

Cincom

SUPRA SERVER PDM

RDM PL/I Programming Guide
(OS/390 & VSE)

P26-8331-62



SUPRA[®] Server PDM RDM PL/I Programming Guide (OS/390 & VSE)

Publication Number P26-8331-62

© 1987, 1991, 1993, 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This document contains unpublished, confidential, and proprietary information of Cincom.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 RDM PL/I Programming Guide (OS/390 & VSE)*, P26-8331-62, 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	ix
Revisions to this manual	x
Conventions	xi
SUPRA Server documentation series	xiv
Overview of PL/I application programming with RDM	17
Application programming overview	18
Using RDM to write PL/I programs.....	21
Understanding RDM views.....	22
Creating user views.....	23
Understanding columns and keys	23
Introduction to the Relational Data Manipulation Language (RDML)	25
Introduction to the DBAID Utility subset	26

Using the DBAID Utility subset	27
DBAID commands list.....	28
DBAID formatting guidelines.....	31
DBAID commands	32
= command.....	32
BYE command.....	33
BY-LEVEL command.....	34
CAUTIOUS command.....	36
COLUMN-TEXT command.....	37
COMMIT command.....	39
DELETE command.....	40
ERASE command.....	42
FIELD-DEFN command.....	43
FORGET command.....	45
GET command	46
GO command	50
INSERT command.....	53
KEEP command	56
LINESIZE command.....	57
MARK command	58
MARKS command.....	59
OPEN command.....	60
PAGESIZE command.....	62
RELEASE command	63
RESET command.....	64
SIGN-OFF command.....	65
SIGN-ON command	66
SURE command.....	67
UPDATE command	68
USER-LIST command.....	70
USERS command	71
VIEW-DEFN command	72
VIEWS command	73
VIEWS-FOR-USER command.....	74

Coding RDM PL/I application programs	75
Using the programmer's report	76
Coding INCLUDE statements	77
Specifying views and user views	77
Specifying TIS_CONTROL	78
RDM status indicators	79
Signing on/off	84
Maintaining storage	85
Retrieving rows using the GET statement	85
Retrieving rows containing unique keys	86
Retrieving rows containing nonunique keys	87
Retrieving rows without keys	87
Accessing multiple views	88
Using the MARK statement	89
Using explicit and automatic record holding	89
Explicit record holding	89
Automatic record holding	90
Handling error conditions	91
Modifying rows	92
Updating rows	92
Deleting rows	93
Using the INSERT statement	94
Using the COMMIT/RESET statements	95
Handling errors requiring a recompile	96
RDM PL/I application program statements	97
INCLUDE statements	98
INCLUDE view-data	98
INCLUDE TIS_CONTROL	102
Coding RDML statements	104
COMMIT	105
DELETE	106
FORGET	108
GET	109
INSERT	116
MARK	119
RELEASE	120
RESET	121
SIGN_OFF	122
SIGN_ON	123
UPDATE	124

Compiling and linking an RDM PL/I application program	125
Executing the RDML precompiler	126
Linking a compiled program	128
OS/390.....	128
VSE.....	128
OS/390 and VSE samples and procedures	129
OS/390 samples and procedures	129
VSE samples	130
Sample RDM PL/I application program	131
Index	135

About this book

Using this document

This manual is for PL/I application programmers who wish to write RDM applications for SUPRA PDM.

Document organization

The information in this manual is organized as follows:

Chapter 1—Overview of PL/I application programming with RDM

Presents an overview of the requirements and considerations that a PL/I programmer needs to be aware of before writing an RDM PL/I program.

Chapter 2—Using the DBAID Utility subset

Describes DBAID utility commands.

Chapter 3—Coding RDM PL/I application programs

Presents requirements and guidelines for coding RDM PL/I application programs.

Chapter 4—RDM PL/I application program statements

Contains format descriptions and usage considerations for the two groups of RDM PL/I program statements.

Chapter 5—Compiling and linking an RDM PL/I application program

Presents information on the RDML precompiler, including instructions for executing the precompiler and linking considerations for each operating system.

Appendix A—OS/390 and VSE samples and procedures

Lists samples and procedures for running certain tasks in OS/390 or VSE environments.

Appendix B—Sample RDM PL/I application program

Displays a sample PL/I application to execute a set of RDML statements.

Index

Revisions to this manual

The following changes have been made for this release:

- ◆ New information has been added regarding the execution of PL/I % statements under “**Executing the RDML precompiler**” starting on page 126.
- ◆ References to CMS have been removed.

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 bbbb 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>	<pre>MONITOR {ON } OFF }</pre>
<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>	<pre>[<u>WAIT</u>] [<u>NOWAIT</u>]</pre> <hr/> <pre><u>STATISTICS</u></pre>
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>	<pre>INTO :host-variable [:ind- variable],...</pre>

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 keystroke. Multiple keystrokes are hyphenated.	ALT-TAB

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 DBA Functions, DBAID, precompilers, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed. For a brief synopsis of each manual, refer to the *SUPRA Server Digest (OS/390 & VSE)*, P26-9062.

Overview

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

Getting started

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

General use

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

Database administration tasks

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

Application programming tasks

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

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561



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



Educational material is available from your regional Cincom education department.

1

Overview of PL/I application programming with RDM

SUPRA is an advanced relational database management system for high-volume update-oriented processing. The Relational Data Manager (RDM) is the SUPRA component which accepts and processes requests from end users and application programmers. RDM retrieves the data needed for an application program while providing database security and integrity and insulating the application program from changes to the database. RDM allows the application programmer to write PL/I programs without knowing about the physical structure of the database.

RDM supports the OS/390 and VSE operating systems. Within these operating systems, it supports the following programming languages:

- ◆ COBOL
- ◆ COBOL II
- ◆ CICS/VS COBOL (Command Level)
- ◆ MANTIS
- ◆ PL/I



The catalogued procedures for running DBAID, the Relational Data Manipulation Language (RDML) precompiler and Directory reports in your operating system are supplied on your tailored installation tape.

Specifically, RDM simplifies the process of writing application programs in the following ways:

- ◆ Application programmers and end users need no knowledge of physical database implementation; RDM provides application programs with data independence
- ◆ RDM allows the DBA to change and restructure the database without requiring the programmer to rewrite or recompile applications
- ◆ Programmers have a simplified RDML for retrieving and modifying the database contents
- ◆ The DBA can control the security of the database on a user-by-user basis

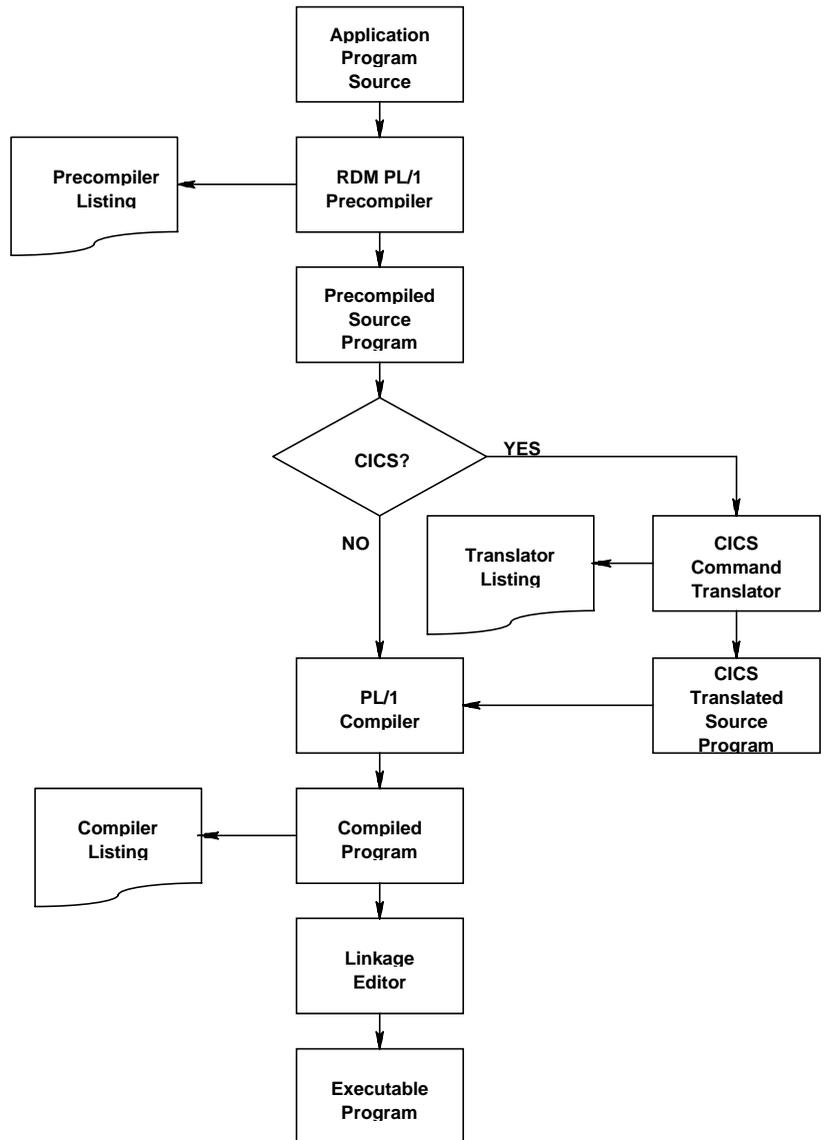
Because RDM is controlled by the DBA who decides how the database can be accessed and who can modify its contents, individual programmers are relieved of maintaining database validity. This results in a simpler access language, programs which are easier to write and debug and, because changes to the database do not affect programs, fewer maintenance tasks.

Application programming overview

This chapter presents an overview of the requirements and considerations that a PL/I programmer needs to be aware of before writing an RDM PL/I program. This chapter addresses:

- ◆ The steps for writing an application
- ◆ Understanding RDM views
- ◆ Introduction to the RDML
- ◆ Introduction to using the DBAID utility subset

The figure on the following page illustrates the PL/I application programming process.



The DBA maintains complete control over the definition and generation of views of the data, determines data needs, and assists in determining the best method of structuring views and relationships. The DBA controls changes made to view definitions and should distribute copies of definitions and changes that affect the application programmer. The responsibilities of the DBA to the application programmer can be summed up as follows:

- ◆ **Defining views.** The DBA defines each view by determining the data that should be in the view and how that data should be accessed. The DBA determines which columns are required or key columns and which contain fixed values. The DBA also determines whether keys are unique or nonunique and the access you have to the data in the row—read only, update, and so on. After creating the view, the DBA defines the view on the Directory.
- ◆ **Changing database contents.** When you use RDML commands to modify a view, you also modify the contents of the database. The DBA has the choice of allowing you to make such modifications to the views. In addition, the DBA controls the ordering of the rows and may, at any time, change the content or column length of a row.
- ◆ **Changing View Design.** The DBA can change the view design at any time. The DBA can alter existing views, add new views, change keys or required columns, and add to or delete information from existing views.
- ◆ **Providing the Programmer's Report.** The DBA can provide you with a RDM PL/I Programmer's Report that tells you the views available for your use. This report indicates any changes that have been made to the views and includes any special instructions for their use. For more detailed information about the RDM PL/I Programmer's Report, see ["Coding RDM PL/I application programs"](#) on page 75.

Using RDM to write PL/I programs

When you write PL/I application programs using RDM, you do so using views assigned by the DBA. Views are defined by the DBA in the Directory by describing the required fields and by providing access to the file(s) containing these fields. When you begin to write your program, the DBA assigns a view that fulfills all of the data requirements and functions of your program.

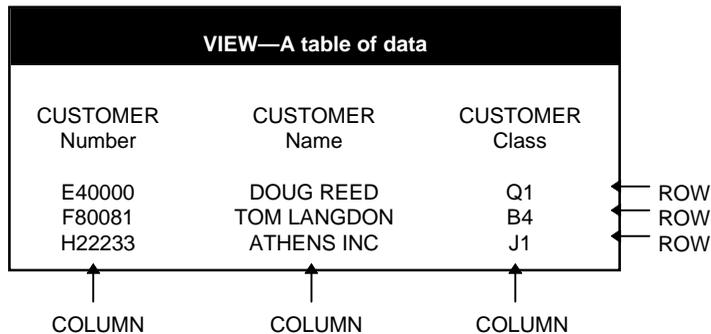
To write effective application programs, you need to understand views. A view is a set of one or more rows describing physical field values within the database. The following terms are integral to a discussion of views:

- ◆ **Row.** A set of one or more related data items stored in computer memory.
- ◆ **Column.** In a row, a specified area used for a particular category of data.
- ◆ **Value.** A quantity assigned to a constant, a variable, a parameter or a symbol.
- ◆ **Key.** One or more data items, the contents of which identify the type or location of a row, or the ordering of data.
- ◆ **User View.** A subset of a view which may consist of all or part of the view.

Understanding RDM views

RDM provides a view of physical fields from one or many files in a flat, two-dimensional format. This set of field values is a row. A view consists of one or more rows. RDM provides a view of data in the form of a table which consists of rows and columns.

In a view, columns are mapped between the externally constructed row and the physical database. During physical navigation of the database, RDM collects certain occurrences of the physical records based on the row definition. RDM then selects the appropriate physical fields and maps them to the constructed row. The following figure illustrates a view:



Picture a set of views stored sequentially as a flat file. You can retrieve records from the file according to relative positioning within the file or by selecting key values. The operations available to RDM (GET, INSERT, UPDATE, and DELETE) are those needed to manipulate the records on an occurrence-by-occurrence basis.

Although a view resembles a flat file, there are two important differences:

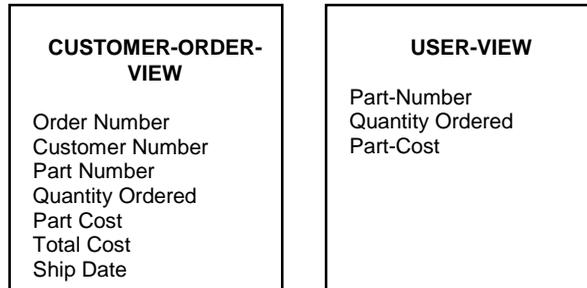
- ◆ The ordering of rows within the file is not always controlled by your maintenance operations
- ◆ Fields (columns) can have null values

The DBA sets up on the Directory views available to you. Refer to the Relational Data Manager PL/I Programmer's Report for a list of views available to you. See "Using the programmer's report" on page 76 for information about this report.

Creating user views

You can subset the row(s) or reorder the column(s) within a view according to your needs. This is done in the application program through RDM language specification. This subset of data is called a user view. Once the DBA has defined the columns included in a view, you can use all or part of the view as a user view.

The following figure illustrates a view and a user view:



Understanding columns and keys

Each view contains one or more columns that the DBA can designate as keys to the view. The keys can appear anywhere in the view. The DBA can define four different types of keys:

- ◆ Unique key
- ◆ Compound unique key
- ◆ Simple nonunique key
- ◆ Compound nonunique key

The simplest view has one unique key. This key value allows you to select and retrieve data. A unique key must have a valid, non-null value.

In a compound unique key, several columns are designated as unique logical keys, and the combination of the key values is unique—an “and” connection between the columns is implied. For example, to check customer orders for a certain part number, you would use a view with both customer number and part number as key values. RDM will retrieve the specific customer number and part number combination if it is present.

A nonunique key allows more than one row to contain the same value in a key column. An example is a customer file where you keep a list of notes or comments concerning each customer. You do not date the comments, and you do not want more than one key; for each customer, you want to retrieve a list of comments that may have been posted. In this case, the customer number could be defined as a single nonunique key. When the program does its first GET using a customer number, it will retrieve the first comment for that customer. A subsequent GET will retrieve the second comment; the third GET, the third comment, and so on. After RDM reaches the last comment for that customer, it will reach a boundary condition and return a NOT FOUND to the program.

A compound nonunique key is an extension of the simple nonunique key. Here, more than one column is defined as a nonunique key. However, all the nonunique keys together still do not completely describe the record occurrence as unique. You can still have more than one record with the same compound nonunique key.

You can access a set of rows by assigning values to the keys of the view (if there are any). The DBA determines which columns are keys and defines them on the Directory. The keys can be used to locate a specific record or to perform a generic read. Both types of reads return all qualifying rows from the views. You can also access a set of rows sequentially by not supplying any values for the keys.

A required column must contain a valid, non-null value in order for the record to be included in the view. By default, a column is not required and does not need a value.

Introduction to the Relational Data Manipulation Language (RDML)

While your application programs are written in PL/I, RDM uses the Relational Data Manipulation Language (RDML) to sign on and off the system, to maintain storage, to manipulate data, and to control data recovery. These functions are accomplished by the following RDML statements:

- ◆ **SIGN-ON/SIGN-OFF.** The SIGN-ON statement establishes communication between the programmer and RDM and identifies you as the user of the system. The SIGN-OFF statement informs RDM that you want to terminate your session.
- ◆ **RELEASE/FORGET.** The RELEASE and FORGET statements free internal storage without signing off the system.
- ◆ **DELETE.** Removes a row from the view.
- ◆ **GET.** Retrieves a row from the view.
- ◆ **INSERT.** Inserts a new row into the view.
- ◆ **UPDATE.** Updates column values in an existing row.
- ◆ **COMMIT/RESET.** Control database recovery. These statements function differently depending on the environment and recovery system supported.

Using the Directory to supply working storage, RDML statements are converted into standard PL/I source code by the RDML Compiler. Standard compilers then convert the PL/I source code into object code. When the program executes, the Directory uses the physical data descriptions, and RDM uses the logical data descriptions, to access the database and present the data in the view requested by the application program.

For information on using the RDML, see [“Coding RDM PL/I application programs”](#) on page 75; for the syntax of the RDML statements, see [“RDM PL/I application program statements”](#) on page 97.

Introduction to the DBAID Utility subset

The DBAID Utility, an online and batch tool, allows the DBA to define a new view, open the view, issue RDML statements and examine the results. The DBA can then change the view, if necessary, reorder for efficiency, or try different access methods. These activities have no impact on the Directory.

Certain DBAID commands are also available to non-DBA users. With this subset of commands, you can use the DBAID Utility when constructing programs that use views. You can use this subset to test a view before actual production runs. You can run DBAID in a batch or online environment to make sure the view fits your particular requirements. With DBAID, you can run test cases until you are sure that the view is correctly defined. You can also use the DBAID subset as an educational tool for immediate hands-on experience with the views being accessed.

DBAID has three command categories:

- ◆ System commands
- ◆ Built-in view commands
- ◆ RDML commands

System commands display information about the DBAID Utility currently executing. Use system commands to display current users and active views. Built-in view commands allow you to inspect the view after it is opened. RDML commands let you use test data with a defined view to make sure the view has been properly defined.

See [“Using the DBAID Utility subset”](#) on page 27 for more information on the DBAID subset of commands available to you.

2

Using the DBAID Utility subset

A subset of the DBAID utility commands is available to the application programmer to use for testing a view before actual production runs. To make sure a view fits your specific requirements, DBAID can be run in a batch or online environment. Using DBAID, you can run test cases until you are satisfied that the view is correctly defined.

The DBAID utility subset has three command categories:

- ◆ **System commands.** These commands display information about the currently executing DBAID Utility such as current users and active views.
- ◆ **Built-in view commands.** Use these commands to inspect a view after it is opened.
- ◆ **RDML commands.** Use RDML commands to test data with a defined view to ensure the view has been properly defined.

DBAID commands list

The following table lists all the commands available to you by category with a brief description and a section reference for detailed information.

Some DBAID Utility subset commands have specific underlying, file-system restrictions. For more information on the restrictions for PDM file systems, refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221. For information on VSAM restrictions, refer to the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222.

Command	Description	Section
System commands		
LINESIZE	Specifies line width for DBAID output	" LINESIZE command " on page 57
MARKS	Lists all open MARKs and the views they are marking	" MARKS command " on page 59
PAGESIZE	Specifies the number of lines on the page/screen for DBAID output	" PAGESIZE command " on page 62
USER-LIST	Displays column list for the view named	" USER-LIST command " on page 70
USERS	Displays the current users of the system	" USERS command " on page 71
VIEWS	Displays all views active in DBAID	" VIEWS command " on page 73

Command	Description	Section
Built in view commands		
BY-LEVEL	Displays the column names in the view by level of occurrence	"BY-LEVEL command" on page 34
COLUMN-TEXT	Displays the short and long text for a column in a view. (You can also code this as FIELD-TEXT.)	"COLUMN-TEXT command" on page 37
FIELD-DEFN	Displays the full description of a column in a view	"FIELD-DEFN command" on page 43
VIEW-DEFN	Displays a condensed description of the view	"VIEW-DEFN command" on page 72
VIEWS-FOR-USER	Lists the views related to the signed-on user and the short text for the view	"VIEWS-FOR-USER command" on page 74
RDML commands		
=	Reissues previous RDML command	"= command" on page 32
BYE	Causes the DBAID Utility to exit	"BYE command" on page 33
CAUTIOUS	Prohibits an automatic COMMIT	"CAUTIOUS command" on page 36
COMMIT	Makes all updates since the last commit permanent in the database	"COLUMN-TEXT command" on page 37
DELETE	Removes a View Record occurrence from database	"DELETE command" on page 40
ERASE	Issues an RDM RESET if an 'X' FSI is returned	"ERASE command" on page 42
FORGET	Frees the storage allocated by a previously issued MARK command	"FORGET command" on page 45
GET	Retrieves and displays the requested row for the view indicated	"GET command" on page 46
GO	Issues multiple GET commands and displays the rows in tabular format	"GO command" on page 50
INSERT	Places a view row in the physical database on relative location specified	"INSERT command" on page 53

Command	Description	Section
RDML commands (cont.)		
KEEP	Prohibits an automatic RESET	"KEEP command" on page 56
MARK	Marks the current position of the view name established by the previous GET	"MARK command" on page 58
OPEN	Readies either a virtual or stored view for use by the DBAID Utility	"OPEN command" on page 60
RELEASE	Closes one or all views that have been opened and releases the occupied storage	"RELEASE command" on page 63
RESET	Forces task-level abend and rolls back any database updates since the last commit	"RESET command" on page 64
SIGN-OFF	Signs off a user from the DBAID Utility	"SIGN-OFF command" on page 65
SIGN-ON	Identifies a user to the DBAID Utility	"SIGN-ON command" on page 66
SURE	Causes a COMMIT after each successful INDERST, UPDATE, or DELETE	"SURE command" on page 67
UPDATE	Updates data values in the database	"UPDATE command" on page 68

DBAID formatting guidelines

DBAID format is a series of commands, with one command per line and a maximum of 72 characters per command. The guidelines for formatting DBAID in a batch environment or in an online environment are the same except that batch output is to a line printer.

Several DBAID syntax options simplify the use of DBAID:

- ◆ The FOR option used with the GO command (see “GO command” on page 50)
- ◆ The := syntax used with the UPDATE command (see “UPDATE command” on page 68)
- ◆ The MASS option used with the INSERT command (see “INSERT command” on page 53)

You can use an asterisk (*) in DBAID for two functions. An * entered in column 1 denotes a comment line. For example:

```
OPEN VIEW
GET VIEW                               Performs GET on VIEW
*THIS IS A TEST VIEW
```

You can also use the * in a command as a substitute for the last view name used. For example:

```
OPEN VIEW
GET *                                   Performs GET on VIEW
OPEN VIEW2 = * FIELD1, FIELD5          Performs OPEN of user view
                                       VIEW2 where VIEW2 is created
                                       from VIEW1 specifying columns
                                       FIELD1, FIELD5 only.
GET *                                   Performs GET on VIEW2
```



Using the * as a substitute for the last view name used is described in each supported command’s explanation.

DBAID commands

The following sections describe the individual DBAID commands. Each section contains a description of the command's format, a list of considerations for using the command, if necessary, and a coding example or the expected output.

= command

The = command reissues the previous RDML command.

=

Example

In the following example, = causes another GET NEXT CUST-PROD.

```
GET NEXT CUST-PROD
```

```
=
```

BYE command

The BYE command causes you to exit the DBAID Utility.

BYE

General considerations

- ◆ In an online environment, the BYE command returns you to the RDM sign-on screen or other user-installed menu screens.
- ◆ If you entered DBAID with the task already signed-on to RDM, the BYE command does not perform a **SIGN-OFF**. If you entered DBAID with the task signed-off from RDM, the BYE command performs a sign-off.
- ◆ In a batch environment, the BYE command terminates the task.
- ◆ DBAID erases any unsaved virtual views.

BY-LEVEL command

The BY-LEVEL command displays the column names in a view by level of occurrence, starting with the 0 level, followed by level 1, and so on. RDM generates the column number when displaying this data.

BY-LEVEL [*view-name* [*column-number*]

view-name

Description *Optional.* Specifies the name of the view whose column names you want to display.

Format Must be a valid, open view.

Considerations

- ◆ If you omit this parameter, the BY-LEVEL command displays all column names for all of your open views.
- ◆ You can enter an * instead of a view name. This substitutes the last view-name used.

column-number

Description *Optional.* Specifies the number of the column whose name is to be displayed.

Format Numeric characters

Considerations

- ◆ If you use this parameter, you must have specified a view name.
- ◆ If you omit this parameter, the BY-LEVEL command displays all column names of the specified view.

Example

BY-LEVEL			
NUMBER	VIEW NAME	COLUMN NAME	LEVEL
1	CUST-PROD	CUST-NO	0
2	CUST-PROD	PROD-NO	1
3	CUST-PROD	RENT	1
4	CUST-PROD	MAINT	1
5	CUST-PROD	INSTALL-DATE	1
6	CUST-PROD	CANCEL-DATE	1
7	CUST-PROD	PURCHASE-PRICE	1
1	CUSTOMER	CUST-NO	0
2	CUSTOMER	NAME	0
3	CUSTOMER	STATE	0
1	TEST	ZONED5	1
2	TEST	PACKED5	1
3	TEST	KEY2	1

CAUTIOUS command

The CAUTIOUS command prohibits an automatic COMMIT. This command is the opposite of the SURE command. DBAID does not issue a COMMIT when an RDML modifying command returns an '** FSI. Instead, you must issue the COMMIT.

CAUTIOUS

General consideration

DBAID normally issues a COMMIT after every successful RDML modification. The CAUTIOUS command is not required; however, you can use it when you want manual control over COMMIT commands when updating the database.

COLUMN-TEXT command

The COLUMN-TEXT command displays the short and long text for a column in a view.

COLUMN-TEXT [*view-name* [*column-name*]]

view-name

Description *Optional.* Specifies the view to be used.

Format Must be a valid, open view.

Considerations

- ◆ If you omit this parameter, the COLUMN-TEXT command displays all column descriptions for all of your open views.
 - ◆ You can enter an * instead of a view name. This substitutes the last view-name used.
-

column-name

Description *Optional.* Identifies the column whose text is to be displayed.

Format The column must already be a part of the view.

Considerations

- ◆ If you use this parameter, you must have specified a view name.
- ◆ If you omit this parameter, the COLUMN-TEXT command displays the short and long text for all columns.
- ◆ You can substitute FIELD-TEXT as the command for COLUMN-TEXT.

Example

```
COLUMN-TEXT CUST-PROD PROD-NO
VIEW NAME          COLUMN NAME
-----
CUST-PROD          PROD-NO
-----
                SHORT TEXT
PRCU-PROD-NUM SHORT TEXT
67890123456789012345678901234567890123456789012
-----
                LONG TEXT
-----
PRCU-PROD-NUM LONG TEXT 100
PRCU-PROD-NUM LONG TEXT 200
PRCU-PROD-NUM LONG TEXT 300
12345678901234567890123456789012345678901234567890123456789012
```

COMMIT command

The COMMIT command makes all updates since the last COMMIT permanent in the database.

COMMIT

General consideration

DBAID issues a COMMIT after every successful RDML modification unless you have issued a **CAUTIOUS** command. You can use the COMMIT command if you have issued a CAUTIOUS command.

DELETE command

The DELETE command removes a view row from the database.

DELETE [ALL] *view-name*

ALL

Description *Optional.* Deletes all rows retrieved by automatically generated GET NEXTs using the logical-key qualification of the GET command issued before the DELETE.

Consideration If a program specifies a GET without a USING phrase, DELETE ALL deletes all rows in a view.

view-name

Description *Required.* Identifies the name of the view containing the row(s) to be deleted.

Format Must be a valid, open view.

General considerations

- ◆ Before performing the DELETE, you must perform a successful GET command.
- ◆ You can enter an * instead of a view name. This causes DBAID to substitute the last view-name used.

Examples

- ◆ This example deletes the one occurrence of SAMPLE-VIEW you obtained based on the value in KEY1:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE SAMPLE-VIEW
```

- ◆ This example deletes all occurrences of rows you obtained based on the value in KEY1:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE ALL SAMPLE-VIEW
```

- ◆ The previous code processes in the manner shown in this code:

```
GET FIRST SAMPLE_VIEW FOR UPDATE USING KEY1;
MORE DELETE SAMPLE_VIEW;
GET NEXT SAMPLE_VIEW FOR UPDATE USING KEY1
    NOT FOUND GOTO CONTINUE.
GOTO MORE;
DONE:    .
        .
        .
```

ERASE command

The ERASE command causes DBAID to automatically issue an RDM **RESET** if an RDML command results in an 'X' FSI. This command is the opposite of the **KEEP** command and causes RDM to automatically issue a RESET if an 'X' FSI is returned.

ERASE

FIELD-DEFN command

The FIELD-DEFN command displays the full description of columns in a view.

FIELD-DEFN [*view-name* [*column-name*]]

view-name

Description *Optional.* Specifies the view to be used.

Format Must be a valid, open view.

Considerations

- ◆ If you omit this parameter, the FIELD-DEFN command displays all column descriptions for all your open views.
- ◆ You can enter an * instead of a view name. This causes DBAID to substitute the last view name used.

column-name

Description *Optional.* Identifies the column whose description is to be displayed.

Format The column must already be a part of the view.

Considerations

- ◆ If you use this parameter, you must specify a view name.
- ◆ If you omit this parameter, the FIELD-DEFN command displays all column descriptions of the specified view.

Example

```
FIELD-DEFN
VIEW-NAME          (+) CUSTOMER
COLUMN-NAME        (+) CUST-NO
COLUMN-POS         (+) 0
COLUMN-LEN         (+) 5
ASI-POS            (+) 60
COLUMN-DEC         (+) 0
OUTPUT-LEN        (+) 5
MASK-LEN          (+) 15
FORMAT            (+) Z
EDIT-MASK         (+) ZZZZZZZZZZZZZZZ9
READING           (+) CUST;NO
DELETABLE         (+) Y
INSERTABLE        (+) Y
REPLACEABLE       (+) N
COLUMN-LVL        (+) 0
KEY-NUMBER        (+) 1
REQUIRED          (+) Y
UNIQUE            (+) Y
EDIT-TRANS        (+) E
ORDERING          (-)
***MORE***
```

FORGET command

The FORGET command frees the storage allocated by a previously issued **MARK** command.

FORGET *mark-name*

mark-name

Description *Required.* Specifies the mark information that should be forgotten.

Format 1–30 alphanumeric characters

Consideration Must be a name you assigned with the MARK command.

General consideration

Once you issue a FORGET command, you release the indicated mark and cannot regain it without issuing a new MARK command.

GET command

The GET command retrieves and displays a row for the indicated view.

```
GET [ NEXT  
    LAST  
    SAME view-name  
    FIRST  
    PRIOR ]  
  
[ FOR UPDATE ]  
[ AT mark-name ]  
[ USING literal1 [ literal2 ... literaln ] ]
```

```
[ NEXT  
  LAST  
  SAME  
  FIRST  
  PRIOR ]
```

Description *Optional.* Modifies the order of row retrieval.

Default NEXT If no current position exists, NEXT defaults to FIRST.

Considerations

- ◆ For a unique key:
 - GET NEXT retrieves either the row immediately after the current row or the first row, if no current position exists.
 - GET LAST retrieves the last row.
 - GET SAME retrieves the latest row if a current position exists.
 - GET FIRST retrieves the first row in the view.
 - GET PRIOR retrieves either the row immediately before the current row or the last row, if no current position exists.
 - Use GET PRIOR only in connection with a USING KEY phrase for predictable results.
 - If the underlying file system cannot perform the GET PRIOR and GET LAST functions, an error is returned.
- ◆ For a nonunique key:
 - GET NEXT retrieves the next occurrence of the row within the generic group.
 - GET LAST retrieves the last occurrence of the row.
 - GET SAME retrieves the latest row if a current position exists.
 - GET FIRST retrieves the first occurrence of the row with the indicated key.
 - GET PRIOR will perform a read reverse within the group of nonunique keyed rows.

view-name

Description *Required.* Specifies the view to be used.

Format Must be a valid, open view.

Consideration You can enter * instead of a view name. This causes DBAID to substitute the last view name used.

FOR UPDATE

Description *Optional.* Allows you to lock out other users' modifications to the row you are retrieving.

Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications dependent on the current contents of the row.
- ◆ If you do not need to be certain of the content of the row, you can use a GET without the FOR UPDATE phrase. When you use an **UPDATE** or **DELETE** function, the automatic-hold facility of the system performs the lock before modifying the row.
- ◆ FOR UPDATE implies that all physical resources will be locked until you issue another GET or an **INSERT**, **UPDATE**, **DELETE**, **COMMIT**, or **RESET**. This practice may lead to system inefficiency.
- ◆ When you issue a GET, RDM can return data to you that is currently being updated by another task. If you subsequently issue a FOR UPDATE, that update may fail in the following ways:
 - The other task has not yet committed. The update will fail with an FSI=U status and message VIEW HELD BY ANOTHER TASK - RETRY LATER.
 - The other task committed between the GET and the update. The update will fail with an FSI=D and the message COLUMN VALUES HAVE BEEN CHANGED.

AT *mark-name*

Description *Optional.* Repositions a view previously marked with the **MARK** command.

Consideration The USING and AT phrases cannot be used with the same GET command.

USING *literal1*[*literal2* ... *literaln*]

Description *Optional.* Identifies a value or set of values to be used for a keyed GET.

Format Either character, hexadecimal, or numeric data. Character and hexadecimal data must be enclosed in quotes; numeric data does not. For example:

<code>USING 'ABCD'</code>	Character data
<code>USING 1234</code>	Numeric data
<code>USING X'A10C'</code>	Hexadecimal
<code>USING 123 'ABC'</code>	Combination (two keys)

Considerations

- ◆ The number of keys specified in the GET statement must be less than or equal to the number of keys in your specified column list. No more than nine keys are allowed in one view.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys is a convenient feature for allowing both direct access to a view and a sequential scan of many rows. RDM returns all occurrences of a particular, unspecified column as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your column list.
- ◆ The USING and AT phrases cannot be used with the same GET command.

GO command

The GO command issues a penetration **GET** request followed by a series of sweeping GET requests and displays the rows in tabular format.

GO $\left[\begin{array}{l} \text{NEXT} \\ \text{PRIOR} \end{array} \right]$ *view-name*

$\left[\begin{array}{l} \text{START} \left\{ \begin{array}{l} \text{NEXT} \\ \text{LAST} \\ \text{SAME} \\ \text{FIRST} \\ \text{PRIOR} \\ \text{AT } \textit{mark-name} \end{array} \right\} \end{array} \right]$

FOR *number-of-rows*

$\left[\left\{ \begin{array}{l} \text{FROM} \\ \text{USING} \end{array} \right\} \textit{literal}_1(\textit{literal}_2 \dots \textit{literal}_n) \right]$

$\left[\begin{array}{l} \text{NEXT} \\ \text{PRIOR} \end{array} \right]$

Description *Optional.* Specifies the **GET** command modifier to be used in retrievals after the initial penetration.

Default NEXT

view-name

Description *Required.* Specifies the view to be accessed.

Format Must be a valid, open view.

Consideration You can enter an * instead of a view name. This causes DBAID to substitute the last view name used.

START {
 NEXT
 LAST
 SAME
 FIRST
 PRIOR
 AT *mark-name*}

Description *Optional.* Specifies the **GET** command modifier to be used for the initial penetration of the database.

Default FIRST

FOR *number-of-rows*

Description *Optional.* Indicates the number of rows (or the number of GET NEXTs minus 1) to be performed.

Format Numeric characters

Consideration GET NEXTs repeat until the count is exhausted or until the last row is retrieved, whichever occurs first.

[{
 FROM
 USING } *literal*₁ [*literal*₂ ... *literal*_n]]

Description *Optional.* Identifies a value or set of values to use for a keyed **GET**.

Format Either character or numeric data. Character data, if it includes blanks, must be enclosed in quotes; numeric data does not.

Options FROM Uses the key values only on the initial penetration; the scan is unqualified.

USING Uses the key values for both the initial penetration and the subsequent scan.

Considerations

- ◆ The number of keys specified in the **GET** statement must be less than or equal to the number of keys in your specified column list.
- ◆ RDM treats any omitted keys as generic keys. The use of generic keys allows for both direct access to a view and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified key as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your column list.

General considerations

- ◆ RDM displays the output in columnar fashion. If more data is to be displayed than will fit on a screen/page, RDM uses an alternate format.
- ◆ After the GO command displays a page of rows (see “**PAGESIZE command**” on page 62), the “****MORE****” prompt will be issued. Enter a blank line for each additional page in batch mode, and press ENTER for each additional page in online mode.
- ◆ At the end of the series of rows retrieved by GO, the “****END****” prompt is issued.
- ◆ Do not use “For number-of-rows” for online because it does not pause until the last screen.
- ◆ The GO command always looks ahead one more row so it can determine whether to display the ****MORE**** or ****END**** message. It can be confusing if you issue a **GET** after the GO because a row may appear to have been skipped.

INSERT command

The INSERT command places a view row in the physical database based on the relative location specified.

```
INSERT [ NEXT
        LAST
        FIRST
        PRIOR ] view-name [MASS]
```

```
NEXT
LAST
FIRST
PRIOR
```

Description *Optional.* Specifies the relative location of the row to be inserted in relation to existing rows. The Access Set Description (ASD) may override this specification.

Default NEXT If not positioned in the view, NEXT defaults to LAST, and PRIOR defaults to FIRST.

Considerations For nonuniquely keyed values:

- ◆ INSERT FIRST places a row in the first position in the view.
- ◆ INSERT NEXT places a row after the current row. If no current position exists, the row is placed in the last position in the view.
- ◆ INSERT PRIOR places a row before the current row. If no current position exists, the row is placed in the first position in the view.
- ◆ INSERT LAST places a row in the last position of the view.
- ◆ If the DBA specified ordering in the view definition, or if the Physical Data Manager (PDM) does not allow program control of ordering, the specification on the INSERT statement is ignored.

view-name

- Description** *Required.* Specifies the name of the view in which you want the rows inserted.
- Format** Must be a valid, open view.
- Consideration** You can enter an * instead of a view name. This causes DBAID to substitute the last view-name used.
-

MASS

Description *Optional.* Allows multiple rows to be inserted in the physical database.

Considerations

- ◆ The positioning parameter specified (NEXT, LAST, FIRST, or PRIOR) is used on every RDM INSERT command issued by MASS insert.
- ◆ Views are entered immediately following this command after the two prompt lines, “MASS INSERT PROCESSING INITIATED” and “ENTER ‘END.’ TO EXIT MASS INSERT,” are displayed (see the second example, below).
- ◆ Rows are inserted as flat records. Separate the column values with commas. To insert rows longer than one line, terminate the list of values with a comma and continue the input on the next line.
- ◆ Place multiple rows on a single line by leaving a blank between rows.
- ◆ Use a pair of single quote marks to insert columns containing spaces.
- ◆ If you have columns with no values, enter two consecutive commas to indicate their absence. This value is treated as a null value for packed or zoned fields, as a large number (X'40404040' or 67372036 integer) for binary fields, and as blanks for a character field.
- ◆ Specify END. after entering all rows to be inserted into the view. The period after END is required.

General considerations

- ◆ After entering column values on a single insert (not using MASS), the row is displayed. The message INSERT (Y/N)? appears. Enter a Y to insert the row. Any other response does not insert the row.
- ◆ Processing stops if 10 errors are detected while using the MASS insert; otherwise, enter END. to terminate insert processing.

Examples

◆ Example of a single INSERT:

```

> INSERT *
NUMBER
> 9998
PRODUCT
> AAAA
INSTALLED
> 100883
NUMBER ( ) 9998
PRODUCT ( ) AAAA
INSTALLED ( ) 100883
INSERT (Y/N)?
> Y
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION

```

◆ Example of a MASS INSERT (single row):

```

> INSERT * MASS
MASS INSERT PROCESSING INITIATED.
ENTER "END." TO EXIT MASS INSERT.
> 9997,BBBB,100783
FSI: * VSI + MSG: SUCCESSFUL COMPLETION
* Example of MASS insert (using comma to continue to next
line):
> 9996,CCCC,
> 100683
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
* Example of a MASS insert (multiple rows on a single line):
> 9995,DDDD,100583 9994,EEEE,100483 9993,FFFF,100383
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
* Example of ending the MASS insert processing:
> END.
MASS INSERT PROCESSING COMPLETED.

```

KEEP command

The KEEP command prohibits an automatic **RESET**. This command is the opposite of the **ERASE** command. KEEP causes DBAID not to issue a RESET when it receives an 'X' FSI from RDM. Instead, DBAID keeps the database as it is and allows the user to decide to RESET or not.

KEEP

LINESIZE command

The LINESIZE command specifies the number of characters to display in a line.

LINESIZE [*number-of-characters*]

number-of-characters

Description *Optional.* Indicates the number of characters to display on a line.

Default 79 characters per line

Format 2–3 numeric characters

Options 10–132

Considerations

- ◆ In an online environment, the line size maximum is restricted to the line capacity of the screen.
- ◆ If you omit *number-of-characters*, the command displays the current LINESIZE setting.

MARK command

The MARK command marks the current position of the view row established by the previous **GET** command.

MARK *view-name* AT *mark-name*

view-name

- Description** *Required.* Identifies the view name established by the previous **GET** command.
- Format** Must be a valid, open view.
- Consideration** You can enter an * instead of a *view-name*. This substitutes the last view-name used.
-

AT *mark-name*

- Description** *Required.* Assigns a name to the location where the position of the current view will be marked.
- Format** 1–30 alphanumeric characters
- Consideration** The name assigned is the name you use in a later GET AT request to retrieve this same view row.
-

General considerations

- ◆ Use the AT phrase in the **GET** command to reposition the view at the position set by the MARK command.
- ◆ You can create any number of MARKs for a view, but to save internal memory space, it is best to reuse *mark-name* when possible.
- ◆ A *mark-name* may be reused.
- ◆ The number of MARKs you can create is limited by the amount of internal memory space allocated to your task.

MARKS command

The MARKS command lists all open **MARKS** and the views they are marking.

MARKS

Example output

MARKS	MARK NAME	VIEW NAME
MARK6		CUST-PROD
MARK5		CUST-PROD
MARK4		CUST-PROD
MARK3		CUST-PROD

OPEN command

The OPEN command readies a saved or virtual view for use by DBAID.

OPEN [*user-view-name*=]*view-name*[*column1*[,...,*columnn*]]

user-view-name=

- Description** *Optional.* Gives an existing view a name for use in DBAID.
- Format** 1–30 alphanumeric characters and the special characters # and \$. The first character must be alphabetic or a special character. If the first character is a special character, the second character must be alphabetic.

Considerations

- ◆ If *user-view-name* is not specified, it will be the same name as the view name.
 - ◆ You can use this method (together with the column parameter) to create many smaller views from one common view.
 - ◆ To OPEN a view that has not been listed or defined in the same session of DBAID, the user must be related to the view in the Directory.
-

view-name

- Description** *Required.* Identifies the virtual or stored view to be readied for use.
- Format** Must be a valid view.

Considerations

- ◆ You can enter an * instead of a *view-name*. This substitutes the last view-name used.
- ◆ The list of column names can be continued on successive lines by ending the line you are entering with a comma. The command **USER-LIST** displays the list of columns used to open the view after it has been opened.
- ◆ Issuing an OPEN request on a view without first issuing a LIST request causes RDM to directly open the view with the user relations checked but without text available to DBAID.
- ◆ If a virtual view has the same name as a saved view, the virtual view is used.

column1[,...,columnn]

Description *Optional.* Identifies the column or list of columns to be included in the user view. If omitted, all columns in the view are in the user view.

Format The columns must already be part of the view being opened.

General consideration

The OPEN returns information about the storage used in the form of the message:

```
nnnnn BYTES USED IN OPENING VIEW
```

where *nnnn* is the amount of storage used by the view.

Example

```
OPEN CP-ONLY = CUST-PROD CUST-NO, PROD-NO
```



Only CUST-NO and PROD-NO are returned when you do GET CP-ONLY, even though CUST-PROD has six defined columns.

PAGESIZE command

The PAGESIZE command specifies the number of lines to display on a screen/page.

PAGESIZE [*number-of-lines*]

number-of-lines

Description *Optional.* Indicates the number of lines to display on a screen/page.

Default 24 lines

Options Must be greater than 10, with no maximum limit.

Considerations

- ◆ In an online environment, the PAGESIZE maximum should be no more than the screen capacity.
- ◆ If you omit *number-of-lines*, the command displays the current PAGESIZE setting.

RELEASE command

The RELEASE command closes a specific view or all views that are open, and releases the occupied storage within RDM.

RELEASE [*view-name*]

view-name

Description *Optional.* Specifies the view to release.

Format Must be a valid, open view.

Considerations

- ◆ You can enter an * instead of a view name. This substitutes the last view-name used.
- ◆ If you omit this parameter, the RELEASE command releases all of your open views.

General consideration

This command does not affect virtual view text of the view(s).

RESET command

The RESET command rolls back any database updates since the last COMMIT point.

RESET

General considerations

- ◆ Use RESET only after unsuccessful RDML updates. DBAID does not automatically issue a RESET command when an 'X' FSI is returned. See "KEEP command" on page 56 and "ERASE command" on page 42.
- ◆ In CICS, a RESET backs out any database updates since the last COMMIT point but does not restart DBAID.
- ◆ The RESET command restores your database to the last COMMIT point and you lose position on all views. Therefore, the GET SAME, DELETE, or UPDATE commands are not valid after a RESET. A GET NEXT command positions you on the first record while a GET PRIOR command positions you on the last record after a RESET.

SIGN-OFF command

The SIGN-OFF command signs off the user from DBAID.

SIGN-OFF

General consideration

Use the SIGN-OFF command to remove yourself as a user without terminating DBAID.

SIGN-ON command

The SIGN-ON command identifies the user to DBAID.

SIGN-ON *user-name* [*password*]

user-name

Description *Required.* Indicates the name of the user.

Format Must be a valid user name already defined on the Directory.

password

Description *Optional.* Indicates the user's password.

Format Must be a valid password defined on the Directory.

General considerations

- ◆ In an online environment, initializing DBAID completes the SIGN-ON before you enter DBAID and need not be repeated.
- ◆ In a batch environment, DBAID blanks the password field before printing the output.

Example

```
SIGN-ON JDOE PRGMPSWD
```

SURE command

The SURE command causes a **COMMIT** after each successful **INSERT**, **UPDATE**, or **DELETE**. The SURE command is the opposite of the **CAUTIOUS** command and causes RDM to automatically issue a **COMMIT** if an '** FSI that alters the database is returned by an RDML command. SURE is the default.

SURE

UPDATE command

The UPDATE command updates data values in the database.

UPDATE *view-name*

[*column1:=literal1[,...,columnn:=literaln]*]

view-name

Description *Required.* Identifies the view you wish to update.

Format Must be a valid, open view.

Consideration You can enter an * instead of a view name. This causes DBAID to substitute the last view-name used.

column1:=literal1[,...,columnn:=literaln]

Description *Optional.* Identifies a column in the view that is to have the value of the literal.

Format *column* The column must already be part of the view being updated.

:= Must be coded as shown.

literal Character or numeric data. A hexadecimal value is not allowed.

Considerations

- ◆ In an online environment, DBAID displays each updateable column and accepts replacement values. Entering a null line leaves the column unchanged; entering new data replaces the column value in the row. After all updateable columns are processed, DBAID displays the “UPDATE (Y/N)” prompt and requires a response.
- ◆ In a batch environment, use the column:= literal syntax when updating columns in the row. DBAID updates only the columns you specify; all others remain the same. To update a row, indicate the column you want to update, the :=, and the new value for the column.
- ◆ Do not use single quotes around character or numeric literals.
- ◆ Use single quotes to change the value of a column to blanks. A literal of spaces (keyed in) must be in single quotes. Pressing ENTER does not affect the column’s value.
- ◆ You cannot use the UPDATE function to modify key column values.
- ◆ To UPDATE a row, you must first retrieve the row using the GET command.
- ◆ You cannot use UPDATE to change all the values in a defined column to a specific value—you cannot change all PROD-CODES to “T100.”

Example

```
UPDATE CUST-PROD RENT: = 175.00, MAINT = 50.00
```

USER-LIST command

The USER-LIST command displays the column list for the user view named.

USER-LIST *view-name*

view-name

Description *Required.* Identifies the view or user view to display.

Format Must be a valid view.

Consideration You can enter an * instead of a *view-name*. This substitutes the last view-name used.

Example output

```
USER-LIST PO-CODE-ONLY
USER VIEW NAME :    PO-CODE-ONLY
VIEW NAME : CUSTOMER-PURCHASE-ORDER
USER VIEW LIST :
CUST-NO , PURCHASE-ORDER-CODE , END.
```

USERS command

The USERS command displays information about the current users of the system.

USERS

General considerations

- ◆ The information displayed with this command includes:
 - Station Number—The number of the user's station.
 - User Name—The name of the user for that station.
 - Time of Sign-on—The sign-on time of that user.
 - Processing Time—The total CPU time that user has used.
 - Request Count—The number of requests that user has issued.
 - Duration of Last Request—The duration of the user's last request.
- ◆ This command is operational only in the online environment.

Example output

In a non-CICS system, a USERS display looks like this:

STN.#	USER NAME	REQ. #	I/O TIME	SIGN-ON
LAST REQ.				
102	Character Name of User	572	12:05:32	12:06:50
09:02:35				

VIEW-DEFN command

The VIEW-DEFN command displays a condensed description of a view.

VIEW-DEFN [*view-name*]

view-name

Description *Optional.* Specifies the view whose condensed description you want to display.

Format Must be a valid, open view.

Considerations

- ◆ You can enter an * instead of a *view-name*. This causes DBAID to substitute the last view-name used.
- ◆ Omitting this parameter displays a condensed description of all your open views.

Example

```
> VIEW-DEFN
VIEW-NAME          (+) CUSTOMER
INS-ORDER          (+) N
TOTAL-SIZE         (+) 63
TOTAL-COLUMNS    (+) 3
TOTAL-LEVELS      (+) 1
TOTAL-DELETABLE   (+) 3
TOTAL-INSERTABLE  (+) 3
TOTAL-REPLACABLE  (+) 3
TOTAL-REQUIRED    (+) 1
TOTAL-KEYS        (+) 1
TOTAL-NONUNIQUE   (+) 0
***MORE***
```

VIEWS command

The VIEWS command displays all of the views currently active in DBAID.

VIEWS

General consideration

The information displayed with this command includes:

- ◆ **User view.** The name of the user view.
- ◆ **View.** The name of the view of which this user view is a part.
- ◆ **Status.** Indicates whether the user view is open or released.

Example output

USER VIEW	VIEW	STATUS
CUSTOMER-PURCHASE-ORDER	CUSTOMER-PURCHASE-ORDER	OPENED
PO-CODE-ONLY	CUSTOMER-PURCHASE-ORDER	OPENED

VIEWS-FOR-USER command

The VIEWS-FOR-USER command lists the names and short text for the views related to the signed-on user.

VIEWS-FOR-USER

Example output

```
> VIEWS-FOR-USERS
VIEW NAME          DATE          TIME
          SHORT DESCRIPTION
-----
CUST-QUERY          08/17/99     17:27:32
-----
PROD-QUERY          08/17/99     17:36:22
-----
PRCU-QUERY          08/17/99     17:30:18
-----
EDUC-QUERY          08/17/99     17:33:20
-----
INST-QUERY          03/16/99     17:45:39
-----
SYST-QUERY          10/19/99     13:34:58
.....
***MORE***
```

3

Coding RDM PL/I application programs

This chapter presents the requirements and general guidelines for coding RDM PL/I application programs. RDM PL/I application programs are made up of two types of statements:

- ◆ INCLUDE statements
- ◆ RDML statements

The RDML statements and their associated optional phrases are written as part of the source language. Each RDML statement starts on a line by itself, ends with a semicolon, and can have a label.

This chapter addresses the following RDM PL/I programming topics:

- ◆ [Using the programmer's report](#)
- ◆ [Using INCLUDE statements](#)
- ◆ [Signing on/off](#)
- ◆ [Maintaining storage](#)
- ◆ [Retrieving rows](#)
- ◆ [Modifying rows](#)
- ◆ [Controlling database recovery](#)
- ◆ [Handling error conditions](#)

After you code your application programs, the RDML compiler converts the statements into the proper set of assignments and CALL statements to utilize RDM. Each RDML statement must start on a line by itself and must end with a period. Under CICS, you must use command level PL/I; macro level PL/I is not supported. See [“Compiling and linking an RDM PL/I application program”](#) on page 125 for information on using the RDML Compiler and for linking the compiled program. See [“Sample RDM PL/I application program”](#) on page 131 for a sample RDM PL/I application program.

Using the programmer's report

RDM provides a programmer's report which describes the layout of a view and the column names in a view. The DBA provides the report which should be used when you are constructing keyed GETs or when you are determining the columns to subset for a user view.

The programmer's report provides information about the column type, column name, and the picture clause generated by the RDML compiler. The column type may be KEY, NONUNIQUE KEY, or REQUIRED. See the following example programmer's report.

```
*** RELATIONAL DATA MANAGER PL/I PROGRAMMER'S REPORT FOR SCHEMA QUERYDBM ***  
VIEW:  CUST-PROD
```

COLUMN TYPE	COLUMN NAME	PICTURE
KEY	CUST-NO	9(05) CUST-NO
KEY	PROD-NO	X(0004) PRCU-PROD_NUM
	RENT	9(07)V9(02) USAGE COMP-3
	MAINT	9(07)V9(02) USAGE-COMP-3
	INSTALL-DATE	9(06)
	CANCEL-DATE	9(06)
	PURCHASE-PRICE	9(07)V9(02) USAGE COMP-3

Coding INCLUDE statements

INCLUDE statements indicate which views and which columns in those views you want to use in an application program. You can use an INCLUDE statement anywhere you can use a PL/I DECLARE statement. You can also have both INCLUDE and RDML statements at different levels in your program.

Specifying views and user views

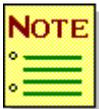
To use a specific view in your program, code an **INCLUDE** statement with the name of the view. For example:

```
01 INCLUDE CUST.
```

You can create your own user view by selecting columns from a view. This subset of the view is also specified with the INCLUDE statement. Indicate the name of your user view as well as the columns to be included from the view. For example:

```
01 CUST-MAIL INCLUDE CUST (NAME, ADDRESS, CITY, STATE, ZIP).
```

Unless you are accessing data values that are shared between views, each user view can be positioned independently and can act independently. If you create your own user view, you cannot change the effect of required columns in terms of their being available for inserts. However, you can modify the column order in a view.



Be aware that rearranging key columns may adversely affect your application's performance, and we do not recommend it.

Each INCLUDE statement also has associated row- and status-data areas generated by RDM. The row-data area specifies where data for each included view will be placed in the program.

The status-data area has one-to-one mapping to the fields in the row-data area and contains one byte of information indicating whether the data is valid, has changed since your last access, or is missing. See "**RDM status indicators**" on page 79 for information on RDM status indicators.



View names and column names used in PL/I programs must not contain hyphens (-) but may contain the underscore (_). The DBA enters all names into the Directory using hyphens only. Code any names in PL/I programs with underscores, and the RDML compiler takes care of all necessary conversion.

Specifying TIS_CONTROL

You must include the special view TIS_CONTROL in each program issuing an RDML request. TIS_CONTROL is used for passing parameters between the application and RDM and contains operation, status, and other information required to control access to all views. When you specify an **INCLUDE for TIS_CONTROL**, the RDML compiler generates the following:

```

/*
INCLUDE  TIS_CONTROL;
*/

DCL 1  TIS_CONTROL,
      2  TIS_OBJECT_NAME          CHAR(30),
      2  TIS_OPERATION            CHAR(6),
      2  TIS_FSI CHAR(1),
      2  TIS_VSI CHAR(1),
      2  TIS_FILLER                CHAR(2),
      2  TIS_MESSAGE              CHAR(40),
      2  TIS_PASSWORD             CHAR(8),
      2  TIS_OPTIONS              CHAR(4),
      2  TIS_CONTEXT             CHAR(4),
      2  TIS_LVCONTEXT            CHAR(4);

DCL TIS_ID          CHAR(2) DEF TIS_OPERATION;
DCL TIS_OPCODE     CHAR(1) DEF TIS_OPERATION POS(3);
DCL TIS_POSITION  CHAR(1) DEF TIS_OPERATION POS(4);
DCL TIS_MODE       CHAR(1) DEF TIS_OPERATION POS(5);
DCL TIS_KEYS       CHAR(1) DEF TIS_OPERATION POS(6);

```

If you want to pass rows to external modules (subroutines) from your application, code an INCLUDE statement in the LINKAGE SECTION of the subroutine instead of the WORKING_STORAGE SECTION. If a subroutine issues any RDML commands, it must define, or be passed, a TIS_CONTROL area.

Use the TIS_OPTIONS field to specify DEBUG and TRACE. To debug an RDM call, place DBUG in the TIS_OPTIONS field before the call to RDM. To trace an RDM call, place TRAC in the TIS_OPTIONS field before the call to RDM. DEBUG and TRACE output is written to the DMLPRINT dataset. To close the DMLPRINT file within an application, place NBUG in the TIS_OPTIONS field before the call to RDM.

RDM status indicators

RDM returns status indicators to the application program to indicate RDML processing results. The type and validity of the values you can place in the columns of your program statements are determined by the DBA. If you code a value which does not meet established criteria, you will receive a value error in the form of a status indicator. There are three kinds of status indicators:

- ◆ Function Status Indicator (FSI)
- ◆ Attribute (Column) Status Indicator (ASI)
- ◆ Validity Status Indicators (VSI)

Function Status Indicator (FSI)

FSI reflects the success or failure of your RDML request. (An associated message is provided in the TIS_MESSAGE area.) The FSIs are obtained from TIS_CONTROL. The following code shows an example of the generation of this control region. The RDML compiler changes RDML requests into comments by placing an asterisk in column 7 of each statement.



All other statements are generated by the RDML compiler.

```

/*
  INCLUDE TIS_CONTROL;
*/
DCL 1 TIS_CONTROL,
.
.
      2 TIS_FSI           CHAR (1),
      2 TIS_VSI           CHAR (1),
.
.
      2 TIS_MESSAGE       CHAR (40),

```

FSIs have the following meanings:

FSI value	Meaning
*	Successful.
D	Data error. The request would have run with valid values in the columns. You need to check the ASIs to find the column(s) that contains the invalid value.
F	Fail. Indicates a major error. Something may be wrong with the database, or you may have attempted to perform an illegal function on the user view.
N	Failure due to occurrence problem. May be due to a GET not found or an INSERT duplicate found.
S	Security.
U	Unavailable resources.
X	RESET recommended. While processing, RDML function modifications were made to the database before the error condition was detected. Issue a RESET to restore the database. This code overrides D, F, S, or U indicators.

Attribute (Column) Status Indicator (ASI)

ASI reflects the status of each column defined in your view. ASIs have one-to-one mapping to each column in the user view and are placed immediately following the last column in your user view.

You can access ASIs through PL/I-assigned names generated by the RDML compiler. When you code your program, an **INCLUDE** statement for the user view is required. The RDML compiler generates a field for each column included in the user view. The RDML compiler also generates a field for each required ASI column by preceding each column name with the characters ASI_. When you have a data column name containing more than 26 characters, the RDML compiler truncates any trailing characters when forming the column status field. The following is an example of this generation:

```

/*
INCLUDE CUST_PROD;
*/

DCL 1 TIS_CUST_PROD,
    2 FILLER CHAR(30) INIT ('CUST_PROD'),
    2 FILLER0001 CHAR(10) INIT (5ZCUST_NO,'),
    2 FILLER0002 CHAR(10) INIT ('4CPROD_NO,'),
    2 FILLEND CHAR(04) INIT ('END.');
```

```

DCL 1 RDM_CUST_PROD,
    2 CUST_PROD,
    3 CUST_NO PIC '(4)9T',
    3 PROD_NO CHAR(4),
    2 ASI_CUST_PROD,
    3 ASI_CUST_NO CHAR(1),
    3 ASI_PROD_NO CHAR(1);
```



The asterisk indicates the statement you code; the RDML compiler generates all other statements.

ASIs have the following meanings:

ASI value	Meaning
=	The column exists and has not changed since the last access. (Valid for GET processing only.)
-	The column is missing. It has a null value. (Valid for GET processing only.)
+	The column exists but has changed since the last access. (Valid for GET processing only.)
V	The column contains an invalid value.
C	Column value changed by another view.
N	Used to set a column to its null value. (INSERT and UPDATE processing only. RDM never returns this.)

There are four ways to use ASIs:

- ◆ When you issue a **GET** command, certain returned columns may not have a value. You can check this status (on unaltered columns) with the ASI.
- ◆ If you receive an FSI indicating a data error, you can use the ASI to find the columns that have illegal values.
- ◆ When a view contains packed or zoned values, the ASIs allow you to avoid unintentional abends. You can do this by examining each ASI for such columns before performing arithmetic or move operations. If the ASI for a column is 'V', the value is actually placed in the row even though it is not in a valid format. When a '-' ASI is returned, the field value is a valid zero value for packed, zoned, and binary fields.



Note that +, - and = are only meaningful on **GET** processing. Your application program ignores these values on **INSERT**, **DELETE** and **UPDATE** processing. **INSERT**, **DELETE**, and **UPDATE** processing returns ASIs of +, C, or V.

- ◆ For **INSERT** or **UPDATE** processing, moving 'N' to the ASI for a column before the function is performed indicates the column is null.

Validity Status Indicator (VSI)

VSI reflects the validity of the user view row returned by your last RDM request. RDM returns the VSI to the program in an area generated as part of the programmer-supplied TIS_CONTROL statement. The VSI helps you determine if any additional processing of ASIs is needed to correct invalid data or to fill in missing values. FSIs help you determine the most significant ASI returned by RDM, according to the hierarchy indicated in the chart below:

VSI value	Meaning
C	Column value changed by another view.
V	At least one invalid ASI was returned.
-	No invalid ASIs were returned, but at least one missing ASI was returned.
+	No invalid or missing ASIs were returned, but at least one new physical occurrence in the database was returned.
=	No invalid, missing, or new physical occurrences were returned by this RDM function.

Signing on/off

The only requirement for coding RDM PL/I statements is that you must supply a user name with the **SIGN_ON** statement. A password is optional, but you must supply it if you have been assigned a password in the Directory. At run time, RDM checks the Directory for the validity of the user name (and password, if necessary).

The **SIGN_ON** statement establishes communication between a task and the RDM. A task can be a batch job or an online task. When an RDM PL/I program is a subroutine to another RDM task, the task as a whole is already signed on, and an additional sign-on is required only if you are changing the user I.D.

Sign off an RDM PL/I subroutine only if the logical unit of work is complete and will do no further RDM processing before the next **SIGN_ON**.

The **SIGN_OFF** statement tells RDM that you want to terminate your session. RDM will release the storage areas acquired. Issue a **SIGN_OFF** at the end of every application program.

An example of an RDM PL/I subroutine is an interface to MANTIS. MANTIS using RDM takes care of the **SIGN_ON** (when processing a view statement) and the **SIGN_OFF** (when the task is terminating). If the RDM PL/I interface issues a **SIGN_OFF**, the logical unit of work may be invalid, and the task as a whole will no longer be communicating with the RDM.

For pseudoconversational applications under CICS, rules for signing on/off are different. RDM supports pseudoconversational applications by keeping available to the next program executed (at the same terminal) the internal content areas acquired for a program. An application program must issue an RDM **COMMIT** instead of a **SIGN_OFF** before releasing control to CICS. The next application program must not issue a **SIGN_ON**; omitting the **SIGN_ON** indicates a continuation of the previous task.

Maintaining storage

Use the **RELEASE** and **FORGET** statements to free internal storage without signing off the system. Use **FORGET** to release the storage allocated by a **MARK** statement (see “Using the **MARK** statement” on page 89).

Use **RELEASE** to close a specific view and free the storage allocated for that one view. If you do this, you lose any **MARKs** associated with that view. The **RELEASE** statement is also useful when you are accessing multiple views and want to remove all **MARKs**. You can use **RELEASE** (without specifying a view name) to close all views and free all allocated storage. If you do this, you remove all **MARKs**, and you lose the current position in all views you are using.

Retrieving rows using the **GET** statement

You can retrieve three types of rows using the **GET** statement:

- ◆ Rows containing unique keys
- ◆ Rows containing nonunique keys
- ◆ Rows containing no keys

The **USING** phrase in the **GET** statement indicates which key values to use to access the view. The system goes to the indicated view and retrieves the row for that particular customer.

Retrieving rows containing unique keys

If the row you want to retrieve has a unique key (ACCOUNT_NUMBER) and your program supplies a value for the unique key, the **GET** command retrieves the specific row having that key. For example:

```
ACCOUNT_NUMBER = 71560;  
GET ACCOUNT_DATA USING ACCOUNT_NUMBER;
```

RDM retrieves the row in the view, ACCOUNT_DATA, for the ACCOUNT_NUMBER indicated.

You can retrieve in sequential order all user rows with unique keys. The statement GET FIRST instructs the system to retrieve the first row in the user view. For example:

```
GET FIRST ACCOUNT_DATA.
```

The statement GET NEXT retrieves the next row. For example:

```
GET NEXT ACCOUNT_DATA.
```

A GET NEXT statement automatically retrieves the first row in a user view if no current position exists (no other **GET** statements have been issued). The GET SAME statement retrieves the same row as accessed on the previous GET statement, GET PRIOR retrieves the previous row, and GET LAST retrieves the last row.

After the last user row has been retrieved, a NOT FOUND condition results. Indicate what should be done in your program:

```
GET NEXT ACCOUNT_DATA;  
NOT FOUND GOTO STOP;
```

Retrieving rows containing nonunique keys

You can also retrieve a row with a nonunique key in sequential order. Again, the GET FIRST statement retrieves the first row, and the GET NEXT statement retrieves the next row.

GET NEXT will also automatically retrieve the first row in the view if no other commands have been issued. Keep two considerations in mind if you use the GET NEXT statement to retrieve the first row:

- ◆ GET NEXT operates as GET FIRST if no current position exists.
- ◆ The DBA may define some nonuniquely keyed views without a logical key for performing a direct read to the first row. In this case, the USING phrase is invalid and causes an error. The Relational Data Manager Programmer's Report shows you if no columns can be used as keys.

A NOT FOUND condition results when you reach the end of the view. Supply a NOT FOUND clause on the GET request to tell the system what to do.

Another method for retrieving a nonuniquely keyed user row is to include a USING phrase and a key value with your GET command.

```
GET ACCOUNT_TRANS USING ACCOUNT_NO.
```

The remaining rows with the same key can be retrieved with a GET NEXT command that contains a USING phrase.

```
GET NEXT ACCOUNT_TRANS USING ACCOUNT_NO.
```

The NOT FOUND condition appears after the last row with the specified key has been retrieved.

The GET PRIOR, GET LAST, and GET SAME commands also operate on nonuniquely keyed user rows. GET PRIOR retrieves the previous row, GET LAST retrieves the last row, and GET SAME retrieves the same row.

Retrieving rows without keys

You can also retrieve a row that does not contain a key. For example, by repeatedly issuing the request, GET CUST_INFO, you can retrieve in sequential order every row in the view, CUST_INFO. You can also use GET FIRST, GET NEXT, GET PRIOR, GET LAST, and GET SAME to retrieve rows without a key.

Accessing multiple views

You may want to use more than one view in a program. For example, you may want to code a program which will print a customer's name and the name of the part that customer ordered. Assume that you have a customer number and order number, and you want to use the views shown below. The columns in each view are listed below the view name.

CUSTOMER-ORDER-VIEW	CUSTOMER-VIEW	PRODUCT-VIEW
Order Number Customer Number Part Number Quantity Ordered Part Cost Total Cost Ship Date	Customer Number Customer Name Customer Address Customer Telephone	Part-Number Part-Name Part-Cost Quantity in Stock

First, retrieve the CUSTOMER_ORDER_VIEW (using the customer number and the order number as keys) to find the number of the part ordered. Next, retrieve the CUSTOMER_VIEW (using the customer number as a key) to find the customer's name. Finally, using the part number as a key, retrieve the PRODUCT_VIEW to find the name of the part.

```
CUSTOMER_NUMBER = 12345:  
ORDER_NUMBER = 67890:  
GET CUSTOMER_ORDER_VIEW USING CUSTOMER_NUMBER ORDER_NUMBER:  
GET CUSTOMER_VIEW USING CUSTOMER_NUMBER:  
GET PRODUCT_VIEW USING CUSTOMER_ORDER_VIEW PART_NUMBER:
```

Using the MARK statement

The **MARK** statement tells RDM to mark the current position of the view established by the previous **GET**, **UPDATE**, or **INSERT**. For example:

```
MARK CUSTOMER_VIEW AT SAVE_LV;
```

The AT phrase specifies where the view MARK should be saved. You must define the field used (SAVE_LV) in your program as a PICTURE X(4) field. You may use the AT phrase in the GET statement to reread the record at the position set by the MARK statement:

```
GET CUSTOMER_VIEW AT SAVE_LV;
```

Using explicit and automatic record holding

With RDM, you can choose explicit or automatic record holding. This decision depends on program requirements and the process you use to modify a row.

Explicit record holding

To specify explicit record holding, use the FOR UPDATE clause with the **GET** statement (GET FOR UPDATE). Explicit record holding invokes the record holding and enqueueing facilities of the underlying PDM and prevents other tasks from modifying the row.

Explicit record holding can cause a number of problems. A row is composed of selected physical fields from many physical files. Holding each of these physical fields can affect views in other tasks, even though the held view does not need the fields the other task wants to modify. Explicit record holding can also tie up resources for long periods and requires elaborate measures to release needed resources.

Automatic record holding

Automatic record holding allows you to do the following:

- ◆ Access a row using a **GET** statement
- ◆ Update or delete the row without an explicit request to hold the physical records.

This allows more efficient processing because the required record is held immediately before the actual database modification occurs.

Modifications that affect your row can be made by other tasks between the time you access the row and the time you update or delete it. To prevent such modifications from being undetected, RDM checks each column value in the row. This ensures that the column value is the original value you retrieved. If any columns have been changed, the following occurs:

- ◆ A data error
- ◆ You receive a 'D' FSI
- ◆ RDM flags the changed columns with a 'C' ASI and produces a 'C' VSI
- ◆ The column values marked with 'C' will not contain the new values but will contain the original values. This allows you to save the column values and retrieve the altered row to resolve the conflict.

Handling error conditions

When anything other than an FSI value of '*' is returned, RDM performs an automatic RESET and repositions you at the top of the view. (See "RDM status indicators" on page 79 for a list of possible FSI values and their meanings.) For example, if you perform a GET and then an UPDATE on a read-only view, the UPDATE will fail and RDM will reposition you at the top of the view. The next unqualified GET will return the first row in the view.

To avoid an automatic RESET, you need to code an error paragraph containing a NOT FOUND statement. The following example illustrates a sample error-handling paragraph:

```
PROD_TRAN.
  GET PROD FOR UPDATE USING TRAN_PROD
  NOT FOUND DO;
    PUT SKIP EDIT ('PROD NOT FOUND') (A);
    CALL ERROR_ON_PROD;
  END;
ERROR_ON_PROD: PROCEDURE;
  IF TIS_FSI='F' THEN
    DISPLAY 'DUE TO A MAJOR PROBLEM ENCOUNTERED
           WHILE ACCESSING THE LOGICAL USER
           VIEW PROD, THIS TASK IS NOW
           SIGNED OFF. ');
  SIGN_OFF;
  STOP;
END;
```



If you do not include an ERROR_ON_PROD paragraph in the program, the RDML compiler would have generated an automatic RESET as follows:

```
ERROR_ON_PROD: PROCEDURE;
  RESET;
END;
```

You can also add phrases (INVALID KEY, ELSE, NOT FOUND, etc.) to your basic program statements to handle common exception conditions in your paragraph.

When coding an error handling paragraph, the paragraph name should have the format ERROR_ON_ *viewname* where *viewname* is the name of the view.

If you start your view processing with a GET NEXT (default) followed by a USING phrase, (GET NEXT USING KEY1), you have qualified the row, so GET NEXT USING returns a single row with the designated key.

If only one row has the specified logical key, a repeat of the same GET returns a NOT FOUND error. Because an error repositions you at the top of the view, another execution of the GET returns the correct row.

Modifying rows

The DBA decides upon the modifications you can make to a row. There are three ways to modify a row:

- ◆ Update the data that already exists in the row (see “[Updating rows](#),” below)
- ◆ Delete the row (see “[Deleting rows](#)” on page 93)
- ◆ Insert a new row (INSERT statement) (see “[Using the INSERT statement](#)” on page 94)

Issue a **COMMIT** command after each logical transaction (which may involve more than one change) to establish the modifications in the database.

Updating rows

The **UPDATE** statement allows you to modify a column’s content. Before performing UPDATE, you must access the view by using a **GET** statement. For example:

```
GET ACCOUNT_DATA USING KEY1;  
UPDATE ACCOUNT_DATA;
```

You cannot modify a view key using the UPDATE command. RDM does not permit replacing a view key because you need the view key to locate the view row to be replaced. To change a view key, first **DELETE** the old row, then **INSERT** a new one.

Deleting rows

The **DELETE** statement removes a row from the system. Before performing DELETE, you must access the view by using a **GET** statement. For example:

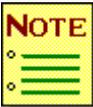
```
GET SAMPLE_VIEW USING KEY1;
DELETE SAMPLE_VIEW;
```



This example deletes the one occurrence of SAMPLE_VIEW obtained, based on the value of KEY1.

The phrase DELETE ALL deletes all rows that would have been retrieved by a GET FIRST followed by GET NEXTs using the parameters of the GET statement just prior to the DELETE. In other words, the DELETE ALL will delete all rows that depend on the key value specified on the latest GET:

```
GET SAMPLE_VIEW USING KEY1;
DELETE ALL SAMPLE_VIEW;
```



This example deletes all rows with the key value specified.

Certain constraints apply to deletions from the database. To delete an entity from the database means to remove an object (a product). This differs from removing a relationship. If you have an employee who transfers from one department to another, you do not remove the department; you remove the relationship between the employee and the first department.

Typically, a relationship delete can occur at any time. However, an entity must be unrelated to any other objects before you can remove it. Therefore, to remove the employee's record, you must first remove the relationship to the department and to any other objects.

Situations may arise where you want to delete an entity and the entity's relationships from the database. For example, if a customer cancels all outstanding orders and wants to be removed from your customer file, you first delete all relationships and then delete the customer. You can do this in one RDML statement by coding DELETE ALL in the application program, provided the DBA has allowed such an operation to occur.

Using the INSERT statement

The **INSERT** statement adds a new user row to the database:

```
INSERT ACCOUNT_DATA;
```

If you are inserting a user row in nonuniquely keyed rows, you can control the placement of the new row within the set of rows with the same key value. You cannot determine the location if the DBA has already defined an order for the view. The phrases NEXT, FIRST, LAST, or PRIOR may be added to the INSERT command. For example, INSERT NEXT ACCOUNT_DATA instructs the system to insert the new row after the current row (the last row accessed) in the view.

If the view is uniquely keyed, order is already determined. If the value of the keys to be inserted already exists, the DUP KEY condition results and RDM performs the action specified on the DUP KEY phrase in the INSERT statement:

```
INSERT ACCOUNT_DATA;  
DUP KEY GO TO ALREADY_THERE;
```

The constraints that apply when inserting information are the inverse of the constraints that apply to a deletion. You can always add a new entity (a customer), assuming you have space on the database. Typically, you cannot add a new relationship until all the entities being related exist. You cannot add a relationship between an employee and a department until you have added the department and employment entities.

However, you can add an entity and a relationship in one operation. For example, you can add a new employee and his first department assignment in a single INSERT request, provided the DBA has allowed this operation.

Using the COMMIT/RESET statements

The **COMMIT** statement makes the changes to the database (**INSERT**, **DELETE**, and **UPDATE**) permanent. The **RESET** statement instructs the system to perform the standard error-recovery procedure for dealing with the previous RDML request (to undo all database changes made by this task since the last COMMIT).

When Task Level Recovery (TLR) is active, the COMMIT statement sends all pending updates to the disk. A RESET backs out any database updates since the last COMMIT and continues processing from the RESET.

If TLR is not active, a RESET statement prints an error message, and the task abends. The task abend is intentional, and the system prints messages on the job log indicating the last function statement issued prior to the RESET. Normally, standard database-recovery procedures are performed, depending on the PDM being used.

In the CICS environment, RDM COMMIT/RESET logic works according to Dynamic Transaction Backout (DTB) processing. A COMMIT makes all updates permanent to the database and takes a CICS syncpoint. A RESET backs out any database updates since the last COMMIT and continues processing from the RESET.

In the CICS environment, RDM COMMIT/RESET logic works according to DTB processing. A COMMIT makes all updates permanent to the database and takes a CICS syncpoint. A RESET backs out any database updates since the last COMMIT and continues processing from the RESET.

Under CICS DTB, a rollback is performed. If you encounter an error condition in an online environment, you can back out of the modification by using the RESET function. This erases all modifications issued since the last COMMIT command.

For more information about DTB processing, refer to the *SUPRA Server PDM CICS Systems Programmer's Guide (OS/390 & VSE)*, P26-7452.

Handling errors requiring a recompile

RDM has several checks to ensure that the program you are running is current and that the user view it uses is the same as other applications in the system. When an RDML command is issued in an application program, RDM checks to see if the columns in the view, as defined in the Directory, are the same as when the program was last compiled. If not, an FSI status code is returned, and the program must be recompiled.

Changes requiring a recompile are:

- ◆ Data type change (packed to zoned decimal, etc.)
- ◆ Deleted column (if the column is not part of your user view, you need not recompile)
- ◆ Column length change
- ◆ A change in the number of decimal places

Application systems are often composed of several separately compiled programs that depend on common definitions of data items. These programs call each other to perform special tasks. RDM checks on each RDML call to make sure that the definition of the user view is the same for each program. If you compile a program or subroutine with the same user view name as another program or subroutine and the user view definition does not match, RDM generates an error message. The data used to perform this error checking is contained in the field list generated at compile time by the RDML compiler.

For information on executing the RDML compiler, see [“Compiling and linking an RDM PL/I application program”](#) on page 125.

4

RDM PL/I application program statements

This chapter contains the format of and usage considerations for RDM PL/I application program statements. The statements are divided as follows:

- ◆ INCLUDE statements
- ◆ RDML statements

Some RDM PL/I program statements have specific underlying file-system restrictions. For more information on the restrictions for PDM file systems, refer to the *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221. For information on VSAM restrictions, refer to the *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222.

INCLUDE statements

INCLUDE statements describe the information or data that the application program is to process. This information includes format and characteristics of input and output records, their data fields, and miscellaneous work areas.

INCLUDE view-data

Use the INCLUDE statement to indicate which views your program needs and where to place them.

```
INCLUDE [user-view-name] = view-name [user-column-list];
```

user-view-name

- | | |
|--------------------|---|
| Description | <i>Optional.</i> Specifies the name to assign to the user view. This name is used in the RDML commands. |
| Format | Must follow PL/I naming standards |

view-name

- | | |
|--------------------|---|
| Description | <i>Required.</i> Indicates the view you want to use. If you do not specify a <i>user-view-name</i> , this name is used as the <i>user-view-name</i> in RDML commands. |
| Format | Must be a valid view name. |

user-column-list

Description *Optional.* Indicates the columns from the particular view you want to use.

Format Must be part of a valid view.

Considerations

- ◆ If you do not include user columns that are required columns in the view, you cannot perform **INSERTs** and some **UPDATES** on the view.
- ◆ Modifying key order in your user view could adversely affect performance. The DBA has defined the key order on the Directory to maximize performance.

General considerations

- ◆ You can place an **INCLUDE** statement anywhere you can place a **DECLARE** statement in a PL/I program. PL/I scoping rules apply, so views at different program levels can have the same names.
- ◆ You must specify a **DECLARE** statement for the **SYSPRINT** file in order to specify a **PRINT** attribute with a record size of 133. This is required to override the default of 121 that PL/I uses if you do not **DECLARE** a **SYSPRINT**.
- ◆ User column lists can enhance the performance of your application. RDM uses lists of user columns supplied in the **INCLUDE** statement to optimize the physical accesses. Performance may be improved if you do not use all columns in the view.
- ◆ Place the **INCLUDE** statement for a particular view within the scope of all RDML statements using that view.

Examples

```

/*
INCLUDE VIEW2 = CUST_PROD;
*/

      DCL  1 TIS_VIEW2,
          2 FILLER  CHAR(30) INIT      ('CUST_PROD      '),
          2 FILL0001 CHAR(11) INIT
              ('5ZCUSTOMER, '),
          2 FILL0002 CHAR(10) INIT
              ('4CPROD_NO, '),
          2 FILL0003 CHAR(04) INIT      ('END. ');

DCL  1 RDM_VIEW2,
      2 VIEW2,
          3 CUSTOMER          PIC '(4)9T',
          3 PROD_NO          CHAR(4),
      2 ASI_VIEW2,
          3 ASI_CUSTOMER      CHAR(1),
          3 ASI_PROD_NO       CHAR(1);

TIS_OPERATION = 'LVY---';
TIS_OBJECT_NAME = 'VIEW2      ';
CALL CSVIRDM(TIS_CONTROL, TIS_CONTROL,
             TIS_DATE_STAMP,
             TIS_VIEW2);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_VIEW2;

```

```

/*
  INCLUDE  CUST_PROD;
  */
      DCL  1  TIS_CUST_PROD,
      2  FILLER  CHAR(30) INIT  ('CUST_PROD
'),
      2  FILL0001 CHAR(10) INIT
          ('5ZCUST_NO, '),
      2  FILL0002 CHAR(10) INIT
          ('4CPROD_NO, '),
      2  FILL0003 CHAR(08) INIT
          ('5P2RENT, '),
      2  FILL0004 CHAR(09) INIT
          ('5P2MAINT, '),
      2  FILL0005 CHAR(15) INIT
          ('6ZINSTALL_DATE, '),
      2  FILL0006 CHAR(14) INIT
          ('6ZCANCEL_DATE, '),
      2  FILL0007 CHAR(18) INIT
          ('5P2PURCHASE_PRICE, '),
      2  FILLEND  CHAR(04) INIT  ('END:');

      DCL  1  RDM_CST_PROD,
          2  CUST_PROD,
          3  CUST_NO          PIC '(4)9T',
          3  PROD_NO         CHAR(4),
          3  RENT            DEC FIXED (9,2),
          3  MAINT           DEC FIXED (9,2),
          3  INSTALL_DATE    PIC '(5)9T',
          3  CANCEL_DATE     PIC '(5)9T',
          3  PURCHASE_PRICE  DEC FIXED (9,2),
          2  ASI_CUST_PROD,
          3  ASI_CUST_NO     CHAR(1),
          3  ASI_PROD_NO    CHAR(1),
          3  ASI_RENT       CHAR(1),
          3  ASI_MAINT      CHAR(1),
          3  ASI_INSTALL_DATE CHAR(1),
          3  ASI_CANCEL_DATE CHAR(1),
          3  ASI_PURCHASE_PRICE CHAR(1);

      TIS_OPERATION = 'LVY---';
      TIS_OBJECT_NAME = 'CUST_PROD';
      CALL CSVIRDM(TIS_CONTROL, TIS_CONTROL,
                  TIS_DATE_STAMP,
                  TIS_CUST_PROD);
      IF TIS_FSI NE '*' THEN
          CALL ERROR_ON_CUST_PROD;

```

INCLUDE TIS_CONTROL

Use the INCLUDE TIS_CONTROL statement to include the special view, TIS_CONTROL, in a program.

INCLUDE TIS_CONTROL;

General considerations

Use the TIS_OPTIONS field to specify DEBUG and TRACE; code as follows:

DEBUG - DEBUG is on

NDEBUG - DEBUG is off

TRAC - TRACE is on

See “[Specifying TIS_CONTROL](#)” on page 78 for instructions on coding DEBUG and TRACE in your application program.

Example To add the special view TIS_CONTROL to your program, code the following statement:

```

/*
INCLUDE TIS_CONTROL;
*/
DCL 1 TIS_CONTROL,
    2 TIS_OBJECT_NAME      CHAR(30),
    2 TIS_OPERATION        CHAR(6),
    2 TIS_FSI              CHAR(1),
    2 TIS_VSI              CHAR(1),
    2 TIS_FILLER           CHAR(2),
    2 TIS_MESSAGE          CHAR(40),
    2 TIS_PASSWORD         CHAR(8),
    2 TIS_OPTIONS          CHAR(4),
    2 TIS_CONTEXT          CHAR(4),
    2 TIS_LVCONTEXT        CHAR(4);

DCL TIS_ID                  CHAR(2) DEF TIS_OPERATION;
DCL TIS_OPCODE              CHAR(1) DEF TIS_OPERATION      POS(3);
DCL TIS_POSITION            CHAR(1) DEF TIS_OPERATION      POS(4);
DCL TIS_MODE                 CHAR(1) DEF TIS_OPERATION      POS(5);
DCL TIS_KEYS                 CHAR(1) DEF TIS_OPERATION      POS(6);

DCL 1 TIS_VER_DATA,
    2 TIS_DATE_STAMP,
    3 TIS_DATE                CHAR(8) INIT('19840224'),
    3 TIS_TIME                 CHAR(6) INIT('145128');

```

Coding RDML statements

RDML statements need to be on a line by themselves and end with a semicolon (;). RDML statements can have labels. Because the expansion of RDML statements into PL/I statements usually involves several statements, use a do-group when placing RDML statements in THEN and ELSE clauses.

```
IF expression THEN DO;  
    GET VIEW1 USING KEY1;  
    .  
    .  
    .  
    UPDATE VIEW1;  
    END;  
ELSE DO;  
    INSERT VIEW1;  
    END;
```

When you compile an RDM PL/I program, a special CALL to RDM is issued for each view included in the program. The CALL statements appear as the first statements after the declarations of the view's data and status areas. RDM checks each view to ensure no columns were modified in the Directory since the last time the program was compiled, and that the rows accessed by multiple programs have the same format (the same column lists). If such a Directory change exists, the system does not allow the program to execute until it has been recompiled.

COMMIT

Use the COMMIT statement to identify a synchronized recovery point in your program. In environments where TLR is supported, the COMMIT statement results in a COMMIT function being issued to the PDM.

COMMIT;

General considerations

- ◆ In those environments where TLR is supported, the COMMIT statement returns either a successful or restart status. In other environments, the COMMIT statement always returns a successful status.
- ◆ In a CICS environment, a CICS syncpoint function is performed.
- ◆ To maintain view context in the CICS pseudoconversational mode, issue a COMMIT rather than a **SIGN_OFF** before task termination. The next program executed from the same terminal can continue to use RDM as though the task had not terminated.

Example

The COMMIT statement identifies a recovery point for the task which issues it.

```

/*
  COMMIT;
*/

TIS_OPERATION = 'LVC---';
CALL CSVIRDM(TIS_CONTROL,TIS_CONTROL,
            TIS_CONTROL,TIS_CONTROL)
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_TIS_CONTROL;

```

DELETE

Use the DELETE statement to remove a view row from the database.

DELETE [ALL] *view-name*;

ALL

Description *Optional.* Deletes all view rows that depend on the logical keys specified by the previous GET for this view.

Consideration This statement uses the parameters of the GET statement issued just prior to the DELETE.

view-name

Description *Required.* Specifies the view you want to use in your deletion.

Format Must be a valid, open view.

General considerations

- ◆ The DELETE statement removes an entire row.
- ◆ You cannot perform a DELETE if it compromises data integrity. For example, you cannot delete a customer's record until you delete all outstanding orders for that customer.
- ◆ To ensure database integrity, a **RESET** must follow an X failure status from a DELETE request. If you provide an error-handling procedure that does not RESET following an X status on DELETE, part of the modification may be done, and part may not be done.
- ◆ The DELETE ALL command deletes all records in a view if the program specifies a GET without a qualifying USING phrase.
- ◆ The RDML compiler recognizes DELETE as a view request whenever it is followed by something other than FILE. In this case, the preprocessor issues a message that the statement was skipped and is assumed to be a PL/I statement.
- ◆ The DBA may disallow DELETES.

Examples

- ◆ The following example deletes the one occurrence of `SAMPLE_VIEW` based on the value of `KEY1`:

```
GET SAMPLE_VIEW USING KEY1;
DELETE SAMPLE_VIEW;
```

- ◆ The next example deletes all user-view records retrieved by automatically generated `GET NEXT`s, using the parameters of the `GET` statement just prior to the `DELETE`.

```
GET SAMPLE_VIEW FOR UPDATE USING KEY1;
DELETE ALL SAMPLE_VIEW;
```



This example has the same effect as the following set of statements:

```
GET FIRST SAMPLE_VIEW FOR UPDATE USING KEY1;
MORE:  DELETE SAMPLE_VIEW;

GET NEXT SAMPLE_VIEW FOR UPDATE USING KEY1

      NOT FOUND GOTO DONE;

GOTO MORE;

DONE:
```

```
.
.
.
```

FORGET

The FORGET statement frees the storage allocated by a previously issued **MARK** statement.

FORGET *data-item* [**NOT FOUND PL/I do-group**]
[**ELSE PL/I do-group**];

data-item

Description *Required.* Specifies the MARK information that should be forgotten.

Format Must be defined as a CHAR(4) and must contain information passed to it by a previously issued MARK statement

NOT FOUND PL/I do-group

Description *Optional.* Indicates what should be done if the mark information cannot be released.

Considerations

- ◆ RDM may not find a mark value if one of the following conditions is true:
 - The mark has previously been forgotten by another FORGET statement or by a **RELEASE** statement.
 - The data-item was never marked by a **MARK** statement.
 - The marked data-item was somehow changed or moved.
 - ◆ Do not precede with a semicolon.
-

ELSE PL/I do-group

Description *Optional.* Indicates what to do if the mark information release is done.

Consideration The program falls through to the next statement if you do not specify an ELSE clause.

General considerations

- ◆ Issuing a FORGET statement releases the indicated mark, and you cannot regain it without issuing a new **MARK** statement.
- ◆ After a successful FORGET, set the data-item field to spaces.

GET

Use the GET statement to identify the row to retrieve from the indicated view.

```

GET 
  NEXT
  LAST
  SAME
  FIRST
  PRIOR
 view-name [FOR UPDATE] 
  USING data-item1[...data-item9]
  AT mark – data – item


  [NOT FOUND PL / 1 do – group]
  [ELSE PL / 1 do – group];


```

```


  NEXT
  LAST
  SAME
  FIRST
  PRIOR


```

Description	<i>Optional.</i> Indicates row-retrieval order.
Default	NEXT
Options	<p>GET NEXT Retrieves the next row with the specified keys. If you supply no keys, GET NEXT returns the next sequential row. If no current row exists, GET NEXT operates as GET FIRST.</p> <p>GET LAST Retrieves the last row in the view with the specified keys. If you give no keys, RDM returns the last row.</p> <p>GET SAME Retrieves the row just accessed, if a current row exists. If no current row exists, a NOT FOUND condition results.</p> <p>GET FIRST Retrieves the first row in the view with the specified keys. If no keys are given, RDM returns the first row.</p> <p>GET PRIOR Retrieves the previous row with the specified keys. If no current row exists, GET PRIOR operates as GET LAST.</p>

Considerations

- ◆ If the underlying file system cannot perform the GET PRIOR or GET LAST functions, an error results.
- ◆ A series of GET NEXTs loops back to the first row and continues if the statement has no NOT FOUND.
- ◆ A GET PRIOR view without a USING phrase returns a row, if there is a currently established position for a given key in a row. However, after processing all prior rows for the key, RDM returns the message: PDM DOES NOT SUPPORT THIS OPERATION.

view-name

Description *Required.* Specifies the name of the view you want to use.

Format Must be a valid view name.

FOR UPDATE

Description *Optional.* Allows you to lock out other users' modifications to the row record you are retrieving.

Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications that depend on the current contents of the row.
- ◆ If you do not need to be certain of the content of the row, you can use a GET without the FOR UPDATE phrase. When RDM performs the UPDATE or DELETE function, the automatic hold facility performs the lock before modifying the row.
- ◆ Using FOR UPDATE may decrease overall system performance. If any column values have changed before performing a DELETE or UPDATE, the function produces a data error ('D' FSI) and flags the changed columns with a 'C' ASI.

USING *data-item1*[...*data-item9*]

Description *Optional.* Specifies the key values to use in accessing the view.

Format The data items must be part of a valid view

Considerations

- ◆ The number of keys specified in the GET statement must be less than or equal to the number of keys in your specified column list.
- ◆ RDM treats any omitted keys as generic keys. Generic keys are convenient for allowing both direct access to a row and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified column as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase must correspond to the order of key declarations (left to right) in your Programmer's Report or your user view (see the INCLUDE statement, "**INCLUDE view-data**" on page 98). You cannot omit a key that occurs between two keys you want to specify. (For example, you cannot include KEY1 and KEY3 without including KEY2.)
- ◆ The USING phrase cannot be used with a GET SAME statement or with an AT phrase.
- ◆ If there is only one row for a given key and you try to use the same key with a GET USING statement (to access the row a second time), you receive an OCCURRENCE NOT FOUND message. This message indicates there are no more occurrences with this particular logical-key specification. In order to access this same row most efficiently, use a GET SAME statement instead.
- ◆ The logical key can use up to nine data items.

AT mark-data-item

Description *Optional.* Repositions a view based on the mark obtained by a previous MARK statement.

Format Must be a CHAR(4) data-item.

Considerations

- ◆ The data-item contains information generated by a previous MARK statement.
- ◆ You cannot use the USING and AT phrases in the same GET statement.
- ◆ You cannot specify the AT phrase in a statement using the FIRST, NEXT, PRIOR, LAST, or SAME positional qualifiers.

NOT FOUND PL/I do-group

Description *Optional.* Indicates what RDM is to do if it finds no data.

Considerations

- ◆ Data may not be found due to one or more of the following reasons:
 - No data is available for a keyed GET.
 - All the existing data is exhausted for a generic GET.
 - All the data available to the user view is exhausted for a nonkeyed GET.
 - A series of GET NEXTs loops back to the first row and continues if the program does not check for a NOT FOUND.
- ◆ Do not precede with a semicolon.

ELSE PL/I *do-group*

Description *Optional.* Indicates what RDM is to do if good data is found.

Consideration The program falls through to the next statement if you do not specify an ELSE clause.

General consideration

The RDML compiler does not recognize GET as an RDML statement if it is followed by any of the following:

- ◆ FILE, STRING, LIST, COPY, DATA, EDIT, or SKIP
- ◆ A left parenthesis (()

In these cases, a warning message results, indicating that the RDML compiler skipped the statement and assumed it to be a PL/I statement.

Examples

- ◆ The following statement retrieves the first row in the view PROD that matches the supplied key value. The PROD_TRAN field contains the key value used for retrieving the row.

```

/*
  GET PROD USING PROD_TRAN;
*/
      TIS_OBJECT_NAME = 'PROD
      PROD.PROD_NO
          = PROD_TRAN;
TIS_OPERATION = 'LVG_R1';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_PROD;

```

- ◆ This statement retrieves the view using the KEY PROD_TRAN. The USING phrase indicates that a key is used to retrieve the view.

```

/*
  GET PROD FOR UPDATE USING PROD_TRAN;
*/
      TIS_OBJECT_NAME = 'PROD
      PROD.PROD_NO
          = PROD_TRAN;
TIS_OPERATION = 'LVG_U1';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_PROD;

```

- ◆ This statement retrieves a view marked and saved for later access.

```

/*
  GET PROD AT PROD_MARK;
*/
      TIS_OBJECT_NAME = 'PROD                               ';
      TIS_CONTEXT     = PROD_MARK;
      TIS_OPERATION   = 'LVGARO';
      CALL CSVIRDM(TIS_CONTROL,
                  RDM_PROD,
                  TIS_DATE_STAMP,
                  TIS_PROD);
IF TIS_FSI NE '*' THEN
  CALL ERROR_ON_PROD;

```

- ◆ Repeatedly issuing this request retrieves all PROD rows in the view.

```

/*
  GET PROD;
*/
      TIS_OBJECT_NAME = 'PROD                               ';
      TIS_OPERATION   = 'LVG_RO';
      CALL CSVIRDM(TIS_CONTROL,
                  RDM_PROD,
                  TIS_DATE_STAMP,
                  TIS_PROD);
IF TIS_FSI NE '*' THEN
  CALL ERROR_ON_PROD;

```

- ◆ This request retrieves the row for update.

```

/*
  GET PROD FOR UPDATE;
*/
      TIS_OBJECT_NAME = 'PROD                               ';
      TIS_OPERATION   = 'LVG_UO';
      CALL CSVIRDM(TIS_CONTROL,
                  RDM_PROD,
                  TIS_DATE_STAMP,
                  TIS_PROD);
IF TIS_FSI NE '*' THEN
  CALL ERROR_ON_PROD;

```

- ◆ The following statements retrieve rows in the specified order (NEXT, LAST, SAME, FIRST, and PRIOR):

```

/*
GET NEXT PROD;
*/
TIS_OBJECT_NAME = 'PROD                                ';
TIS_OPERATION = 'LVGNRO';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
CALL ERROR_ON_PROD;

/*
GET LAST PROD;
*/
TIS_OBJECT_NAME = 'PROD                                ';
TIS_OPERATION = 'LVGLRO';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
CALL ERROR_ON_PROD;

/*
GET SAME PROD;
*/
TIS_OBJECT_NAME = 'PROD                                ';
TIS_OPERATION = 'LVGSRO';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
CALL ERROR_ON_PROD;

/*
GET FIRST PROD;
*/
TIS_OBJECT_NAME = 'PROD                                ';
TIS_OPERATION = 'LVGFRO';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
CALL ERROR_ON_PROD;

/*
GET PRIOR PROD;
*/
TIS_OBJECT_NAME = 'PROD                                ';
TIS_OPERATION = 'LVGPRO';
CALL CSVIRDM(TIS_CONTROL,
             RDM_PROD,
             TIS_DATE_STAMP,
             TIS_PROD);
IF TIS_FSI NE '*' THEN
CALL ERROR_ON_PROD;

```

INSERT

The INSERT statement inserts a new row into the view.

```

INSERT [
  NEXT
  LAST
  FIRST
  PRIOR
] view-name [DUP KEY PL / 1 do-group];
    
```

NEXT
LAST
FIRST
PRIOR

Description	<i>Optional.</i> Specifies where to insert the row relative to its current position.
Default	NEXT
Options	<p>INSERT NEXT Places the row after the current row, provided the keys are the same. If the keys are different or if no current row exists, INSERT NEXT operates as INSERT LAST.</p> <p>INSERT LAST Places the row into the view so that a subsequent GET LAST command using the same key values retrieves it.</p> <p>INSERT FIRST Places the row in the view so that subsequent GET FIRST commands using the same key values retrieve it.</p> <p>INSERT PRIOR Places the row in the view before the current row, provided the keys are the same. If the key values are different or if there is no current row, INSERT PRIOR operates as INSERT FIRST.</p>

Considerations

- ◆ If the DBA specified ordering in the view definition, or if the PDM does not allow program control of ordering, the specification on the INSERT statement is ignored.
- ◆ To ensure database integrity, a **RESET** must follow an X-failure status from an INSERT request. If you provide an error handling procedure which does not RESET following an X status on INSERT, it is possible that only part of the modification will be done.

view-name

Description	<i>Required.</i> Specifies the name of the view into which you want the rows inserted.
Format	Must be a valid, open view.

DUP KEY PL/I do-group

Description *Optional.* Indicates what RDM should do if the row to be inserted is uniquely keyed, and if the value of the keys to be inserted already exists in the database.

Consideration Do not precede this clause with a semicolon.

General considerations

- ◆ The DBA and/or the PDM being used may disallow ordering.
- ◆ You must supply all keys and required columns for the INSERT to be successful.
- ◆ Your application program can update a column with a null value by changing the ASI to N or by supplying the null value in the column.

Examples

The following examples show various ordering possibilities available to use with the INSERT statement:

```

/*
  INSERT NEXT PROD;
*/
    TIS_OPERATION = 'LVIN--';
    TIS_OBJECT_NAME = 'PROD';
    CALL CSVIRDM(TIS_CONTROL,
                RDM_PROD,
                TIS_DATE_STAMP,
                TIS_PROD);
    IF TIS_FSI NE '*' THEN
        CALL ERROR_ON_PROD;

/*
  INSERT LAST PROD;
*/
    TIS_OPERATION = 'LVIL--';
    TIS_OBJECT_NAME = 'PROD';
    CALL CSVIRDM(TIS_CONTROL,
                RDM_PROD,
                TIS_DATE_STAMP,
                TIS_PROD);
    IF TIS_FSI NE '*' THEN
        CALL ERROR_ON_PROD;

/*
  INSERT FIRST PROD;
*/
    TIS_OPERATION = 'LVIF--';
    TIS_OBJECT_NAME = 'PROD';
    CALL CSVIRDM(TIS_CONTROL,
                RDM_PROD,
                TIS_DATE_STAMP,
                TIS_PROD);
    IF TIS_FSI NE '*' THEN
        CALL ERROR_ON_PROD;

/*
  INSERT PRIOR PROD;
*/
    TIS_OPERATION = 'LVIP--';
    TIS_OBJECT_NAME = 'PROD';
    CALL CSVIRDM(TIS_CONTROL,
                RDM_PROD,
                TIS_DATE_STAMP,
                TIS_PROD);
    IF TIS_FSI NE '*' THEN
        CALL ERROR_ON_PROD;

```

MARK

Use the MARK statement to record the current position of the view established by the last **GET**, **UPDATE**, or **INSERT** statement.

MARK *view-name* **AT** *data-item*;

view-name

Description	<i>Required.</i> Indicates the view you want to mark.
Format	Must be a valid, open view.

AT *data-item*

Description	<i>Required.</i> Specifies where to save the MARK information.
Consideration	Must be defined in the program as a CHAR(4) data item.

General considerations

- ◆ The AT phrase in the GET statement (see “**GET**” on page 109) is used to reposition the view at the position set by the MARK statement.
- ◆ You can create any number of MARKs for a view, but to conserve internal memory space, it is best to reuse MARKs or to **FORGET** them whenever possible.
- ◆ The number of MARKs that a program can have outstanding at any time is limited by the size of the available slot. When the program no longer requires a particular MARK, issue a FORGET command for the *data-item*.

Example In this example the current position of the user view PROD is marked and saved at PROD_MARK:

```
DCL  PROD_MARK CHAR(4);
.
.
.
MARK PROD AT PROD_MARK;
.
.
.
GET  PROD AT PROD_MARK;
.
.
.
```

RELEASE

Use the RELEASE statement to close a specific view or all views that have been opened, and to free internal storage space allocated to the RDML compiler.

RELEASE [*view-name*];

view-name

Description *Optional.* Specifies the view to be released.

Format Must be a valid, open view.

Consideration If you omit this parameter, all your opened views are released.

General considerations

- ◆ The RELEASE statement is helpful when you are accessing multiple views. However, if you issue it without a *view-name*, RELEASE removes all MARKs (see “MARK” on page 119) and loses the current position in all views being used.
- ◆ The RDML compiler only recognizes a RELEASE request as valid for a view if a semicolon follows the statement. If it finds anything other than a semicolon, the preprocessor issues a warning message that it skipped the statement and assumed it to be a PL/I statement.
- ◆ If you issue RELEASE without a *view-name*, reset the MARK fields in the application to spaces.

Example

```
/*
RELEASE;
*/
TIS_OPERATION = 'LVR--';
CALL CSVIRDM(TIS_CONTROL,TIS_CONTROL,
             TIS_CONTROL,TIS_CONTROL);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_TIS_CONTROL;
```

RESET

Use the RESET statement to undo any **UPDATE**, **DELETE**, or **INSERT** requests issued since the last **COMMIT**.

RESET;

General considerations

- ◆ If you do not supply an error-handling procedure, the preprocessor generates an error routine that issues a RESET request to RDM, if an error occurs.
- ◆ In a non-TLR batch program, this operation prints an error message and the task abends. Normally, RDM then runs a batch recovery program.
- ◆ In the CICS environment, a CICS rollback function is performed.
- ◆ The RESET command restores your database to the last COMMIT point, and you lose position on all views. Therefore, the GET SAME, DELETE, or update commands are not valid after a RESET. A GET NEXT command positions you on the first record while a GET PRIOR command positions you on the last record after a RESET.

Example

In this example, you indicate a reset:

```

/*
  RESET;
*/
TIS_OPERATION = 'LVA---';
CALL CSVIRDM(TIS_CONTROL,TIS_CONTROL,
            TIS_CONTROL,TIS_CONTROL);
IF TIS_FSI NE '*' THEN
  CALL ERROR_ON_TIS_CONTROL;

```

SIGN_OFF

The SIGN_OFF statement informs RDM that access to the system is no longer desired.

SIGN_OFF;

General considerations

- ◆ The SIGN_OFF statement releases all storage areas acquired to service RDML requests.
- ◆ Issue a SIGN_OFF at the end of every application program, unless it is a CICS/VS pseudoconversational application program. A CICS/VS pseudoconversational application program transfers its context to the next program run from the same terminal (see “[Signing on/off](#)” on page 84). Use COMMIT instead of a SIGN_OFF for pseudoconversational operation.
- ◆ SIGN_OFF also causes a **COMMIT**.

Example

In this example, USER1 signs off the system:

```
/*
SIGN_OFF;
*/
TIS_OPERATION = 'LVC---';
CALL CSVIRDM(TIS_CONTROL,TIS_CONTROL,
             TIS_CONTROL,TIS_CONTROL);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_TIS_CONTROL;
TIS_OPERATION = 'LVF---';
CALL CSVIRDM(TIS_CONTROL,TIS_CONTROL,
             TIS_CONTROL,TIS_CONTROL);
IF TIS_FSI NE '*' THEN
    CALL ERROR_ON_TIS_CONTROL;
```

SIGN_ON

The SIGN_ON statement identifies the user to RDM.

SIGN_ON *user-name* [*password*];

user-name

Description *Required.* Indicates the user's name.

Format Must be assigned in the Directory.

Consideration The user-name must be a PL/I data-item name and not a literal.

password

Description *Optional.* Indicates the user's password. The password is required if the user has an assigned password in the Directory.

Format Must be assigned in the Directory.

Consideration If a password is specified, it must be a PL/I data item name and not a literal.

General consideration

A SIGN_ON request implicitly issues a release request and frees any previously allocated storage space.

Example In this example, SST signs on to the system:

```
DCL USER_IS CHAR(3) INIT ('SST');
.
.
.
.
SIGN_ON USER_ID;
```

UPDATE

The UPDATE statement updates column values in the database.

UPDATE *view-name*;

view-name

Description *Required.* Indicates view name you want to update.

Format Must be a valid, open view.

General considerations

- ◆ Before performing an UPDATE, you must access the view using a **GET** statement.
- ◆ Use the GET FOR UPDATE before you use the UPDATE function when computing a new value for a row (incrementing a counter, etc.). If you use the UPDATE function to place a value in a row, you need not issue a GET FOR UPDATE statement that does not depend on the values already present.
- ◆ You cannot update a view key. By changing the view key, you are requesting a repositioning of the view, not a modification of the current row. To update a view key, you must first delete the old row; then insert a new one.
- ◆ To ensure database integrity, a **RESET** must follow an X failure status from an UPDATE request. If you provide an error-handling procedure that does not RESET following an X status on **DELETE**, only part of the modification may be done.
- ◆ The use of UPDATE other than as part of a view statement is unaffected by the RDML compiler.
- ◆ Your application program can update a column with a null value by changing the ASI to N or by supplying the null value in the column.

Example

The statement UPDATE PROD indicates that you want to update the view PROD:

```
GET PROD USING ---;  
PRODUCT_FIELD = NEW_DATA;  
UPDATE PROD;  
  
.  
.  
.
```

5

Compiling and linking an RDM PL/I application program

This chapter presents information on the RDML precompiler, including instructions for executing the precompiler and linking considerations for each operating system.

Using the Directory to supply working storage, RDML statements are converted into standard PL/I source code by the RDML precompiler. Standard compilers then convert the PL/I source code into object code. When the program executes, the Directory uses the physical data descriptions, and RDM uses the logical data descriptions, to access the database and present the data in the view requested by the application program.

You cannot use the standard PL/I compiler until the application-program source statements (including the RDML) have gone through a preprocess phase using the supplied RDML precompiler.

See [“OS/390 and VSE samples and procedures”](#) on page 129 for the samples and procedures for executing the RDML precompiler in your operating environment.

Executing the RDML precompiler

There are two required parameters and two optional parameters for precompiling RDM PL/I application programs:

- ◆ **Actual Compiler Name.** *Required.* You designate this name as determined by the environment being used (either Batch or CICS).

Batch: To precompile a batch program, supply a compiler name of IEL0AA on the parm of the RDML precompiler execution step.

CICS: To precompile a IEL0AA PL/I RDM program for the CICS environment, specify the CICS command-level, preprocessor name on the parm of the RDML compiler step. In the CICS environment, two preprocess steps are necessary prior to executing the PL/I compiler. The RDML preprocess must occur before the CICS preprocess. Use a subsequent job step to execute the PL/I Compiler. Refer to procedure TISCPLBL (OS/390), and refer to TXJPLIPP and TXJPLICI samples (VSE) for sample JCL:

```
FIRST: SECOND: THIRD: PROCEDURE OPTIONS (MAIN);
```

A compilation of many external procedures results in all procedures being enrolled under the name of the first procedure. In the example above, the program is enrolled as FIRST.

- ◆ **BOOTMOD Name.** Not used.
- ◆ **Schema Name.** *Required.* The DBA furnishes this parameter.
- ◆ **Miscellaneous Parameters.** *Optional.* Includes any parameters you want to give to the PL/I Compiler or the CICS preprocessor.

The expansion of INCLUDE and RDML statements generates additional statements in the PL/I program. To assist you when referring to the program listing, the RDML precompiler places a 7-character identifier (TIS****) in columns 74-80 of the generated source statements. If the program includes a comment in the form /* \$\$ NUMBS \$\$ */, the 7-character identifier becomes TISnnnn, where nnnn is the line number of the INCLUDE or RDML statement in the original PL/I program.

If you use PL/I preprocessor % statements, you need to break the procedure into separate steps as follows:

1. Create a job stream with a first step that executes the PL/1 compiler with certain EXEC PARMs that tell the compiler to only do the preprocess phase.
 - Specify MACRO to invoke the PL/1 preprocessor.
 - Specify MDECK to generate preprocessor output. This output goes to SYSPUNCH DD.
 - Specify NODECK and NOOBJECT so this step generates no object output. Use a temporary data set for SYSPUNCH.
2. Execute the RDM & PL/1 compile job you've been running up to now but pass in the SYSPUNCH file from the first step as the input to CSVPL1PP.

Linking a compiled program

The following operating-system-dependent considerations apply when you are linking a compiled PL/I application program:

OS/390

In batch OS/390, COBOL applications are linked with CSVILUV, a composite containing CSVILOAD and DATBAS.

In OS/390 CICS, COBOL applications are linked with CSVCLUV, CSVICICS, or CSVNICIC. If the application is to run above the 16-megabyte line or if the application allocates RDM parameters above the 16-megabyte line, then the application must be linked with CSVNICIC.

VSE

In batch VSE, COBOL applications are linked with CSVIOSVS, a composite containing CSVJLUV and DATBAS.

In VSE CICS, COBOL applications are linked with CSVCLUV, CSVICICS, or CSVNICIC. If the application is to run above the 16-megabyte line or if the application allocates RDM parameters above the 16-megabyte line, then the application must be linked with CSVNICIC.

A

OS/390 and VSE samples and procedures

This appendix presents the samples and procedures for running the following tasks in OS/390 or VSE environments:

- ◆ RDML precompiler
- ◆ Run-time support
- ◆ Batch DBAID
- ◆ Batch reports

OS/390 samples and procedures

Description	Single-task		Central	
	Sample	Procedure	Sample	Procedure
Batch PL/I RDML precompiler and PL/I compile		TISPL1BL		TISPL1CL
CICS PL/I RDML precompiler and PL/I compile		TISCPLBL		TISCPLCL
Batch DBAID	TXJBDAID	TISAIDBL	TXJCDAID	TISAIDCL
Batch RDM Impact of Change Report	TXJICRPT	TISICRBL		TISICRCL
Batch RDM Reports	TXJREPRT	TISRPTBL		TISRPTCL

VSE samples

Description	Single-task	Central
RDML PL/I Precompiler	TXJPL1PP	
PL/I compile and link of precompiled RDML batch PL/I applications	TXJPL1CL	
CICS precompile PL/I compile and link of precompiled RDML CICS PL/I applications	TXJPL1CI	
Execute batch RDM PL/I applications	TXJPL1GO	
Batch DBAID	TXJBDAID	TXJCDAID
Batch RDM Impact of Change Report	TXJICRPT	
Batch RDM Reports	TXJREPRT	

B

Sample RDM PL/I application program

```
MANFPL1:  PROCEDURE OPTIONS(MAIN);                                00010022
                                                    00020022
INCLUDE TIS_CONTROL;                                           00030022
INCLUDE RDML_MANIFEST;                                         00040022
                                                    00050022

ERROR_ON_RDML_MANIFEST: PROCEDURE;                               00060023
    PUT SKIP EDIT('TIS_FSI= ',TIS_FSI);                         00070036
    PUT SKIP EDIT('TIS_VSI= ',TIS_VSI);                         00080036
    PUT SKIP EDIT('TIS_OBJECT_NAME= ',TIS_OBJECT_NAME);        00090036
    PUT SKIP EDIT('TIS_OPERATION= ',TIS_OPERATION);            00100036
    PUT SKIP EDIT('TIS_MESSAGE= ',TIS_MESSAGE);                00110036
    PUT SKIP EDIT('TIS_PASS_WORD= ',TIS_PASS_WORD)              00120036
    END ERROR_ON_RDML_MANIFEST;                                  00130036
                                                    00140036
                                                    00150036
    DCL USER_ID CHAR(2);                                        00160036
    USER_ID='DR';                                             00170036
                                                    00180036
    DCL SKIP_CONTROL FIXED DECIMAL;                             00190036
                                                    00200036
    DCL PRINTR FILE PRINT;                                     00210036
                                                    00220036
```

DCL (GET_1ST_MANIFEST, END_OF_REPORT,	00240023
(FIRST_DETAIL_LINE) CHAR (1) EXTERNAL;	00250023
	00260023
	00270023
GET_1ST_MANIFEST='Y';	00280023
END_OF_REPORT="Y";	00290023
FIRST_DETAIL_LINE='Y'	00300023
	00310023
SINON: PROCEDURE;	00320023
SIGN_ON USER_ID;	00330036
END SINON;	00340032
	00350032
SINOFF: PROCEDURE;	00360023
CLOSE FILE (PRINTR);	00370036
	00380036
STOP;	00390036
	00400036
SIGN_OFF	00410023
	00420023
END SINOF;	00430023
	00440023
DETAIL: PROCEDURE;	00450023
	00460023
IF FIRST_DETAIL_LINE 'Y';	00470023
SKIP_CONTROL=4;	00480023
ELSE DO;	00490023
SKIP_CONTROL=1;	00500023
END;	00510023
PUT EDIT(MANLINE_PRODUCT,PRODUCT_DESC,MANLINE_QNTY,	00530023
MANLINE_VALUE) ((SKIP(SKIP_CONTROL), A,X(2),A,	00540023
X(5),F(3),X(11),F(6,2);	00550023
	00560023
	00570023

```

//
00580023
00590023
HEADER:  PROCEDURE; 00600023
    PUT PAGE EDIT('BURRYS') (X(36)); 00610024
    PUT EDIT(MANIFEST') (SKIP(3),X(10),A); 00620024
    PUT SKIP EDIT('-----') (X(10),A); 00630036
    PUT SKIP (4) EDIT('BRANCH NUMBER : ', BRANCH_ZIPCODE), 00640024
        'MANIFEST NUMBER : ',MANIFEST_NO. 00650024
        (X(16),A,A,X(35), A,A); 00660024
    PUT SKIP EDIT(BRANCH_NAME) (X(16),A); 00670024
    PUT SKIP EDIT(BRANCH_ADDR) (X(16),A); 00680024
    PUT SKIP EDIT(BRANCH_CITY, BRANCH_STATE,BRANCH_ZIPCODE) 00690024
        (X(16),A,A,A); 00700024
    PUT SKIP EDIT('PRODUCT','DESCRIPTION','QNTY','VALUE') 00710024
        (X(11),A,X,(13),A,X(14),A); 00720024
    PUT SKIP EDIT('-----','-----','----','----') 00730036
        (X(11),A,X,(3),A,X(13),A,X(14),A); 00740036
00750024
END HEADER; 00760023
00770023
00780023
TOTAL:  PROCEDURE; 00790023
00800023
    PUT EDIT(MANIFEST TOTAL :',MANIFEST_TOTAL) (SKIP,X(34),A,A); 00810024
00820023
END TOTAL: 00830023
00840023
LSTMNF: PROCEDURE; 00850023
00860023

```

```

LOOP: DO WHILE (TIS_FSI NE "n");                                00870036
    IF (GET_1ST_MANIFEST='Y' THEN DO;                            00880023
        GET FIRST RDML_MANIFEST;                                00890023
        END;                                                    00900024
    ELSE DO:                                                    00910024
        GET NEXT RDML_MANIFEST;                                  00920023
        NOT FOUND DO;                                           00930036
        CALL TOTAL;                                              00940036
        CALL SINOF;                                              00950036
        END;                                                    00960024
    END:                                                         00970036
    IF ((ASI_MANIFEST_NO='+') & (GET_1ST_MANIFEST='Y'))THEN DO: 00980024
        GET_1ST_MANIFEST=' ';                                    00990024
        CALL HEADER;                                            01000024
        END;                                                    01010024
    ELSE DO:                                                    01020024
        IF (ASI_MANIFEST_NO='+') THEN DO:                        01030024
            CALL TOTAL;                                          01040024
            FIRST_DETAIL.LINE='Y';                                01050024
            CALL HEADER;                                          01060024
        END;                                                    01070024
    END;                                                         01080024
END LOOP;                                                       01090023
                                                                01110023
                                                                01120022
END LSTMNF;                                                      01130000
END MANFPL1;
//

```

Index

*

*, in DBAID 31

=

= command
description 29
example 32
syntax 32

A

abends 82, 95
ALL clause
in DELETE command 40
in DELETE statement 106
ASI. See Attribute Status
Indicator
asterisk, in DBAID 31
AT phrase
example of 89
in GET command 48
in GET statement 112
in MARK command 58
in MARK statement 119
Attribute (Column) Status
Indicator (ASI)
and binary fields 82
and current program 82
and packed vlaues 82
and zoned values 82
description 81
automatic hold 110
automatic record holding 90

B

batch recovery 95
binary fields, and ASI 82
built-in view commands. See
DBAID built-in view
commands

BYE command
description 29
syntax 33
BY-LEVEL command
description 29
example 35
syntax 34

C

catalogued procedures, for
OS/390 129
CAUTIOUS command
description 29
syntax 36
characters per line, displaying 57
CICS recovery
column definition of 21
use of 23
using COMMIT/RESET
statements 95
coding RDML statements 104
COMMIT 105
DELETE 106
FORGET 108
GET 109
INSERT 116
MARK 119
RELEASE 120
RESET 121
SIGN-OFF 122
SIGN-ON 123
UPDATE 124
column names, displaying 34, 43
column values, updating 124
column-list, displaying 70
COLUMN-TEXT command
description 29
example 38
syntax 37
COMMIT
automatic 67
prohibiting 36
COMMIT command
description 29
syntax 39
COMMIT statement 105
COMMIT/RESET logic 95
compiler 125
current program, checking for 96

- D**
- data validation 77
 - DBAID
 - signing-off 65
 - signing-on 66
 - DBAID built-in view commands
 - BY-LEVEL 29
 - COLUMN-TEXT 29
 - FIELD-DEFN 29
 - VIEW-DEFN 29
 - VIEWS-FOR-USER 29
 - DBAID RDML commands
 - = 29
 - BYE 29
 - CAUTIOUS 29
 - COMMIT 29
 - DELETE 29
 - ERASE 29
 - FORGET 29
 - GET 29
 - GO 29
 - INSERT 29
 - KEEP 30
 - MARK 30
 - OPEN 30
 - RELEASE 30
 - SIGN-OFF 30
 - SIGN-ON 30
 - SURE 30
 - UPDATE 30
 - DBAID system commands
 - LINESIZE 28
 - MARKS 28
 - PAGESIZE 28
 - USER-LIST 28
 - USERS 28
 - VIEWS 28
 - DBAID Utility subset
 - command categories 27
 - description 26
 - DBAID utility, exit from 33
 - DEBUG 102
 - DECLARE statement 77, 99
 - DELETE command
 - description 29
 - examples 41
 - syntax 40
 - DELETE statement
 - examples 107
 - syntax 106
 - using 93
 - DMLPRINT file 78
 - DUP KEY clause, in INSERT statement 117
 - duration of last request, displaying 71
- E**
- ELSE clause
 - in FORGET statement 108
 - in GET statement 113
 - ERASE command
 - description 29
 - syntax 42
 - error handling 91, 95, 96
 - explicit record holding 89
- F**
- FIELD-DEFN command
 - description 29
 - example 44
 - syntax 43
 - FOR clause, in GO command 51
 - FOR UPDATE phrase
 - in GET command 48
 - in GET statement 89
 - FORGET command
 - description 29
 - syntax 45
 - FORGET statement, syntax 108
 - formatting guidelines, for DBAID 31
 - FROM clause, in GO command 51
 - FSI. *See* Function Status Indicator (FSI)
 - Function Status Indicator (FSI) 80
- G**
- GET command
 - description 29
 - syntax 46
 - GET statement
 - examples 113
 - syntax 109
 - GO command
 - description 29
 - syntax 50

I

INCLUDE statements, syntax 98
 INCLUDE TIS-CONTROL
 statement
 example 103
 syntax 102
 INCLUDE view-data statement
 examples 100
 syntax 98
 INSERT command
 description 29
 examples 55
 syntax 53
 INSERT statement
 examples 118
 syntax 116
 using 94

K

KEEP command
 description 30
 syntax 56
 key
 compound nonunique 23
 compound unique 23
 definition of 21
 nonunique 87
 simple nonunique 23
 simple unique 23
 unique 23, 86
 keys, order of, in USING phrase
 111

L

lines per screen, specifying 62
 LINESIZE command
 description 28
 syntax 57
 linking a compiled program 128

M

MARK command
 description 30
 syntax 58
 MARK statement
 example 119
 syntax 119

MARKS command
 description 28
 example 59
 syntax 59
 MASS clause, and INSERT
 command 53, 54
 memory, conserve space in 58

N

NBUG 77, 102
 NOT FOUND clause
 in FORGET statement 108
 in GET statement 112

O

OPEN command
 description 30
 example 61
 syntax 60
 OS/390, cataloged procedures
 129

P

packed values, and ASI 82
 PAGESIZE command
 description 28
 syntax 62
 parameters
 passing between application
 and RDM 78
 using 126
 with RELEASE statement 120
 with UPDATE statement 124
 PL/I Programmer's Report
 description 75
 using 76
 PL/I programs, and RDM 21
 precompiling PL/I application
 programs 126
 processing time, displaying 71
 program statements 79, 91
 pseudoconversational mode, and
 COMMIT statement 105
 pseudoconversational program,
 and SIGN-OFF statement
 122

- R**
- RDM overview 17
 - RDM PL/I sample application program 131
 - RDML commands listed. See DBAID
 - RDML commands, reissue 32
 - RDML precompiler
 - executing 126
 - with DELETE statement 106
 - with GET statement 109
 - recompile, when required 96
 - record holding
 - automatic 90
 - explicit 89
 - recovery point, identifying 105
 - recovery, database 95
 - relationship
 - delete 93
 - insert 94
 - RELEASE command
 - description 30
 - syntax 63
 - RELEASE statement
 - example 120
 - syntax 120
 - request count, displaying 71
 - RESET command
 - description 30
 - syntax 64
 - RESET statement
 - example 121
 - syntax 121
 - rollback
 - and RESET statement 121
 - of database updates 64
 - row
 - adding 94
 - definition of 21
 - deleting 93
 - modifying 92
 - retrieving 46
 - save position of 89
 - updating 92
 - using DELETE statement 93
 - using INSERT statement 94
 - using UPDATE statement 92
 - with GET FIRST 85, 87
 - with GET LAST 87
 - with GET NEXT 85, 86, 87
 - with GET PRIOR 87
 - with GET SAME 87
 - with unique key 86
 - without key 87
 - ROW
 - WITH NONUNIQUE KEY 87
- S**
- sample RDM PL/I application program 131
 - samples
 - for OS/390 129
 - for VSE 130
 - signing off 84
 - signing on 84
 - SIGN-OFF command
 - description 30
 - syntax 65
 - SIGN-OFF statement
 - described 84
 - example 122
 - syntax 122
 - SIGN-ON command
 - description 30
 - syntax 66
 - SIGN-ON statement
 - described 84
 - example 123
 - syntax 123
 - sign-on time, displaying 71
 - START clause, in GO command 51
 - station number, displaying 71
 - status data area 77
 - status indicators 78
 - storage
 - amount used information 60
 - freeing 85, 108
 - with RELEASE statement 120
 - with SIGN-OFF statement 122
 - with SIGN-ON statement 123
 - SURE command
 - description 30
 - syntax 67
 - syncpoint 84
 - SYSPRINT file 99
 - system commands. See DBAID RDML commands

T

tabular format display 50
 Task Level Recovery (TLR)
 and COMMIT statement 105
 in CICS environment 95
 text, of column, displaying 43
 TIS_CONTROL
 and FSI 80
 specifying 78
 TLR. *See* Task Level Recovery
 (TLR)
 TRACE 78, 102

U

unsuccessful function, in RDM 91
 UPDATE command
 description 30
 example 69
 syntax 68
 UPDATE statement
 example 124
 syntax 124
 to modify a row 92
 user name, displaying 71
 user view, definition of 21
 user views
 creating 77
 specifying 77
 user-column-list, in INCLUDE
 statement 99
 USER-LIST command
 description 28
 example 70
 syntax 70
 USERS command
 description 28
 example 71
 syntax 71
 user-view-name, in INCLUDE
 statement 98
 USING phrase
 in GET command 49
 in GET statement 111
 in GO command 51

V

Validity Status Indicator (VSI)
 and automatic record holding
 90
 description 83
 value, definition of 21
 view
 closing 85
 creating a user 23
 current position of 89
 marking a position 119
 preparing for use by DBAID 60
 testing 27
 with RELEASE command 63
 view data values, updating 68
 view definition, and INSERT
 command 53
 view description, displaying 72
 view key, updating 124
 view row
 adding to physical database 53
 deleting from database 40
 multiple, inserting 54
 removing from database 106
 view, multiple, accessing 88
 VIEW-DEFN command
 description 29
 example 72
 syntax 72
 view-name, in INCLUDE
 statement 98
 views
 active, displaying 73
 listing be signed-on user 74
 logical 77
 user 77
 VIEWS command
 description 28
 example 73
 syntax 73
 VIEWS-FOR-USER command
 description 29
 example 74
 syntax 74
 VSI. *See* Validity Status Indicator
 (VSI)

W

warning message, with
 RELEASE statement 120

X

X failure 106

Z

zoned values, and ASI 82