DataRefresher

Version 1

Exit Routines

Part of the Information Warehouse family

SH19-6998-00

DataRefresher
Version 1

Exit Routines

Note!

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

First Edition (October 1994)

This edition applies to Version 1 of DataRefresher, Program Number 5696-703, and to all releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

A form for readers' comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Software Solutions,
Information Development (IISL),
2 Burlington Road,
Dublin 4,
Ireland.
Fax: (Ireland) +353 - 1 -6614246
IBMMAIL: IEIBM3FL at IBMMAIL
INTERNET: IEIBM3FL@IBMMAIL.COM

Contents

* IBM and OS/2 are trademarks of the International Business Machines Corporation.

Notices

References in this publication to IBM* products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Corporation, IBM Director of Licensing, 208 Harbor Drive, Stamford, Connecticut, United States, 06904.

Authorized use of IBM online books

For online versions of this book, we authorize you to:

Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.

Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Programming Interface information

This book is intended to help database administrators and application programmers write and use exit routines with DataRefresher.

This book documents Product-Sensitive Programming Interface and Associated Guidance Information.

Product-Sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of

this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-Sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-Sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Product-Sensitive Programming Interface

Product-Sensitive Programming Interface and Associated Guidance Information...

End of Product-Sensitive Programming Interface

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle
DATABASE 2
DB2
IBM
MVS/ESA
RACF
SQL/DS
VTAM
CICS
DataPropagator
DXT
Language Environment
OS/2
SAA
Systems Application Architecture

About this book

This book is intended for database administrators and application programmers using DataRefresher* to write and use exit routines. It describes how you can use exit routines to extend the range of functions DataRefresher can perform, and covers the requirements which must be met before you can use exit routines at your site.

Information on creating Exits on the workstation for accessing exit routines is provided in the DataRefresher *OS/2 User's Guide*.

What you should know

This book assumes you are familiar with:

- The structure of your current database environment
- Programming languages: PL/I, COBOL, or Assembler
- Multiple Virtual Storage (MVS) operating system
- MVS job control language (JCL)
- IBM* DATABASE 2* (DB2*)
- IBM Operating System/2* (OS/2*) operating system
- Virtual Storage Access Method (VSAM) data sets
- Information Management System (IMS) DL/I database
- International Organization for Standardization (ISO) data standards

How to use this book

The following table will help you to find the sections of this book that provide the information you need. A complete index is provided in this book.

Table 1 (Page 1 of 2). How to use this book

For information on	Read
Determining which exit routines you should use.	Chapter 1, "Understanding exit routines" on page 1.
Working with exit routines and control blocks.	Chapter 2, "Using exit routines" on page 11.
Compiling and link-editing exit routines and language load modules.	Chapter 2, "Using exit routines" on page 11.
Including exit routines in extract requests submitted using DataRefresher OS/2.	Chapter 2, "Using exit routines" on page 11.
Using Accounting exit routines to keep track of DataRefresher use at your site.	Chapter 3, "Writing Accounting exit routines" on page 29.
Using Data exit routines to convert records or segments in your source data.	Chapter 4, "Writing Data exit routines" on page 47.
Using User Data Type exit routines to convert fields in your source data.	Chapter 5, "Writing User Data Type exit routines" on page 69.

* IBM, DATABASE 2, DB2, Operating System/2 and OS/2 are trademarks of the International Business Machines Corporation.

Table 1 (Page 2 of 2). How to use this book

For information on	Read
Using Date/Time Conversion exit routines to convert date and time data in your source data.	Chapter 6, "Writing Date/Time Conversion exit routines" on page 77.
Using Generic Data Interface (GDI) exit routines to access data from sources that are not supported by DataRefresher.	Chapter 7, "Writing Generic Data Interface (GDI) exit routines" on page 83.
Using Generic Output Interface (GOI) exit routines to format data for targets that are not supported by DataRefresher.	Chapter 8, "Writing Generic Output Interface (GOI) exit routines" on page 125.
Using Map Capture exit routines to provide data definition and extract request information for further processing or validation.	Chapter 9, "Writing Map Capture exit routines" on page 133.

DataRefresher library overview

The following describes the contents and organization of information in the DataRefresher Version 1 library.

DataRefresher Version 1 (5696-703)	Order number
An Introduction This book provides an overview of DataRefresher. It describes the uses, benefits, and requirements of DataRefresher to help you evaluate the product.	GH19-6993-00
Licensed Program Specifications This document briefly describes the technical information for DataRefresher and is the warranty for the product.	GH19-9994-00
Administration Guide This book describes how to plan for the installation and the use of DataRefresher in your organization. It describes how to set up and administer DataRefresher.	SH19-6995-00
MVS and VM User's Guide This book describes how to use DataRefresher in an MVS or VM environment. In particular, it describes how to use the DataRefresher Administrative Dialogs and End User Dialogs to create and run extract requests.	SH19-6996-00
OS/2 User's Guide This book describes how to use DataRefresher on a workstation. It describes how to register your host data sources and create an extract which can be run on the host MVS system.	SH19-6997-00
Exit Routines This book describes how to write user exit routines to be used by DataRefresher when an extract is processed.	SH19-6998-00
Command Reference This book provides detailed reference information for all of the DataRefresher commands and procedures.	SH19-6999-00

DataRefresher Version 1 (5696-703)	Order number
Messages and Codes This book lists the DataRefresher messages with explanations and suggested responses.	SC19-5000-00
Diagnosis Guide This book contains the information required to diagnose problems with DataRefresher. It also contains information that can help you communicate with the IBM Support Center to isolate and solve problems with DataRefresher.	LY19-6386-00

Sources of related information

The following books are referenced in this publication.

DataPropagator NonRelational publications
DataPropagator NonRelational Customization Guide (SH19-5042).

DATABASE 2 publications
IBM DATABASE 2 Version 2: Application Programming and SQL Guide (SC26-4377).

Chapter 1. Understanding exit routines

DataRefresher is a powerful tool for gathering and distributing data across a diverse range of systems and formats. By writing exit routines, you can increase the range of functions DataRefresher can perform and improve DataRefresher's responsiveness to your site's data handling needs. Exit routines can only be used for extracts processed by the general Data Extract feature. The Relational Extract Manager (REM) does not support exit routines.

This chapter introduces exit routines and the functions they can perform.

What is an exit routine?

An exit routine is a user-written program that enables DataRefresher to perform tasks that it does not ordinarily handle. If you specify an exit routine in your extract request or data description, DataRefresher passes control to that exit routine at one or more points in the extract process. After the exit routine has completed its task, it returns control to DataRefresher, which continues processing your extract request.

DataRefresher supports seven types of exit routine:

- Accounting exit routines
- Data exit routines
- User Data Type exit routines
- Date/Time Conversion exit routines
- Generic Data Interface (GDI) exit routines
- Generic Output Interface (GOI) exit routines
- Map Capture exit routines

Before you can include an exit routine in your extract requests, the exit routine must be written, compiled and link-edited on the host (MVS) system. Your exit routines can be included in extract requests created using the following methods:

- DataRefresher OS/2
- DataRefresher online commands
- DataRefresher dialogs
- User-written extract requests

Deciding which exit routine to use

Each exit routine can be used to carry out a specific range of tasks. There is, however, some overlap between the functions that the various exit routines can perform.

The following sections, each devoted to a single type of exit routine, should enable you to determine which exit routines will best suit your needs. For a summary of the tasks each exit routine can perform, see Appendix A, "Exit routine task overview" on page 141.

Accounting exit routines

Accounting exit routines keep track of the resources used by the Data Extract Manager (DEM). You can write your Accounting exit routine to track virtually anything related to DEM use, and to reject extract requests that do not meet the criteria you specify.

You can use Accounting exit routines to:

- Record information about DEM use by specific users or accounts; you can, for example, keep track of the reasons for each request, or record the number of rows extracted during a particular run
- Keep track of resource utilization costs by users or accounts
- Identify authorized DEM users, and prevent unauthorized users from running the DEM
- Change the priority of an extract request
- Change the output limits of an extract request

For information about using Accounting exit routines, see Chapter 3, "Writing Accounting exit routines" on page 29.

Data exit routines

Data exit routines manipulate source data not yet examined by DataRefresher to make it match a data description in the DataRefresher file description table library (FDTLIB). You can use Data exit routines to clean up or change segments or records in your source data before DataRefresher translates the data into the target format. With Data exit routines, including those for summarization, you can manipulate data in many ways, as long as the result matches the data description in the FDTLIB when the exit routine completes processing.

You can use Data exit routines to:

- Suppress source fields or create new fields for extraction
- Create several logical records or segments from one physical source record or segment
- Create one logical record or segment from several physical source records or segments
- Alter or combine data
- Summarize data
- Decompress or decrypt data

Data exit routines can also be used to make nonstandard data in IMS databases uniform before DataRefresher examines it. If you need to interpret data types or encoded data, or change data types or data lengths, use a User Data Type exit routine. If you need to convert date or time data into International Standards Organization (ISO) format, use a Date/Time Conversion exit routine.

You cannot use Data exit routines to change key and sequence fields of source data in IMS DL/I databases or VSAM key-sequenced data set (KSDS) files. DataRefresher uses the key or sequence field in its search strategy, and must know the (in-storage) data type and length of the key or sequencing field.

For information about using Data exit routines, see Chapter 4, "Writing Data exit routines" on page 47.

User Data Type exit routines

Data in source format may contain a mixture of DataRefresher and user-defined data types. You can write User Data Type exit routines to convert specific fields in user-defined formats into standard DataRefresher data type formats.

You can use User Data Type exit routines to convert:

- Bit data
- Unsigned packed numeric data
- Three-byte binary data
- Encoded data (such as a 2-byte state ID that you want expanded)
- Numeric data stored as character data

User Data Type exit routines are similar to field level exit routines, while Data exit routines are equivalent to segment or record level exit routines. If necessary, you can write a User Data Type exit routine to convert more than one user-defined field.

Note: User Data Type exit routines cannot be used in combination with GDI Select exit routines.

For information about using User Data Type exit routines, see Chapter 5, "Writing User Data Type exit routines" on page 69.

Date/Time Conversion exit routines

Date/Time Conversion exit routines reformat date and time data into ISO format. ISO is the accepted format for date and time values in IBM relational databases, and is supported by DataRefresher. ISO date and time formats are shown in Table 2.

Table 2. ISO date/time formats

Data Type	Format	Description
DATE	yyyy-mm-dd	Always a 10-byte character string
TIME	hh.mm.ss	Always an 8-byte character string

where:

- `yyyy = year`
- `mm = month`
- `dd = day`
- `hh = hours`
- `mm = minutes`
- `ss = seconds`

DataRefresher supplies its own Date/Time Conversion exit routine to convert certain types of date and time formats to ISO. DataRefresher converts date/time data in the following formats:

```
Date formats:
 • mm/dd/yyyy (USA format)
 • dd.mm.yyyy (EUR format)
 • mm/dd/yy   (local format)
 • mm-dd-yy   (local format)

Time formats:
 • hh:mm xM   where x=A or P (USA format)
 • hh:mm:ss                  (JIS format)
```

Time data in EUR format (hh.mm.ss.) is the same as ISO and does not need to be converted.

A Date/Time Conversion exit routine is required if your data has date or time fields that are not ISO accepted or converted automatically by DataRefresher

Date/Time Conversion exit routines do not support the conversion of timestamp data. If necessary, timestamp data can be converted with a User Data Type exit routine.

For information about using Date/Time Conversion exit routines, see Chapter 6, "Writing Date/Time Conversion exit routines" on page 77.

Generic Data Interface exit routines

Generic Data Interface (GDI) exit routines can access data from many data sources that are not accessed directly by the DEM. A GDI exit routine retrieves data from the source and moves it to an output buffer, one row or record at a time, for further processing by DataRefresher.

You can use GDI exit routines to:

- Extract data from non-IBM data sources
- Extract data from Integration Exchange Format (IXF) files
- Perform a two-stage extract for DB2 data
- Join data between these and other data sources

The main difference between Data exits and GDI exits is the program that obtains the source data. With a Data exit, DataRefresher obtains the source data and passes it to the Data exit routine. With a GDI exit, DataRefresher calls the exit routine, and the exit routine reads the data and passes it back to DataRefresher. The advantage of GDI exit routines is that they can extract data from nearly any access method your installation has under MVS.

For information about which data sources are accessed directly by DataRefresher, see *DataRefresher MVS and VM User's Guide*.

Choosing between GDI Select and GDI Record exit routines

The following GDI exit routines are available:

- GDI Select exit routines which return the columns and rows of data that satisfy the SELECT statement in the SUBMIT command of your extract request. GDI Select exit routines can act like the dynamic SQL* of relational databases. GDI Select exit routines can also enable you to conserve system resources during peak-use times by doing two-stage extractions from DB2 databases.

  GDI Select exit routines cannot be used in combination with User Data Type exit routines.

- GDI Record exit routines which act as a user-written access method. They return records from any user-specified location to DataRefresher. DataRefresher then applies the WHERE clause criteria and selects fields. DataRefresher can join these records with data from other sources, such as DL/I and VSAM.

  GDI Record exit routines can also be used in combination with User Data Type exit routines.

The following provides more information on selecting the exit routine to use that best suits a particular task.

Using GDI Select exit routines

GDI Select exit routines can be used when you want to extract from a non-IBM database management system (DBMS) that has an SQL interface. In most cases, a DBMS with an SQL interface retrieves data more efficiently than a GDI Record exit routine written for the same purpose. When you write a GDI Select exit routine, you are free to use SQL dialects not supported by DataRefresher.

Using GDI Select exit routines, you can also perform two-stage extractions from a DB2 database instead of using the Relational Extract Manager (REM). The REM, normally used to extract from DB2, executes extract requests immediately.

Using GDI Record exit routines

GDI Record exit routines can be used when you want to extract from a non-IBM DBMS that does *not* have an SQL interface and it would be too costly to develop such an interface. With GDI Record exit routines, you can only use the SQL supported by DataRefresher in your extract requests.

GDI Record routines can also be used to join data from diverse databases and file types using DataRefresher. For example, using GDI Record exit routines DataRefresher can join DB2, IMS, non-IBM DBMS, physical sequential, VSAM, and IXF data into a combination you require. However if a conflict exists between two or more environments required to access your source data, a join may not be possible if you cannot run a DEM in this environment.

You can also extract data from a source that is neither self-defining, nor dependent on a data dictionary or DBMS catalog for its data descriptions using GDI Record exit routines.

---

* SQL is a trademark of the International Business Machines Corporation.

For information about using GDI exit routines, see Chapter 7, "Writing Generic Data Interface (GDI) exit routines" on page 83.

Generic Output Interface exit routines

Generic Output Interface (GOI) exit routines are used to manipulate data that has been extracted by DataRefresher and to convert it into a user-defined format. Just as GDI exit routines enable you to extract data from IBM and non-IBM sources, GOI exit routines enable you to format data for IBM and non-IBM targets.

You can use a GOI exit routine to:

- Prepare data for loading into a non-IBM database
- Prepare data for downloading to a workstation DBMS
- Summarize extracted data

After converting the data, your GOI exit routine can write the data to a file and, for example, load it into a database.

For information about using GOI exit routines, see Chapter 8, "Writing Generic Output Interface (GOI) exit routines" on page 125.

Map Capture exit routines

Map Capture exit routines provide you with DataRefresher definition and extract request information. For example, you can use Map Capture exit routines to:

- Save the information
- Use the information as a basis for some further processing, such as that performed by IBM's DataPropagator NonRelational.*
- Perform additional validation of the extract request, such as verifying that the table into which you are extracting data has the correct table name, column names, and data types

Your Map Capture exit routine, and the User Input Manager (UIM) by extension, can be run as a DB2 application.

For information about using Map Capture exit routines, see Chapter 9, "Writing Map Capture exit routines" on page 133.

---

* DataPropagator is a trademark of the International Business Machines Corporation.

How exit routines work

The various exit routines interact with DataRefresher in different ways depending on the functions they perform.

Accounting exit routines

Accounting exit routines enable you to keep track of DEM usage at your site. DataRefresher calls an Accounting exit routine at different stages of the extract process to gather information about DataRefresher processing. According to the conditions you have specified, the exit routine can record information about resource utilization, approve an extract request, modify an extract request, or reject an extract request and terminate processing.

Figure 1 shows how DataRefresher passes extract information to the Accounting exit routine for processing.

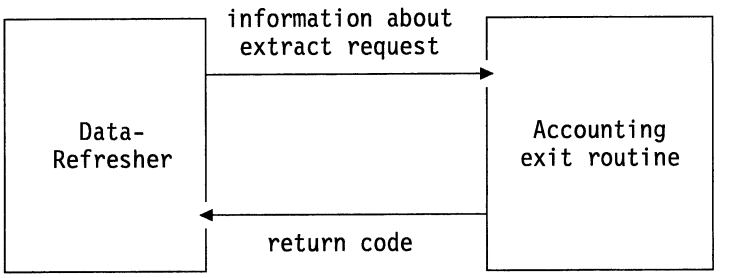

Figure 1. How DataRefresher interacts with Accounting exit routines

DataRefresher passes information about the extract request to the Accounting exit routine. If the exit routine approves the extract request, it completes its tasks, and returns control to DataRefresher to complete the extract process.

For information about how DataRefresher calls Accounting exit routines, see "Calling Accounting exit routines" on page 30.

Data, User Data Type and Date/Time Conversion exit routines

Data, User Data Type and Date/Time Conversion exit routines convert non-standard source data into a format that DataRefresher can process. As shown in Figure 2, these exit routines receive unprocessed source data from DataRefresher, convert the data into a format DataRefresher can understand, and then return it to DataRefresher for processing.

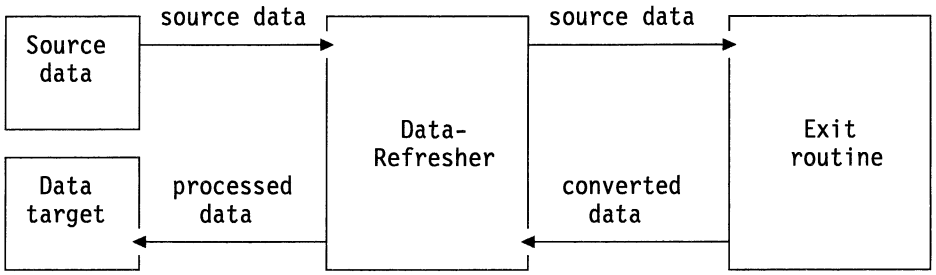

Figure 2. How DataRefresher interacts with Data, User Data Type and Date/Time Conversion exit routines

Once the exit routine has completed its tasks, it returns control to DataRefresher to complete the extract process.

Information about how DataRefresher calls these exit routines, is contained in:

- "Calling Data exit routines" on page 48
- "Calling User Data Type exit routines" on page 70
- "Calling Date/Time Conversion exit routines" on page 78

Generic Data Interface (GDI) exit routines

GDI exit routines enable DataRefresher to access data from many data sources that are not directly supported by DataRefresher. DataRefresher calls your GDI exit routine to extract the data one row or record at a time. Then DataRefresher receives that data for further processing before sending it to the target.

When you use GDI exit routines, the GDI exit routines obtains the source data as shown in Figure 3.

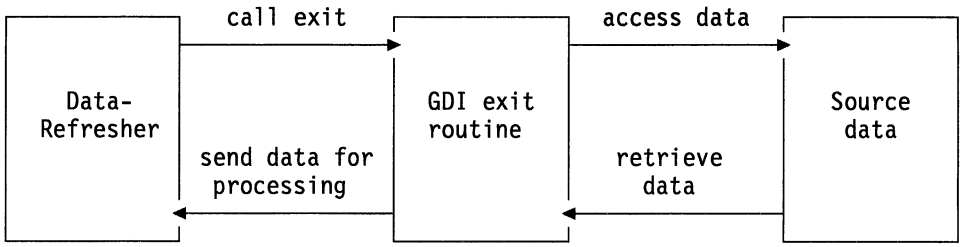

Figure 3. How DataRefresher interacts with GDI exit routines

Once the GDI exit routine has completed, DataRefresher completes the extract process. Information on how DataRefresher calls GDI exit routines is contained in "How DataRefresher calls a GDI exit routine" on page 86.

Generic Output Interface (GOI) exit routines

GOI exit routines enable you to convert data that has been extracted by DataRefresher into a format appropriate for targets that are not directly supported by DataRefresher. After extracting data from the source, DataRefresher passes the data to the GOI exit routine to reformat one row or record at a time. The exit routine sends the reformatted data to the target you have specified in the exit routine, as shown in Figure 4.

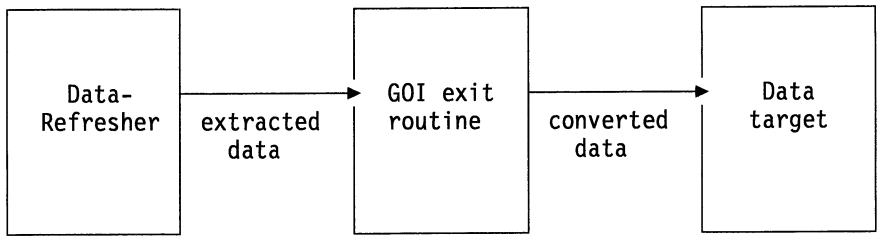

Figure 4. How DataRefresher interacts with GOI exit routines

After the GOI exit routine has completed its tasks, it returns control to DataRefresher to complete the extract process. Information on how DataRefresher calls GOI exit routines is contained in "Calling GOI exit routines" on page 126.

Map Capture exit routines

Map Capture exit routines capture DataRefresher definition or extract request information for validation or further processing by other programs, for example, DataPropagator NonRelational. Figure 5 shows how DataRefresher passes extract request information to the Map Capture exit routine for processing.

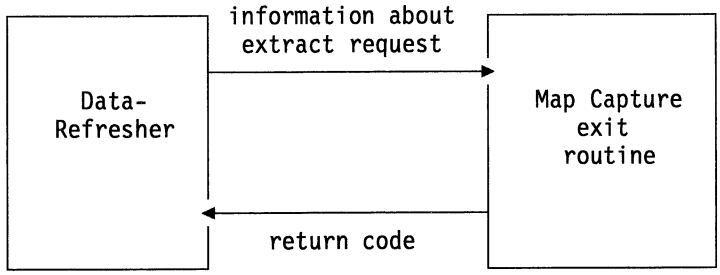

Figure 5. How DataRefresher interacts with Map Capture exit routines

DataRefresher passes information about the extract request to the Map Capture exit routine. If the exit routine approves the extract request, it completes its tasks, and returns control to DataRefresher to complete the extract process.

Information on how DataRefresher calls Map Capture exit routines is contained in "Calling Map Capture exit routines" on page 134.

---

* SAA, Systems Application Architecture, AD/Cycle, Language Environment, MVS/ESA, RACF, and DXT are trademarks of the International Business Machines Corporation.

Chapter 2. Using exit routines

This chapter contains practical guidelines for writing and using exit routines.

Task Overview

Writing and using exit routines with DataRefresher consists of:

1. Determining which types of exit routines you want to use.
   - For information on the functions that exit routines can perform, see "Deciding which exit routine to use" on page 1.
2. Writing your exit routines using the sample exit routines and control blocks provided with DataRefresher.

   For information on:
   - The control blocks and exit routines provided with DataRefresher, see "Using control blocks" on page 12 and "Copying and printing sample exit routines and control blocks" on page 14.
   - Using multiple exit routines, using exit routine messages, making exit routines reentrant, debugging your exit routines, and maintaining the security of your exit routines, see "Exit routine options" on page 24.
3. *If you are using exit routines written in OS PL/I or VS COBOL II,* compiling and link-editing the language load module. If you are writing your exit routines in Assembler, you do not need to set up a special language environment.
   - For information on how to prepare DataRefresher for exit routines written in languages other than Assembler, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. *If your exit routines will access DB2,* precompiling and binding your GDI and Map Capture exit routines.
   - For information on how to prepare your exit routines for DB2 applications, see "Precompiling and binding exit routines for DB2 applications" on page 21.
5. Compiling and link-editing your exit routines.
   - For information on how to compile and link-edit your exit routines, see "Compiling and link-editing your exit routines" on page 22.

6. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on how to:

   - Register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Specify which exit routine to use when creating a source folder, see the *DataRefresher OS/2 User's Guide.*

7. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresher using CREATE DXTPSB, CREATE DATATYPE, SUBMIT and INITDEM commands.

   - For information on the syntax of these commands, see the *DataRefresher Command Reference.*

Using control blocks

DataRefresher communicates with your exit routines through control blocks. A *control block* is a storage area used by a program to hold status, control, and descriptive information. Control blocks pass information back and forth between DataRefresher and your exit routines.

Each type of exit routine has its own type of control block. GDI exit routines use two control blocks: a Data Communication control block and a Data Description control block.

The following illustration shows the interaction between DataRefresher, a control block, and an exit routine:

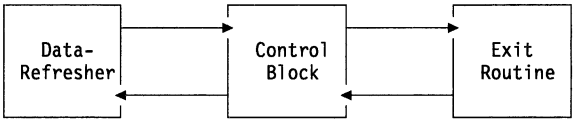

Figure 6. Interaction between DataRefresher, a control block and an exit routine

DataRefresher can also communicate with your exit routines through other storage areas, such as:

- Input and output buffers (Date/Time Conversion, Data, User Data Type and GOI exit routines)
- FETCH buffers (GDI exit routines)
- Global work areas (Data, GDI and GOI exit routines)

Accessing control blocks

DataRefresher provides a set of control blocks for each supported programming language (PL/I, COBOL and Assembler). These control blocks are shipped with DataRefresher for your use in writing exit routines. The control blocks for each programming language are contained in the DVR110.DVRINTER library.

DataRefresher uses the control blocks identified in Table 3 on page 13 to communicate with the various exit routines. Only the control blocks identified in this table should be used to request or receive services of DataRefresher.

Product Sensitive Programming Interface

Table 3. DataRefresher exit routine control blocks.

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Accounting exit	DVRACXP	DVRACXC	DVRACXA
Data exit	DVRDAXP	DVRDAXC	DVRDAXA
Date/Time Conversion exit	DVRXCCXP	DVRXCCXC	DVRXCCXA
GDI Record exit: Data Communication Data Description	DVRXCRCP DVRXCRDP	DVRXCRCC DVRXCRDC	DVRXCRCA DVRXCRDA
GDI Select exit: Data Communication Data Description	DVRXCSCP DVRXCSDP	DVRXCSCC DVRXCSDC	DVRXCSCA DVRXCSDA
Map Capture exit	DVRXCMXP	DVRXCMCC	DVRXCMXA
User Data Type exit	DVRXCUTP	DVRXCUTC	DVRXCUTA
GOI exit	DVRXCGOP	DVRXCGOC	DVRXCGOA

End of Product Sensitive Programming Interface

Modifying control blocks

You can write your exit routines to change the contents of certain unprotected fields in the control blocks. For example, you can change certain return codes, scratchpad space, and message text areas. The Accounting, Data, and User Data Type exit routine control blocks have protected areas that the exit routine is forbidden to change. If your exit routine modifies a protected field in the control block, DataRefresher issues a message and stops processing of the extract request.

Exit routine return codes

When your exit routine returns control to DataRefresher, it tells DataRefresher what happened. Your exit routines use *return codes* to communicate this information to DataRefresher. Each control block (except the GDI Data Description control block) contains a field in which the exit routine includes a return code for DataRefresher. The return codes for GDI exit routines are placed in the Data Communication control block.

Return codes for each exit routine type are listed in Appendix B, "Exit routine return codes" on page 143.

Copying and printing sample exit routines and control blocks

DataRefresher provides sample exit routines and control blocks for your use in writing your exit routines. The sample exit routines and control blocks are contained in the DVR110.DVRINTER library. You can copy or print these samples using the usual procedures for printing partitioned data set (PDS) members at your site.

Table 4 lists the sample exit routines provided with DataRefresher:

Table 4. DataRefresher sample exit routines contained in the DVR110.DVRINTER library

Exit Routine	PL/I	COBOL	Assembler
Accounting exit	DVRXYACX	DVRXOACX	DVRXAACX
Data exit - conversion	DVRXYDPM	DVRXODPM	DVRXADPM
Data exit - summarization	DVRXYXDS	DVRXOXDS	DVRXAXDS
Date/Time Conversion exit	DVRXYTD	DVRXOTD	DVRXATD
GDI Record exit	DVRXYHIX	DVRXOGRX	
GDI Select exit		DVRXOGSX	DVRXAGSX
Map Capture exit	DVRXYMAP	DVRXOMAP	
User Data Type exit	DVRXYXUT	DVRXOXUT	DVRXOXVC
DVRXAXUT			
GOI exit	DVRXYGOI	DVRXOGOI	DVRXAGOI

Setting up DataRefresher to use programming languages

DataRefresher supports exit routines written in Assembler. To use exit routines written in OS PL/I or VS COBOL II, you must configure DataRefresher to set up the proper processing environment. If you use a language version that supports Language Environment/370 (LE/370), you need to configure DataRefresher to take advantage of LE/370's improved performance and error handling.

Extra configuration is not required if your exit routines are written in Assembler.

When writing exit routines in PL/I:

- The value MAIN on the OPTIONS clause of the PROC statement in the exit routine must not be used
- If you want to use SYSPRINT with GDI exits, the parameters DECLARE, OPEN, and CLOSE SYSPRINT must be specified
- If you are using an LE/370 version of PL/I, specify the 'FETCHABLE' procedure option for all exit routines except Accounting exit routines.

When writing exit routines in COBOL:

- Use the compiler option RESIDENT. This option is not required if you are using an LE/370 version of COBOL.
- Code the GOBACK statement to return from the exit routine to DataRefresher.
- Use the QUOTE compiler option when compiling your COBOL exit routines as the copybooks supplied for the DataRefresher control blocks use double quotes to delimit literals.

Informing DataRefresher about PL/I and COBOL

To inform DataRefresher about the language in which an exit routine is written, code the EXITLANG keyword. If you do not code an EXITLANG keyword, DataRefresher assumes that all exit routines it calls are written in Assembler.

The EXITLANG keyword can be included in various commands depending on whether you are running the UIM or the DEM:

- When running the UIM, code the EXITLANG keyword in one of the following:
  - PARM keyword of the JCL EXEC PGM statement. If you are using the catalog procedure DXTINPUT, the corresponding keyword is EXITLNG.
  - PARM keyword of TSO batch RUN PROGRAM command.
  - TSO batch RUN CP command.

  For example, code the EXITLANG keyword in the UIM JCL as follows to specify that your exit routine is written in PL/I:

  ```
  EXEC PGM=DVRU0000,PARM='EXITLANG=PLI'
  ```

  You can run the UIM using TSO batch commands if you must run the UIM as a DB2 application. The RUN PROGRAM command runs the UIM as a TSO batch program, while the RUN CP command runs the UIM as a TSO command processor. For example, code the RUN PROGRAM command as follows to specify that your exit routine is written in COBOL:

  ```
  RUN PROGRAM (DVRU0000)
      PLAN    (EXITPLAN)
      LIBRARY ('DVR110.DVRLOAD')
      PARM    ('EXITLANG=COBOL')
  ```

  Code the RUN CP command as follows to specify that your exit routine is written in COBOL:

  ```
  RUN CP
      PLAN    (EXITPLAN)
      LIBRARY ('DVR110.DVRLOAD')
      DVRU0000 EXITLANG=COBOL
  ```

- When running the DEM, code the EXITLANG keyword in the INITDEM command.

  For example, code the EXITLANG keyword on the INITDEM command as follows to specify that your exit routine is written in VS COBOL II:

  ```
  INITDEM NAME=DEM1,EXITLANG=COBOL;
  ```

Table 5 shows you when to code the EXITLANG keyword for non-Assembler exit routines:

Table 5. When to code the EXITLANG keyword

Exit Routine	In UIM JCL	On INITDEM Command
Accounting exit	yes	yes
Data exit	no	yes
Date/Time Conversion exit	no	yes
GDI Select exit GDI Record exit	yes see note 1 below	yes yes
Map Capture exit	yes	yes
User Data Type exit	see note 2 below	yes
GOI exit	no	yes

Note:

- If you choose to support DESCRIBE calls to a GDI Record exit routine by specifying DETAIL=EXIT on the CREATE DXTFILE command, you must code the EXITLANG keyword in both the UIM JCL and the INITDEM command. DESCRIBE calls return field descriptions supplied by the source data (a self-defining file, for example) or by an auxiliary of the source data (a data dictionary, for example) to DataRefresher.
- For User Data Type exit routines, if you have coded VARIES as the value on any of the CREATE DATATYPE commands, the UIM calls the exit routine when it processes a CREATE DXTFILE or CREATE DXTPCB command which defines a field using this data type, in order to get the length of the field from the exit routine. You must code the EXITLANG keyword in the UIM JCL in this case.

If you use multiple exit routines, some written in OS PL/I and others in VS COBOL II, you can specify both languages when you code the UIM JCL or on the INITDEM command. For example, you could code:

```
EXEC PGM=DVRU0000,PARM='EXITLANG=(PLI,COBOL)'
```

If you have exit routines written in an LE/370 HLL, `'EXITLANG=LE'` is the only valid option.

If you do not code an EXITLANG keyword, DataRefresher assumes that all exit routines it calls are written in Assembler. A message is issued by DataRefresher to remind you of this. If you use a PL/I or VS COBOL II exit routine but fail to notify DataRefresher to set up the appropriate language environment, you might get unpredictable results, or your job step might stop processing and abend. It is not an error, however, to set up a single language environment (just PL/I or VS COBOL II) or a dual language environment (PL/I and VS COBOL II) and then use only Assembler exit routines.

Note: However, performance may be degraded as the language environment is reset for each call to the exit.

Informing DataRefresher about Language Environment/370

Using Systems Application Architecture* (SAA*) AD/Cycle* Language Environment*/370 in conjunction with DataRefresher provides improved performance and better error handling for the exit routine and for DataRefresher. To use a language version that supports LE/370, specify LE/370 as the language for DataRefresher to use.

To use an LE/370 supported HLL for the UIM, code the EXITLANG keyword on the JCL EXEC PGM statement as follows:

```
//UIMSTEP    EXEC  PGM=DVRU0000,PARM=('DEBUG=1,EXITLANG=LE')
//FDTLIB     DD    DSN=DVR110.FDTLIB,DISP=SHR
//EXTLIB     DD    DSN=DVR110.EXTLIB,DISP=SHR
```

To use an LE/370 supported HLL for the DEM, code the EXITLANG keyword on the INITDEM command as follows:

```
//DEM.DXTIN    DD DATA,DLM=$$
  INITDEM NAME=DEMNAME,
          RUNMODE=...,
          :
          :
          EXITLANG=(LE);
```

To upgrade an exit routine written in OS PL/I or VS COBOL II to a language version that supports LE/370, you must recompile the exit routine under LE/370. You can then specify LE/370 as the language for the exit routine. Otherwise you will still need to specify PL/I or COBOL.

Preparing the language load module for OS PL/I and VS COBOL II

Before you can use your exit routines in OS PL/I or VS COBOL II, you must first set up the connection between DataRefresher and the language environment. For example, if you write your exit routines in:

- PL/I, the language load module needs to be compiled and link-edited.
- VS COBOL II, the language load module needs to be link-edited.

Note: If you use Assembler, no compiling or link-editing is required.

You must link-edit the DataRefresher language load module once for the PL/I or VS COBOL II language environment (independent of how many exit routines you have). You must link-edit the DEM and depending on the exit routine, the UIM., Table 6 shows the types of exit routines accessed by UIM and DEM.

Table 6. Types of Exit Routines accessed by the UIM and the DEM

Exit Routine	UIM	DEM
Accounting exit	X	X
Data exit		X
Date/Time Conversion exit		X
GDI exit	X	X
Map Capture exit	X	X
User Data Type exit	X	X
GOI exit		X

Use either JCL procedures or the `Linkage Edit` options of ISPF/PDF. to establish the connection between DataRefresher and the language load module. "Compiling and link-editing the language load modules for OS PL/I" describes how to use JCL procedures to perform these tasks. Information on ISPF/PDF is available via the ISPF/PDF online help.

Compiling and link-editing the language load modules for OS PL/I

The language load modules are automatically added to the DVR110.DVRINTER library when DataRefresher is installed. For OS PL/I exit routines, you must compile and link-edit the appropriate language load modules to the UIM and DEM.

For the UIM

For the UIM, compile the PLIOPT module named DVRXYUPL. The module DVRXYUPL is a PL/I MAIN module that ensures that DataRefresher initializes the PL/I environment only once for the run of the UIM. After compiling DVRXYUPL, you link-edit the object deck resulting from the compilation with the UIM load module, as shown in Figure 7.

```
//*****************************************************************
//* THIS JCL DOES A PL/I COMPILE/LINK EDIT, REPLACING THE SHIPPED
//* DUMMY PLICALLB CSECT IN THE UIM LOAD MODULE (DVRU0000)
//* WITH THE PL/I MAIN PROCEDURE PRODUCED BY COMPILING DVRXYUPL.
//*****************************************************************
//PLI     EXEC PGM=IEL0AA,PARM='OBJECT,NODECK,XREF,ATTRIBUTES(FULL)'
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(800,(300,300))
//SYSPRINT DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSTERM  DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT4   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSIN    DD  DISP=SHR,DSN=DVR110.DVRINTER(DVRXYUPL) SOURCE FOR COMPILE!
//*****************************************************************
//*   LINK EDIT IF THE COMPILE RETURN CODE IS 4 OR LESS -
//*   REPLACES THE DVRU0000 (UIM) LOAD MODULE
//*****************************************************************
//LKED    EXEC PGM=IEWL,PARM='LIST,XREF,LET,RENT,AMODE=NN,RMODE=ANY', 1
//      COND=(4,LT,PLI)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.PLIBASE
//         DD  DISP=SHR,DSN=DVR110.DVRLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DVRLOAD   TARGET LIB FOR LOAD MOD
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD  *
  REPLACE PLICALLB
  INCLUDE SYSLIB(DVRU0000)
 ENTRY   DVRXSUSP
 NAME    DVRU0000(R)
/*
```

Figure 7. Sample link-edit of PL/I language load module to the UIM

1 Specify the same AMODE as the original module. For information on AMODE and RMODE restrictions, see "AMODE and RMODE dependencies" on page 23.

This link-edit produces a return code of 0.

For the DEM

For the DEM, compile the PLIOPT module named DVRXYPLI. The module DVRXYPLI is a PL/I MAIN module that ensures that DataRefresher initializes the PL/I environment only once for the run of the DEM. After compiling DVRXYPLI, you link-edit the text deck resulting from the compilation with the DEM load module, as shown in Figure 8.

```
//*****************************************************************
//* THIS JCL DOES A PL/I COMPILE/LINK EDIT, REPLACING THE SHIPPED
//* DUMMY PLICALLB CSECT IN THE DEM LOAD MODULE (DVRX0000)
//* WITH THE PL/I MAIN PROCEDURE PRODUCED BY COMPILING DVRXYPLI.
//*****************************************************************
//PLI     EXEC PGM=IEL0AA,PARM='OBJECT,NODECK,XREF,ATTRIBUTES(FULL)'
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(800,(300,300))
//SYSPRINT DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSTERM  DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT4   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSIN    DD  DISP=SHR,DSN=DVR110.DVRINTER(DVRXYPLI) SOURCE FOR COMPILE!
//*****************************************************************
//*****************************************************************
//*   LINK EDIT IF THE COMPILE RETURN CODE IS 4 OR LESS -
//*   REPLACES THE DVRX0000 (DEM) LOAD MODULE
//*****************************************************************
//LKED    EXEC PGM=IEWL,PARM='LIST,XREF,LET,RENT,AMODE=NN,RMODE=ANY', 2
//      COND=(4,LT,PLI)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.PLIBASE
//         DD  DISP=SHR,DSN=DVR110.DVRLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DVRLOAD   TARGET LIB FOR LOAD MOD
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD  *
 REPLACE PLICALLB
 INCLUDE SYSLIB(DVRX0000)
 ENTRY   DVRXSSSU
 NAME    DVRX0000(R)
/*
```

Figure 8. Sample link-edit of PL/I language load module to the DEM

2 Specify the same AMODE and RMODE as the original module. For information on AMODE and RMODE restrictions, see "AMODE and RMODE dependencies" on page 23.

This link-edit produces a return code of 0.

Link-editing the language load modules for VS COBOL II

For VS COBOL II exit routines, you must link-edit the appropriate language load modules to the UIM and DEM. When you run the JCL to link-edit the language load modules, you should receive a return code of 0.

DataRefresher uses the VS COBOL II initialization modules in the COBOL II program library when link-editing the language load modules to both the UIM and the DEM, as shown in Figure 9 and Figure 10.

```
//*************************************************************
//*  ADD CBL INIT TO UIM LOAD MODULE DVRU0000                 ***
//*************************************************************
//LINKUIM  EXEC PGM=IEWL,PARM='XREF,LIST,LET,AMODE=NN,RMODE=ANY' 3
//SYSPRINT DD  SYSOUT=A
//SYSLIB   DD  DSN=SYS1.COB2LIB,DISP=SHR
//         DD  DSN=DVR110.DVRLOAD,DISP=SHR
//SYSLMOD  DD  DSN=DVR110.DVRLOAD,DISP=OLD
//SYSUT1   DD  DISP=(NEW,DELETE),DSN=&&WORKDS,SPACE=(1024,(200,20))
//SYSLIN   DD  *
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 REPLACE ILBOSTP0
 INCLUDE SYSLIB(DVRU0000)
 ENTRY   DVRXSUSP
 NAME    DVRU0000(R)
/*
```

Figure 9. Sample link-edit of VS COBOL II language load module to the UIM

```
//*************************************************************
//*  ADD CBL INIT TO DEM LOAD MODULE DVRX0000                 ***
//*************************************************************
//LINKDEM  EXEC PGM=IEWL,PARM='XREF,LIST,LET,AMODE=NN,RMODE=ANY' 3
//SYSPRINT DD  SYSOUT=A
//SYSLIB   DD  DSN=SYS1.COB2LIB,DISP=SHR
//         DD  DSN=DVR110.DVRLOAD,DISP=SHR
//SYSLMOD  DD  DSN=DVR110.DVRLOAD,DISP=OLD
//SYSUT1   DD  DISP=(NEW,DELETE),DSN=&&WORKDS,SPACE=(1024,(200,20))
//SYSLIN   DD  *
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 REPLACE ILBOSTP0
 INCLUDE SYSLIB(DVRX0000)
 ENTRY   DVRXSSSU
 NAME    DVRX0000(R)
/*
```

Figure 10. Sample link-edit of VS COBOL II language load module to the DEM

3 Specify the same AMODE and RMODE as the original module. For information on AMODE and RMODE restrictions, see "AMODE and RMODE dependencies" on page 23.

Precompiling and binding exit routines for DB2 applications

To access or join data from certain databases, you must prepare your GDI and Map Capture exit routines. For example, to join data from a DB2 table with data from an IMS database, you must precompile and bind the GDI Record exit routine used to access the DB2 database.

Precompiling your exit routine

The DB2 precompiler scans your exit routine source code and produces a modified program in which every SQL statement has been replaced by one or more host language statements. The precompiler also produces a database request module (DBRM) for binding.

Binding your exit routine to DB2

Before you can execute your exit routine, you must establish a relationship between the program and its DB2 data. This process is called *binding*. You must bind programs on the same system that invokes DB2 and creates a DB2 plan for the exit routine. DataRefresher jobs that call the exit routine in the TSO batch and IMS environments specify this DB2 plan.

Figure 11 shows an example of JCL to precompile a GDI Select exit routine:

```
//PRECOMP  EXEC PGM=DSNHPC,PARM='HOST(COB2) XREF SOURCE',REGION=2048K
//STEPLIB  DD DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DVR110.DVR110.DBRMLIB(DVRXOGSX),DISP=SHR
//SYSCIN   DD SYSOUT=B
//SYSPRINT DD SYSOUT=A
//SYSTERM  DD DUMMY,DCB=BLKSIZE=800
//SYSUT1   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
*** ADD THE EXIT SOURCE FOLLOWING THE SYSIN DDCARD
//SYSIN    DD *
```

Figure 11. JCL to precompile a GDI Select exit routine as a DB2 application

Figure 12 shows an example of JCL to bind a GDI Select exit routine:

```
***STEP0 EXEC PGM=CP,PARM='M STEIN  REMBIND STARTED'
***  DB2 MUST BE UP BEFORE RUNNING THIS JOB
//TSOTMP   EXEC  PGM=IKJEFT01,REGION=2048K
//STEPLIB  DD DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSTSPRT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
*** ADD THE BIND COMMAND FOLLOWING THE SYSTSIN DDCARD
//SYSTSIN  DD *
```

Figure 12. JCL to bind an exit routine as a DB2 application

An example of this process is shown in "GDI exit routine examples" on page 103. For more information on DB2 precompiling and binding, see *IBM DATABASE 2 Version 2: Application Programming and SQL Guide*.

Compiling and link-editing your exit routines

The purpose of compiling and link-editing is to produce an executable load module. With the exception of Accounting exit routines (see "Compiling and link-editing your Accounting exit routines" on page 37), all exit routines are compiled and link-edited by the same process.

To compile and link-edit your exit routines, you can use either JCL procedures or the `Linkage Edit` options of the ISPF/PDF. The following section describes how to use JCL procedures to compile and link-edit your exit routines. Information on ISPF/PDF is available via the ISPF/PDF online help.

JCL procedures for compiling and link-editing exit routines

The JCL needed to compile and link-edit exit routines is a standard compile and link-edit for a stand-alone module. With the exception of Accounting exit routines (see "Compiling and link-editing your Accounting exit routines" on page 37),

When the DEM executes an extract request, DataRefresher automatically loads the appropriate module (named in the data description) and calls it via its main entry point. The main entry point is:

- PLISTART for OS PL/I, CEESTART for LE/370 HLLs
- The program identification you assign to your COBOL exit routine
- The CSECT name you assign to your Assembler exit routine

To simplify maintenance and security, you should store all exit routine load modules in one load library. When you write the JCL for calling the DEM, you must specify a DD statement for the load library. You can do this by:

- Adding the load library to the JOBLIB or STEPLIB DD concatenation so that the DEM can access the exit routines
- Passing it on the PGMLIBx parameter of the DXTRUN procedure

DataRefresher allows up to 4 substitution libraries to be passed on the DXTRUN procedure.

If you do not specify the load library in the JCL, DataRefresher is unable to execute any extract requests requiring that exit routine. DataRefresher requeues the requests for later processing. The extract request can also fail if the DEM attempts to load a module containing an exit routine and there is not enough storage available. If so, the DEM is unable to execute any extract request requiring that exit routine. The DEM requeues the request for later processing.

Accessing DB2 with GDI exit routines using DataRefresher OS/2

If your GDI exit routine accesses DB2, and you are including your exit routine in extract requests submitted via DataRefresher OS/2, your link-edit JCL will differ slightly from the usual link-edit for the TSO environment.

For information on the link-edit procedures for accessing DB2 with GDI exit routines via DataRefresher OS/2, see "Accessing DB2 when using DataRefresher OS/2" on page 102.

AMODE and RMODE dependencies

There are restrictions on using MVS functions related to RMODE/AMODE link-edit options. For information on these restrictions, see *MVS/ESA Linkage Editor and Loader User's Guide*.

The RMODE environment encountered by Accounting exit routines will be the same as the RMODE used to link-edit the UIM and DEM. This environment is normally RMODE=ANY in MVS/ESA*. All other types of exit routines will be given control in the RMODE/AMODE specified at link-edit time. To execute correctly, COBOL and PL/I exit routines must be in the same mode as DataRefresher because general COBOL and PL/I initialization is performed in the DataRefresher code. Assembler exit routines do not have the same restrictions because they do not depend on DataRefresher storage to run.

If you are running in 31-bit mode, you can:

- Link-edit exit routines to run in 31-bit mode by including the following link editor control statements in the link-edit stream after the exit routine:

```
ENTRY entryname
MODE  AMODE(31), RMODE(ANY)
NAME  exitname(R)
```

- Specify the AMODE and RMODE parameters on the PARM value of the EXEC statement that runs the link editor. For sample JCL that specifies AMODE and RMODE parameters for compiling and link-editing an Accounting exit routine, see "Compiling and link-editing your Accounting exit routines" on page 37.

Registering your exit routines for use with DataRefresher OS/2

Exit routines are written, compiled and link-edited on your host MVS system. After they have been written, compiled and link-edited on the host system, your exit routines can be included in extract requests submitted from a workstation via DataRefresher OS/2. However, you must register the following types of exit routines before you can include them in your DataRefresher OS/2 extract requests:

- Data exit routines
- User Data Type exit routines
- Date/Time Conversion exit routines
- Map Capture exit routines
- GDI exit routines
- GOI exit routines

You do not have to register Accounting exit routines to include them in extract requests submitted via DataRefresher OS/2.

Exit routines are registered using the User-Defined World templates. To access these templates, open the Templates icon in the User-Defined World window. The Templates window contains the following icons:

- GDI Exits
- GOI Exits
- User Exits, which includes Data, User Data Type, Date/Time Conversion, and Map Capture exit routines

To register an exit routine with DataRefresher OS/2, drag and drop the icon representing the exit routine type you want from the Templates window into the User Defined World–Icon View window. A new window appears requesting information about the exit routine you wish to register, such as:

- The name you want to give the object representing the exit routine on the DataRefresher OS/2 interface
- The load library name
- The load library member name, that is, the name of the exit routine on the host system
- The language in which the exit routine is written
- Input and output data types, record lengths, and number of decimal places (for User Data Type exit routines)
- Source data set and ddnames (for GDI exit routines)
- DB2 subsystem and application plan names (for GDI exit routines)
- Target data set and ddnames (for GOI exit routines)

Online help is provided for each entry field.

When you have completed all the information required to register the exit routine, select the REGISTER pushbutton. A message is returned to tell you that the exit routine has been registered.

After you have registered an exit routine, it appears as an object in the User Defined World window. The exit routine becomes available to all users of DataRefresher OS/2 the next time they either:

- Re-open the window
- Log on to DataRefresher OS/2

For more information on including exit routines in extract requests submitted using DataRefresher OS/2, see the *DataRefresher OS/2 User's Guide*.

Exit routine options

Before you write exit routines for your site for use with DataRefresher, you should consider the following:

- Using multiple exit routines
- Using exit routine messages
- Making your exit routines reentrant
- Debugging your exit routines
- Maintaining the security of your exit routines

Using multiple Accounting, Data and User Data Type exit routines

You can write and use more than one exit routine of each type. For example, you may want to have separate Accounting exit routines for the UIM and the DEM. DataRefresher could then call the UIM Accounting exit routine for validation and the DEM Accounting exit routine for the rest of the processing. You might want a Data exit routine for each segment type to be altered, or a User Data Type exit routine for each data type.

Using exit routine messages

All of the exit routines, except Date/Time Conversion exit routines, can cause DataRefresher to write messages to DXTPRINT. The exit routine will move the message text to the control block before returning to DataRefresher. DataRefresher then imbeds the text into one of its own messages and writes it to the DXTPRINT data set.

Table 7 gives the variable name, the number of messages allowed and the length of messages for each type of exit routine.

Table 7. Exit routine messages

Exit Routine	Variable Name	Number of Messages	Length of Text
Accounting exit	ACXSMESG ACXUMESG (see note)	1 1 per extract request	64 64
Data exit	DAXSMESG	1	64
GDI Select exit (Data Communication control block)	GSCMSG	5	72
GDI Record exit (Data Communication control block)	GRCMSG	5	72
Map Capture exit	MXCMESG	10	64
User Data Type exit	UDTXMESG	1	64
GOI exit	GOIMSG	1	72

Note: Message text from an Accounting exit routine placed in ACXUMESG goes to DXTPRINT, and also goes to the user if the SUBMIT command for the extract request includes the MSGS keyword. Message text placed in ACXSMESG only goes to DXTPRINT.

The control blocks for Accounting and Data exit routines contain the address of the DXTPRINT DCB, so the exit routines can write directly to DXTPRINT, if necessary.

Making your exit routines reentrant

A reentrant exit routine can be used concurrently by more than one program. At times, you might want to write your exit routine to be *reentrant*.

Reentrant exit routines should not:

- Limit access to prevent concurrent use by multiple programs
- Change themselves
- Contain data areas that can be modified in local storage

Accounting exit routines are link-edited with the UIM and DEM. If Accounting exit routines are not reentrant, you lose the benefits of having the UIM and DEM being reentrant, but DataRefresher still runs.

VS COBOL II assumes that the default run-time environment option supplied by IBM is not reusable. For more information, refer to the *VS COBOL II Application Programming Guide*.

Each DataRefresher exit routine control block contains work space you can use for working storage. Your Assembler exit routines can issue a GETMAIN instruction to acquire additional storage to hold saved or modifiable data, buffers, data control blocks (DCBs), or anything else the exit routine is designed to use. Normally, it is best to acquire the additional storage at the first call of the exit routine; you then can store the address of this additional storage in the work space of the control block.

Debugging your exit routines

DataRefresher provides a trace facility that can help you detect errors in your exit routines. You can specify *debug tracing* with the DEBUG value in the:

- OPTIONS keyword of your EXTRACT statement
- INITDEM command used to initialize the DEM
- TSO batch RUN CP command used to run the UIM
- PARM keyword of the JCL EXEC statement used to run the UIM
- PARM keyword of TSO batch RUN PROGRAM command used to run the UIM

Use debug level 3 to trace all module calls and returns. This includes calls and returns to and from exit routines. Level 4 returns the same information as level 3 and also includes dumps of the control blocks before and after DataRefresher calls the exit routine.

Two switches useful for debugging—the EXIT-ENTERED switch and the EXIT-IN-CONTROL switch—are in the control blocks for Accounting, Data, User Data Type, GDI, GOI and Map Capture exit routines. (The Date/Time Conversion exit routine control blocks do not include these switches.) DataRefresher does not require that you use these switches. The switches can, however, help you to determine where the problem occurred if you have an abend. DataRefresher sets these one-byte switch fields to blank (`' '`) before the first time the UIM or DEM calls your exit routine.

As your exit routine is entered, it should turn on the EXIT-ENTERED switch by setting it to X. DataRefresher does not alter that field again, so if an abend occurs during the current run of the UIM or DEM, you can tell whether the exit routine has been called by checking the appropriate control block dump.

The exit routine should also turn on the EXIT-IN-CONTROL switch by setting it to X. When control returns to DataRefresher, the field is reset to blank. Thus, this switch can help you determine whether DataRefresher or the exit routine had control if an abend occurs.

Table 8 on page 27 shows the debugging switches listed by exit routine type in the DataRefresher control blocks.

Table 8. DataRefresher exit routine debugging switches		
Exit Routine	**Exit-Entered Switch**	**Exit-In-Control Switch**
Accounting exit	ACXENTRD	ACXINCTL
Data exit	DAXENTRD	DAXINCTL
GDI Select exit	GSCENTRD	GSCINCTL
GDI Record exit	GRCENTRD	GRCINCTL
Map Capture exit	MXCENTRD	MXCINCTL
User Data Type exit	UDTENTRD	UDTINCTL
GOI exit	GOIENTRD	GOIINCTL

Maintaining the security of your exit routines

You can ensure the security of your exit routines by placing the exit load modules in a program library that is protected by the Resource Access Control Facility (RACF*). With RACF, only users who have the proper authority can access or change exit routines stored in the library. See the *DataRefresher Administration Guide* for more information about how to protect a program library using RACF.

You can also control who runs extract requests at your site by using an Accounting exit routine that requires an ACCOUNT value, and by setting up that Accounting exit routine to approve only extract requests that meet your requirements. Any extract requests that do not meet your requirements can be handled by the exit routine as you choose.

For information on specifying the disposition of rejected or unprocessed extract requests, see "Handling unprocessed extract requests and batches" on page 45.

Additional security functions required by a particular data source can be enforced with the help of a GDI exit routine. DataRefresher can pass the Data Communication control block to the GDI exit routine for possible assistance in checking the values of these SUBMIT command keywords: ACCOUNT, EXTID, USERID, and NODE. These values could also be validated by your Accounting exit routine. The GDI exit routine can also assist in checking the values of GDI user parameters specified in the UIM JCL or on the INITDEM command.

Using exit routines written for DXT

To use exit routines written to work with DXT* Version 2 Releases 4 and 5:

- You must recompile and link-edit the Assembler Accounting exit routine, DVRXAACX. In addition, any other Accounting exit routines written to work with DXT must be link-edited to include the new control blocks.
- Map Capture exit routines written to work with DXT must be recompiled and link-edited to include the new control blocks. New fields have been added to the control block to increase the versatility of the information passed to the exit routine.

The interface between DataRefresher Version 1 and Data, User Data Type, Date/Time Conversion and GDI exit routines is unaffected.

Chapter 3. Writing Accounting exit routines

Accounting exit routines enable you to keep track of DEM utilization. With an Accounting exit routine, you can save information about extract requests, or approve or reject extract requests according to the criteria you have specified. Accounting exit routines also enable you to override the output limits and priority values specified on extract requests.

This chapter contains information on writing and using Accounting exit routines.

Task overview

Writing and using Accounting exit routines with DataRefresher consists of:

1. Determining what functions you want your Accounting exit routine to perform.

   For information on:

   - What functions Accounting exit routines can perform, see "Accounting exit routines" on page 2.
   - How you can use Accounting exit routines at your site, see "Using Accounting exit routines" on page 44.

2. Writing your Accounting exit routine.

   For information on:

   - How Accounting exit routines interact with the control block, see "Using the Accounting exit routine control block" on page 32 and "Relationship of exit call types to the control block" on page 33.
   - Using the sample exit routines provided with DataRefresher, see "Writing Accounting exit routines using sample exit routines" on page 37.

3. *If your Accounting exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.

   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.

4. Compiling and link-editing your Accounting exit routine.

   - For information on compiling and link-editing Accounting exit routines, see "Compiling and link-editing your Accounting exit routines" on page 37.

5. Determining what you want DataRefresher to do with rejected extract requests.

   For information on what to do with rejected extract requests, see "Handling unprocessed extract requests and batches" on page 45.

Calling Accounting exit routines

DataRefresher automatically calls an Accounting exit routine when you submit an extract request to the UIM or when you run the DEM. The exit routine called can be either the:

- Nonfunctional Accounting exit routine that is shipped with DataRefresher
- Accounting exit routine that you write, and that replaces the sample shipped with DataRefresher

The UIM and DEM make the following types of calls to an Accounting exit routine to account for DEM use:

1. VALIDATE
2. OPEN
3. BEGIN-BATCH
4. END-BATCH
5. CLOSE

Figure 13 shows how the UIM and DEM call an Accounting exit routine.

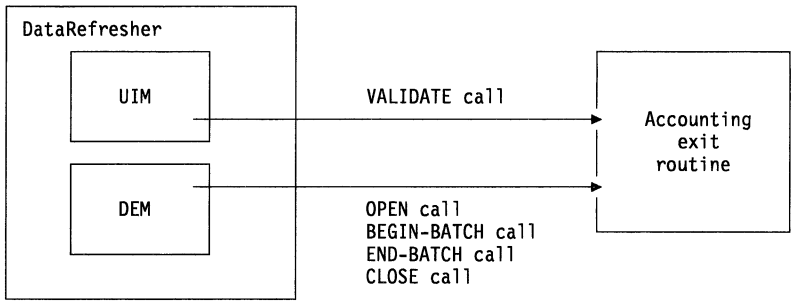

Figure 13. How DataRefresher calls Accounting exit routines

VALIDATE

When you submit an extract request to the UIM, the UIM checks the syntax of the SUBMIT command, including the ACCOUNT parameter. After the UIM has determined that the syntax is correct, the UIM calls the Accounting exit routine with a VALIDATE call.

During the VALIDATE call, the exit routine can:

- Approve the request, including the accounting information that you specified
- Reject the request, including the accounting information that you specified, and prevent the extract request from being placed in the EXTLIB
- Change the output limit of the extract request established (or the default value) on the SUBMIT/EXTRACT command of the extract request
- Change the priority of the extract request established (or the default value) on the SUBMIT/EXTRACT command of the extract request

After the Accounting exit routine has completed the VALIDATE call and returned control to the UIM, and if the return code allows, the extract request is placed in the EXTLIB to await execution by the DEM.

OPEN

During the OPEN call, the Accounting exit routine determines whether the DEM can continue running, according to the accounting information you have specified. The exit routine passes the result, in the form of a return code, to the DEM. Also during the OPEN call, the exit routine performs all one-time initialization functions, such as opening any datasets that it may use and acquiring any additional storage it may need.

After the DEM is initialized, the DEM obtains a batch of extract requests from the EXTLIB to process, and then translates the batch into executable form. The number of extract requests in a batch can vary from one to the number of DXTOUT datasets you have defined in the JCL used to run the DEM.

At this point, the control block, containing information about each of the extract requests in the batch, is built.

BEGIN-BATCH

When a BEGIN-BATCH call is made to the Accounting exit routine, the exit routine sends a return code to tell DataRefresher whether or not to execute the batch and its individual extract requests, according to the accounting information you have specified. Also during the BEGIN-BATCH call, the exit routine can cause DataRefresher to remove or re-queue individual extract requests in the EXTLIB according to the priorities you have specified.

For information on batching extract requests, see the *DataRefresher Administration Guide*.

END-BATCH

After the DEM has finished processing the batch of extract requests, DataRefresher records information in the control block about each extract request and about the batch (such as counts of source data accesses, extract records written, and extract request completion codes). DataRefresher then makes an END-BATCH call to the Accounting exit routine. Your exit routine can be written to record the information that DataRefresher stores in the control block.

If one or more batches are awaiting processing, DataRefresher makes multiple BEGIN-BATCH and END-BATCH calls to the exit routine until the DEM has processed all batches.

CLOSE

After the DEM has completed all the batches of extract requests it can process for the current run, and has cleaned up the remnants of processing, the DEM makes a CLOSE call to your exit routine. During the CLOSE call, the exit routine closes any data sets it may have used and issues a FREEMAIN instruction to free any additional storage it may have required.

When DataRefresher transfers control to your Accounting exit routine, register 1 points to the address of the control block. The exit routine returns to DataRefresher using an address pointed to by register 14. This is standard IBM linkage.

Using the Accounting exit routine control block

The Accounting exit routine control block is the interface between DataRefresher and your Accounting exit routine. There is a different map of the Accounting exit routine control block for each supported programming language. Accounting exit routine control blocks are contained in the DVR110.DVRINTER library, as shown in Table 9:

Table 9. Accounting exit routine control blocks

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Accounting exit	DVRACXP	DVRACXC	DVRACXA

The Accounting exit routine control block contains a fixed-length header with a variable number of fixed-length entries following it. The header contains general information that the exit routine and DataRefresher need. Each entry following the header contains information specific to a single extract request in the batch that is currently processing.

The header at the beginning of the control block contains the following types of information:

- Type of call being made to the exit routine
- Number of requests in the batch
- Text field for messages the exit routine wants DataRefresher to write to DXTPRINT
- Name of the source data for the batch
- Priority range of the DEM
- Output limit of the DEM

A varying number of fixed-length entries follow this standard information. There is one entry for each of the extract requests in the batch that the DEM is currently processing. Table 10 shows the number of fixed-length entries that follow each type of call to an Accounting exit routine.

Table 10. Number of fixed-length entries for calls to Accounting exit routines

Type of call	Number of fixed-length entries
VALIDATE	1
OPEN	0
BEGIN	variable
END	variable
CLOSE	0

Each of the fixed-length entries contains information about the extract request, for example the:

- Identification of the request (node, user, and extract ID)
- Length and content of the ACCOUNT value information
- Number of rows extracted for the request
- Completion code for the request

Your Accounting exit routine can modify the following fields on the Accounting exit routine control block:

- The Return code field (ACXRETC)
- The 114-byte scratchpad area (ACXSCRT1) in which the Accounting exit routine can save addresses and other information it uses
- The 47-byte scratchpad area (ACXSCRT2) for each extract request
- Two flag fields (ACXENTRD and ACXINCTL) which you can use for tracing and debugging
- Two DXTPRINT message text areas (ACXSMESG and ACXUMESG)
- A request disposition field (ACXRDISP)

Relationship of exit call types to the control block

Tables 11, 12, and 13 show the relationship of the exit call types to the different parts of the control block. The field usage codes in the tables indicate the following:

A DataRefresher fills in this field before this call; the exit routine should not modify this field

C DataRefresher fills in this field before this call; the exit routine may modify this field

T Set by exit routine for trace purposes

X Maintained by exit routine for its own purposes

– Information not available or applicable

Each programming environment has its own version of the control block. Table 11 shows the relationship of the exit call types to the fields in the Accounting exit routine control block.

Table 11. ACXPFX section of the control block

Field	Vali-date	Open	Begin	End	Close	Field Description
	A	A	A	A	A	DataRefresher control block prefix (not all fields shown)
ACXTNAME	A	A	A	A	A	Control block name
ACXELEN	A	–	A	A	–	Length of ACX entry
ACXCURRP	A	–	A	A	–	Address of first ACX entry

Table 12 shows the relationship of the exit call types to the ACXPFXE section of the Accounting exit routine control block.

Table 12 (Page 1 of 2). ACXPFXE section of the control block

Field	Vali-date	Open	Begin	End	Close	Field Description
	A	A	A	A	A	Non-repeating data
ACXCALL	A	A	A	A	A	Type of call to exit
ACXDEMNM	–	A	A	A	A	Name of DEM
ACXCOMTM	A	A	A	A	A	Time component started
ACXSYSPR	A	A	A	A	A	Address of DXTPRINT DCB
ACXPSBNM	–	A	A	A	A	Name of DXTPSB, if applies
ACXPCBLS	–	A	A	A	A	Address of list of PCB addresses (if IMS)
ACXBATNO	–	–	A	A	–	Number of this batch
ACXREQNO	–	–	A	A	–	Number of requests in batch (equals number of ACX entries below)
ACXBATTM	–	–	A	A	–	Time batch started
ACXSRCE1	A	–	A	A	–	Name of first file or PCB
ACXPSB1	A	–	A	A	–	Name of PSB if first is PCB
ACXSRCE2	A	–	A	A	–	Name of second file or PCB
ACXPSB2	A	–	A	A	–	Name of PSB if second is PCB
ACXSRCE3	A	–	A	A	–	Name of third file or PCB
ACXPSB3	A	–	A	A	–	Name of PSB if third is PCB
ACXSRCE4	A	–	A	A	–	Name of fourth file or PCB
ACXPSB4	A	–	A	A	–	Name of PSB if fourth is PCB
ACXSRCE5	A	–	A	A	–	Name of fifth file or PCB
ACXPSB5	A	–	A	A	–	Name of PSB if fifth is PCB
ACXSRCE6	A	–	A	A	–	Name of sixth file or PCB
ACXPSB6	A	–	A	A	–	Name of PSB if sixth is PCB
ACXSRCE7	A	–	A	A	–	Name of seventh file or PCB
ACXPSB7	A	–	A	A	–	Name of PSB if seventh is PCB
ACXSRCE8	A	–	A	A	–	Name of eighth file or PCB
ACXPSB8	A	–	A	A	–	Name of PSB if eighth is PCB
ACXSRCE9	A	–	A	A	–	Name of ninth file or PCB
ACXPSB9	A	–	A	A	–	Name of PSB if ninth is PCB
ACXSRCEA	A	–	A	A	–	Name of tenth file or PCB
ACXPSBA	A	–	A	A	–	Name of PSB if tenth is PCB
ACXSRCEB	A	–	A	A	–	Name of eleventh file or PCB
ACXPSBB	A	–	A	A	–	Name of PSB if eleventh is PCB
ACXSRCEC	A	–	A	A	–	Name of twelfth file or PCB
ACXPSBC	A	–	A	A	–	Name of PSB if twelfth is PCB
ACXSRCED	A	–	A	A	–	Name of thirteenth file or PCB

Table 12 (Page 2 of 2). ACXPFXE section of the control block						
Field	**Vali-date**	**Open**	**Begin**	**End**	**Close**	**Field Description**
ACXPSBD	A	–	A	A	–	Name of PSB if thirteenth is PCB
ACXSRCEE	A	–	A	A	–	Name of fourteenth file or PCB
ACXPSBE	A	–	A	A	–	Name of PSB if fourteenth is PCB
ACXSRCEF	A	–	A	A	–	Name of fifteenth file or PCB
ACXPSBF	A	–	A	A	–	Name of PSB if fifteenth is PCB
ACXSRCEG	A	–	A	A	–	Name of sixteenth file or PCB
ACXPSBG	A	–	A	A	–	Name of PSB if sixteenth is PCB
ACXSAMCT	–	–	–	A	–	Count of VSAM gets/batch
ACXVSMCT	–	–	–	A	–	Count of VSAM gets per batch
ACXDLICT	–	–	–	A	–	Count of IMS DL/I calls per batch
ACXGRXCT	–	–	–	A	–	Number of GDI Record exit FETCH calls
ACXGSXCT	–	–	–	A	–	Number of GDI Select exit FETCH calls
ACXPRILO	–	–	A	A	–	Low DEM priority limit
ACXPRIHI	–	–	A	A	–	High DEM priority limit
ACXOUTLM	–	–	A	A	–	DEM output limit
ACXRETC	C	C	C	C	C	Exit return code to DataRefresher
ACXSA	X	X	X	X	X	Exit register save area
ACXENTRD	T	T	T	T	T	"Exit-entered" flag
ACXINCTL	T	T	T	T	T	"Exit-in-control" flag
ACXSMESG	C	C	C	C	C	DataRefresher DXTPRINT message text
ACXSCRT1	X	X	X	X	X	Exit scratchpad space

Table 13 shows the relationship of the exit call types to the repeating ACX section (each occurrence represents an extract request) of the Accounting exit routine control block.

Table 13. ACX section of the control block

Field	Vali-date	Open	Begin	End	Close	Field Description
–	A	A	A	A	A	One entry per request (first addressed by ACXCURRP), one entry (for VA), ACXREQNO entries (for BB, EB)
ACXEID	A	–	A	A	–	Extract ID (from SUBMIT)
ACXUID	A	–	A	A	–	User ID (from SUBMIT)
ACXNID	A	–	A	A	–	Node ID (from SUBMIT)
ACXEXTNM	A	–	A	A	–	Formatted extract name
ACXACTSZ	A	–	A	A	–	Length of account information (from SUBMIT)
ACXACCT	A	–	A	A	–	Account information (from SUBMIT)
ACXROWCT	–	–	–	A	–	Number of extract rows
ACXCCODE	–	–	–	A	–	Request completion code
ACXRSNCD	–	–	–	A	–	Request reason code
ACXROWLM	C	–	A	A	–	Request output limit
ACXPRI	C	–	A	A	–	Request priority
ACXUMESG	C	–	C	C	–	DataRefresher user message text
ACXRDISP	–	–	C	–	–	Request disposition if stopped by exit
ACXSCRT2	X	–	X	X	–	Exit scratchpad space

Identifying your Accounting exit routine to DataRefresher

When you submit or run an extract request, DataRefresher automatically calls its own Accounting exit routine, a nonfunctional control section (CSECT) named DVRSACX that is shipped with DataRefresher. You can replace this sample CSECT with an Accounting exit routine that you write. DataRefresher will then call *your* Accounting exit routine instead of the sample CSECT, as long as you leave the name (the entry point) the same.

You do not have to register your Accounting exit routines to include them in extract requests submitted via DataRefresher OS/2.

Writing Accounting exit routines using sample exit routines

Included with DataRefresher are sample Accounting exit routines in PL/I, COBOL, and Assembler. These sample exits are contained in the DVR110.DVRINTER library. You can copy or print the sample exits and modify them to suit the functions you need. These sample Accounting exit routines are called:

DVRXYACX PL/I Accounting exit routine
DVRXOACX COBOL Accounting exit routine
DVRXAACX Assembler Accounting exit routine

Note: Before an Accounting exit routine can be link-edited with DataRefresher and used, you must rename it as DVRSACX.

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

For an example of how an Accounting exit routine can be used, see "Accounting exit routine example" on page 46.

Compiling and link-editing your Accounting exit routines

When you link-edit your Accounting exit routine, you replace the DataRefresher-supplied sample control section (CSECT) with your Accounting exit routine of the same name—DVRSACX. After you complete the link-edit, all extract requests submitted and executed at your installation call the Accounting exit routine instead of the sample CSECT shipped with DataRefresher.

To establish the connection between DataRefresher and your Accounting exit routine, you can use either JCL procedures or the `Linkage Edit` options of ISPF/PDF. The following section describes how to use JCL procedures for compiling your Accounting exit routine and link-editing it to DataRefresher. For information on using ISPF/PDF, see the ISPF/PDF online help.

The order of the JCL control statements in the following samples is important. In the examples, the modules are link-edited into the DVR110.DVRLOAD library.

Link-editing Accounting exit routines in PL/I

Figure 14 gives an example of the JCL you can use to compile your PL/I Accounting exit routine and link-edit the resulting text deck with the UIM and DEM.

```
/*
//*****************************************************************
//* THIS JCL DOES A PL/I COMPILE/LINKEDIT, REPLACING THE SHIPPED
//* DUMMY ACCOUNTING EXIT CSECT (DVRSACX) IN THE UIM AND DEM
//* WITH THE USER'S ACCOUNTING EXIT ROUTINE.
//*****************************************************************
/*
//PLI     EXEC PGM=IEL0AA,PARM='OBJECT,NODECK,XREF,ATTRIBUTES(FULL)'
//SYSLIB   DD  DISP=SHR,DSN=DVR110.DVRINTER     CONTAINS DVRXCAXP
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(800,(300,300))
//SYSPRINT DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSTERM  DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSUT4   DD  UNIT=SYSDA,SPACE=(400,(300,300),,,ROUND)
//SYSIN    DD  *

    PL/I SOURCE GOES HERE
/*
//*****************************************************************
//*   LINKEDIT IF THE COMPILE RETURN CODE IS 4 OR LESS
//*****************************************************************
//LKEDUIM EXEC PGM=IEWL,PARM='LIST,XREF,LET,RENT,RMODE=ANY,
//      AMODE=NN', 1  COND=(4,LT,PLI)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.PLIBASE 2
//         DD  DISP=SHR,DSN=DVR110.DVRLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DVRLOAD 3  TARGET LIB FOR LOAD MOD
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLINXX DD  DSN=&&LOADSET,DISP=(OLD,PASS)
//SYSLIN   DD  *
 REPLACE DVRSACX 4
 INCLUDE SYSLIB(DVRU0000)
 INCLUDE SYSLINXX
 ENTRY   DVRXSUSP 5
 NAME    DVRU0000(R) 6
/*
//*****************************************************************
//*   LINKEDIT IF THE COMPILE RETURN CODE IS 4 OR LESS
//*****************************************************************
//LKEDDEM EXEC PGM=IEWL,PARM='LIST,XREF,LET,RENT,RMODE=ANY,
//      AMODE=NN', 1  COND=(4,LT,PLI)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.PLIBASE
//         DD  DISP=SHR,DSN=DVR110.DVRLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DVRLOAD   TARGET LIB FOR LOAD MOD
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLINXX DD  DSN=&&LOADSET,DISP=(OLD,PASS)
//SYSLIN   DD  *
 REPLACE DVRSACX
 INCLUDE SYSLIB(DVRX0000)
 INCLUDE SYSLINXX
 ENTRY   DVRXSSSU 5
 NAME    DVRX0000(R) 6
/*
```

Figure 14. Sample link-edit of a PL/I Accounting exit routine

1 Specify the same AMODE and RMODE as the original module. For information on AMODE and RMODE restrictions, see "AMODE and RMODE dependencies" on page 23.

2 The PL/I program library is SYS1.PLIBASE. This library name should be supplied along with the load library name on the JOBLIB or STEPLIB DD statement in the JCL used to run the UIM and DEM.

3 The modules are link-edited into the DVR110.DVRLOAD library.

4 The SYSLIN data set contains the REPLACE statement. The REPLACE statement removes the Accounting exit routine from the particular DataRefresher program, before the INCLUDE SYSLINXX statement reinserts the exit routine. This ensures that all the required PL/I CSECTs remain intact.

5 DVRXSUSP is the main entry point of the UIM. DVRXSSSU is the main entry point of the DEM.

6 DVRU0000 is the name of the UIM load module containing the exit routine. DVRX0000 is the name of the DEM load module containing the exit routine.

Link-editing Accounting exit routines in COBOL

Figure 15 shows the JCL you can use to compile and link-edit your COBOL Accounting exit routine with the UIM and DEM.

```
//*-----------------------------------------------------------------*/
//* THESE JOB STEPS COMPILE AND LINK-EDIT A COBOL ACCOUNTING EXIT   */
//* ROUTINE TWICE- FOR UIM AND FOR DEM                              */
//*
//*- FIRST, FOR UIM ------------------------------------------------*/
//UIMSTEP       EXEC  COBUCL 1 ,PARM.COB='LIB'     COMPILE AND LINK
//COB.SYSLIB    DD  DISP=SHR,DSN=DVR110.DVRINTER
//COB.SYSIN     DD  *                  THE COBOL SOURCE CODE FOLLOWS:

  COBOL SOURCE GOES HERE

/*
//*- LINK EDIT ADDS ACCOUNTING EXIT TO UIM LOAD MODULE -------------*/
//LKED.SYSLMOD DD DISP=SHR,DSN=DVR110.DVRLOAD 2
//LKED.SYSLIB  DD DISP=SHR,DSN=SYS1.COB2LIB 3
//             DD DISP=SHR,DSN=DVR110.DVRLOAD
//LKED.SYSIN  DATA                   CONCAT. BEHIND SYSLIN BY PROC
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 REPLACE DVRSACX
 INCLUDE SYSLIB(DVRU0000)
 ENTRY   DVRXSUSP
 NAME    DVRU0000(R)
/*
//*
//*- SECOND, FOR DEM-----------------------------------------------*/
//DEMSTEP       EXEC  COBUCL           THIS IS A COMPILE AND LINK
//COB.SYSIN     DD  *                  THE COBOL SOURCE CODE FOLLOWS:

  COBOL SOURCE GOES HERE

/*
//*- LINK EDIT ADDS ACCOUNTING EXIT TO DEM LOAD MODULE -------------*/
//LKED.SYSLMOD DD DISP=SHR,DSN=DVR110.DVRLOAD
//LKED.SYSLIB  DD DISP=SHR,DSN=SYS1.COB2LIB
//             DD DISP=SHR,DSN=DVR110.DVRLOAD
//LKED.SYSIN  DATA                   CONCAT. BEHIND SYSLIN BY PROC
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 REPLACE DVRSACX
 INCLUDE SYSLIB(DVRX0000)
 ENTRY   DVRXSSSU
 NAME    DVRX0000(R)
/*
```

Figure 15. Sample link-edit of a COBOL Accounting exit routine

1 COBUCL is a procedure to compile and link-edit COBOL programs.

2 The modules are link-edited into the DVR110.DVRLOAD library.

3 The COBOL program library used is SYS1.COB2LIB. This library name should be supplied along with the load library name on the JOBLIB or STEPLIB DD statement in the JCL used to run the UIM and DEM.

Link-editing Accounting exit routines in Assembler

Figure 16 shows how to assemble and link-edit an Assembler Accounting exit routine with DataRefresher.

```
//*-----------------------------------------------------------------
//* THIS JCL PERFORMS AN ASSEMBLY LINKEDIT,
//* REPLACING THE DUMMY ACCOUNTING EXIT CSECT IN THE UIM AND DEM
//*-----------------------------------------------------------------
//*
//*-- FIRST, THE UIM ---------------------------------------------
//ACCTEXIT      EXEC  ASMHCL           THIS IS AN ASSEMBLE AND LINK
//ASM.SYSLIB    DD DSN=DVR110.DVRINTER,DISP=SHR CONTAINS DVRXCAXA
//              COPY FILE
//ASM.SYSIN     DD  *                  THE SOURCE CODE FOLLOWS:

  PLACE ASSEMBLER SOURCE HERE

/*
//*- LINK EDIT UIM, REPLACING ACCTING EXIT ------------------------ */
//LKED.SYSLMOD  DD  DSN=DVR110.DVRLOAD 1 ,DISP=OLD
//LKED.SYSLIB   DD DISP=SHR,DSN=DVR110.DVRLOAD
//SYSIN         DD  DATA     CONCATENATED BEHIND SYSLIN BY PROCEDURE
 REPLACE DVRSACX
 INCLUDE SYSLIB(DVRU0000)
 ENTRY   DVRXSUSP 2
 NAME    DVRU0000(R) 3
/*
//*
//*-- SECOND, THE DEM --------------------------------------------
//ACCTEXIT      EXEC  ASMHCL           THIS IS AN ASSEMBLE AND LINK
//ASM.SYSLIB    DD DSN=DVR110.DVRINTER,DISP=SHR CONTAINS DVRXCAXA
//              COPY FILE
//ASM.SYSIN     DD  *                  THE SOURCE CODE FOLLOWS:

  PLACE ASSEMBLER SOURCE HERE

/*
//*- LINK EDIT DEM, REPLACING ACCTING EXIT ------------------------ */
//LKED.SYSLMOD  DD  DSN=DVR110.DVRLOAD 1 ,DISP=OLD
//LKED.SYSLIB   DD DISP=SHR,DSN=DVR110.DVRLOAD
//SYSIN         DD  DATA     CONCATENATED BEHIND SYSLIN BY PROCEDURE
 REPLACE DVRSACX
 INCLUDE SYSLIB(DVRX0000)
 ENTRY   DVRXSSSU 4
 NAME    DVRX0000(R) 5
/*
```

Figure 16. Sample link-edit of a Assembler Accounting exit routine

1 The load library, DVR110.DVRLOAD, is the name of the library containing all the Accounting exit routines at your site. Write this name on the JOBLIB or STEPLIB DD statement in the JCL that you use to run the UIM and DEM.

2 DVRXSUSP is the main entry point of the UIM.

3 DVRU0000 is the name of the UIM load module containing the exit routine.

4 DVRXSSSU is the main entry point of the DEM.

5 DVRX0000 is the name of the DEM load module containing the exit routine.

Link-editing Accounting exit routines in LE/370 supported languages

The following examples show sample JCL to compile and link-edit Accounting exit routines in LE/370 supported PL/I and COBOL.

Link-editing Accounting exit routines in PL/I (LE/370)

Figure 17 shows how to compile and link-edit an Accounting exit routine in LE/370 supported PL/I.

```
//*******************************************************************
//* THIS JCL IS AN EXAMPLE OF A PLI (LE370) COMPILE AND LINK EDIT   *
//* OF AN ACCOUNTING EXIT TO REPLACE THE DUMMY DVRSACX SHIPPED WITH *
//* THE UIM AND DEM.                                                *
//*******************************************************************
//PLI370   EXEC  PGM=IEL1AA,REGION=512K,
//         PARM=(OBJECT,NODECK,SYSTEM(MVS))
//STEPLIB  DD DSN=HLQ370.ADPLI370.SIELCOMP,DISP=SHR 1
//SYSLIB   DD DSN=DVR110.DVRINTER,DISP=SHR 2
//SYSIN    DD DSN=yourown.DVR.PLI(DVRSACX),DISP=SHR
//SYSLIN   DD DSN=&&OBJCODE,DISP=(,PASS),
//            UNIT=SYSDA,SPACE=(CYL,(1,1)),
//            DCB=BLKSIZE=80
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DUMMY
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(2,2)),DCB=BLKSIZE=1024
//*******************************************************************
//* LINK IN YOUR ACCOUNTING EXIT REPLACING SHIPPED VERSION IN UIM   *
//*******************************************************************
//LINKUIM  EXEC PGM=IEWL,COND=(4,LT,PLI370),
//         PARM='LIST,LET,XREF,MAP,RENT,AMODE=NN,RMODE=ANY' 3
//SYSLIB   DD DSN=DVR110.DVRLOAD,DISP=SHR 4
//         DD DSN=HLQ370.ADLE370.SCEELKED,DISP=SHR 5
//SYSLMOD  DD DSN=DVR110.DVRLOAD,DISP=SHR 4
//SYSUT1   DD DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//            DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//SYSOBJ   DD DSN=&&OBJCODE,DISP=(OLD,PASS)
//SYSLIN   DD *
  REPLACE DVRSACX
  INCLUDE SYSLIB(DVRU0000)
  INCLUDE SYSOBJ
  ENTRY   DVRXSUSP
  NAME    DVRU0000(R)
/*
//*******************************************************************
//* LINK IN YOUR ACCOUNTING EXIT REPLACING SHIPPED VERSION IN DEM   *
//*******************************************************************
//LINKDEM  EXEC PGM=IEWL,COND=(4,LT,PLI370),
//         PARM='LIST,LET,XREF,MAP,RENT,AMODE=NN,RMODE=ANY' 3
//SYSLIB   DD DSN=DVR110.DVRLOAD,DISP=SHR 4
//         DD DSN=HLQ370.ADLE370.SCEELKED,DISP=SHR 5
//SYSLMOD  DD DSN=DVR110.DVRLOAD,DISP=SHR 4
//SYSUT1   DD DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//            DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//SYSOBJ   DD DSN=&&OBJCODE,DISP=(OLD,PASS)
//SYSLIN   DD *
  REPLACE DVRSACX
  INCLUDE SYSLIB(DVRX0000)
  INCLUDE SYSOBJ
  ENTRY   DVRXSSSU
  NAME    DVRX0000(R)
/*
```

Figure 17. Sample link-edit of a PL/1 (LE/370) Accounting exit routine

1 HLQ370.ADPLI370.SIELCOMP contains the compiler.

2 DVR110.DVRINTER contains the control blocks.

3 Specify the same AMODE and RMODE as the original module. See "AMODE and RMODE dependencies" on page 23 for further information.

4 The modules are link-edited into the DVR110.DVRLOAD library.

5 HLQ370.ADLE370.SCEELKED contains the LE/370 link library.

Link-editing Accounting exit routines in COBOL (LE/370)

Figure 18 shows how to compile and link-edit an Accounting exit routine in LE/370 supported COBOL.

```
//*************************************************************************
//* THIS JCL IS AN EXAMPLE OF A COBOL (LE370) COMPILE AND LINK EDIT OF AN    *
//* ACCOUNTING EXIT TO REPLACE THE DUMMY DVRSACX SHIPPED WITH THE UIM AND DEM*
//*************************************************************************
//COBOL370 EXEC  PGM=IGYCRCTL,REGION=640K,
//         PARM=(QUOTE,NOTERM,LIB,MAP,XREF,RENT,NOLIST)
//STEPLIB  DD DSN=HLQ370.ADCOB370.SIGYCOMP,DISP=SHR
//SYSIN    DD DSN=yourown.DVR.COBOL(DVRSACX),DISP=SHR
//SYSLIB   DD DSN=DVR110.DVRINTER,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DUMMY
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT5   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT6   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSUT7   DD UNIT=SYSDA,SPACE=(CYL,(2,2))
//SYSLIN   DD DSN=&&OBJCODE,DISP=(,PASS),
//            UNIT=SYSDA,SPACE=(CYL,(1,1)),
//            DCB=BLKSIZE=80
//*************************************************************************
//* LINK IN YOUR ACCOUNTING EXIT REPLACING SHIPPED VERSION IN UIM          *
//*************************************************************************
//LINKUIM  EXEC PGM=IEWL,COND=(4,LT,COBOL370),
//         PARM='LIST,LET,XREF,MAP,RENT,AMODE=NN,RMODE=ANY' 1
//SYSLIB   DD DSN=DVR110.DVRLOAD,DISP=SHR
//         DD DSN=HLQ370.ADLE370.SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=DVR110.DVRLOAD,DISP=SHR
//SYSUT1   DD DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//            DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//SYSOBJ   DD DSN=&&OBJCODE,DISP=(OLD,PASS)
//SYSLIN   DD *
  REPLACE DVRSACX
  INCLUDE SYSLIB(DVRU0000)
  INCLUDE SYSOBJ
  ENTRY   DVRXSUSP
  NAME    DVRU0000(R)
//*************************************************************************
//* LINK IN YOUR ACCOUNTING EXIT REPLACING SHIPPED VERSION IN DEM           *
//*************************************************************************
//LINKDEM  EXEC PGM=IEWL,COND=(4,LT,COBOL370),
//         PARM='LIST,LET,XREF,MAP,RENT,AMODE=NN,RMODE=ANY' 1
//SYSLIB   DD DSN=DVR110.DVRLOAD,DISP=SHR
//         DD DSN=HLQ370.ADLE370.SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=DVR110.DVRLOAD,DISP=SHR
//SYSUT1   DD DSN=&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//            DCB=BUFNO=1
```

Figure 18 (Part 1 of 2). Sample link-edit of a COBOL (LE/370) Accounting exit routine

```
//SYSPRINT DD SYSOUT=*
//SYSOBJ   DD DSN=&&OBJCODE,DISP=(OLD,PASS)
//SYSLIN   DD *
  REPLACE DVRSACX
  INCLUDE SYSLIB(DVRX0000)
  INCLUDE SYSOBJ
  ENTRY   DVRXSSSU
  NAME    DVRX0000(R)
/*
```

Figure 18 (Part 2 of 2). Sample link-edit of a COBOL (LE/370) Accounting exit routine

1 Specify the same AMODE and RMODE as the original module. See "AMODE and RMODE dependencies" on page 23 for further information.

Using Accounting exit routines

This section tells you how to use Accounting exit routines to perform various functions.

Keeping track of DataRefresher use

To keep track of which accounts have been using DataRefresher, you can use the ACCOUNT parameter on the SUBMIT command of your extract requests. The account information can be any term or name that is 32 or fewer characters, and it must have single quotation marks around it. If your account information contains an apostrophe, you must write it as two apostrophes so that DataRefresher correctly interprets the value. Some examples of ACCOUNT values are:

```
ACCOUNT='Account 41835'
ACCOUNT='091583 423640 D85/C50'
ACCOUNT='Sales 896''698'
```

The UIM passes the extract request, including the ACCOUNT value, to the Accounting exit routine for validation. This information is passed to the exit routine in the ACXACCT field of the control block.

Note: The ACCOUNT value is optional on a SUBMIT command.

Screening DataRefresher users

You can approve or reject users of DataRefresher by requiring that the ACCOUNT information keyword is always coded on an extract request. This value can then be validated during the UIM's VALIDATE call and during the DEM's BEGIN call.

During the VALIDATE call, the exit routine can check the ACXACCT field of the control block for a value. If that field is blank (that is, if an account value has not been coded on the extract request), or if it contains any deviation from what is acceptable, the exit routine sends a return code to DataRefresher to prevent this extract request from being stored in the EXTLIB.

You can also write your exit routine to tell the DEM to run only those extracts with specific accounting information. During the BEGIN call, the DEM passes information about each extract request in the batch to the exit routine in the control block. Based on this information, the exit routine sends a return code to indicate to the DEM which (if any) of the extracts in the batch are to be run.

Changing output limits and priorities on extract requests

The output limits and priority values specified on your extract requests can be changed during a VALIDATE call from the UIM. To change the output limit, change the DCXROWLN field of the Accounting exit routine control block to the maximum number of output rows you want to generate when the DEM runs this extract request. To change the priority of the extract request, change the ACXPRI field in the control block to a number between 0 and 255.

For more information on specifying output limits and priorities for your extract requests, see *DataRefresher Command Reference*.

Accounting exit routine return codes

The Accounting exit routine communicates with DataRefresher via return codes placed in the ACXRETC field of the control block. The meanings of the return codes vary with the types of calls made to the exit routine. Table 14 explains the meaning of each return code for each type of call to the Accounting exit routine.

Table 14. Accounting exit routine return codes

Type of Call	RC = 0	RC = 4	RC = 8	RC = 12
VALIDATE	Normal completion	Extract request not included in EXTLIB	Extract request not included in EXTLIB	UIM prevented from running
OPEN	Normal completion	Normal completion	Normal completion	DEM prevented from running
BEGIN	Normal completion	Batch will be executed but not all extract requests included	Batch will not be executed	DEM prevented from running
END	Normal completion	Normal completion	Normal completion	DEM prevented from running
CLOSE	Normal completion	Normal completion	Normal completion	DEM prevented from running

If the exit routine returns a code that is not listed above, DataRefresher terminates processing.

Handling unprocessed extract requests and batches

When DataRefresher makes a BEGIN call to your Accounting exit routine, the exit routine may send a return code to prevent one or more extract requests from being processed by DataRefresher. A return code of 4 prevents some of the requests in the batch from being processed while a return code of 8 prevents all of the requests from being processed. You can specify what DataRefresher is to do with unprocessed extract requests by assigning a value to the ACXRDISP field in the control block, as shown in Table 15:

Table 15. Handling unprocessed extract requests

Constant Value	Receiving Field	Extract Request Disposition
ACXRDREQ	ACXRDISP	Requeue extract request in EXTLIB This is the default.
ACXRDDEL	ACXRDISP	Delete extract request from EXTLIB

Accounting exit routine example

A sample COBOL Accounting exit routine is contained in member DVRXOACX of the DVR110.DVRINTER library. This exit routine is designed to:

- Approve or reject a given UIM request
- Record the ID of a given extract request, along with the starting time of the extract and the number of data rows extracted

The extract request used for this example is shown in Figure 19. Notice the value on the ACCOUNT keyword is `'APPROVE THE REQUEST'`:

```
SUBMIT EXTID=EXT02, USERID=DVRSQL2, CD=JCS,JCS=EXT01JCS,
       DBS=SQLDS, ACCOUNT='APPROVE THE REQUEST'
EXTRACT    WITH CREATE
   INTO EXT02TABLE (COLUMN1, COLUMN2)
   SELECT FE1PSCNT, FE1PSLOC
   FROM BE1VIEW1;
```

Figure 19. Extract request

The extract request is approved. The messages shown in Figure 20 are contained in the UIM output:

```
DVRS2428  ACCOUNTING EXIT "VA" CALL HAS PRODUCED THE FOLLOWING TEXT
          FOR DXTPRINT (SYSPRINT):  VALIDATE ROUTINE ENTERED.
DVRU1201  THE EXTRACT REQUEST WITH EXTID=EXT02 AND USERID=DVRSQL2
          IS ADDED TO EXTLIB SUCCESSFULLY.
DVRU0101  UIM PROCESSING IS COMPLETE. RETURN CODE 0.
```

Figure 20. Diagnostic messages if extract request approved

The UIM output also shows the ID of the extract request, the starting time of the extract, and the number of rows extracted.

If the value specified on the ACCOUNT keyword is `'DISAPPROVE THE REQUEST'`, the extract request is disapproved. The UIM returns a code of 4 and the DEM does not run. The messages shown in Figure 21 are contained in the UIM output:

```
DVRS2424  RETURN CODE 4 FROM "VA" CALL TO ACCOUNTING EXIT ENTRY
          POINT DVRSACX.
DVRS2428  ACCOUNTING EXIT "VA" CALL HAS PRODUCED THE FOLLOWING
          TEXT FOR DXTPRINT (SYSPRINT): VALIDATE ROUTINE ENTERED.
DVRS2406  THE EXTRACT REQUEST WITH EXTID=EXT02 AND USERID=DVRSQL2
          HAS BEEN STOPPED BY THE ACCOUNTING EXIT ROUTINE, AND
          WILL NOT BE PLACED INTO EXTLIB.
```

Figure 21. Diagnostic messages if extract request not approved

Chapter 4. Writing Data exit routines

Data exit routines manipulate source data that has been read, but not yet examined by DataRefresher, to make it match a DataRefresher data description. Data exit routines work at a segment or record level. You can use Data exit routines to clean up or change segments or records of your source data before DataRefresher translates the data into the target format.

Task overview

Writing and using Data exit routines with DataRefresher consists of:

1. Determining which segments or records in your source data you want your Data exit routine to convert. Information on the segments or records that Data exit routines can convert is contained in "Data exit routines" on page 2.
2. Writing your Data exit routine. For information on:
   - How your Data exit routine interacts with the control block, see "Using the Data exit routine control block" on page 49.
   - Using the sample Data exit routines provided, see "Writing Data exit routines using sample Data exit routines" on page 49.
   - Using IMS information or data services from within your Data exit routine, see "Using IMS facilities from your Data exit routine" on page 52.
3. *If your Data exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - Information on using non-Assembler exit routines is contained in "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. Compiling and link-editing your Data exit routine. See "Compiling and link-editing your exit routines" on page 22 for further information.
5. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on how to:
   - Register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Specify which exit routine to use when creating a source folder, see the *DataRefresher OS/2 User's Guide.*
6. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresher using CREATE DXTPSB or CREATE DXTFILE commands.

   For information on:
   - Specifying a Data exit routine on your CREATE DXTFILE or CREATE DXTPSB command, see "Identifying your Data exit routine to DataRefresher" on page 50.
   - The syntax of the CREATE DXTFILE and CREATE DXTPSB commands, see the *DataRefresher Command Reference.*

Calling Data exit routines

You identify your Data exit routine to DataRefresher on your CREATE DXTFILE or CREATE DXTPSB commands.

Data exit routines communicate with DataRefresher through two buffers and the control block. Before calling your exit routine, DataRefresher copies the source data to be altered into the input buffer. When called, the Data exit routine takes the data from the input buffer and alters it to conform to the data description. When the exit routine is finished, the data is put into the output buffer. The DEM can then map the data with its FDTLIB definition to perform record qualification and to extract field values into the output data set for the extract request.

When the DEM calls your Data exit routine, it passes as parameters the addresses of the:

- Control block
- Input buffer for the raw source data
- Output buffer for the processed source data
- Anchor area (64-bytes) which the Data exit routine can use as an anchor for exit-obtained storage for the life of the DEM, not just that of the extract request

As in a standard module linkage, your Data exit routine returns to the DEM using an address passed to the exit routine in register 14. This address is passed to the exit routine on entry.

The DAXCALL field in the control block tells the Data exit routine what type of call is being made to it:

NORMAL
A *normal* call is made by DataRefresher to the Data exit routine when it reads a new record or segment occurrence.

RETURN
If DataRefresher needs another occurrence of the record or segment, the exit routine uses a return code of 4 to request a *return* call to the exit routine. DataRefresher then gets the occurrence from the exit routine rather than from the file or database. Return code values are the same for both normal and return calls, including the use of return code 4 to request yet another return call.

END-OF-DATA
An *end-of-data* call can be made by DataRefresher to the Data exit routine. You can request an end-of-data call when specifying the exit in the DXTFILE or DXTPSB description. Normally such a call is requested for data summarization.

For more information about end-of-data calls, see "Using IMS facilities from your Data exit routine" on page 52.

Using the Data exit routine control block

The Data exit routine control block, together with the input and output data buffers, is the interface between DataRefresher and the Data exit routine that you write.

Because DataRefresher associates a copy of the control block with each segment or record for which you have defined a Data exit routine, the control block contains information about the:

- Type of call being made to the exit routine (normal, return or end-of-data).
- Name, type and length of the source data.
- Lengths of the input and output data buffers.
- Address of the DXTPRINT DCB.

  Although not a recommended practice, this field can be used if you code your Data exit routine to write messages to the DXTPRINT data set.

There is a different sample Data exit routine control block for each programming language. The control blocks for each programming language are contained in the DVR110.DVRINTER library, as shown in Table 16.

Table 16. Data exit routine return codes

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Data exit	DVRDAXP	DVRDAXC	DVRDAXA

Your Data exit routine can modify the following fields within the control block:

- Return code field (DAXRETC)
- A 128-byte scratchpad area (DAXSCRT1), in which the exit routine can save addresses and other information that it uses
- Two flag fields (DAXENTRD and DAXINCTL), which you can use for tracing and debugging
- Message field (DAXSMESG)
- A 64-byte area (the fourth parameter field), which the Data exit routine can use as an anchor for exit-obtained space

Writing Data exit routines using sample Data exit routines

Included with DataRefresher are sample Data exit routines in PL/I, COBOL, and Assembler. You can copy or print the sample exit routines and modify them to fit the functions you need. The samples are contained in the DVR110.DVRINTER library:

DVRXYDPM PL/I Data exit routine for conversion

DVRXYXDS PL/I Data exit routine for summarizing an entire VSAM file

DVRXODPM COBOL Data exit routine for conversion

DVRXOXDS COBOL Data exit routine for summarizing data in a subordinate IMS segment

DVRXADPM Assembler Data exit routine for conversion

DVRXAXDS Assembler Data exit routine for summarizing groups of records in a physical sequential file

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147. For some examples of how Data exit routines can be used, see "Data exit routine examples" on page 58.

Identifying your Data exit routine to DataRefresher

To process source data with a Data exit routine, you must indicate to the DEM which Data exit routine it should call for the particular source record or segment. You must also tell the DEM the exact or maximum length the data will be after it has been modified by the Data exit routine. You identify your exit routine to DataRefresher and include your data description in the FDTLIB by writing a CREATE DXTFILE or CREATE DXTPSB command with the DATAEXIT and XBYTES parameters.

Figure 22 shows how you can specify a Data exit routine on the CREATE DXTPSB command to convert data. It deals with an example of salary data, already in a format created to be unreadable to unauthorized users, that you converted to ISO format.

```
CREATE
DXTPSB  NAME=DATAXPSB,
        DESC='CREATE A DESCRIPTION SPECIFYING DATA EXIT'
DXTPCB  NAME=DATAXPCB, DBACCESS=HDAM
  SEGMENT   NAME=DEPT, PARENT=0, BYTES=42 1 , FREQ=10,
            DATAEXIT=DATAX, 2
            XBYTES=12, 3
            DESC='SEGMENT NAMED ORGANIZATION'
    FIELD 4 NAME=ASALARY,  START=1, TYPE=P,
            BYTES=4, SEQFLD=R,
            DESC='STARTING SALARY OF EMPLOYEE'
    FIELD   NAME=ASALARY2,  START=5, TYPE=E,
            BYTES=4,
            DESC='SECOND SALARY OF EMPLOYEE'
    FIELD   NAME=ASALARY3,  START=9, TYPE=E,
            BYTES=4,
            DESC='THIRD SALARY OF EMPLOYEE';
```

Figure 22. Sample CREATE DXTPSB command for a Data exit routine

1 The BYTES keyword, included in the SEGMENT statement of the CREATE DXTPSB command, tells DataRefresher the length of the source data *before* it is changed by a Data exit routine or extracted by the DEM. For DXTFILEs, you do not need to tell DataRefresher the length of the record before extraction. DataRefresher gets the length from the access method.

2 Include the DATAEXIT keyword (or the EXIT keyword—the two keywords are interchangeable) with either the:

- SEGMENT statement in the CREATE DXTPSB command for IMS source databases
- CREATE DXTFILE command for source files

The value you give to the DATAEXIT keyword must match the name of the load module containing the Data exit routine you have written to change the source data. Load module names are alphameric or special characters names with a maximum length of eight characters. For information on DataRefresher naming conventions, see the *DataRefresher Command Reference*.

3 The XBYTES keyword tells the DEM the length of the data in bytes *after* the Data exit routine has interpreted or modified it. Allowable values for the XBYTES keyword are integers from 1 to 32760. The value you give to the XBYTES keyword can be the same as, larger than, or smaller than the length of the source record or segment. For variable-length constructs, use the XBYTES keyword to specify the maximum length, in bytes, of the record after processing by the data exit.

4 The FIELD keywords in your CREATE DXTFILE and CREATE DXTPSB commands show the fields of the records or segments as they appear *after* processing by the Data exit routine, not *before*.

You can use the Dictionary Access Program (DAP) or the Structures Access Program (SAP) to help generate statements for describing files or PSBs to DataRefresher, but these programs do not generate DATAEXIT and XBYTES values. You will have to edit DAP/SAP output before you can use it as UIM input.

For detailed information about the syntax of the CREATE DXTPSB and CREATE DXTFILE commands, see *DataRefresher Command Reference*.

Data exit routine return codes

Your Data exit routine communicates with DataRefresher using return codes placed in the DAXRETC field of the control block. Table 17 contains an explanation of the meaning of the return codes.

Table 17. Data exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
4	Successful completion; DataRefresher should obtain next occurrence of source record or segment from the exit routine instead of from the file or database
8	DataRefresher should ignore data; data does not qualify for extraction
12	Error; terminate batch
16	Error; terminate DEM
Other	Error; terminate batch

Note: If the Data exit routine returns a code other than 0, 4, 8, 12, or 16, DataRefresher cannot interpret the code and discontinues processing the batch of extract requests.

Using IMS facilities from your Data exit routine

If your exit is running in IMS Batch or Batch Message Processing (BMP) environment, you may want to take advantage of the information or data services that IMS can offer from within your exit. For example, you might want to:

- Look at a program communication block (PCB) key feedback area to examine a concatenated key
- Use a PCB to make a DL/I call, perhaps to perform a lookup in another database
- Use an end-of-data call to summarize data

This section tells you how to use IMS facilities to perform these tasks. It assumes that you have a basic familiarity with IMS application programming concepts and techniques.

"Accessing PCBs" tells you how to access PCBs through the Data exit routine control block.

"Using an end-of-data call to summarize data" on page 53 describes end-of-data calls and provides MINSEGM, subordinate segment, WHERE clause, input data and sequence or key field considerations.

Accessing PCBs

Applications use a PCB to communicate with IMS. You can gain access to the PCBs through two variables, DAXPCBAD and DAXPCBLS, in the Data exit routine control block DVRXDAX.

DAXPCBAD This variable points to the PCB that was used to fetch the data segment that is being manipulated on this call to the Data exit routine. DAXPCBAD only contains a value if the exit routine is being called to transform a segment from a DL/I database. It does not contain a value if the exit routine is being called to transform a record or segment of a DXTFILE.

If you use the PCB addressed by DAXPCBAD to make a DL/I call, you must not alter IMS's position in the database. Successful DataRefresher processing depends on correct IMS positioning, and there are a number of ways the exit routine can incorrectly alter the position. In order to avoid changing the positioning of IMS when you issue IMS calls for the same database, add another PCB to the PSB, and use that PCB for your DL/I calls within your exit routine. The added PCB can be found via DAXPCBLS.

You do not need to go through the PCB if you only need to access the key feedback area in the PCB used to access the segment being transformed by the exit routine. You can use variables DAXKFBAD and DAXKFBLN (the address and length of the key feedback area, respectively). Again, these variables only contain a value if a DL/I segment is being transformed by the exit routine. They are not used in the interface to Data exit routines operating on DXTFILE data.

DAXPCBLS This variable points to the list of PCB addresses given to the DataRefresher DEM when it was called as an IMS Batch or BMP application. Using standard DL/I application techniques, the exit routine can access and use any PCB that appears on the list. You should not make any DL/I calls that may alter IMS's positioning in a database currently being processed by DataRefresher.

DAXPCBLS points to the PCB address list even if the data being transformed is DXTFILE data, as long as the DEM was called as an IMS application. This could be useful if, for example, your lookup data was stored in an IMS database, and you are extracting from a DXTFILE and need the lookup data. Depending on your data, you might alternatively be able to handle such a lookup as a normal DataRefresher join across the database and the file, without requiring an exit routine at all.

Note: If you write your Data exit routines in VS COBOL II, you must alter the DVRDAXC control block definition. Redefine the DAXPCBAD, DAXPCBLS and DAXKFBAD variables as pointers via USAGE IS POINTER phrases.

Using an end-of-data call to summarize data

You can request an end-of-data call to enable your Data exit routine to summarize data. End-of-data calls, if requested, are made in addition to the normal calls for segment or record occurrences.

What is end-of-data?

End-of-data can be defined two different ways:

- Root level segments

  For root level segments from a DL/I database, and for records from a DXTFILE, end-of-data means at the end of the file or data base, subject to any search strategy. For example, if your WHERE clause is `WHERE ROOTKEY < 'JONES'`, DataRefresher builds a qualified segment search argument and will only see root segments meeting that standard. If your exit routine requests an end-of-data call, the end-of-data call is made when DL/I gives a GB or GE status. In this situation, the exit routine never sees any roots with `ROOTKEY >= 'JONES'`.

  On the other hand, if the WHERE clause is `WHERE NONKEY < 'JONES'`, then all the root segments are passed to the Data exit routine. DataRefresher cannot be sure where the `NONKEY` field is until after the exit routine has processed the data, so it cannot pre-qualify the segment. DataRefresher makes the end-of-data call (if requested by the exit routine) only at the end of database (GE or GB status from DL/I).

- Lower level segments

  For lower level segments from a DL/I database, end-of-data is relative to each occurrence of the parent. If end-of-data calls are requested for a non-root segment, an end-of-data call is made for each occurrence of the immediate parent segment. In addition, regardless of the WHERE clause details, all occurrences of the segment for which the Data exit routine is identified are passed to the exit routine.

For both root and lower level segments, if there are zero occurrences of the segment or record, the exit routine is called with an end-of-data call. DataRefresher does not know the logic of the exit routine, and assumes the “no

occurrences" has meaning for the exit routine. For example, if the exit routine is being used to count or summarize subordinate segments, zero can be a possible and meaningful result.

How to specify an end-of-data call

The EODCALL keyword can be defined for the CREATE DXTFILE command, and for the SEGMENT statement of the CREATE DXTPSB command. The EODCALL keyword may be used in either of these commands only if the DATAEXIT (or EXIT) keyword is present. The value given to the EODCALL keyword is either `Y` or `N`, with `N` as the default.

Figure 23 shows the framework of the CREATE DXTFILE and CREATE DXTPSB commands including the EODCALL keyword:

```
CREATE DXTFILE NAME=..., DATAEXIT=..., XBYTES=..., EODCALL=value
       ...
       ;

CREATE DXTPSB ...
   DXTPCB ...
     SEGMENT ...
       FIELD ...
       ...
     SEGMENT NAME=..., DATAEXIT=..., XBYTES=..., EODCALL=value,
       ...
       ...
       ...
       ;
```

Figure 23. How to specify an end-of-data call

Whenever a DXTVIEW is defined over a hierarchic path containing a segment for which a Data exit routine with an end-of-data call is defined, that segment must be the bottom segment of the path, that is, it must be the one named on the SEGMENT keyword.

Return codes for an end-of-data call

When an end-of-data condition occurs, DataRefresher calls the exit routine using a special call type: `DAXCALL = DAXCEOD ('ED')`.

The valid return codes for this call are:

- DAXRCOK (value of 0) means that the exit routine is providing a segment or record occurrence for DataRefresher to use in extraction, and a segment or record qualification, even though there is no segment or record occurrence at the current position in the file or database.
- DAXRCNQ (value of 8) means that the exit routine is not providing a segment or record occurrence for DataRefresher to use in extraction, or a qualification from this end-of-data call.

MINSEGM considerations

There are some limitations on the utility of the MINSEGM keyword for a DXTVIEW that has a bottom segment with a Data exit routine and an end-of-data call. MINSEGM may still be specified as any segment of the hierarchic path, but the following MINSEGM values cause the same effect:

- Segment with Data exit routine with end-of-data call
- Parent segment of the segment just described

By using an exit routine with an end-of-data call, you are taking over end-of-data processing and are creating a type of "logical unit" of the two segments. DataRefresher treats the two segments as a unit for purposes of MINSEGM processing. DataRefresher then extracts from incomplete hierarchic paths.

Figure 24 shows a sample hierarchy:

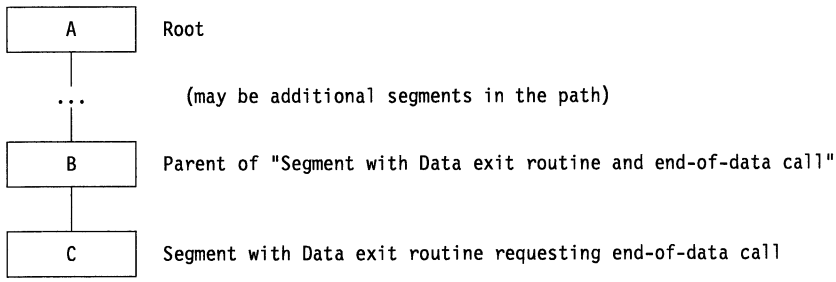

Figure 24. Sample hierarchy

In this example, DataRefresher is processing an extract request that selects data from segment C (output from the Data exit routine). Assume for the purposes of discussion that MINSEGM is equal to A in a three-level hierarchy. If there is no B for a particular A, then DataRefresher generates a row of output in the normal manner for missing segments beneath the MINSEGM segment. The exit routine is not called in this case.

With MINSEGM equal to B or C, if there is no B occurrence for a particular A occurrence, then nothing is extracted.

Equally with MINSEGM equal to C, if there is no B occurrence for a particular A occurrence, then nothing is extracted. If there is a B occurrence with no C occurrences, then the end-of-data call is made to the exit routine.

For any value of MINSEGM, once a B (that is, a parent) segment occurrence is successfully accessed, then DataRefresher will make an end-of-data call to the exit routine when it runs out of C occurrences, whether there are any C occurrences or not.

Subordinate segment considerations

When defining a DXTPCB or DXTFILE for a Data exit routine with an end-of-data call, you should consider using a different DXTPCB or DXTFILE from the one that contains the detailed database or file definition. Define a separate DXTPCB or DXTFILE:

- If you have a Data exit routine defined with an end-of-data call, any segments subordinate to the segment with the exit routine are not accessible through that DXTPCB or DXTFILE.
- The fields defined to DataRefresher for the segment must reflect the contents of the segment buffer after the Data exit routine has given the buffer back to DataRefresher to process (with a return code of 0). The buffer would contain summarized, concatenated, or otherwise aggregated data and not the detail from any particular segment occurrence. The detail level data would not be available for extraction.

WHERE clause considerations

When using Data exit routines, you should be aware of some special considerations if you want to use the type of conditional criteria that can be specified in a WHERE clause.

DataRefresher causes WHERE criteria to be processed to enhance performance, and to produce the correct result, when DataRefresher is extracting without a Data exit routine. When the source data has an index that can be used to limit the search of the file or database (for example, a VSAM KSDS, or an IMS database with an index), WHERE clause criteria involving the key or sequence field are passed to the access method (or DL/I). DataRefresher never sees the records or segments which do not meet the criteria, and therefore cannot pass such records or segments to a Data exit routine. Such criteria are evaluated by the access method *before* the exit routine has examined or transformed the data. WHERE clause criteria that DataRefresher itself evaluates are evaluated *after* the exit routine has manipulated the data, and only if the exit routine has returned a return code of 0.

For example, if `WHERE VSAMKEY > '300'`, is specified in an extract request from a VSAM KSDS, DataRefresher uses VSAM to fetch all records with `VSAMKEY >= '300'`, and passes them to the Data exit routine. DataRefresher then makes sure that any record with key value `'300'` is not included, by executing a `VSAMKEY > '300'` evaluation later, after processing by the exit routine.

When using Data exit routines together with WHERE clause criteria:

- All data occurrences that are given to DataRefresher by the access method (or DL/I) after application of search strategy are given to the Data exit routine for normal calls to the exit routine (DAXCALL = 'NO').

  Exit routines that provide summarization only may make mismatches difficult to locate as the detail extracted between what DataRefresher gives the exit routine, and what you expect DataRefresher to give, may not be obvious.
- End-of-data calls to the exit routine (DAXCALL = 'ED'), WHERE clause criteria not involving the key field are always evaluated by DataRefresher after the exit routine returns data to DataRefresher. This occurs even with summary data.

- All WHERE clause criteria involving fields from the summarized segment are evaluated after processing by the Data exit routine for summarizing data from a hierarchically subordinate segment in a DL/I (or internal segment) hierarchy.

When using a Data exit routine on a root level segment of a DL/I database or an unstructured DXTFILE, avoid using the KEY, SORT, or SEQUENCE fields in the WHERE clause. Instead, look at the input records in your Data exit routine and check them for inclusion in the extract processing. Return a code of DAXRETC=0 if they are to be included, and DAXRETC=8 if they are not to be included. If you must include such fields in your WHERE clause for performance reasons, make sure that you run tests to check that your exit routine is getting the records that it expects from DataRefresher.

When using a Data exit routine on a lower level segment of a hierarchy, be aware that all WHERE clause criteria against the segment are evaluated by DataRefresher after it receives data from the exit routine. When the exit routine returns a return code of 0, from either a normal call or from an end-of-data call, DataRefresher will then evaluate the WHERE criteria.

Input data considerations

The second parameter passed to the exit routine is normally the input data that the exit routine manipulates. This parameter has a different meaning for the end-of-data call, since there is no input data. The parameter passed is the most recent occurrence of the record or segment that DataRefresher has processed. The contents will depend on the type of call previously made to the exit routine, and the return code from that call, as shown in Table 18.

Table 18. Contents of second parameter passed to Data exit routine on end-of-data call

Type of previous call to exit routine	Return Code	Contents of second parameter
Normal call	0	Last record or segment processed by the exit routine (as transformed by the exit routine)
Normal call	8	Last record or segment processed by the exit routine (in raw input format)
End-of-data call	0, 4 or 8	Last data returned by the exit routine after the end-of-data call

Because the exit routine is given the last occurrence on the end-of-data call, if necessary you can extract from only the last occurrence using an end-of-data call. The exit routine only needs to identify and disqualify the case where no detail data has been passed between end-of-data calls, and then copy the input buffer to the output buffer on the end-of-data call. A simple switch in DAXSCRT1 can be used to do this.

Sequence or key field considerations

For Data exit routines that manipulate DL/I segments or records of a VSAM KSDS file, DataRefresher has a rule that demanding that the exit routine should not alter the contents of the sequence or key field. The rule also applies to normal calls that pass data back to DataRefresher. The rule does not apply For end-of-data calls. The data passed back from an end-of-data call can change or omit the sequence or key field present in the detail records or segments.

Data exit routine examples

This section contains examples to help explain how to use Data exit routines. Each of examples discussed are contained in the DVR110.DVRINTER library. The first example contains a sample data conversion, while the rest of the examples use Data exit routines to summarize a subordinate IMS segment, groups of records in a physical sequential file, and an entire VSAM file,

Conversion of IMS data

The sample COBOL Data exit routine contained in member member DVRXODPM in the DVR110.DVRINTER library is used to convert IMS data. It converts from database to DataRefresher format and from DataRefresher to the database format, and also performs a field edit rejecting fields containing invalid data.

Figure 25 contains an example of the DXTPSB definition of SAMPPSB1 data used in the sample COBOL Data exit routine, DVRXODPM. The data converted are the SALARY and COMMISSION fields in the staff segment of the SAMPPSB1 and SAMPPSB2 sample data.

```
CREATE DXTPSB NAME=SAMPPSB1,DESC='PSB OVER THE HSAM DBASE'
       DXTPCB NAME=SAMPPCB1, DBACCESS=HSAM,
                             DESC='ONE PCB OVER THIS HSAM DBASE'
 SEGMENT NAME=ORG,PARENT=0,BYTES=41,DESC='ORGANIZATIONAL DATA'
  FIELD NAME=DEPT,START=1,TYPE=H,BYTES=2, DESC='DEPT NUMBER'
  FIELD NAME=DEPTNAME,START=3,TYPE=C,BYTES=14,SEQFLD=R,DESC='PRODUCT NO'
  FIELD NAME=MANAGER,START=17,TYPE=H,BYTES=2,DESC='MNGR. NUMBER'
  FIELD NAME=DIVISION,START=19,TYPE=C,BYTES=10,DESC='DIVISION'
  FIELD NAME=LOCATION,START=29,TYPE=C,BYTES=13,DESC='LOCATION'
 SEGMENT NAME=PROJ,PARENT=ORG,BYTES=52,FREQ=4,
                   DESC='CURRENT BUSINESS PROJECTS'
  FIELD NAME=PRODNUM,START=1,TYPE=H,BYTES=2,DESC='PRODUCT NO'
  FIELD NAME=PROJNUM,START=3,TYPE=C,BYTES=4,SEQFLD=R,DESC='PROJECT NO'
  FIELD NAME=STARTD,START=7,TYPE=A,BYTES=10,DESC='START DATE'
  FIELD NAME=ENDD,START=17,TYPE=A,BYTES=10,DESC='END DATE'
  FIELD NAME=TIMESTMP,START=27,TYPE=S,BYTES=26,DESC='TIME STAMP'
/*                                                          */
/* STAFF SEGMENT MODIFIED WITH DATA EXIT ROUTINE            */
/*                                                          */
 SEGMENT NAME=STAFF,PARENT=ORG,BYTES=28,FREQ=10,
         DATAEXIT=DVRXODPM,XBYTES=34,DESC='EMPLOYEE STATISTICS'
  FIELD NAME=ID,START=1,TYPE=H,BYTES=2,DESC='EMP ID'
  FIELD NAME=NAME,START=3,TYPE=C,BYTES=9,SEQFLD=R,DESC='EMP NAME'
  FIELD NAME=JOB,START=14,TYPE=C,BYTES=5,DESC='JOB NAME'
  FIELD NAME=YEARS,START=19,TYPE=B,BYTES=1,DESC='YRS WORKED'
/*FIELD NAME=SALARY,START=20,TYPE=P,BYTES=4,SCALE=2,
                             DESC='YEARLY SALARY'          */
/*FIELD NAME=COMM,START=24,TYPE=P,BYTES=4,SCALE=2,
                           DESC='YEARLY COMMISSION'        */
  FIELD NAME=SALARY,START=20,TYPE=Z,BYTES=7,SCALE=2,DESC='YEARLY SALARY'
  FIELD NAME=COMM,START=27,TYPE=Z,BYTES=7,SCALE=2,DESC='YEARLY COMMISSION'
  FIELD NAME=NUMHIST,START=34,TYPE=B,BYTES=1,
                     DESC='NUMBER OF HIST SEGMENTS'
 SEGMENT NAME=HIST,PARENT=STAFF,BYTES=24,FREQ=1,
                       DESC='EMPLOYMENT HISTORY'
  FIELD NAME=DATE,START=1,TYPE=A,BYTES=10,DESC='CHANGE DATE'
  FIELD NAME=PASTJOB,START=11,TYPE=C,BYTES=5,DESC='JOB NAME'
  FIELD NAME=YRSWKED,START=16,TYPE=B,BYTES=1,DESC='YRS WORKED'
  FIELD NAME=SSAL,START=17,TYPE=P,BYTES=4,SCALE=2,
                           DESC='STARTING SALARY'
  FIELD NAME=ESAL,START=21,TYPE=P,BYTES=4,SCALE=2,
                           DESC='ENDING SALARY';
```

Figure 25. DXTPSB definition of SAMPPSB1 data

Figure 26 contains the DXTPSB definition of SAMPPSB2 data used in the sample COBOL Data exit routine, DVRXODPM.

```
CREATE DXTPSB NAME=SAMPPSB2,DESC='PSB OVER THE HISAM DBASE'
       DXTPCB NAME=SAMPPCB2, DBACCESS=HISAM,
                              DESC='ONE PCB OVER THIS HISAM DBASE'
 SEGMENT NAME=ORG,PARENT=0,BYTES=41,DESC='ORGANIZATIONAL DATE'
  FIELD NAME=DEPT,START=1,TYPE=H,BYTES=2,DESC='DEPT NUMBER'
  FIELD NAME=DEPTNAME,START=3,TYPE=C,BYTES=14,SEQFLD=R,
                               DESC='PRODUCT NO'
  FIELD NAME=MANAGER,START=17,TYPE=H,BYTES=2,DESC='MNGR. NUMBER'
  FIELD NAME=DIVISION,START=19,TYPE=C,BYTES=10,DESC='DIVISION'
  FIELD NAME=LOCATION,START=29,TYPE=C,BYTES=13,DESC='LOCATION'
 SEGMENT NAME=PROJ,PARENT=ORG,BYTES=52,FREQ=4,
                  DESC='CURRENT BUSINESS PROJECTS'
  FIELD NAME=PRODNUM,START=1,TYPE=H,BYTES=2,DESC='PRODUCT NO'
  FIELD NAME=PROJNUM,START=3,TYPE=C,BYTES=4,SEQFLD=R,DESC='PROJEC
  FIELD NAME=STARTD,START=7,TYPE=A,BYTES=10,DESC='START DATE'
  FIELD NAME=ENDD,START=17,TYPE=A,BYTES=10,DESC='END DATE'
  FIELD NAME=TIMESTMP,START=27,TYPE=S,BYTES=26,DESC='TIME STAMP'
  SEGMENT NAME=STAFF,PARENT=ORG,BYTES=76,FREQ=10,
                               DESC='EMPLOYEE STATISTICS'
  FIELD NAME=ID,START=1,TYPE=H,BYTES=2,DESC='EMP ID'
  FIELD NAME=NAME,START=3,TYPE=C,BYTES=9,SEQFLD=R,DESC='EMP NAME'
  FIELD NAME=JOB,START=14,TYPE=C,BYTES=5,DESC='JOB NAME'
  FIELD NAME=YEARS,START=19,TYPE=B,BYTES=1,DESC='YRS WORKED'
  FIELD NAME=SALARY,START=20,TYPE=P,BYTES=4,SCALE=2,
                              DESC='YEARLY SALARY'
  FIELD NAME=COMM,START=24,TYPE=P,BYTES=4,SCALE=2,
                            DESC='YEARLY COMMISSION'
  FIELD NAME=NUM_HIST,START=28,TYPE=B,BYTES=1,
                            DESC='# OF INTERNAL SEGMENTS'
 SEGMENT NAME=HIST,FORMAT=FI,PARENT=STAFF,
         BYTES=24,START=NUM_HIST+1,FREQ=1,OCCURS=NUM_HIST,
         DESC='INTERNAL HISTORY SEGMENT'
  FIELD NAME=DATE,START=1,TYPE=A,BYTES=10,DESC='CHANGE DATE'
  FIELD NAME=PASTJOB,START=11,TYPE=C,BYTES=5,DESC='JOB NAME'
  FIELD NAME=YRSWKED,START=16,TYPE=B,BYTES=1,DESC='YRS WORKED'
  FIELD NAME=SSAL,START=17,TYPE=P,BYTES=4,SCALE=2,
                            DESC='STARTING SALARY'
  FIELD NAME=ESAL,START=21,TYPE=P,BYTES=4,SCALE=2,
                            DESC='ENDING SALARY';
```

Figure 26. DXTPSB definition of SAMPPSB2 data

Figure 27 shows the description of the DXTVIEWs for both SAMPPSB1 and SAMPPSB2 in the sample COBOL Data exit routine, DVRXODPM.

```
CREATE DXTVIEW NAME=NEWSTAFF,DXTPSB=SAMPPSB1,DXTPCB=SAMPPCB1,
       FIELD=(ID, NAME, JOB, YEARS, SALARY, COMM, NUMHIST),
       SEGMENT=STAFF,
       DESC='HSAM, VIEW TO STAFF';

CREATE DXTVIEW NAME=IMSSTAFF,FIELD=*,DXTPSB=SAMPPSB1,DXTPCB=SAMPPCB1,
       SEGMENT=HIST,MINSEGM=ORG,
       DESC='HSAM, VIEW TO STAFF';

CREATE DXTVIEW NAME=IMSHIST,FIELD=*,DXTPSB=SAMPPSB2,DXTPCB=SAMPPCB2,
       SEGMENT=HIST,MINSEGM=ORG,
       DESC='HISAM, VIEW TO HIST';

CREATE DXTVIEW NAME=IMSPROJECT,FIELD=*,DXTPSB=SAMPPSB1,
               DXTPCB=SAMPPCB1,SEGMENT=PROJ,MINSEGM=ORG,
               DESC='HSAM, VIEW TO PROJ';
```

Figure 27. DXTVIEW definitions

Figure 28 shows the extract request for the COBOL Data exit routine, DVRXODPM.

```
SUBMIT USERID=TESTER, EXTID=EXT01, CD=JCS, JCS=EXT01JCS
EXTRACT INTO MYTABLE (ID, NAME, JOB, YEAR, SALARY, COMM, NUMHIST)
        SELECT *
        FROM NEWSTAFF;
```

Figure 28. Extract request (EXT01)

Figure 29 shows the output from the extract request:

```
00090 KOONITZ   SALES  00006  18001.75  01386.70  00000
00100 PLOTZ     MGR    00007  18352.80  00000.00  00001
00200 SCOUTTEN  CLERK  00000  11508.60  00084.20  00000
00130 YAMAGUCHI CLERK  00006  10505.90  00075.60  00000
00240 DANIELS   MGR    00005  19260.25  00000.00  00000
00260 JONES     MGR    00012  21234.00  00000.00  00002
00210 LU        MGR    00010  20010.00  00000.00  00002
00160 MOLINARE  MGR    00007  22959.20  00000.00  00000
00080 JAMES     CLERK  00000  13504.60  00128.20  00000
00020 PERNAL    SALES  00008  18171.25  00612.45  00001
00010 SANDERS   MGR    00007  18357.50  00000.00  00002
00190 SNEIDER   CLERK  00008  14252.75  00126.50  00000
00300 DAVIS     SALES  00005  15454.50  00806.10  00001
00340 EDWARDS   SALES  00007  17844.00  01285.00  00000
00350 GAFNEY    CLERK  00005  13030.50  00188.00  00000
00290 QUILL     MGR    00010  19818.00  00000.00  00001
00050 HANES     MGR    00010  20659.80  00000.00  00001
00170 KERMISCH  CLERK  00004  12258.50  00110.10  00000
00110 NGAN      CLERK  00005  12508.20  00206.60  00000
00070 ROTHMAN   SALES  00007  16502.83  01152.00  00001
00330 BURKE     CLERK  00001  10988.00  00055.50  00000
00320 GONZALES  SALES  00004  16858.20  00844.00  00000
00310 GRAHAM    SALES  00013  21000.00  00200.30  00000
00270 LEA       MGR    00009  18555.50  00000.00  00000
00280 WILSON    SALES  00009  18674.50  00811.50  00001
00140 FRAYE     MGR    00006  21150.00  00000.00  00002
00230 LUNDQUIST CLERK  00003  13369.80  00189.65  00000
00220 SMITH     SALES  00007  17654.50  00992.80  00001
00250 WHEELER   CLERK  00006  14460.00  00513.30  00000
00214 WILLIAMS  SALES  00006  19456.50  00637.65  00001
00180 ABRAHAMS  CLERK  00003  12009.75  00236.50  00000
00030 MARENGHI  MGR    00005  17506.75  00000.00  00001
00120 NAUGHTON  CLERK  00000  12954.75  00180.00  00000
00040 O'BRIEN   SALES  00006  18006.00  00846.55  00000
00060 QUIGLEY   SALES  00000  16808.30  00650.25  00000
```

Figure 29. Output from the extract request (EXT01)

Summarization of IMS data in a subordinate segment

The example discussed in this section is a sample Data exit routine in COBOL named STAFSUMM, which is contained in member DVRXOXDS in the DVR110.DVRINTER library. This example shows a straightforward summarization of the data in a subordinate IMS segment. The extract request produces one row of output for each parent occurrence.

Figure 30 is a hierarchy diagram of the DXT1DBD sample HSAM database. The STAFF segment has salary and commission information that the customer wishes to sum for each ORG segment, producing only one extract output record per ORG segment.

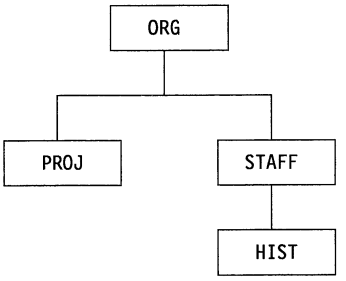

Figure 30. DXT1DBD sample HSAM database

Figure 31 shows the PCB definition that is embedded into the DXTPSB data description corresponding to the IMS PSB that will be used to access the data.

```
DXTPCB NAME=SUMMPCB1, DBACCESS=HSAM, DBNAME=DXT1DBD

SEGMENT NAME=ORG,PARENT=0,BYTES=41
 FIELD NAME=DEPTNUMB,START=1,TYPE=H,BYTES=2
 FIELD NAME=DEPTNAME,START=3,TYPE=C,BYTES=14,SEQFLD=R
 FIELD NAME=MANAGER,START=17,TYPE=H,BYTES=2
 FIELD NAME=DIVISION,START=19,TYPE=C,BYTES=10
 FIELD NAME=LOCATION,START=29,TYPE=C,BYTES=13

SEGMENT NAME=PROJ,PARENT=ORG,BYTES=52,FREQ=4
 FIELD NAME=PRODNUM,START=1,TYPE=H,BYTES=2
 FIELD NAME=PROJNO,START=3,TYPE=C,BYTES=4,SEQFLD=R
 FIELD NAME=STARTD,START=7,TYPE=A,BYTES=10
 FIELD NAME=ENDD,START=17,TYPE=A,BYTES=10
 FIELD NAME=TIMESTMP,START=27,TYPE=S,BYTES=26

SEGMENT NAME=STAFF,PARENT=ORG,BYTES=28,FREQ=10,

/* DEFINE EXIT 'STAFSUMM', WITH AN EOD CALL, ON THE STAFF SEGMENT..*/
    DATAEXIT=STAFSUMM, XBYTES=12, EODCALL=Y

/* THESE NEXT FIELDS ARE IN THE IMS SEGMENT, AND ARE INPUT TO THE
    EXIT ROUTINE, BUT IT IS THE OUTPUT OF THE EXIT ROUTINE THAT
    MUST BE DESCRIBED TO DATAREFRESHER, SO THESE FIELDS
    ARE COMMENTED OUT!                                          */
/*  FIELD NAME=ID,START=1,TYPE=H,BYTES=2
    FIELD NAME=NAME,START=3,TYPE=C,BYTES=9,SEQFLD=R
    FIELD NAME=JOB,START=14,TYPE=C,BYTES=5
    FIELD NAME=YEARS,START=19,TYPE=B,BYTES=1
    FIELD NAME=SALARY,START=20,TYPE=P,BYTES=4,SCALE=2
    FIELD NAME=COMM,START=24,TYPE=P,BYTES=4,SCALE=2
    FIELD NAME=NUMHIST,START=28,TYPE=B,BYTES=1                  */
```

Figure 31 (Part 1 of 2). DXTPCB definition (SUMMPCB1)

```
/* THESE NEXT FIELDS ARE PRODUCED BY EXIT ON EOD CALL...
   AND THESE ARE WHAT MUST BE DESCRIBED TO DATREFRESHER            */
FIELD NAME=STAFFCOUNT,  START=01, TYPE=H
FIELD NAME=TOTALSALARY, START=03, TYPE=P, BYTES=5
FIELD NAME=TOTALCOMM,   START=08, TYPE=P, BYTES=5

SEGMENT NAME=HIST,PARENT=STAFF,BYTES=24,FREQ=1
 FIELD NAME=DATE,START=1,TYPE=A,BYTES=10
 FIELD NAME=PASTJOB,START=11,TYPE=C,BYTES=5
 FIELD NAME=YRSWKED,START=16,TYPE=B,BYTES=1
 FIELD NAME=SSAL,START=17,TYPE=P,BYTES=4,SCALE=2
 FIELD NAME=ESAL,START=21,TYPE=P,BYTES=4,SCALE=2
```

Figure 31 (Part 2 of 2). DXTPCB definition (SUMMPCB1)

Note: The segment data output from the exit routine is described to DataRefresher for the STAFF segment, not the segment data input.

The definition of the HIST segment has been left in the DXTPCB. This segment cannot be accessed through this DXTPCB because it is subordinate to a segment with a Data exit routine requesting EODCALL=Y.

Figure 32 shows the description of the DXTVIEW. In this example, MINSEGM defaults to STAFF.

```
CREATE DXTVIEW NAME=SUMMSTAFF, DXTPSB= ... , DXTPCB=SUMMPCB1,
       SEGMENT=STAFF, FIELDS=*,
       DESC='VIEW INCLUDES ORG AND STAFF SEGMENTS';
```

Figure 32. DXTVIEW definition (SUMMSTAFF)

The STAFSUMM exit routine uses the work space (DAXSCRT1) in the control block (DVRDAXC) as an accumulation area. For each occurrence of STAFF, it adds to the accumulators, and sends a return code of 8 to DataRefresher. This return code means that no data is to be extracted for this occurrence. When the exit routine receives the end-of data call, it passes the accumulated values back to DataRefresher with a return code of 0. DataRefresher is then prepared to evaluate any WHERE criteria and extract the data.

In the Linkage Section of the exit routine, the REDEFINES is used to put the accumulators in the DAXSCRT1 area of the control block.

In addition to summing the salary and commission from each STAFF segment, the exit counts the number of STAFF segments within each ORG segment.

The extract shown in Figure 33 request SELECTs:

- Two fields from the ORG segment
- Three accumulated fields:
  - Count of STAFF segments
  - Two total fields

```
SUBMIT   EXTID=SUMM1, JCS=JCS1, ...
EXTRACT ...
    SELECT DEPTNUMB, DEPTNAME,
           STAFFCOUNT, TOTALSALARY, TOTALCOMM
    FROM SUMMSTAFF
    ;
```

Figure 33. Extract request (SUMM1)

In this example, only the relevant information is shown for the SUBMIT command and EXTRACT statement. Figure 34 shows the data portion of the extract output.

```
00042 GREAT LAKES     00004  0005836905  0000154650
00010 HEAD OFFICE     00004  0008346345  0000000000
00020 MID ATLANTIC    00004  0006428610  0000086715
00084 MOUNTAIN        00004  0006614700  0000227910
00015 NEW ENGLAND     00004  0006192933  0000146870
00066 PACIFIC         00005  0008607620  0000191130
00051 PLAINS          00005  0008609080  0000233340
00038 SOUTH ATLANTIC  00005  0007728555  0000191330
```

Figure 34. Output from the extract request (SUMM1)

Summarization of groups of records from a physical sequential file

The example discussed in this section is a sample Data exit routine in Assembler named SUMMDEPT. It is contained as member DVRXAXDS in the DVR110.DVRINTER library. This example summarizes groups of records from a physical sequential file. In the file, each sale is represented by a record, and the records are in order by department. You want a summary record for each department.

You want your sales data, in order by DEPT, summarized to produce a single record per DEPT value. The following diagram is a conceptual picture of the data:

```
DPTSALES file record format:          Desired extract output:

  DEPT INV# QUANT  $AMT                 DEPT TOTQUANT #SALES TOT$AMT

  A    xyz     3  10.00  ─┐
  A    abc    13  30.00   │
  A    uvw     4  19.00   │──────>  A       29      5    88.95
  A    123     8  20.00   │
  A    rst     1   9.95  ─┤
  B    qrs     9  70.00   │
  B    543     8  94.00   │──────>  B       22      3   169.70
  B    def     5   5.70  ─┘
  C    hgf     3 119.95  ──────────>  C        3      1   119.95
```

Figure 35 shows the DXTFILE definition.

```
CREATE DXTFILE NAME=DPTSALES, ACCESS=PS, DATAEXIT=DEPTTOTL, XBYTES=16,
               EODCALL=Y
   /* FIELD DESCRIPTIONS ARE OF THE DATA *AFTER* IT IS SUMMARIZED
      AND RETURNED TO DATAREFRESHER BY THE EXIT AT A CHANGE IN DEPT
      VALUE       */
   FIELD NAME= DEPT, START= 1,  TYPE=C, BYTES=4, SEQFLD=R, UNIQUE=N
   FIELD NAME= DEPTQNT,  START=5, TYPE=F
   FIELD NAME= DEPTNBR,  START=9, TYPE=H
   FIELD NAME= DEPTAMT,  START=11,TYPE=P, BYTES=6, SCALE=2;
```

Figure 35. DXTFILE definition (DPTSALES)

The defined fields map the data after it is returned to DataRefresher by the exit routine. Because the returned data is the summarized data, the Inventory Number field is not represented. Also, the fields for the accumulated values are of different lengths and data types (zoned decimal) than the corresponding fields in the input records.

Figure 36 shows the DXTVIEW definition.

```
CREATE DXTVIEW NAME=VDEPT, DXTFILE=DPTSALES, FIELDS=*;
```

Figure 36. DXTVIEW definition (VDEPT)

This exit routine uses the Data exit routine anchor area as an accumulation area. This anchor area is the 4th parameter passed to the Data exit routine. When the Data exit routine detects a change in value of DEPT, it returns the accumulated values for the old DEPT back to DataRefresher. The exit routine then zeros the accumulators and begins getting the next DEPT values. Finally, the EOD call is used to pass the accumulated values for the last DEPT back to DataRefresher for processing.

Although the CSECT name for this exit routine is SUMMDEPT, the DATAEXIT keyword in the DXTFILE definition identifies the exit as DEPTTOTL. The DATAEXIT keyword identifies the exit routine by the name of its load module, and in this example, DEPTTOTL is how the exit is link-edited. (The link-edit job is not shown here.)

To maintain reentrancy, the Data exit routine uses the scratch space in the control block (DAXSCRT1) for two purposes:

- Work area PWORK
- Register save area (shown for illustrative purposes only, as this particular exit routine does not use any external services and therefore does not really need its own register save area)

Addressability to the constant values in the DVRDAXA control block is established via the VCONSTS VCON defined in the exit.

Figure 37 shows the extract request.

```
SUBMIT   EXTID=GMW1, JCS=JCS1, ...
EXTRACT ...
    SELECT *
    FROM VDEPT
    ;
```

Figure 37. Extract request (GMW1)

These SUBMIT and EXTRACT statements show only the options that are vital to the points being made here.

Figure 38 shows the input data file.

```
DEPT INVT#  QNTY $AMT
----+------+----+------
AAAA 1PA100 0001 019999
AAAA 1PA1D4 0003 008451
AAAA 0DOG01 0001 001495
AAAA 4BAGHY 0028 028028
AAAA 0BAHAL 0001 002360
AAAA 1PA100 0002 039998
BBBB OGAMLG 0800 080000
CCCC JWALKR 0003 007659
CCCC CHOLER 0011 000099
CCCC FWINGR 0001 003388
CCCC 00HULK 0008 006400
CCCC 4MULA1 0001 022295
CCCC 000000 0013 000013
CCCC STUBBY 0002 003999
CCCC GARDER 0020 001499
EEEE MRAWDR 0003 012345
EEEE UNGUNT 0001 008977
EEEE UBARS  0001 050000
EEEE 005415 0009 005432
JJJJ 393900 0006 004999
JJJJ 221ST  0001 091315
JJJJ BAYSID 0100 038877
JJJJ NYNYNY 0050 004000
JJJJ BAYSID 0050 019439
JJJJ FLUSHG 0001 005000
NNNN 9LIVES 0020 008188
RRRR 988858 0030 004300
RRRR ADFFAD 0031 012998
RRRR 988858 0033 001995
RRRR BGHHJK 0020 010000
RRRR GGHHKK 0018 007995
RRRR AS0088 0039 006066
RRRR SARWAK 0029 004000
RRRR BULOVO 0030 003330
TTTT TORNCE 0010 007011
TTTT TORNCE 0020 014022
UUUU GROVLR 0002 018000
```

Figure 38. Input data file

Figure 39 shows the output from the extract request.

```
AAAA  0000000036  00006  000001003.31
BBBB  0000000800  00001  000000800.00
CCCC  0000000059  00008  000000453.52
EEEE  0000000014  00004  000000767.54
JJJJ  0000000208  00006  000001636.30
NNNN  0000000020  00001  000000081.88
RRRR  0000000230  00008  000000506.84
TTTT  0000000030  00002  000000210.33
UUUU  0000000002  00001  000000180.00
```

Figure 39. Output from the extract request (GMW1)

Example 4. Summarization of a VSAM data set

The example discussed in this section is a sample Data exit routine in PL/I, which is contained as member DVRXYXDS in the DVR110.DVRINTER library. It describes a summarization of the entire sample VSAMDEPT file. Only one output record is produced at the end specifying how many PROJ type and ORG records there are in the file.

This VSAM key sequenced data set (KSDS) contains two record types:

- PROJ records where the suffix of the key is nonzero (the product number of these records)
- ORG records where the suffix is zero

You want one row of output containing the counts of each of these record types.

Figure 40 shows the DataRefresher file definitions used in this example. The first definition shows the data formats of the VSAMDEPT records. The second definition, called VSAMDSUM, shows the definition that specifies the exit routine, and whose fields represent the data output from the exit routine.

```
/* THIS DEFINITION OF VSAMDEPT SHOWS THE RECORD FORMATS, AND
   IS NOT THE ONE WHICH DEFINES THE DATA EXIT DOING THE SUMMARIZATION*/
CREATE DXTFILE NAME=VSAMDEPT, ACCESS=VK, FREQ=20,DESC='PROJECT INFORMATION'
SEGMENT NAME=PROJSEG,
 DESC='SEGMENT FOR THE PROJ RECORDS'
 FIELD NAME= DEPTKEY, START=1, TYPE=C, BYTES=5, SEQFLD=R,
            DESC='VSAM KEY IS DEPT/PRODNUM'
 FIELD NAME= DEPT,START= 1,  TYPE=Z, BYTES= 2, DESC='DEPT NUMBER'
 FIELD NAME= PRODNUM, START= 3,  TYPE=Z, BYTES= 3, DESC='PRODUCT NO'
 FIELD NAME= PROJNUM, START= 6,  TYPE=C, BYTES= 4, DESC='PROJECT NO'
 FIELD NAME= STARTD,  START=10,  TYPE=A, BYTES=10, DESC='START DATE'
 FIELD NAME= ENDD,    START=20,  TYPE=A, BYTES=10, DESC='END DATE'
 FIELD NAME= TIMESTMP,START=30,  TYPE=S, BYTES=26, DESC='TIME STAMP'
SEGMENT NAME=ORGSEG,
 DESC='SEGMENT FOR THE DEPT RECORDS'
 FIELD NAME= DEPTKEY, START=1, TYPE=C, BYTES=5, SEQFLD=R,
            DESC='VSAM KEY IS DEPT/SUFFIX'
 FIELD NAME= DEPT,START= 1, TYPE=Z,  BYTES= 2, DESC='DEPT NUMBER'
 FIELD NAME= SUFFIX,  START= 3, TYPE=Z, BYTES= 3, DESC='SUFFIX: 000'
 FIELD NAME= DEPTNAME,START= 6, TYPE=C, BYTES=14, DESC='NAME OF DEPT'
 FIELD NAME= MANAGER, START=20, TYPE=H, BYTES= 2, DESC='MNGR. NUMBER'
 FIELD NAME= DIVISION,START=22, TYPE=C, BYTES=10, DESC='DIVISION'
 FIELD NAME= LOCATION,START=32, TYPE=C, BYTES=13, DESC='LOCATION';
/* THIS DEFINITION IS THE ONE WHICH WILL BE USED TO PERFORM THE SUMMARIZATION  */
CREATE DXTFILE NAME=VSAMDSUM, ACCESS=VK, FREQ=20,
 DATAEXIT=DEPTNBRS, XBYTES=5, EODCALL=Y,DESC='SUMMARY INFORMATION'
```

Figure 40 (Part 1 of 2). DXTFILE definitions

```
FIELD NAME= DEPTKEY, START=1, TYPE=C, BYTES=5, SEQFLD=R,
            DESC='VSAM KEY IS DEPT/PRODNUM'
FIELD NAME= DEPT,START= 1,  TYPE=Z, BYTES= 2, DESC='DEPT NUMBER'
FIELD NAME= PRODNUM,  START= 3, TYPE=Z, BYTES= 3, DESC='PRODUCT NO'
FIELD NAME= PROJREC#, START= 1, TYPE=H, BYTES= 2, DESC='# PROJ RECS'
FIELD NAME= ORGREC#,  START= 3, TYPE=H, BYTES= 2, DESC='# ORG RECS';
```

Figure 40 (Part 2 of 2). DXTFILE definitions

Figure 41 shows the DXTVIEW definition.

```
CREATE DXTVIEW NAME=VDEPTSUM, DXTFILE=VSAMDSUM, FIELDS=*,
       DESC='DEPARTMENT RECORD SUMMARY INFORMATION';
```

Figure 41. DXTVIEW definition (VDEPTSUM)

This exit routine uses the Data exit routine anchor area for an accumulation area. This anchor area is the fourth parameter passed to the Data exit routine. When the exit routine detects a change in value of DEPT, it returns the accumulated values for the old DEPT back to DataRefresher. The exit routine then zeros the accumulators and begins getting the next DEPT values. Finally, the end-of-data call is used to pass the accumulated values for the last DEPT back to DataRefresher for processing.

Although the PROCEDURE label for this exit routine is DAXPLI, the DATAEXIT keyword in the DXTFILE definition identifies the exit as DEPTNBRS. The DATAEXIT keyword identifies the exit routine by the name of its load module, and in this example, DEPTNBRS is how the exit routine is link-edited.

To help debug the exit routine, the record key is placed in DAXSMESG. This causes DataRefresher to include the key value in DXTPRINT in a message. You can remove it if you do not require it.

The accumulators are zeroed after moving the contents to the output buffer in the DAXCALL=DAXCEOD section of this Data exit routine. Clearing the accumulators is recommended as there may be multiple extract requests going against the file at a single time. DataRefresher will not modify the anchor area after the original setting of all zeros, so the anchor area values may stay in storage across multiple extract requests. Figure 42 shows the extract request.

```
SUBMIT   EXTID=GMW3, JCS=JCS3, ...
EXTRACT ...
    SELECT PROJREC#, ORGREC#
    FROM VDEPTSUM
    ;
```

Figure 42. Extract request (GMW3)

The SUBMIT and EXTRACT statements show only the options related to the points discussed. Figure 43 shows the output from the extract request.

```
   00012  00008
```

Figure 43. Output from the extract request (GMW3)

Chapter 5. Writing User Data Type exit routines

User Data Type exit routines enable you to process specific fields in your source data that are in a user-defined format.

User Data Type exit routines are field level exit routines, in contrast to Data exit routines, which are segment or record level exit routines.

You can write User Data Type exit routines to convert as many user data type fields as you need. Although you may write User Data Type exit routines to convert more than one field type, it is advisable to have one exit routine per field type to reduce the possibility of errors. User Data Type exit routines cannot be used in combination with GDI Select exit routines.

Task overview

Writing and using User Data Type exit routines with DataRefresher consists of:

1. Determining which fields in your source data will need a User Data Type exit routine.
   - For information on the functions that User Data Type exit routines can perform, see "User Data Type exit routines" on page 3.
2. Writing your User Data Type exit routine.

   For information on:
   - How User Data Type exit routines interact with the control block, see "Using the User Data Type exit routine control block" on page 71.
   - Using the sample User Data Type exit routines provided with DataRefresher, see "Writing User Data Type exit routines using sample exit routines" on page 71.
3. *If your User Data Type exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. Compiling and link-editing your User Data Type exit routine.
   - For information on compiling and link-editing your User Data Type exit routines, see "Compiling and link-editing your exit routines" on page 22.
5. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on how to:
   - Register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Specify which exit routine to use when creating a source folder, see the *DataRefresher OS/2 User's Guide.*

6. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresher using CREATE DATATYPE, CREATE DXTFILE, and CREATE DXTPSB commands.

   For information on:

   - Specifying a User Data Type exit routine on your CREATE DATATYPE command and on your CREATE DXTFILE or CREATE DXTPSB commands, see "Identifying your User Data Type exit routine to DataRefresher" on page 72.
   - The syntax of these commands, see the *DataRefresher Command Reference.*

Calling User Data Type exit routines

Identify your User Data Type exit routine to DataRefresher on your CREATE DATATYPE command and on your CREATE DXTFILE or CREATE DXTPSB command. Associate your CREATE DATATYPE command with your CREATE DXTFILE or CREATE DXTPSB command by means of the:

- SRCTYPE parameter (CREATE DATATYPE command)
- TYPE parameters (CREATE DXTFILE or CREATE DXTPSB command)

The UIM calls the exit routine at field definition time to validate the length of the source data and to provide the length of the target field when necessary. *Field definition time* is the time at which a user data type is used in a FIELD statement of a CREATE DXTFILE or CREATE DXTPSB command. You will be notified of any errors with the CREATE DATATYPE command information at field definition time, rather than having to wait until the DEM processes an extract request using that data type.

The DEM calls the User Data Type exit routine during extraction to convert a user data type into the format described by the DATATYPE data description.

Before calling your User Data Type exit routine, DataRefresher copies the source data type field to be altered into the input buffer of the exit routine. When DataRefresher calls the User Data Type exit routine, the exit routine takes the data from the input buffer and alters it. When the User Data Type exit routine finishes processing, it puts the data into the output buffer. The DEM can then continue its processing using the converted values.

When the DEM calls your User Data Type exit routine, it passes as parameters the addresses of the:

- Control block
- Input buffer for the source field
- Output buffer for the target field
- Anchor area (64 bytes)

As in a standard module linkage, your User Data Type exit routine returns control to the DEM using an address passed to the exit routine in register 14. This address is passed to the User Data Type exit routine on entry.

Using the User Data Type exit routine control block

The User Data Type exit routine control block is the interface between DataRefresher and your User Data Type exit routine. You should have a separate instance of the control block for each User Data Type exit routine you write. Control blocks for each programming language are contained in the DVR110.DVRINTER library as shown in Table 19.

Table 19. User Data Type exit routine control blocks

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
User Data Type exit	DVRXCUTP	DVRXCUTC	DVRXCUTA

User Data Type exit routines can modify the following control block fields:

- Number of number of bytes (UDTSBYTV). This field can only be modified when the UIM calls the exit routine at field definition time.
- Number of places after the decimal point in the source (UDTSSCLV). This number can only be modified when the UIM calls the exit routine at field definition time.
- Number of target bytes (UDTTBYTV).
- Number of places after the decimal point in the target (UDTTSCLV).
- A 128-byte scratchpad area (UDTSCRT1) to which the User Data Type exit routine can save addresses and other information that it uses during the life of the exit.
- A return code field (UDTXRETC).
- Two flag fields (UDTENTRD and UDTINCTL) that you can use for tracing and debugging.
- A message field (UDTXMESG).
- A 64-byte area (the fourth parameter field) that the User Data Type exit routine can use as an anchor for exit-obtained space for the life of the DEM.

Writing User Data Type exit routines using sample exit routines

Included with DataRefresher in the DVR110.DVRINTER library are sample User Data Type exit routines in PL/I, COBOL, and Assembler. You can copy or print the following sample exit routines and modify them to perform functions you require:

DVRXYXUT PL/I User Data Type exit routine
DVRXOXUT COBOL User Data Type exit routine
DVRXOXVC COBOL User Data Type exit routine to produce a VC data type
DVRXAXUT Assembler User Data Type exit routine

Information on copying and printing the sample exit routines provided is contained in Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

See "User Data Type exit routine example" on page 73 for an example of how a User Data Type exit routine can be used.

Identifying your User Data Type exit routine to DataRefresher

To process source data with a User Data Type exit routine, you must describe the source data you are referencing to the DEM. Put your data description in the FDTLIB by writing CREATE DATATYPE, CREATE DXTFILE, and CREATE DXTPSB commands.

For example, if you have created three user-defined data types and you want to use them in a DXTFILE associated with a VSAM input file. you would first need to define the user data types. Figure 44 shows how you would specify the three user data types using the CREATE DATATYPE command.

```
CREATE DATATYPE EXIT     = XXEDIT,
                SRCTYPE  = XX, 1
                SRCBYTES = 5, 2
                SRCSCALE = 4, 3
                TRGTYPE  = C,
                TRGBYTES = 10,
                DESC     = 'DATA IS IN XX FORMAT';
CREATE DATATYPE EXIT     = YYEDIT,
                SRCTYPE  = YY,
                SRCBYTES = 7,
                TRGTYPE  = A,
                TRGBYTES = 8,
                DESC     = 'DATA IS IN YY FORMAT';
CREATE DATATYPE EXIT     = ZZEDIT,
                SRCTYPE  = ZZ,
                SRCBYTES = 6,
                TRGTYPE  = G,
                TRGBYTES = 8,
                DESC     = 'DATA IS IN ZZ FORMAT';
```

Figure 44. Sample CREATE DATATYPE command for a User Data Type exit routine

Next, you should write your CREATE DXTFILE or CREATE DXTPSB command. Figure 45 shows how you would specify the three user data types using the CREATE DXTFILE command.

```
CREATE DXTFILE
   NAME=UDTX, ACCESS=VK, DESC='THREE USER DATA TYPES FOR USER DATA TYPE EXIT',
   SEGMENT NAME=DEPT, DESC='DEPARTMENT SALES INFO',
      FIELD NAME=FLD1, TYPE=XX 1 , BYTES=5 2 ,  START=1, SCALE=4 3
      FIELD NAME=FLD2, TYPE=C,    BYTES=10,   START=7
      FIELD NAME=FLD3, TYPE=YY,   BYTES=7,    START=2
      FIELD NAME=FLD4, TYPE=P,    BYTES=8,    START=6
      FIELD NAME=FLD5, TYPE=ZZ,   BYTES=6,    START=3;
```

Figure 45. Sample CREATE DXTFILE command for a User Data Type exit routine

1 The value you define to the SRCTYPE in the CREATE DATATYPE command should be the same as the value you later specify using the TYPE keyword of the FIELD statement in the CREATE DXTFILE or CREATE DXTPSB command.

2 If you define a fixed SRCBYTES value in the CREATE DATATYPE command, you do not need to code a BYTES value in the CREATE DXTFILE or CREATE DXTPSB command for any field of that data type. With variable length data, the SRCBYTES value should be coded as VARIES, and the BYTES keyword you later specify in the FIELD statement of your CREATE DXTFILE or CREATE DXTPSB command provides the length value to be used as input to the User Data Type exit routine.

3 The value you define to the SRCSCALE in the CREATE DATATYPE command should be the same as the value you give to the SCALE keyword in the CREATE DXTFILE or CREATE DXTPSB command.

For detailed information about the syntax of the CREATE DATATYPE, CREATE DXTFILE, and CREATE DXTPSB commands, see the *DataRefresher Command Reference*.

User Data Type exit routine return codes

Your User Data Type exit routine communicates with DataRefresher via return codes placed in the UDTXRETC field of the control block. If your User Data Type exit routine returns a code other than 0, you will receive an error message. Table 20 shows the User Data Type exit routine return code values.

Table 20. User Data Type exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
Other	Output is based on the FLDERR keyword of the SUBMIT command

User Data Type exit routine example

This section shows an example of a User Data Type exit routine. The sample User Data Type exit routine in Assembler used in this section is contained in member DVRXAXUT in the DVR110.DVRINTER library.

All of the sample User Data Type exit routines perform the same function: the exit routines convert a bit string into a character string. Each bit represented by a character is set to 1 or 0 based on the value of the related bit. An alternative representation might be T for true and F for false. This exit routine could be used to convert bit control fields to individual flag bytes.

The source field is converted, one bit at a time, into 1 or 0 characters in the target field. For example, the two-byte character string `A1` is equal to `C1F1` in hexadecimal or `1100000111110001` in binary. This bit string would be converted into the 16-byte character string `1100000111110001`.

The length of the target field determines when processing terminates. When the target length is more than eight times the source length, the exit routine sets the remaining right hand bytes to the character `0`.

The following DATATYPE definitions are used in this example:

- AA has fixed-length fields, as shown in Figure 46.

```
CREATE DATATYPE SRCTYPE = AA, EXIT=DVRXAXUT,
   SRCBYTES = 4,
   TRGTYPE  = C,
   TRGBYTES = 32;
```

Figure 46. DATATYPE definition (AA)

- AB has variable-length fields, as shown in Figure 47, so the SRCBYTES and TRGBYTES keywords have a value of `VARIES`. The target length will be eight times the source length.

```
CREATE DATATYPE SRCTYPE = AB, EXIT=DVRXAXUT,
   SRCBYTES = VARIES,
   TRGTYPE  = C,
   TRGBYTES = VARIES;
```

Figure 47. DATATYPE definition (AB)

- AC will have the target length set to eight by the exit routine, as shown in Figure 48.

```
CREATE DATATYPE SRCTYPE = AC, EXIT=DVRXAXUT,
   SRCBYTES = 1,
   TRGTYPE  = C,
   TRGBYTES = VARIES ;
```

Figure 48. DATATYPE definition (AC)

Figure 49 is the DXTFILE definition used in this example.

```
CREATE DXTFILE NAME=UDTFILE, ACCESS=PS
   FIELD NAME= UDTFLD1,   START= 1,  TYPE=AA, BYTES=4
   FIELD NAME= UDTFLD2,   START= 11,  TYPE=AB, BYTES=2
   FIELD NAME= UDTFLD3,   START= 21, TYPE=AC, BYTES=1
   FIELD NAME= UDTFLD4,   START= 31, TYPE=AB, BYTES=8;
```

Figure 49. DXTFILE definition (UDTFILE)

When the SRCBYTES value is variable, the BYTES keyword provides the length value to be used as input to the exit routine. Since the SRCBYTES value for Data Types AA and AC are fixed, the BYTES value is optional for the fields UDTFLD1 and UDTFLD3.

Figure 50 shows the definition of the DXTVIEW.

```
CREATE DXTVIEW NAME=UDTVIEW1, DXTFILE=UDTFILE, FIELDS=*;
```

Figure 50. DXTVIEW definition (UDTVIEW1)

Four extract requests are run with the sample Assembler User Data Type exit routine (DVRAXUT). In the first extract request, illustrated in Figure 51 on page 75, the source data has a value of `1234` and is 4-bytes long. The target data is equal to `11110001111100101111001111110100` and has a length of 32 bytes.

```
SUBMIT   EXTID=UDTEXT1, JCS=AT018903, DBS=DB2, FORMAT=EBCDIC
EXTRACT
    SELECT UDTFLD1
    FROM UDTVIEW1
    ;
```

Figure 51. Extract request (UDTEXT1)

In the second extract request, illustrated in Figure 52, the source data has a value of 12 and is 2-bytes long. The target data is equal to 1111000111110010 and has a calculated length of 16 bytes.

```
SUBMIT   EXTID=UDTEXT2, JCS=AT018903, DBS=DB2, FORMAT=EBCDIC
EXTRACT
    SELECT UDTFLD2
    FROM UDTVIEW1
    ;
```

Figure 52. Extract request (UDTEXT2)

In the third extract request, illustrated in Figure 53, the source data has a value of 1 and is one-byte long. The target data is equal to 11110001 and has a calculated length of 8 bytes.

```
SUBMIT   EXTID=UDTEXT3, JCS=AT018903, DBS=DB2, FORMAT=EBCDIC
EXTRACT
    SELECT UDTFLD3
    FROM UDTVIEW1
    ;
```

Figure 53. Extract request (UDTEXT3)

In the fourth extract request, illustrated in Figure 54, the source data has a value of 12345678 and is 8-bytes long. The target data has a calculated length of 64 bytes and is equal to:

1111000111110010111100111111010011110101111101101111011111111000

```
SUBMIT   EXTID=UDTEXT4, JCS=AT018903, DBS=DB2, FORMAT=EBCDIC
EXTRACT
    SELECT UDTFLD4
    FROM UDTVIEW1
    ;
```

Figure 54. Extract request (UDTEXT4)

Chapter 6. Writing Date/Time Conversion exit routines

Date/Time Conversion exit routines reformat date and time fields from your source data into International Standards Organization (ISO) format, or into a format that DataRefresher can recognize. You may not need to use a Date/Time Conversion exit routine even though date or time fields in your source data are not in ISO format. DataRefresher supplies its own Date/Time Conversion exit routine to convert certain date and time formats into ISO format. See "Date/Time Conversion exit routines" on page 3 for further information.

Task overview

Writing and using Date/Time Conversion exit routines consists of:

1. Determining which fields in your source data will need a Date/Time Conversion exit routine.
   - For further information, see "Date/Time Conversion exit routines" on page 3.
2. Writing your Date/Time Conversion exit routine.

   For information on:
   - How Date/Time Conversion exit routines interact with the control block, see "Using the Date/Time Conversion exit routine control block" on page 78.
   - Using the sample Date/Time Conversion exit routines provided with DataRefresher, see "Writing Date/Time Conversion exit routines using sample exit routines" on page 79.
3. *If your Date/Time Conversion exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. Compiling and link-editing your Date/Time Conversion exit routine.
   - For further information, see "Compiling and link-editing your exit routines" on page 22.
5. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on how to:
   - Register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Specify which exit routine to use when creating a source folder, see the *DataRefresher OS/2 User's Guide.*

6. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresher with the CONV keyword of a CREATE DXTFILE or CREATE DXTPSB command.

   For information on:

   - Specifying your Date/Time Conversion exit routine on your CREATE DXTFILE or CREATE DXTPSB command, see "Identifying your Date/Time Conversion exit routine to DataRefresher" on page 80 .
   - The syntax of these commands, see the *DataRefresher Command Reference.*

Calling Date/Time Conversion exit routines

Identify your Date/Time Conversion exit routine to DataRefresher with the CONV keyword on your CREATE DXTFILE or CREATE DXTPSB commands.

When executing an extract request that requires source data associated with a Date/Time Conversion exit routine, the DEM loads and calls the appropriate exit routine and gives it the address of the data buffer. The exit routine gets the source data from this buffer, converts it, and places the data in another buffer. The exit routine then passes control and the address of the new data buffer to the DEM.

When the DEM calls your Date/Time Conversion exit routine, it passes as parameters the addresses of the:

- Source data to be converted
- Target area (where the converted data is to be placed)
- Control block

As in a standard module linkage, your Date/Time Conversion exit routine returns to the DEM using an address passed to the exit routine in register 14. This address is passed to the Date/Time Conversion exit routine on entry.

Using the Date/Time Conversion exit routine control block

The control block is the interface between DataRefresher and your Date/Time Conversion exit routines. A copy of the Date/Time Conversion exit routine control block is automatically associated with each field for which you have written a Date/Time Conversion exit routine.

There is a different Date/Time Conversion exit routine control block for each programming language. The control blocks for each language are contained in the DVR110.DVRINTER library, as shown in Table 21:

Table 21. Date/Time Conversion exit routine control blocks

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Date/Time Conversion exit routine	DVRXCCXP	DVRXCCXC	DVRXCCXA

The control block contains:

- The length of each date or time value before conversion
- The data type (DATE or TIME)
- A field for the return code

Your Date/Time Conversion exit routine can modify the following fields within the control block:

- A return code field (CXXCVNRC)
- A 1000-byte scratch area that can be used by the exit routine (CXXWORK).

Writing Date/Time Conversion exit routines using sample exit routines

Included with DataRefresher are sample Date/Time Conversion exit routines in PL/I, COBOL, and Assembler. These sample exit routines are contained in the DVR110.DVRINTER library. You can copy or print the following sample exit routines and modify them to perform the functions you need:

DVRXYTD PL/I Date/Time Conversion exit routine
DVRXOTD COBOL Date/Time Conversion exit routine
DVRXATD Assembler Date/Time Conversion exit routine

You can supply a Date/Time Conversion exit routine for each date or time field existing in your source data. The date and time output from that exit routine must be in the (ISO) format shown in Table 22:

Table 22. ISO date and time formats

Data Type	Format	Description
DATE	yyyy-mm-dd	Always a 10-byte character string
TIME	hh.mm.ss	Always an 8-byte character string

where:

- `yyyy = year`
- `mm = month`
- `dd = day`
- `hh = hours`
- `mm = minutes`
- `ss = seconds`

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

For an example of how a Date/Time Conversion exit routine can be used, see "Date/Time Conversion exit routine example" on page 81.

Identifying your Date/Time Conversion exit routine to DataRefresher

The DEM gets the name of your Date/Time Conversion exit routine from the data description. You identify your exit by naming it on the CONV keyword in the FIELD statement of your CREATE DXTFILE or CREATE DXTPSB command.

For example, if you have some time data in a local format that you wanted to convert to standard ISO format. Figure 55 shows how you would specify a Date/Time Conversion exit on the CREATE DXTPSB command to convert time type data.

```
CREATE
DXTPSB  NAME=DXTTIME,
        DESC='CREATE A DESCRIPTION SPECIFYING
              DATE/TIME CONVERSION EXIT.'
  DXTPCB  NAME=DTCXPCB, DBACCESS=HISAM
  SEGMENT NAME=DEPT,   PARENT=0, BYTES=42, FREQ=10,
          DESC='SEGMENT FOR ARRIVETIM'
    FIELD   NAME=ARVTIME,
            START=1,
            TYPE=T 1 ,
            CONV=ISOCONV 2 ,
            BYTES=10 3 ,
            DESC='WHEN THE PLANES ARRIVE';
```

Figure 55. Sample CREATE DXTPSB command for a Date/Time Conversion exit routine

1 TYPE=T identifies the data as time type data. (TYPE=A would identify the data as date type data.)

2 You should include the CONV keyword with the FIELD statement in one of the following:

- CREATE DXTPSB command for source databases (as in the example above)
- CREATE DXTFILE command for source files

The value of the CONV keyword is the name of the load module containing your Date/Time Conversion exit routine. Load module names are alphameric-national names and must be eight characters or less in length. For information on DataRefresher naming conventions, see the *DataRefresher Command Reference*.

CONV=ISOCONV identifies ISOCONV as the Date/Time Conversion exit routine which converts the data from local to standard format. If the data was in a USA or JIS format, you would not specify the CONV keyword, because DataRefresher automatically converts these formats.

3 BYTES=10 shows that the size of the field containing the time data to be converted is ten bytes. Your Date/Time Conversion exit routine is given the BYTES value as the *input* length of the data. DataRefresher always assumes that the *output* length for time data is eight bytes. For date data, the output length is assumed to be ten bytes.

For detailed information about the syntax of the CREATE DXTFILE and CREATE DXTPSB commands, see *DataRefresher Command Reference*.

Date/Time Conversion exit routine return codes

Your Date/Time Conversion exit routine communicates with DataRefresher using return codes placed in the CXXCVNRC field of the control block. If the Date/Time Conversion exit routine returns a code other than 0, you will receive an error message. DataRefresher continues processing, but produces no output for the field that produced the error.

Table 23. Date/Time Conversion exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
Other	Error; continue processing

Date/Time Conversion exit routine example

This section shows you an example of the use of a Date/Time Conversion exit routine. The exit routine used in this example is the sample PL/I Date/Time Conversion exit routine, which is contained as member DVRXYTD in the DVR110.DVRINTER library. The sample exit routine converts the date and time data provided by the system, so no sample input data is provided.

The exit routine in this example converts the date and time from the format provided by the system to a format that DataRefresher can use. The system date format is `yymmdd`. The exit converts this into the type A DataRefresher format of `yyyy-mm-dd`. For this conversion, `yyyy` is equal to `19` plus the year, `mm` is the month, and `dd` is the day.

The system time format is `hhmmsstt`. The exit routine converts this into the type T DataRefresher format of `hh.mm.ss`. For this conversion, `hh` is equal to the hour, `mm` are the minutes, `ss` are the seconds, and `tt` are the hundredths of a second.

Figure 56 contains the DXTFILE definition used in the sample exit routine.

```
CREATE DXTFILE NAME=DATETIME, ACCESS=PS
  FIELD NAME= DATEXX,START= 1,  TYPE=A, BYTES= 06, CONV=DVRXYTD
  FIELD NAME= TIMESX,START=07,  TYPE=T, BYTES= 08, CONV=DVRXYTD;
```

Figure 56. DXTFILE definition (DATETIME)

Figure 57 contains the description of the DXTVIEW used in the sample exit routine.

```
CREATE DXTVIEW NAME=DATE_TIME, DXTFILE=DATETIME, FIELDS=*;
/*                                                          */
```

Figure 57. DXTVIEW definition (DATE_TIME)

Figure 58 contains the extract request used in the sample exit routine.

```
SUBMIT EXTID= EXTRACT_T, NODE=MLPMVS1, USERID=VNDORN2,
       EXTDATA=EXTDATA
EXTRACT
       SELECT  *
       FROM DATE_TIME;
```

Figure 58. Extract request (EXTRACT_T)

Chapter 7. Writing Generic Data Interface (GDI) exit routines

Generic Data Interface (GDI) exit routines enable you to extract data from many data sources not directly supported by DataRefresher. When DataRefresher invokes a GDI exit routine, the exit routine retrieves the data from the source and passes it back to DataRefresher for further processing.

There are two types of GDI exit routines: Select and Record. GDI Select exit routines return only the columns and rows of data that satisfy the SELECT statement in the SUBMIT command of your extract request. GDI Record exit routines act as a user-written access method and return records from a user-specified location to DataRefresher, which then applies WHERE clause criteria and selects fields.

A single GDI exit routine can be written to access one particular file, or it can be written in a general way to handle a broad class of files or databases. There is no limit to the number of GDI exit routines you can write.

Task overview

Writing and using GDI exit routines with DataRefresher consists of:

1. Determining whether you will use a GDI Select exit routine or a GDI Record exit routine to extract your data.
   - For information on the functions that GDI Select and Record exit routines can perform, see "Generic Data Interface exit routines" on page 4.
2. Writing your GDI exit routine.

   For information on:
   - How GDI exit routines interact with the control blocks, see "Using the GDI exit routine control blocks" on page 90 and "Relationship of exit call types to the control blocks" on page 93
   - Using the sample GDI exit routines provided with DataRefresher, see "Writing GDI exit routines" on page 97.
3. Precompile and binding your exit routine if your GDI exit routine will access DB2.
   - For information on preparing your GDI exit routine to access DB2, see "Precompiling and binding exit routines for DB2 applications" on page 21.
4. *If your GDI exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
5. Compiling and link-editing your GDI exit routine.

   For information on compiling and link-editing your GDI exit routine, see "Compiling and link-editing your exit routines" on page 22.

6. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on:

   - How to register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Accessing DB2 with GDI exit routines included in extract requests submitted with DataRefresher OS/2, see "Accessing DB2 when using DataRefresher OS/2" on page 102.

7. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresher using a CREATE DXTFILE command.

   For information on:

   - The syntax of the CREATE DXTFILE command, see the *DataRefresher Command Reference.*
   - Specifying your GDI exit routine on your CREATE DXTFILE command, see "Identifying your GDI exit routine to DataRefresher" on page 100.

How GDI exit routines work

When running an extract request that requires data associated with a GDI exit routine, DataRefresher calls the appropriate GDI exit routine to access the source data. DataRefresher looks for the GDI exit routine load module in a load library specified in the JCL used to run the UIM or DEM.

DataRefresher loads your GDI exit routine into storage (unless it is already resident) and then calls the exit routine through its main entry point. If the GDI exit routine cannot be loaded, DataRefresher issues a diagnostic message and marks the GDI exit routine as not usable for the current run of the UIM or DEM.

There are two types of GDI exit routines:

- GDI Select exit routines
- GDI Record exit routines

GDI Select exit routines

A *GDI Select exit routine* works like a relational database application that supports a structured query language. You build the extract request to be passed to the GDI Select exit routine the same way you would to extract data from a DB2 system. A data source described by a GDI Select exit routine is called a GDI Select file.

GDI Select exit routines let you submit an SQL-like SELECT statement to DataRefresher without previously storing field descriptions in the FDTLIB. The UIM does not process the SELECT statement. The SELFILE value on the SUBMIT command names the data source to be accessed and triggers DataRefresher to pass the full SELECT statement to your GDI Select exit for processing. This means that the SQL supported by a GDI Select exit routine is not restricted to the DataRefresher subset.

Your SELECT statement can contain other SQL constructs such as those that support grouping and aggregation. The result is that you can code the SELECT statement that is appropriate to your GDI Select exit routine and the system that actually reads the data. The system does not have to accept the SQL; it only needs to be able to accept the exit routine. The exit routine could translate the SQL into whatever interface the data system supports.

When called by DataRefresher, the GDI Select exit routine returns the next row of selected data from the source to the FETCH buffer. This row will have been determined by the predicate of the SQL statement passed to the buffer. DataRefresher then includes all the selected column data in its output. How the GDI Select exit routine retrieves the data does not affect DataRefresher. Furthermore, DataRefresher does not verify that the data meets any WHERE keyword criteria. This is performed by the exit routine.

DataRefresher does not let you create DXTVIEW descriptions for GDI Select files. DataRefresher automatically generates sample views (no fields are defined) for GDI source data. The sample view that DataRefresher generates has the same name as the NAME keyword of the DXTFILE keyword. If the GDI select file is deleted, DataRefresher automatically deletes the associated view. You can get the effect of multiple views over a GDI Select file by writing multiple CREATE DXTFILE commands which, in turn, automatically generate associated views.

DataRefresher cannot join GDI Select files with other types of DXTFILEs. GDI Select files can be joined with each other, if they name the same GDI Select exit routine and that exit routine (in conjunction with its underlying data system) can perform a join. Other considerations when using GDI Select exit routines are that DataRefresher:

- Can process GDI Select files with fields containing nulls
- Cannot batch extract requests involving GDI Select files

GDI Record exit routines

A *GDI Record exit routine* works like a custom-made access method, supporting the DEM as, for example, VSAM does. The GDI Record exit routine could be used to extract data managed by a database, or any other file organization. A data source described by a GDI Record exit routine is called a GDI Record file.

This type of exit is similar to an access method that reads records one at a time from a data source. When called by DataRefresher, the GDI Record exit routine returns the next data record from the source to the FETCH buffer. DataRefresher then checks the record against the WHERE keyword criteria, extracts the requested fields from the FETCH buffer, and sends them to the designated target.

GDI Record exit routines are not passed SQL SELECT statements. If the WHERE keyword contains a field defined as a sequence field of the DXTFILE keyword, DataRefresher places operators and values in the control block passed to the GDI Record exit routine on the calls. The exit routine can then be written to make use of this key field comparison information. WHERE keyword information for other, non-key fields is not given to the exit routine.

The predicate may include either of the following:

- Any of the comparison operators (`=`, `¬=`, `>`, `<`, `>=`, `<=`) and a value
- The set membership operator (`IN`) and a list of values

The comparison operator (>=) and the value (100) are passed to the GDI Record exit routine as follows:

```
SELECT  FIELD1, FIELD2, FIELD3
  FROM  DXTVIEW1
  WHERE KEYFIELD >= 100
```

The exit routine can then qualify records or optimize data access using the key field.

DataRefresher does not automatically generate view descriptions for GDI Record files as it does for GDI Select files. You can define as many DXTVIEWs of a GDI Record file as you need.

Optionally, your GDI Record exit routine can preprocess the data in GDI Record files. Or, to give your GDI Record exit routine a measure of data independence, you can design it to simply return records to DataRefresher, leaving a Data exit routine to manipulate the fields if necessary. For more information on Data exit routines, see Chapter 4, "Writing Data exit routines" on page 47.

DataRefresher can join data from GDI Record files with data from:

- Other GDI Record files
- VSAM data sets
- Physical sequential data sets
- IMS DL/I databases

The GDI Record exit routine may use self-defining files or the support of a set of system catalogs or dictionary products. If the exit routine can define the file this way to DataRefresher, the user does not need to provide field level detail.

To join data from GDI Record files, your GDI Record exit routine must reposition to the top of files to reread them from the beginning. A join might not be possible if there is a conflict between two or more environments required to access your source data (if two or more environments want to initiate the task, for example). Some other considerations when using GDI Record exit routines are:

- DataRefresher cannot process GDI Record files with fields containing nulls
- DataRefresher can batch extract requests involving GDI Record files

How DataRefresher calls a GDI exit routine

Identify your GDI exit routine to DataRefresher on the GDIEXIT keyword of your CREATE DXTFILE command; there is no GDIEXIT keyword on a CREATE DXTPSB command. When you use a GDI exit routine, your extract is processed normally. This involves the execution of the:

1. UIM to update the FDTLIB with data descriptions and update the EXTLIB with extract requests
2. DEM to extract the data

These steps are followed regardless of where the source data is located. Even when extracting data from DB2 using a GDI exit routine, there is a UIM phase and a DEM phase. The REM is not used.

Table 24 shows whether DataRefresher will call the GDI exit routine and at what point in the extract process:

Table 24. When DataRefresher calls a GDI exit routine

Phase	Program	GDI Select Exit	GDI Record Exit
CREATE DXTFILE command	UIM	no	yes if DETAIL=EXIT
CREATE DXTVIEW command	UIM	N/A (sample)	no
Submit extract request	UIM	yes	no
Extract data	DEM	yes	yes
GETDEF command	UIM	yes	no

When DataRefresher calls a GDI exit routine, register 1 points to a parameter list containing the addresses of four interface areas. The addresses are passed to the GDI exit routine when DataRefresher calls it, as shown in the following example:

```
CALL GDIEXIT (COMCB, DESCB, FETCHB, GLOBA)
```

The parameters (COMCB, DESCB, FETCHB, and GLOBA) are always passed. However, the actual FETCH buffer address is only passed on FETCH calls. Your GDI exit routine returns to DataRefresher using an address stored in register 14.

Calling a GDI Select exit routine

When you submit an extract request, DataRefresher first builds the DVRXCGSC and DVRXCGSD control blocks to hold processing and descriptive information about the extract transaction. Together, the UIM and DEM make the following types of calls to a GDI Select exit routine:

- DESCRIBE
- OPEN
- FETCH
- CLOSE

Figure 59 shows how the UIM and DEM call a GDI Select exit routine.

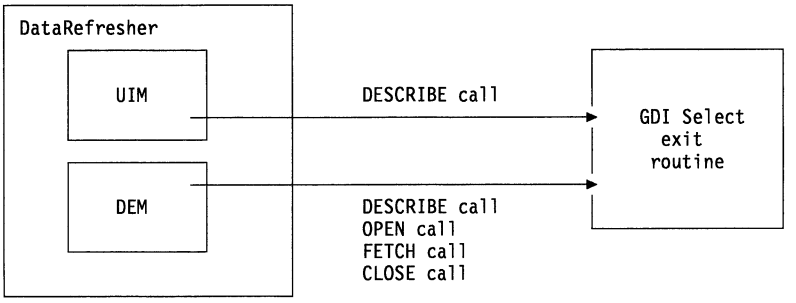

Figure 59. How DataRefresher calls a GDI Select exit routine

DESCRIBE As DataRefresher does not have a description of data for the GDI Select exit routine, the UIM makes a DESCRIBE call to the exit routine and passes, in the control block, the SELECT statement from the SUBMIT command. The exit routine validates the SELECT statement, and describes the output of the SELECT to DataRefresher as output from the DESCRIBE call.

The UIM updates the EXTLIB with the extract request. The DEM later reissues the DESCRIBE call for its own purposes.

OPEN The DEM makes an OPEN call to the exit routine. The exit routine determines whether the DEM can continue running and passes back a return code. The exit routine can also be written to perform one-time initialization functions, for example:

- Opening data sets
- Acquiring main storage using a GETMAIN macro
- Initializing fields
- Preparing or compiling functions on the SELECT statement

FETCH The DEM issues a series of FETCH calls until the exit routine returns an end-of-data indication. The exit routine returns to DataRefresher, row by row, selected data from the source file that meets the WHERE keyword criteria. DataRefresher does not initialize the FETCH buffer before issuing a FETCH call.

DataRefresher extracts the data and places it into the output data set defined for the extract request.

CLOSE The DEM issues a CLOSE call to the exit routine. The exit routine cleans up the remnants of processing, closes any data sets used and issues a FREEMAIN instruction to free any additional storage.

Calling a GDI Record exit routine

When you submit an extract request, DataRefresher first builds the DVRXCGRC and DVRXCGRD control blocks to hold processing and descriptive information. Together, the UIM and DEM make the following types of calls to a GDI Record exit routine:

1. DESCRIBE
2. OPEN
3. FETCH
4. REPOSITN
5. CLOSE

Figure 60 on page 89 shows how the UIM and DEM call a GDI Record exit routine.

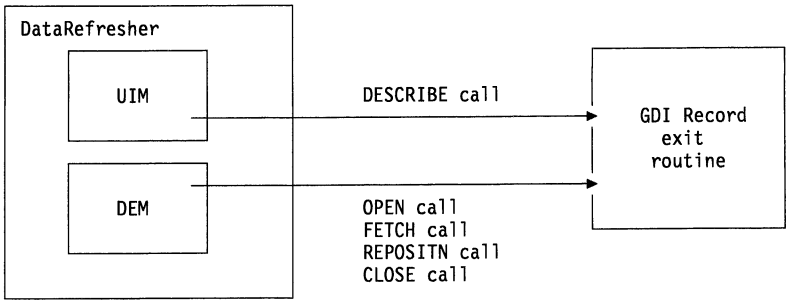

Figure 60. How DataRefresher calls a GDI Record exit routine

DESCRIBE If your exit routine has access to field and segment definitions in a self-defining file, data dictionary, database management system catalog, or another auxiliary of the source data, the UIM can call the exit routine with the DESCRIBE call when processing the CREATE DXTFILE command. You must have specified DETAIL=EXIT in your data description. The exit routine returns the details of the data description and indicates if it needs a larger control block to provide this detail.

The exit routine places the data description in a control block and returns control to DataRefresher.

The UIM validates the field and segment entries returned from the exit routine. For example, the UIM verifies that the length specified for a field is compatible with its data type. The UIM does not validate field names. DataRefresher updates the FDTLIB if the DESCRIBE call completes normally and DataRefresher finds no errors. If the exit routine signals to DataRefresher (through a return code of 4) that the Data Description control block is too small to hold the required number of entries, DataRefresher reformats a larger control block and reissues the DESCRIBE call. Work space in the control block is reinitialized.

OPEN The DEM makes preliminary OPEN and CLOSE calls to the exit routine for its own processing. On OPEN, the exit routine determines whether the DEM can continue running and passes back a return code. The exit routine also performs one-time initialization functions, for example:

- Opening data sets
- Acquiring main storage through a GETMAIN macro
- Initializing fields

The exit routine also returns the fixed or maximum length (if variable) of records in the file and the record format (fixed-length or variable-length) defined for the file. The record format defaults to 'F' for fixed unless otherwise specified.

DataRefresher calculates the necessary length of the FETCH buffer to be passed to your exit routine on FETCH calls and places the result in the control block. DataRefresher then obtains storage for the FETCH buffer by issuing a GETMAIN instruction.

FETCH The DEM issues a series of FETCH calls, until the exit routine returns an end-of-data indication. The exit routine returns to DataRefresher, row by row, selected data from the source file. If a condition on the

sequence field (the field associated with the SEQFLD keyword) is provided in the WHERE clause, DataRefresher will pass this WHERE information to the exit routine. The exit routine can use this information to optimize its search, or just return to DataRefresher the records that satisfy the WHERE clause. DataRefresher checks all the WHERE criteria. If the record returned from a FETCH call is variable-length, a field in the control block will tell DataRefresher the length of the record. DataRefresher does not initialize the FETCH buffer before issuing a FETCH call.

DataRefresher extracts the data and places it into the target.

REPOSITN The REPOSITN call is only issued when DataRefresher performs a join. The DEM issues this call to reposition to the top of source files for re-reading records during a join. In implementing REPOSITN in your exit routine, closing and reopening the underlying file might be unnecessary if the underlying access method or DBMS supports a reposition function.

CLOSE The DEM issues a CLOSE call to the exit routine. The exit routine cleans up the remnants of processing, closes any data sets used and issues a FREEMAIN instruction to free any additional storage required.

Using the GDI exit routine control blocks

Control blocks are used to pass information back and forth between DataRefresher and your GDI exit routine. The interface between DataRefresher and your GDI exit routine consists of a:

- Data Communication control block which holds processing information
- Data Description control block which holds a description of the source data
- FETCH buffer which holds data returned by the GDI exit routine
- User-managed global work area

The interface is synchronous; if the GDI exit routine waits or abends, DataRefresher also does.

Note: If the DEM processes requests submitted by other users, GDI exit routines should be written to be sensitive to the availability of the source data or associated database.

You should have copies of the Data Communication and Data Description control blocks for each data source or extract request. There is a different set of GDI control blocks for each programming language. These control blocks are provided with DataRefresher for your use in writing GDI exit routines. They are contained in the DVR110.DVRINTER library, as shown in Table 25:

Table 25. GDI exit routine control blocks			
GDI Exit Routine	**PL/I Control Block**	**COBOL Control Block**	**Assembler Control Block**
Select exit: Data Communication Select exit: Data Description	DVRXCSCP DVRXCSDP	DVRXCSCC DVRXCSDC	DVRXCSCA DVRXCSDA
Record exit: Data Communication Record exit: Data Description	DVRXCRCP DVRXCRDP	DVRXCRCC DVRXCRDC	DVRXCRCA DVRXCRDA

Your GDI Select exit routine can modify the following fields in the control blocks:

- Return code field (GSCRTNCD)
- Reason code field (GSCRSNCD)
- The flag fields (GSCENTRD and GSCINCTL) can be used for tracing and debugging
- Field (GSCMSG) can be used for the text of up to five DXTPRINT messages
- A save area for each column entry of data (GSDCSAVE)
- A 16384-byte work space (GSCWORK) where your exit can save, for example, addresses.

Your GDI Record exit routine can modify the following fields in the control blocks:

- Return code field (GRCRTNCD)
- Reason code field (GRCRSNCD)
- The flag fields (GRCENTRD and GRCINCTL) can be used for tracing and debugging
- The field (GRCMSG) can be used for the text of up to five DXTPRINT messages
- The save areas for the source file, each segment entry of data, and each field entry of data (GRDFLSAV, GRDSSAVE, and GRDFSAVE respectively)
- A 16384-byte work space (GRCWORK) where your exit can save, for example, addresses.
- Record length field (GRDMAXLN)
- Record format field (GRDRECFM)

Data Communication control blocks

The Data Communication control blocks act as communication areas. The information passed from DataRefresher in the control block consists of:

- Control values
- Processing options
- Key values
- SELECT statements

Information returned by the GDI exit routine in the Data Communication control block consists of:

- Return codes
- Reason codes
- Messages

DataRefresher uses the Data Communication control block to pass processing options and control values to your GDI exit routine, and your exit routine uses the Data Communication control block to pass return codes, reason codes, and message inserts back to DataRefresher.

The Data Communication control blocks contain a fixed-length section, along with a variable-length section used to hold SELECT statements for Select exit routines or key-related information for Record exit routines.

Data Communication control blocks are contained in the DVR110.DVRINTER library.

Data Description control blocks

The Data Description control blocks serve as Data Description areas. They contain the name (and ddnames) of the source data and information about each of its columns, segments, or fields.

DataRefresher uses the Data Description control blocks to pass file descriptions to your GDI exit routine. If your GDI exit routine supports DESCRIBE calls, it uses the Data Description control block to pass column descriptions (GDI Select exit routines) or segment and field descriptions (GDI Record exit routines) back to DataRefresher.

The Data Description control blocks contain a fixed-length section, together with a variable-length section to hold repeating entries that describe fields, segments, and columns. Data Description control blocks are contained in the DVR110.DVRINTER library.

FETCH buffers

The FETCH buffer holds the selected fields (GDI Select exit routines) or data records (GDI Record exit routines) fetched by your GDI exit routine from the source database or file.

Global work areas

DataRefresher provides a user-managed global work area of 64K bytes for one or more GDI exit routines to use during a call by the UIM or DEM. This is a common work area, and any data placed in the area by one GDI exit routine is available to all other GDI exit routines while the UIM or DEM is running.

The global work area is acquired when DataRefresher detects the first attempt to use a GDI exit routine. DataRefresher initializes the global work area to zeros and frees it just before terminating the UIM or DEM.

The global work area is not saved across program boundaries. Different global work space is used by the UIM than from that used by the DEM.

Relationship of exit call types to the control blocks

Tables 26, 27, 28, and 29 contain information on how the GDI exit routine calls map to the GDI control blocks.

The following field usage codes are used in the tables:

DVR Value is assigned by DataRefresher before the call

Exit Value is assigned by the GDI exit routine during the call

– Field is not applicable to the call

Table 26 contains the exit call types to the DVRXCGSC Data Communication control block for GDI Select exit routines.

Table 26. Exit call types to the DVRXCGSC Data Communication control block for GDI Select exit routines

Field	Describe	Open	FETCH	Close	Field Description	Notes
GSCCBNAM	DVR	DVR	DVR	DVR	Name of control block	
GSCCBLEN	DVR	DVR	DVR	DVR	Length of control block	
GSCXNAME	DVR	DVR	DVR	DVR	Name of GDI exit routine	
GSCCALL	DVR	DVR	DVR	DVR	Type of call to exit	
GSCEID	DVR	DVR	DVR	DVR	Extract request EXTID	
GSCUID	DVR	DVR	DVR	DVR	Extract request USERID	
GSCNID	DVR	DVR	DVR	DVR	Extract request NODE	
GSCENTRD	Exit	Exit	Exit	Exit	Exit-entered debugging switch	1
GSCINCTL	Exit	Exit	Exit	Exit	Exit-in-control debugging switch	1
GSCACCTL	DVR	DVR	DVR	DVR	Length of ACCOUNT parameter	
GSCACCT	DVR	DVR	DVR	DVR	ACCOUNT parameter	
GSCRTNCD	Exit	Exit	Exit	Exit	Exit return code	2
GSCRSNCD	Exit	Exit	Exit	Exit	Exit reason code	2
GSCNMSGS	Exit	Exit	Exit	Exit	Number of messages	2
GSCMSG	Exit	Exit	Exit	Exit	Up to five messages	3
GSCPARML	DVR	DVR	DVR	DVR	Length of user parameters	
GSCPARM	DVR	DVR	DVR	DVR	User parameters	
GSCWORK	Exit	Exit	Exit	Exit	Exit work space	4,5
GSCSTMLN	DVR	DVR	DVR	DVR	Length of SELECT statement	
GSCSTMVL	DVR	DVR	DVR	DVR	SELECT statement	

Notes:

1. DataRefresher initializes these (character) fields to blank when the UIM or DEM calls the GDI Select exit routine for the first time. When the exit routine returns, DataRefresher sets GSCINCTL to blank again.
2. DataRefresher initializes this (numeric) field to zero (0) prior to calling the GDI Select exit routine.
3. DataRefresher initializes this (character) field to blank prior to calling the GDI Select exit routine.
4. The GSCWORK field is work space for the sole use of the GDI Select exit; DataRefresher ignores this field. Optionally, the GDI Select exit routine could use this work space to maintain information (pointers to DCBs, for example) from one call of the exit routine to another. However, DataRefresher does not retain this work space across program boundaries—work space used on a call by the UIM is reinitialized on a subsequent call by the DEM.
5. DataRefresher initializes this field to hexadecimal zeros when the UIM or DEM calls the GDI Select exit routine for the first time.

Table 27 contains the exit call types to the DVRXCGRC Data Communication control block for GDI Record exit routines.

Table 27. Exit call types to the DVRXCGRC Data Communication control block for GDI Record exit routines

Field	Describe	Open	FETCH	Close	Repositn	Field Description	Notes
GRCCBNAM	DVR	DVR	DVR	DVR	DVR	Name of control block	
GRCCBLEN	DVR	DVR	DVR	DVR	DVR	Length of control block	
GRCFILNM	DVR	DVR	DVR	DVR	DVR	Name of GDI Record file	
GRCXNAME	DVR	DVR	DVR	DVR	DVR	Name of GDI exit routine	
GRCCALL	DVR	DVR	DVR	DVR	DVR	Type of call to exit	
GRCEID	DVR	DVR	DVR	DVR	DVR	Extract request EXTID	
GRCUID	DVR	DVR	DVR	DVR	DVR	Extract request USERID	
GRCNID	DVR	DVR	DVR	DVR	DVR	Extract request NODE	
GRCENTRD	Exit	Exit	Exit	Exit	Exit	Exit-entered debugging switch	1
GRCINCTL	Exit	Exit	Exit	Exit	Exit	Exit-in-control debugging switch	1
GRCACCTL	DVR	DVR	DVR	DVR	DVR	Length of ACCOUNT parameter	
GRCACCT	DVR	DVR	DVR	DVR	DVR	ACCOUNT parameter	
GRCRTNCD	Exit	Exit	Exit	Exit	Exit	Exit return code	2
GRCRSNCD	Exit	Exit	Exit	Exit	Exit	Exit reason code	2
GRCNMSGS	Exit	Exit	Exit	Exit	Exit	Number of messages	2
GRCMSG	Exit	Exit	Exit	Exit	Exit	Up to five messages	3
GRCPARML	DVR	DVR	DVR	DVR	DVR	Length of user parameters	
GRCPARM	DVR	DVR	DVR	DVR	DVR	User parameters	
GRCWORK	–	Exit	Exit	Exit	Exit	Exit work space	4,5
GRCKEYLN	–	DVR	DVR	DVR	DVR	Length of key	
GRCKEYN	–	DVR	DVR	DVR	DVR	Number of key values	
GRCKEYOP	–	DVR	DVR	DVR	DVR	Qualifier operator for key field	
GRCKEYVL	–	DVR	DVR	DVR	DVR	Key values	

Notes:

1. DataRefresher initializes this (character) field to blank when the UIM or DEM calls the GDI Record exit routine for the first time. When the exit routine returns to DataRefresher, DataRefresher sets GSCINCTL to blank again.
2. DataRefresher initializes this (numeric) field to zero (0) prior to calling the GDI Record exit routine.
3. DataRefresher initializes this (character) field to blank prior to calling the GDI Record exit routine.
4. The GRCWORK field is work space for the sole use of the GDI Record exit routine; DataRefresher ignores this field. Optionally, the GDI Record exit routine could use this work space to maintain information (pointers to DCBs, for example) from one call of the exit routine to another. However, DataRefresher does not retain this work space across program boundaries—work space used on a call by the UIM is reinitialized on a subsequent call by the DEM.
5. DataRefresher initializes this field to hexadecimal zeros when the UIM or DEM calls the GDI Record exit routine for the first time.

Table 28 contains the exit call types to the DVRXCGSD Data Description control block for GDI Select exit routines.

Table 28. Exit call types to the DVRXCGSD Data Description control block for GDI Select exit routines

Field	Describe	Open	FETCH	Close	Field Description	Notes
GSDCBNAM	DVR	DVR	DVR	DVR	Name of control block	
GSDCBLEN	DVR	DVR	DVR	DVR	Length of control block	
GSDDTLTO	DVR	DVR	DVR	DVR	Number of entries available	1
GSDDTLUS	Exit	DVR	DVR	DVR	Number of entries needed	1
GSDBUFLN	–	–	DVR	–	Length of FETCH buffer	2
GSDFLNAM	DVR	DVR	DVR	DVR	Names of GDI Select files	
GSDDDNAM	DVR	DVR	DVR	DVR	DDNAMEs corresponding to files	
GSDCID	Exit	DVR	DVR	DVR	Column entry ID	
GSDCNAME	Exit	DVR	DVR	DVR	Name of column	3
GSDCTYPE	Exit	DVR	DVR	DVR	Type of column	3
GSDCLEN	Exit	DVR	DVR	DVR	Maximum length of column	3
GSDFSCAL	Exit	DVR	DVR	DVR	Scale of column	3
GSDCSPOS	–	DVR	DVR	DVR	Start position of column	2,3
GSDCNULP	Exit	DVR	DVR	DVR	Null column indicator	3
GSDCNULO	–	–	Exit	3	Null column occurrence indicator	3
GSDCDESC	Exit	DVR	DVR	DVR	Description of column	3
GSDCVLNO	–	–	Exit	–	Length of variable-length column occurrence	3
GSDCDTCV	Exit	DVR	DVR	DVR	Name of GDI exit routine	3
GSDCSAVE	Exit	Exit	Exit	Exit	Exit-maintained information about column	3

Notes:

1. When DataRefresher issues a DESCRIBE call to the GDI Select exit routine, DataRefresher passes, using GSDDTLTO, the number of entries for which there is room in the Data Description control block. If this number is insufficient, the exit routine must indicate a return code of 4 and place the number of entries required in GSDDTLUS. DataRefresher then reinitializes the Data Description control block with the required number of entries and reissues the DESCRIBE call.
2. After the DESCRIBE call, DataRefresher calculates the necessary length of the FETCH buffer (the data area into which the GDI exit routine moves column values on FETCH calls) and places the result in GSDBUFLN. After DataRefresher obtains storage for the FETCH buffer, DataRefresher calculates the start position for each column and places the results in GSDCSPOS.
3. DataRefresher initializes all character fields for column entries to blank (' ') and all numeric fields for column entries to zero (0).

Table 29 contains the exit call types to the DVRXCGRD Data Description control block for GDI Record exit routines.

Table 29 (Page 1 of 2). Exit call types to the DVRXCGRD Data Description control block for GDI Record exit routines

Field	Describe	Open	FETCH	Close	Repositn	Field Description	Notes
GRDCBNAM	DVR	DVR	DVR	DVR	DVR	Name of control block	
GRDCBLEN	DVR	DVR	DVR	DVR	DVR	Length of control block	
GRDDTLTO	DVR	DVR	DVR	DVR	DVR	Number of entries available	1
GRDDTLUS	Exit	DVR	DVR	DVR	DVR	Number of entries needed	1
GRDBUFLN	–	–	DVR	–	–	Length of FETCH buffer	2
GRDFILNM	DVR	DVR	DVR	DVR	DVR	Name of GDI Record file	
GRDMAXLN	–	Exit	DVR	DVR	DVR	Maximum record length	4
GRDRECLN	–	–	Exit	–	–	Length of record occurrence	5
GRDKEYIX	–	DVR	DVR	DVR	DVR	Index to key field	
GRDRECFM	–	Exit	DVR	DVR	DVR	Format of record	3,4
GRDDDNAM	DVR	DVR	DVR	DVR	DVR	DDNAMEs corresponding to file	
GRDSID	Exit	DVR	DVR	DVR	DVR	Segment entry ID	6
GRDFLSAV	Exit	Exit	Exit	Exit	Exit	Exit-maintained information about file	
GRDSNAME	Exit	DVR	DVR	DVR	DVR	Name of segment	7
GRDSFMAT	Exit	DVR	DVR	DVR	DVR	Format of the segment	7
GRDSPRNT	Exit	DVR	DVR	DVR	DVR	Name of parent segment	7
GRDSFREQ	Exit	DVR	DVR	DVR	DVR	Frequency of segment in file or parent	7
GRDSIBYT	Exit	DVR	DVR	DVR	DVR	Length of internal segment	7
GRDSOCCN	Exit	DVR	DVR	DVR	DVR	Number of internal segment occurrences	7
GRDSOCCF	Exit	DVR	DVR	DVR	DVR	Count field of internal segment	7
GRDSSPOS	Exit	DVR	DVR	DVR	DVR	Start position of internal segment	7
GRDSSVPN	Exit	DVR	DVR	DVR	DVR	Start base of internal segment	7
GRDSSVPO	Exit	DVR	DVR	DVR	DVR	Start offset of internal segment	7
GRDSNVPN	Exit	DVR	DVR	DVR	DVR	Base of next occurrence of internal segment	7
GRDSNVPO	Exit	DVR	DVR	DVR	DVR	Offset of next occurrence of internal segment	7
GRDSDESC	Exit	DVR	DVR	DVR	DVR	Description of segment	7
GRDSSAVE	Exit	Exit	Exit	Exit	Exit	Exit-maintained information about segment	
GRDFID	Exit	DVR	DVR	DVR	DVR	Field entry ID	
GRDFNAME	Exit	DVR	DVR	DVR	DVR	Name of field	
GRDFTYPE	Exit	DVR	DVR	DVR	DVR	Type of field	
GRDFSFLD	Exit	DVR	DVR	DVR	DVR	Field sequence indicator	
GRDFSEQU	Exit	DVR	DVR	DVR	DVR	Sequence indicator for sequence field	
GRDFUNIQ	Exit	DVR	DVR	DVR	DVR	Uniqueness indicator for sequence field	
GRDFLEN	Exit	DVR	DVR	DVR	DVR	Maximum length of field	
GRDFSCAL	Exit	DVR	DVR	DVR	DVR	Scale of field	
GRDFSPOS	Exit	DVR	DVR	DVR	DVR	Start position of fixed field	
GRDFSVPN	Exit	DVR	DVR	DVR	DVR	Base field for variable-position field	
GRDFSVPO	Exit	DVR	DVR	DVR	DVR	Offset for variable-position field	

Table 29 (Page 2 of 2). Exit call types to the DVRXCGRD Data Description control block for GDI Record exit routines

Field	Describe	Open	FETCH	Close	Repositn	Field Description	Notes
GRDFLFLD	Exit	DVR	DVR	DVR	DVR	Name of size field for variable-length field	8
GRDFDESC	Exit	DVR	DVR	DVR	DVR	Description of field	
GRDCDTCV	Exit	DVR	DVR	DVR	DVR	Name of GDI exit routine	
GRDFSAVE	Exit	Exit	Exit	Exit	Exit	Exit-maintained information about field	

Notes:

1. When DataRefresher issues a DESCRIBE call to the GDI exit routine, DataRefresher passes, using GRDDTLTO, the number of entries for which there is room in the Data Description control block. If the number of entries is insufficient, the exit routine shows a return code of 4 and places the number of entries required in GRDDTLUS. DataRefresher then reinitializes the Data Description control block with the required number of entries and reissues the DESCRIBE call.
2. After the OPEN call, DataRefresher calculates the necessary length of the FETCH buffer, places the result in GRDBUFLN, and then obtains storage for the FETCH buffer. DataRefresher checks to ensure that the GRDMAXLN and GRDRECFM fields are set, when returning from an OPEN call. DataRefresher needs the GRDMAXLN value to acquire the FETCH buffer. GRDMAXLN tells DataRefresher the fixed or maximum record length (cannot exceed 512K characters); GRDRECFM tells DataRefresher the record format.
3. On the OPEN call, the GDI exit routine must provide a fixed-length or variable-length record indicator in GRDRECFM. (If a format indicator is not specified, DataRefresher defaults to fixed-length.)
4. DataRefresher expects the maximum record length (GRDMAXLN) and record format (GRDRECFM) to be set when returning from an OPEN call.
5. If the data being returned from a FETCH call is variable-length, GRDRECLN tells DataRefresher the length for each occurrence of a variable-length record.
6. You use segment entries to define internal segments or multiple record types.
7. This field applies to a structured DXTFILE—one with multiple record types or internal segments.
8. GRDFLFLD holds the name of a field that contains the length of a variable-length data type. The field whose name is in GRDFLFLD is required if variable-length data types are being returned by a GDI Record exit routine. This field, containing the length of a variable-length character or variable-length graphic field, will be part of the data record returned to DataRefresher.

Writing GDI exit routines

You need to evaluate the following before writing GDI exit routines:

- Using sample GDI exit routines
- User work areas
- User parameters
- The execution environment
- The DEM extract request

Using sample GDI exit routines

Included with DataRefresher are sample GOI exit routines in PL/I, COBOL and Assembler. These sample exit routines are contained in the DVR110.DVRINTER library. You can copy or print the sample exit routines and modify them to perform the functions you need. These sample GDI exit routines are:

DVRXYHIX PL/I GDI Record exit routine that extracts from an IXF file
DVRXOGSX COBOL GDI Select exit routine that extracts data from DB2
DVRXOGRX COBOL GDI Record exit routine that extracts data from DB2
DVRXAGSX Assembler GDI Select exit routine that extracts data from DB2

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

See "GDI exit routine examples" on page 103 for some examples of how GDI exit routines can be used.

User work areas

Work space areas for your use are contained in the control blocks and in the global work area.

Work Space in the control blocks

Each control block contains work space that you can use for working storage. For examples of how to use this work space, look at the sample GDI exit routines in the DVR110.DVRINTER library.

Because DataRefresher does not reinitialize the work space after a GDI exit routine is called for the first time, you can use the GSCWORK (GDI Select exit routine) and GRCWORK (GDI Record exit routine) fields in the Data Communication control blocks to save data or addresses that you want to use during a particular run of DataRefresher.

Note: If you use this work space during a call by the UIM, it is reinitialized on a later call by the DEM.

In the Data Description control blocks, you can use the GSDCSAVE field (GDI Select exit routine) to store information about each column entry, and you can use the GRDFLSAV, GRDSSAVE, and GRDFSAVE fields (GDI Record exit routine) to store information about the source file, each segment entry, and each field entry.

Note: If you use this work space during a call by the UIM, it is reinitialized on a later call by the DEM. For example, if you store file-related information in GRDFLSAV during a DESCRIBE call by the UIM, your GDI Record exit routine cannot use that information during later calls by the DEM.

Global work area

You can use the global work area (the fourth parameter passed to the exit routine) in joining DB2 tables using GDI Record exit routines. When joining DB2 tables, GDI exit routines must be able to associate a unique cursor for each DB2 table to be accessed. Both cursors must be open at the same time for DataRefresher to perform the join. One GDI exit routine could potentially use the global area to communicate with another GDI exit routine, as well as keep track of which cursor (or file identifier) is being used for each table (or file) being accessed. See the sample GDI exit routines for examples of PL/I and COBOL GDI record exits that use the global work area in joining IXF files and in joining up to four DB2 tables.

User parameters

Through the UIM JCL EXEC statement or the INITDEM command, the GDI value lets you pass parameters through DataRefresher to your GDI exit routine. For example, the GDI exit routine might need security information, or it might require certain processing options to access the source data.

The parameters are stored in the GSCPARM (GDI Select exit routine) or GRCPARM (GDI Record exit routine) field in the Data Communication control

block. You can use up to 94 characters for GDI user parameters. In the UIM, other EXEC statement PARM keywords come out of these 94 bytes as well.

The execution environment

Neither the UIM nor the DEM is required to be the program that the job scheduler runs to begin a task (or job-step program). Nevertheless, the following requirements on register 1 must be met when the UIM or DEM receives control:

- For the UIM, register 1 is either zero-filled or addresses a pointer to parameters specified on the PARM value of the JCL EXEC statement.
- For the DEM, if access to IMS is required, register 1 addresses a list of pointers to PCBs. If access to IMS is *not* required, register 1 addresses a pointer to `'NONDLI'`.

Note: If you are running in a file or database environment in which another program must be the job-step program, you may have to write a special routine that, when called by the mandatory job-step program, passes required parameters to DataRefresher.

When setting up the programming language environment, DataRefresher assumes that your special routine is written in Assembler. It is not possible to specify another language for these routines. Because these special routines manipulate registers, it is recommended that they be written in Assembler.

As shown in Figure 61, the mandatory job-step program calls the special routine. The routine passes the required parameters to the DEM.

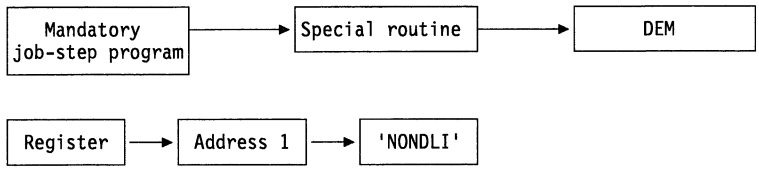

Figure 61. Mandatory job-step program

"Example 3. Calling non-reusable database management systems (DBMSs)" on page 120 contains information on how to handle changes in the normal operating environment of DataRefresher.

The DEM extract request

To run an extract that calls a GDI exit routine, you must ensure that your DEM extract request specifies what data you require from your GDI sources.

Building and submitting the DEM extract request

To produce a DEM extract request, you must code the SUBMIT command, followed by the EXTRACT command. You use values on the SUBMIT and EXTRACT commands to describe to DataRefresher what data you want to extract from the GDI data source and where you want to put that extracted data.

For GDI Select exit routines, you code the SELFILE keyword on the SUBMIT command to name the GDI Select file (or files, if joining) and trigger the UIM to pass the full SELECT statement to the GDI exit routine to process.

Two of the EXTRACT command keywords are SELECT and WHERE. With GDI Select exit routines, you can submit an SQL-like SELECT statement to DataRefresher without previously storing field descriptions in the FDTLIB.

The WHERE keyword specifies the conditions for extracting data. If a key or sequence field has been defined for the data, you can pass key or sequence values to a GDI Record exit routine by coding the WHERE keyword.

DataRefresher can pass key values to a GDI Record exit routine with the WHERE keyword of your extract request, if a key field has been defined for the data source. During FETCH and OPEN calls, the exit routine can qualify records or optimize data access using the key field.

Executing the DEM extract request

To give the DEM access to the necessary file descriptions in the FDTLIB, you must specify which files you want the DEM to access—this time as part of the DEM's DXTIN data set. You must code, on the USE DXTFILE command in the DEM's DXTIN data set, the name of each GDI data source you want the DEM to access. You might also have to include, depending on the nature of the data source, one or more DD statements in a DEM's JCL job stream for each GDI data source you want the DEM to access.

Identifying your GDI exit routine to DataRefresher

The DEM needs a description of the GDI exit routine source data you are referencing. Advise DataRefresher of the GDI exit routine and its source data by using the parameters specified in the CREATE DXTFILE command. Figure 62 and Figure 63 contain examples of the CREATE DXTFILE command for GDI Record and Select exit routines:

```
CREATE DXTFILE
  NAME=VSAMDEPT
  DESC='DESCRIBING VSAM FILE WITH GDI EXIT',
  ACCESS=GDI, 1
  GDIEXIT=GDIVSAM, 2
  GDIXTYPE=RECORD, 3
  DETAIL=HERE, 4
  FIELD NAME= DEPT,START= 1, TYPE=H, BYTES= 2, DESC='DEPT NUMBER'
  FIELD NAME= PRODNUM, START= 3, TYPE=H, BYTES= 2, DESC='PRODUCT NO';
```

Figure 62. Sample CREATE DXTFILE command for a GDI Record exit routine

```
CREATE DXTFILE
  NAME=TDEPT,
  ACCESS=GDI, 1
  GDIEXIT=DVRXOGSX, 2
  GDIXTYPE=SELECT; 3
```

Figure 63. Sample CREATE DXTFILE command for a GDI Select exit routine

1 The ACCESS keyword must equal GDI for all GDI exit routines.

2 The name you write as the value on the GDIEXIT keyword must match the load module name of your GDI exit routine. This can also be the entry point name. Load module names are alphameric/special characters names with a maximum

length of eight characters. For information on DataRefresher naming conventions, see the *DataRefresher Command Reference*.

3 The GDIXTYPE keyword specifies whether the data description is for a Select or Record exit routine.

4 The DETAIL keyword is only used for GDI Record exit routines. This keyword tells DataRefresher where to find the detailed field and segment definitions it needs. The value can be either HERE or EXIT. If you write DETAIL=EXIT, DataRefresher will issue a DESCRIBE call to the exit routine to get the description details. Your GDI exit routine must then obtain the information from the data source or an auxiliary to the source data (a data dictionary, for example). When you write DETAIL=HERE, you must supply the information with the FIELD and SEGMENT keywords as needed. Include a FIELD keyword for each field you want to include in the DXTFILE description.

If you want DataRefresher to pass key or sequence values to a GDI Record exit routine from an extract request WHERE keyword, you must code the SEQFLD keyword.

For detailed information on the syntax of the CREATE DXTFILE command, see the *DataRefresher Command Reference*.

Naming tables when using DataRefresher dialogs and GDI Select exit routines

When using DataRefresher dialogs to build extract requests for data to be extracted using a GDI Select exit routine, avoid using special characters in table names. Otherwise, you may have to edit your extract requests before submitting them to the UIM.

For example, if you created a relational table named MY*VIEW under user ID SMITH, the table is known to the database as SMITH."MY*VIEW". You may need to send the extract request to your own user ID, edit it, and resubmit it to DataRefresher. DataRefresher assumes nothing about the use of special characters (the embedded asterisk) in table names. Using an automatically generated DXTVIEW, End User Dialogs produces an extract request containing SELECT and FROM statements in the following format:

```
SELECT
  SMITH.MY*FILE.COL1,
  SMITH.MY*FILE.COL2
FROM
  SMITH.MY*FILE;
```

For the database to interpret the table name properly, you might need to edit the output, enclosing each instance of MY*FILE in quotation marks as shown in the following example:

```
SELECT
  SMITH."MY*FILE".COL1,
  SMITH."MY*FILE".COL2
FROM
  SMITH."MY*FILE";
```

For information on naming conventions, see the *DataRefresher Command Reference*.

Accessing DB2 when using DataRefresher OS/2

When DataRefresher OS/2 processes a GDI exit routine that accesses DB2, DataRefresher connects to DB2 using the Call Attachment Facility (CAF). Therefore, if your GDI exit routine accesses DB2, and you are including your exit routine in extract requests submitted via DataRefresher OS/2, your link-edit JCL will differ slightly from the usual link-edit for the TSO environment.

To access DB2 using CAF, your GDI exit routine must be link-edited with the Call Attachment Language Interface (DSNALI). The SQL statements are coded as usual in your exit routine. Figure 64 provides sample JCL for link-editing the sample GDI Record exit routine DVRXOGRX for the CAF environment.

```
//*-----------------------------------------------------------------
//*       LINK-EDIT THE GDI RECORD EXIT FOR THE CAF ENVIRONMENT
//*-----------------------------------------------------------------
//LKEDTSO EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP,AMODE=NN,RMODE=ANY',
//             COND=(4,LT,COMPILE)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.COB2LIB
//         DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DXT.EXITLIB(DVRXOGRX)
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DISP=(OLD,PASS),DSN=&&OBJECT 1
//         DD  *
 INCLUDE SYSLIB(DSNALI) 2
 ENTRY   DVRXOGRX
 NAME    DVRXOGRX(R)
```

Figure 64. JCL to link-edit a GDI Record exit routine for the CAF environment

The link editor creates a load module for the GDI Record exit routine to run in the CAF environment. Input to this step consists of:

1. The object module for the GDI Record exit routine (&&OBJECT)
2. The Call Attachment Language Interface module (DSNALI)

In order to be called by the DEM, the exit routine must be link-edited as normal. For sample JCL to link-edit a GDI exit routine, see "GDI exit routine examples" on page 103.

GDI exit routine return codes

Your GDI exit routines communicate with DataRefresher via return codes placed in the Data Communication control block. Return codes are placed in the GSCRTNCD field for GDI Select exit routines and in the GRCRTNCD field for GDI Record exit routines.

Table 30 contains an explanation of the meaning of each return code.

Table 30 (Page 1 of 2). GDI exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
4	End of data reached (on FETCH call); not enough room for field/column entries (on DESCRIBE call)
8	Error; terminate function or extract request
12	Error; terminate DataRefresher

Table 30 (Page 2 of 2). GDI exit routine return codes	
Return Code	**Meaning**
16	Error; terminate DataRefresher
Other	Error; terminate DataRefresher

If your GDI exit routine indicates a return code of 8, 12, or 16 after a FETCH or REPOSITN call, DataRefresher immediately issues a CLOSE call to clean up the remnants of processing. However, if your GDI exit routine indicates a return code of 8, 12, or 16 after another type of call, DataRefresher expects the exit routine to do the cleanup.

Although DataRefresher responds to return codes 12 and 16 in the same way, your GDI exit routine could use these two codes to rank the severity of errors and then set different (error) reason codes.

DataRefresher can also communicate with the GDI exit routine via optional reason codes and messages that you define and the GDI exit routine writes in the Data Communication control block. DataRefresher inserts the codes and messages returned by the GDI exit routine into the messages that DataRefresher issues.

GDI exit routine examples

This section shows you, through detailed examples, how to use GDI exit routines in the following situations:

- Joining DB2 and IMS data
- Extracting DB2 data
- Invoking non-reusable database management systems (DBMSs)

Example 1. Joining DB2 and IMS data

To join data from a DB2 table with data from an IMS database, you must meet the following requirements:

- Use a GDI Record exit routine to access the DB2 data. A sample GDI Record exit routine written in VS COBOL II, DVRXOGRX, accesses DB2 data. You can copy this sample from the DVR110.DVRINTER library.
- Link-edit the UIM and DEM with the appropriate language-dependent modules, so DataRefresher can establish the proper execution environment. The example that follows assumes that you have link-edited the UIM and DEM with the appropriate COBOL modules and that the resulting load modules are contained in DVR110.DVRLOAD. See "Preparing the language load module for OS PL/I and VS COBOL II" on page 17 for more information on this process.
- Install and invoke the DB2-IMS Batch Support Program to access DB2 and IMS data. See *IBM DATABASE 2 Version 2: Application Programming and SQL Guide* for information about this program.

This example shows you how to meet those requirements. It also includes sample JCL for performing required tasks.

Task overview

To join DB2 and IMS data:

1. Prepare to run the GDI record exit routine in the:
   - TSO batch environment
   - IMS environment
2. Create and process the data descriptions for the:
   - DB2 data
   - IMS data
3. Build and submit the DEM extract request
4. Execute the DEM extract request

The following discusses each of the steps in detail.

Step 1. Preparing to run the GDI Record exit routine

The GDI Record exit routine for accessing DB2 data is a DB2 application. Before running this DB2 application:

1. Precompile the GDI Record exit routine as a DB2 application
2. Compile the modified source code
3. Link-edit the generated object module for the TSO environment
4. Link-edit the generated object module for the IMS environment
5. Bind the application program to create a DB2 plan
6. Grant authority to use the DB2 plan

1. Precompile the exit routine as a DB2 application

Figure 65 provides sample JCL for precompiling the GDI Record exit routine as a DB2 application:

```
//*-------------------------------------------------------------------
//*  PREPARE THE GDI RECORD EXIT AS A DB2 APPLICATION
//*-------------------------------------------------------------------
//PREPARE  JOB   MSGLEVEL=(1,1),CLASS=A,USER=userid,PASSWORD=password
//*-------------------------------------------------------------------
//*       PRECOMPILE THE SOURCE CODE FOR THE GDI RECORD EXIT
//*-------------------------------------------------------------------
//PRECOMP EXEC PGM=DSNHPC,PARM='HOST(COB2) XREF SOURCE',REGION=2048K
//STEPLIB  DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//DBRMLIB  DD  DISP=SHR,DSN=DVR110.DBRMLIB(DVRXOGRX) 1
//SYSLIN   DD  DISP=(MOD,PASS),DSN=&&MODSRC 2 ,UNIT=SYSDA,
//             SPACE=(TRK,(3,3))
//SYSPRINT DD  SYSOUT=A
//SYSTERM  DD  DUMMY,DCB=BLKSIZE=800
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSIN    DD  DISP=SHR,DSN=DVR110.DVRINTER(DVRXOGRX)
/*
```

Figure 65. JCL to precompile a GDI Record exit routine as a DB2 application

The DB2 precompiler processes the source code for the GDI Record exit routine contained in the DVR110.DVRINTER library (DVRXOGRX) and produces:

1 A DB2 database request module (DBRM) stored in DBRMLIB (DVRXOGRX) for use in the BIND step to create a DB2 plan

2 Modified source code (&&MODSRC) for use in the COMPILE step

2. **Compile the modified source code**

   Figure 66 provides sample JCL for compiling the GDI Record exit routine.

```
//*-------------------------------------------------------------------
//*       COMPILE THE MODIFIED SOURCE CODE FOR THE GDI RECORD EXIT
//*-------------------------------------------------------------------
//COMPILE EXEC PGM=IGYCRCTL,PARM='OBJECT,LIST,MAP,RES,SOURCE',
//             REGION=1024K,COND=(4,LT,PRECOMP)
//STEPLIB  DD  DISP=SHR,DSN=SYS1.COB2LIB
//         DD  DISP=SHR,DSN=SYS1.COB2MAC
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DISP=(MOD,PASS),DSN=&&OBJECT,UNIT=SYSDA, 3
//             SPACE=(TRK,(3,3))
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD  DISP=(OLD,DELETE),DSN=&&MODSRC
/*
```

Figure 66. JCL to compile the GDI Record exit routine

3 The VS COBOL II compiler processes the modified source code (&&MODSRC) for the GDI Record exit routine and produces an object module (&&OBJECT) for use in the link-edit steps (LKEDTSO and LKEDIMS) to create an executable load module.

3. **Link-edit the exit routine for TSO**

   Figure 67 provides sample JCL for link-editing the GDI Record exit routine for the TSO environment.

```
//*-------------------------------------------------------------------
//*       LINK-EDIT THE GDI RECORD EXIT FOR THE TSO ENVIRONMENT
//*-------------------------------------------------------------------
//LKEDTSO EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP,AMODE=NN,RMODE=ANY',
//             COND=(4,LT,COMPILE)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.COB2LIB
//         DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DXT.EXITLIB(DVRXOGRX)
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DISP=(OLD,PASS),DSN=&&OBJECT 4
//         DD  *
 INCLUDE SYSLIB(DSNELI) 5
 ENTRY   DVRXOGRX
 NAME    DVRXOGRX(R)
```

Figure 67. JCL to link-edit the GDI Record exit routine for the TSO environment

The link editor creates a load module for the GDI Record exit routine stored in DVR110.DXT.EXITLIB (DVRXOGRX) to run in the TSO batch environment. Input to this step consists of:

4 The object module for the GDI Record exit routine (&&OBJECT)
5 The TSO language interface module (DSNELI)

There are special RMODE considerations that you should know about. For more information on RMODE dependencies, see "AMODE and RMODE dependencies" on page 23.

4. **Link-edit the exit routine for IMS**

   Figure 68 provides sample JCL for link-editing the GDI Record exit for the IMS environment.

```
//*-------------------------------------------------------------------
//*       LINK-EDIT THE GDI RECORD EXIT FOR THE IMS ENVIRONMENT
//*-------------------------------------------------------------------
//LKEDIMS EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP,AMODE=NN,RMODE=ANY',
//             COND=(4,LT,COMPILE)
//SYSLIB   DD  DISP=SHR,DSN=SYS1.COB2LIB
//         DD  DISP=SHR,DSN=PRDUCT.IMSVS.RESLIB
//         DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.DXTIMS.EXITLIB(DVRXOGRX)
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DISP=(OLD,DELETE),DSN=&&OBJECT 6
//         DD  *
 INCLUDE SYSLIB(DFSLI000) 7
 ENTRY   DVRXOGRX
 NAME    DVRXOGRX(R)
/*
```

Figure 68. JCL to link-edit the GDI Record exit routine for the IMS environment

   The link editor creates a load module for the GDI Record exit routine stored in DVR110.DXTIMS.EXITLIB (DVRXOGRX) to execute in the IMS environment. Input to this step consists of:

   6 The object module for the GDI Record exit routine (&&OBJECT)
   7 The IMS language interface module (DFSLI000)

   The member name of the GDI Record exit routine load module (DVRXOGRX) must be the same in the TSO batch and IMS environments to execute DataRefresher and the GDI Record exit routine.

   There are special RMODE considerations that you should know about. For more information on RMODE dependencies, see "AMODE and RMODE dependencies" on page 23.

5. **Bind the application to create a DB2 plan**

   Figure 69 provides sample JCL for binding the GDI Record exit routine as a DB2 application.:

```
//*-------------------------------------------------------------------
//*       BIND THE GDI RECORD EXIT TO CREATE A DB2 PLAN
//*-------------------------------------------------------------------
//BIND    EXEC PGM=IKJEFT01,REGION=2048K
//STEPLIB  DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//SYSPRINT DD  SYSOUT=A
//SYSTSPRT DD  SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSTSIN  DD *
DSN   SYSTEM  (DS3)
BIND  PLAN    (GDIPLAN)  -
      MEMBER  (DVRXOGRX) 8 ACTION(REPLACE) RETAIN ISOLATION(CS) -
      LIBRARY ('DVR110.DBRMLIB')
END
/*
```

Figure 69. JCL to bind a GDI Record exit routine as a DB2 application

   The DB2 bind process creates a DB2 plan (GDIPLAN) for the exit routine.

8 Input to this step is the database request module, MEMBER (DVRXOGRX), stored in LIBRARY ('DVR110.DBRMLIB').

This DB2 plan is specified in DataRefresher jobs that invoke the GDI Record exit routine in the TSO batch and IMS environments.

6. **Grant authority to use the DB2 plan**

   Figure 70 provides sample JCL for granting authority to use the DB2 plan..

```
//*-------------------------------------------------------------------
//*       GRANT AUTHORITY TO USE THE DB2 PLAN
//*-------------------------------------------------------------------
//GRANT   EXEC PGM=IKJEFT01,REGION=2048K
//STEPLIB  DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//SYSTSPRT DD  SYSOUT=A,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSTSIN  DD  *
 DSN SYSTEM  (DS3)
 RUN PROGRAM (DSNTIAD)                -
     PLAN    (DSNTIA13)               -
     LIBRARY ('PRDUCT.DSN220.RUNLIB.LOAD')
 END
//SYSIN    DD  *
GRANT EXECUTE ON PLAN GDIPLAN TO PUBLIC; 9
COMMIT;
/*
```

Figure 70. JCL to grant authority to use the DB2 plan

9 This step grants PUBLIC authority to use the DB2 plan for the GDI Record exit routine. Input to this step is the DB2 plan name (GDIPLAN).

Both the bind and grant steps require that DB2 be up before you run this job. To isolate the processing that requires DB2 being up, you can create a separate job for these two steps.

Step 2. Creating and processing the data descriptions

The UIM is called to verify data descriptions for the DB2 and IMS data and to store the data descriptions in the FDTLIB. To connect to DB2 and run the GDI Record exit routine for obtaining the DB2 data descriptions, the UIM is called indirectly via the TSO batch facility. This means that the TSO terminal monitor program (TMP) is called in batch and, in turn, passes control to the UIM. The DB2 and IMS data can be described to DataRefresher during the same call to the UIM.

For the DB2 data, the CREATE DXTFILE and CREATE DXTVIEW commands must be specified as input to the UIM. For the IMS data, the CREATE DXTPSB and CREATE DXTVIEW commands must be specified as input to the UIM.

DB2 must be up before you run this job, so the GDI Record exit routine can obtain the DB2 data descriptions.

The UIM need only be invoked by the TMP to process the CREATE DXTFILE command. Therefore, the UIM can be invoked directly (by the job step) to process the CREATE DXTPSB and CREATE DXTVIEW commands. Figure 71 on page 108 provides sample JCL for describing DB2 and IMS data to DataRefresher:

```
//*-----------------------------------------------------------------
//*  DESCRIBE THE DB2 AND IMS DATA TO DATAREFRESHER
//*-----------------------------------------------------------------
//DESCRIBE JOB  MSGLEVEL=(1,1),USER=userid,PASSWORD=password
//*-----------------------------------------------------------------
//*  RUN DATAREFRESHER'S USER INPUT MANAGER (UIM) TO ADD FDTLIB DATA
//*  DESCRIPTIONS FOR BOTH DB2 AND IMS DATA
//*-----------------------------------------------------------------
//UIM      EXEC PGM=IKJEFT01 1 ,REGION=2048K
//STEPLIB  DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD 2
//         DD  DISP=SHR,DSN=DXT.EXITLIB
//         DD  DISP=SHR,DSN=SYS1.COB2LIB
//SYSTSPRT DD  SYSOUT=A,DCB=BLKSIZE=121
//DSNTRACE DD  SYSOUT=A
//DXTPRINT DD  SYSOUT=A
//DXTPUNCH DD  SYSOUT=A
//DXTOUT   DD  SYSOUT=A
//DXTDUMP  DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSABEND DD  SYSOUT=A
//SYSABOUT DD  SYSOUT=A
//FDTLIB   DD  DISP=SHR,DSN=DXT.FDTLIB
//EXTLIB   DD  DISP=SHR,DSN=DXT.EXTLIB
//SYSTSIN  DD  *
 DSN SYSTEM  (DS3)
 RUN PROGRAM (DVRU0000)                -
     PLAN    (GDIPLAN)                 -
     LIBRARY ('DVR110.DVRLOAD') -
     PARM    ('EXITLANG=COBOL')
 END
//DXTIN    DD DATA,DLM=$$ 3
/*-----------------------------------------------------------------*/
/* DXTFILE DEFINITION USED TO ACCESS DB2 DATA VIA A GDI RECORD     */
/* EXIT. THE DXTFILE NAME MUST MATCH THE NAME OF THE DB2 TABLE.    */
/*-----------------------------------------------------------------*/
CREATE
  DXTFILE NAME    = "DSN82.TEMPL",
          DESC    = 'ACCESS DB2 VIA GDI RECORD EXIT',
          ACCESS  = GDI,
          GDIEXIT = DVRXOGRX,
          GDIXTYPE= RECORD,
          DETAIL  = EXIT
      ;
/*-----------------------------------------------------------------*/
/* DXTVIEW DEFINITION FOR ACCESSING DB2 DATA VIA A GDI RECORD EXIT */
/*-----------------------------------------------------------------*/
CREATE
  DXTVIEW NAME    = "DSN82.TEMPL_VIEW",
          DXTFILE = "DSN82.TEMPL",
          DESC    = 'VIEW OF DB2 DATA',
          FIELDS  = *
      ;
/*-----------------------------------------------------------------*/
/*           DXTPSB DEFINITION OF AN IMS DATABASE                  */
/*-----------------------------------------------------------------*/
CREATE
  DXTPSB      NAME = DXTIMSDB
    DXTPCB    NAME = SALESDN, DBACCESS=HIDAM,
                     DESC='PCB FOR DXTSALES'
   .../* THE DETAILS OF THIS DEFINITION ARE NOT IMPORTANT TO THIS
   ...   EXAMPLE */
/*-----------------------------------------------------------------*/
/*      DXTVIEW DEFINITION OF PATH OVER IMS DATABASE               */
/*-----------------------------------------------------------------*/
CREATE
  DXTVIEW NAME    = STAFF_VIEW,
          DXTPSB  = DXTIMSDB,
          DXTPCB  = SALESDN,
          FIELDS  = *,
          SEGMENT = STAFF,
          MINSEGM = STAFF
          ;
```

Figure 71. JCL for describing DB2 and IMS data to DataRefresher

1 The job step program to be executed is the TMP (IKJEFT01). Input to the TMP consists of:

- Name of the program to be executed, PROGRAM(DVRU0000), stored in LIBRARY('DVR110.DVRLOAD')
- Input for the UIM (EXITLANG=COBOL) to tell DataRefresher that the COBOL environment must be set up for the GDI Record exit routine
- Name of the DB2 plan (GDIPLAN) for the GDI Record exit routine

2 The STEPLIB DD statements reference the following data sets:

- PRDUCT.DSN220.DSNLOAD, which contains the DB2 libraries
- DXT.EXITLIB, which contains the GDI Record exit routine that was prepared for running in the TSO batch environment
- SYS1.COB2LIB, which contains the VS COBOL II libraries

3 Other input to the UIM is specified in the DXTIN DD statement, consisting of:

- CREATE DXTFILE command for describing the DB2 data—the user information specified on the job card must match the authorization for the DB2 table or view specified on this command
- CREATE DXTVIEW commands for describing views of the DB2 and IMS data
- CREATE DXTPSB command for describing the IMS data

Step 3. Building and submitting the DEM extract request

The UIM is invoked to verify the extract request for joining DB2 and IMS data and to store this extract request in the EXTLIB. The SUBMIT command, consisting of both the SUBMIT command and the EXTRACT statements, must be specified as input to the UIM. The UIM is invoked directly since the GDI Record exit routine is not required to validate and store the extract request. Figure 72 on page 110 provides sample JCL for building and submitting the DEM extract request.

```
//*-------------------------------------------------------------------
//*  SUBMIT THE EXTRACT REQUEST FOR JOINING DB2 AND IMS DATA
//*-------------------------------------------------------------------
//SUBMIT   JOB  MSGLEVEL=(1,1),USER=userid,PASSWORD=password
//*-------------------------------------------------------------------
//*  RUN DATAREFRESHER'S USER INPUT MANAGER (UIM) TO
//*  VALIDATE AN EXTRACT REQUEST FOR JOINING DB2 AND IMS DATA, AND
//*  STORE THE EXTRACT REQUEST IN THE EXTLIB
//*-------------------------------------------------------------------
//UIM     EXEC PGM=DVRU0000,REGION=1200K
//STEPLIB  DD  DISP=SHR,DSN=DVR110.DVRLOAD 1
//SYSABEND DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//FDTLIB   DD  DISP=SHR,DSN=DXT.FDTLIB
//EXTLIB   DD  DISP=SHR,DSN=DXT.EXTLIB
//DXTPRINT DD  SYSOUT=A
//DXTDUMP  DD  SYSOUT=A
//DXTOUT   DD  SYSOUT=A
//DXTPUNCH DD  SYSOUT=A
//JCSDD    DD  *,DLM=$$
*CD
*EO
$$
//DXTIN    DD  DATA,DLM=$$ 2
SUBMIT   EXTID=EXTRACT1, NODE=STLVMA, USERID=TESTER,
         DBS=IXF, CD=JCS, JCS=JCSDD 3
EXTRACT
         SELECT *
         FROM   "DSN82.TEMPL_VIEW",
                STAFF_VIEW
         WHERE  ENAME = LASTNAME 4
         ;
$$
/*
```

Figure 72. JCL for building and submitting the DEM extract request

1 The STEPLIB DD statement references the data set DVR110.DVRLOAD which contains the UIM (DVRU0000).

2 Input to the UIM is specified in the DXTIN DD statement. The join condition for extracting data is a match of the employee name fields.

3 The target destination for the extracted data is an output job (JCS=JCSDD). The extracted data is formatted as an IXF file (DBS=IXF).

4 LASTNAME is in the DB2 table; ENAME is in the IMS database.

Step 4. Executing the DEM extract request

To execute the DEM extract request, the DEM must be invoked as follows:

- The IMS Region Controller (DFSRC00) must be invoked as the top task in the job stream.
- The DSNMTV01 program must be the specified IMS application program. This is the DB2-IMS Batch Support Program that lets DataRefresher join DB2 and IMS data.
- Input to the DSNMTV01 program must specify the DEM as the real application program to be loaded and run.
- The INITDEM, USE DXTFILE, and USE DXTPSB commands must be specified as input to the DEM.

Figure 73 on page 111 provides sample JCL for executing the DEM extract request.

```
//*-------------------------------------------------------------------
//*  EXECUTE THE EXTRACT REQUEST FOR JOINING DB2 AND IMS DATA
//*-------------------------------------------------------------------
//EXTRACT  JOB MSGLEVEL=(1,1),USER=userid,PASSWORD=password  1
//*-------------------------------------------------------------------
//*  RUN DATAREFRESHER'S DATA EXTRACT MANAGER (DEM) TO
//*  EXECUTE THE EXTRACT REQUEST FOR JOINING DB2 AND IMS DATA
//* ------------------------------------------------------------------
//DEM      EXEC DLIBATCH,MBR=DSNMTV01,PSB=DXTALL 2
//G.STEPLIB DD  DISP=SHR,DSN=IMSVS.RESLIB 3
//          DD  DISP=SHR,DSN=DXTIMS.EXITLIB
//          DD  DISP=SHR,DSN=DVR110.DVRLOAD
//          DD  DISP=SHR,DSN=PRDUCT.DSN220.DSNLOAD
//          DD  DISP=SHR,DSN=SYS1.COB2LIB
//PRINTDD   DD  SYSOUT=A
//DFSVSAMP  DD  *
4096,6
4096,6
//SYSPRINT  DD  SYSOUT=A
//SYSABOUT  DD  SYSOUT=A
//SYSABEND  DD  SYSOUT=A
//SYSUDUMP  DD  SYSOUT=A
//FDTLIB    DD  DISP=SHR,DSN=DXT.FDTLIB
//EXTLIB    DD  DISP=SHR,DSN=DXT.EXTLIB
//DXTPRINT  DD  SYSOUT=A
//DXTDUMP   DD  SYSOUT=A
//DXTPUNCH  DD  SYSOUT=A
//*-------------------------------------------------------------------
//*  ADD THE DD STATEMENTS FOR THE IMS DATABASE HERE
//* ------------------------------------------------------------------
//DXTOUT1   DD  SYSOUT=A
//G.DDOTV02 DD  DISP=(NEW,PASS,DELETE),DSN=&TEMP,
//             SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
//             DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD  * 4
 DS3,SYS1,DSNMIN10,,R,,DEM,GDIPLAN,DVRX0000
/*
//DXTIN     DD  DATA,DLM=$$ 5
INITDEM NAME=DEMRUN, EXITLANG=COBOL;
USE DXTFILE = "DSN82.TEMPL" ;
USE DXTPSB  = DXTIMSDB ;
$$
/*
//* ------------------------------------------------------------------
//* PRINT - ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET
//* ------------------------------------------------------------------
//PRINT    EXEC PGM=DFSERA10
//STEPLIB   DD  DISP=SHR,DSN=IMSVS.RESLIB
//SYSPRINT  DD  SYSOUT=A
//SYSUT1    DD  DISP=(OLD,DELETE),DSN=&TEMP1
CONTROL  CNTL K=000,H=80000
OPTION   PRINT
/*
//
```

Figure 73. JCL for executing the DEM extract request

1 The user and password that have authority to access the DB2 table are specified on the job card. Otherwise, the IMS PSB is used as the DB2 table qualifier.

2 The DEM step specifies the IMS procedure (DLIBATCH) to be invoked and the PSB name (DXTALL) for the IMS data to be extracted. The PSB name specified here is the name used in the IMS PSB Generation statements.

3 The STEPLIB DD statements reference the following data sets.

Data set	Contents
IMSVS.RESLIB	IMS libraries
DXTIMS.EXITLIB	GDI Record exit routine that was prepared for running in the IMS environment
DVR110.DVRLOAD	DEM (DVRX0000)
PRDUCT.DSN220.DSNLOAD	DB2 libraries
SYS1.COB2LIB	VS COBOL II libraries

4 Input to the DB2-IMS Batch Support Program is specified in the DDITV02 DD statement, which consists of these positional fields:

DS3	The name of the DB2 subsystem.
SYS1	The language interface token for routing SQL calls when operating in the on-line IMS environment. As a batch application can only connect to one DB2 system, you can omit this parameter. 'SYS1' is the recommended value.
DSNMIN10	The name of the DB2 initialization module.
(omitted)	The next parameter is omitted, because a resource translation table will not be used.
R	Indicates that an SQL return code should be returned to the application program, if DB2 is not operational or the plan is not available.
(omitted)	The next parameter is omitted since DB2 commands are not supported in this environment.
DEM	The connection name which should be unique for each job step that accesses DB2.
GDIPLAN	The name of the DB2 plan created by the DB2 bind process.
DVRX0000	The name of the application program (the DataRefresher DEM) to be loaded and given control.

5 Input to the DEM is specified in the DXTIN DD statement. The input data sources are identified in the USE statements. The INITDEM statement tells DataRefresher that the COBOL environment (EXITLANG=COBOL) must be set up to run the GDI Record exit routine.

Example 2. Extracting DB2 data

This example explains how to prepare and use GDI exit routines written in COBOL. Two sample COBOL GDI exit routines (DVRXOGSX and DVRXOGRX) are supplied to access DB2 data. These exit routines have been generalized to allow extracting from any type of DB2 table. DVRXOGSX is a GDI Select exit routine (which means you can specify any SQL SELECT acceptable to DB2 in your extract request).

DVRXOGRX is a GDI Record exit routine (which means you can join the data extracted with DVRXOGRX with any other DEM data, except for data from GDI SELECT exits).

Details about preparing and running DB2 Application programs can be found in *IBM DATABASE 2 Version 2: Application Programming and SQL Guide*.

Using the DVRXOGRX GDI Record exit routine

The source of this sample COBOL GDI exit routine is contained as member DVRXOGRX in the DVR110.DVRINTER library. You can use DVRXOGRX to extract data from a DB2 table, and select any fields within the record returned to DataRefresher by DVRXOGRX. You can also join the data returned from DVRXOGRX with any other DEM data (except data from GDI Select exit routines).

If the DB2 table which DVRXOGRX is extracting from contains a variable length entry, then DVRXOGRX adds an extra field just before the variable length field in the record returned to DataRefresher. This halfword field holds the length of the variable length field (variable graphic or variable character).

For GDI Record exit routines, the DXTFILE name should be the name of the DB2 table the exit routine is extracting from. For example, in Figure 74, DVRXOGRX extracts data from the DB2 table named TDEPT.

```
CREATE
   DXTFILE NAME      = TDEPT,
           ACCESS    = GDI,
           GDIEXIT   = DVRXOGRX,
           GDIXTYPE  = RECORD,
           DETAIL    = EXIT;
```

Figure 74. DXTFILE definition

Task overview

To extract data from DB2 using a COBOL GDI exit routine:

1. Precompile the GDI exit routine
2. Bind the GDI exit routine to the DB2 database
3. Compile and link-edit the exit routine
4. Link the UIM and DEM to the COBOL language module
5. Call the GDI exit routine

Step 1. Precompiling a COBOL GDI exit routine for DB2

Before a DB2 COBOL application program can be compiled, the SQL statements embedded in the program must be prepared by the DB2 precompiler. The precompile step validates the SQL statements, comments them out so they are acceptable to the COBOL compiler, and creates a Database Request Module (DBRM) for use by DB2 during the bind process.

The precompile returns a PRINT file and a PUNCH file. The PUNCH file is used in the COB2 compile step. Figure 75 on page 114 provides sample JCL for performing a precompile.

```
//* THIS IS THE JCL TO PERFORM A DB2 PRECOMPILE ON A COBOL EXIT
//COBOLSEL JOB   MSGLEVEL=(1,1),CLASS=K
//PC EXEC PGM=DSNHPC,PARM='HOST(COB2) XREF SOURCE',REGION=2048K
//STEPLIB DD DSN=PRDUCT.DSN220.DSNLOAD 1 ,DISP=SHR
//DBRMLIB DD DSN=DVR110.DEM.DBRMLIB(DVRXOGRX) 2 ,DISP=SHR
//SYSCIN DD SYSOUT=B
//SYSPRINT DD SYSOUT=A
//SYSTERM DD DUMMY,DCB=BLKSIZE=800
//SYSUT1 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2 DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSIN DD  DSN=DVR110.DVRINTER(DVRXOGRX) 3
```

Figure 75. JCL for performing a precompile

1 PRDUCT.DSN220.DSNLOAD in the STEPLIB is the data set in which the DB2 programs reside.

2 DVR110.DEM.DBRMLIB (DVRXOGRX) is the data set and member in which the DBRM is created.

3 The COBOL source code is stored in DVR110.DVRINTER (DVRXOGRX).

Step 2. Binding the application program to the DB2 database

Binding establishes a relationship between the application program and DB2. This relationship must exist before you can run the application program. Binding validates SQL statements, verifies user authorization, selects DB2 data access paths, and builds a control structure called an application plan required to access the DB2 data when the application program is running. The sample JCL shown in Figure 76 performs a bind for application program DVRXOGRX.

```
//BIND   JOB  1,&user,CLASS=J,USER=&user,PASSWORD=&passwd 1
//*  CODE TO BIND THE DB2 COBOL EXIT TO THE DB2 Database
//*  DB2 MUST BE UP BEFORE RUNNING THIS JOB
//TSOTMP  EXEC  PGM=IKJEFT01,REGION=2048K
//STEPLIB DD DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR 2
//SYSPRINT DD  SYSOUT=A
//SYSTSPRT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSTSIN   DD *
DSN SYSTEM(DS3)  3
BIND  PLAN(DB2REC) -
 MEMBER(DVRXOGRX) - 4
       ACTION(REPLACE) RETAIN ISOLATION(CS)              -
 LIBRARY('DVR110.DEM.DBRMLIB')
END
```

Figure 76. JCL for performing a bind

1 The USER and PASSWORD parameters are the TSO user ID and password which TSO batch uses to connect to DB2.

2 The STEPLIB data set is where the DB2 database resides.

3 DS3 is the subsystem name for the DB2 database. DB2REC is a user-defined name for the application plan. This plan will be named by the DataRefresher job that runs the GDI exit routine (the application program).

4 Member specifies the member name in the DBRM library, DVR110.DEM.DBRMLIB, set up in the precompile JCL.

Step 3. Compiling and link-editing the application program

The PUNCH job created by the precompile is embedded in the JCL shown in Figure 77. This JCL performs a COBOL compile on the embedded source, then link-edits the resulting object code into a data set member.

```
//EXITOBJ JOB    MSGLEVEL=(1,1),REGION=1024K,CLASS=J
//*   PROCESS FOR VS COBOL II - COMPILE AND LINK
//*   FOR THE COBOL DB2 EXIT
//COB2   EXEC  PGM=IGYCRCTL,PARM='OBJECT,LIST,MAP,RES,SOURCE,RENT',
//         REGION=1024K
//STEPLIB  DD  DSNAME=PRDUCT.COBTST1.COB2LIB,DISP=SHR 1
//         DD  DSNAME=PRDUCT.COBTST1.COB2MAC,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),
//             SPACE=(TRK,(3,3))
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD *

      *   COBOL SOURCE CODE FOLLOWS

//*       COBOL SOURCE CODE PRECEDES
//*****************************************************************
//* INVOKE IEBCOPY TO COMPRESS DVR110.EXITLIB
//*****************************************************************

//COMPRSS  EXEC PGM=IEBCOPY 2
//SYSPRINT DD SYSOUT=A
//DXTV22AI DD DSN=DVR110.EXITLIB 3 ,DISP=(MOD,KEEP,KEEP),UNIT=SYSDA,
//         VOL=SER=DVR110
//SYSUT3   DD DSN=&WORK1,UNIT=SYSDA,SPACE=(TRK,1)
//SYSUT4   DD DSN=&WORK2,UNIT=SYSDA,SPACE=(TRK,1)
//SYSIN    DD *,DLM=$$
 COPY INDD=DXTV22AI,OUTDD=DXTV22AI
$$
//*****************************************************************
//*            LINKEDIT IF THE COMPILE RETURN CODE IS 4 OR LESS
//*****************************************************************
//LKED    EXEC PGM=IEWL 4 ,PARM='LIST,XREF,LET,MAP',
//      COND=(4,LT,COB2)
//SYSLIB   DD  DISP=SHR,DSN=PRDUCT.COBTST1.COB2LIB
//         DD  DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR
//SYSLMOD  DD  DISP=SHR,DSN=DVR110.EXITLIB(DVRXOGRX)      LOAD MOD
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN   DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD  *
 ENTRY DVRXOGRX
 NAME DVRXOGRX(R)
/*
//*****************************************************************
```

Figure 77. JCL for performing a COBOL compile

1 The PUNCH output from the precompile is embedded as indicated by the comments in this JCL. IGYCRCTL is the step that performs the COBOL compile. The STEPLIBs are the COBOL libraries required to perform the compile. &&LOADSET is the temporary data set containing the object code generated by the compilation, and is passed to the link-edit step.

2 IEBCOPY is used to compress the DVR110.EXITLIB data set so it does not run out of space if this JCL is called many times.

3 DVR110.EXITLIB is a user-defined data set in which the exit routine object code created by this job is stored.

4 IEWL is the link-edit step in which &&LOADSET, the object code from the compile, is link-edited into the user-defined data set member, DVR110.EXITLIB (DVRXOGRX), with an indicated entry point of DVRXOGRX. The SYSLIBs in this step are the required COBOL language data set and the data set containing the DB2 database. VS COBOL II is used in this example, and the generated load module will be executable in the TSO batch environment for connecting to DB2.

Step 4. Linking the UIM and DEM to the COBOL language modules

The UIM and DEM must be linked to COBOL language modules to set up the correct environment for running COBOL exit routines.

The JCL shown in Figure 78 link-edits the required COBOL language modules to the UIM.

```
//LKCOBUIM JOB  1,&user,CLASS=J
//LKED      EXEC  PGM=IEWL,PARM='LIST,XREF,LET,RENT,SIZE=(765K,256K)'
//SYSLIB    DD  DISP=SHR,DSN=PRDUCT.COBTST1.COB2LIB
//          DD  DISP=SHR,DSN=DVR110.DVR110.DVRLOAD
//SYSLMOD   DD  DISP=SHR,DSN=DVR110.NEWUIM
//SYSPRINT  DD  SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN DD *
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 INCLUDE SYSLIB(DVRU0000)
 ENTRY   DVRXSUSP
 NAME    DVRU0000(R)
/*
```

Figure 78. JCL for link-editing COBOL language modules to the UIM

The JCL shown in Figure 79 link-edits the required COBOL language modules to the DEM.

```
//LKCOBDEM JOB  1,&user,CLASS=J
//LKED      EXEC  PGM=IEWL,PARM='LIST,XREF,RENT,LET,SIZE=(768K,256K)'
//SYSLIB    DD  DISP=SHR,DSN=PRDUCT.COBTST1.COB2LIB  1
//          DD  DISP=SHR,DSN=DVR110.DVR110.DVRLOAD  2
//SYSLMOD   DD  DISP=SHR,DSN=DVR110.NEWDEM  3
//SYSPRINT  DD  SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(800,(100,100),,,ROUND)
//SYSLIN DD *
 INCLUDE SYSLIB(ILBOSTP0)
 INCLUDE SYSLIB(ILBOBEG)
 INCLUDE SYSLIB(ILBOSRV)
 INCLUDE SYSLIB(ILBOCMM)
 INCLUDE SYSLIB(DVRX0000)
 ENTRY   DVRXSSSU
 NAME    DVRX0000(R)
/*
```

Figure 79. JCL for link-editing COBOL language modules to the DEM

1 PRDUCT.COBTST1.COB2LOB is the data set containing the required COBOL libraries.

2 DVR110.DVR110.DVRLOAD contains the DEM.

3 DVR110.NEWDEM is the data set containing the new COBOL-ready DEM.

Step 5. Calling the GDI exit routine

The sample DB2 GDI Record exit routine, DVRXOGRX, has now been prepared. Assume that all the preparation performed for DVRXOGRX has been carried out for the sample DB2 GDI Select exit routine, DVRXOGSX.

At this point, the basic setup for using GDI exit routines is complete. The same procedures were used for both types of GDI exit routines. However, DataRefresher handles each type of GDI exit routine differently, so different jobs are used to call GDI Record and GDI Select exit routines.

Calling a GDI Record exit routine: The sample DB2 GDI Record exit routine, DVRXOGRX, uses the plan name DB2REC, assigned in the bind job. The UIM and DEM are called under the TSO Terminal Monitor Program in background mode.

The JCL shown in Figure 80 calls the GDI Record exit routine.

```
//D110CR01  JOB  MSGLEVEL=(1,1),USER=&user,PASSWORD=&passwd
//*  FUNCTION : TEST THE DATAREFRESHER V1R1 SAMPLE COBOL RECORD EXIT
//*  RELEASE  : DATAREFRESHER V1R1
//*
//*  SETUP ENVIRONMENT:  DATAREFRESHER MUST BE LINKED TO THE PROPER
//*  COBOL RUN TIME LIBRARIES TO ALLOW THE EXECUTION OF A COBOL USER
//*  EXIT.  DB2 TABLE TDEPT MUST EXIST AND BE AVAILABLE
//*  DVRXOGRX, THE COBOL SAMPLE RECORD EXIT MUST BE LINK EDITED IN
//*  THE USER DEFINED DATA SET DVR110.EXITLIB FOR EXECUTION.
//*  DVRXOGRX, THE COBOL SAMPLE RECORD EXIT MUST BE BOUND TO THE DB2
//*  DATABASE.
//*       EXPECTED RESULTS : NORMAL EXTRACTION FROM THE DB2 TABLE.
//*
//*********************************************************************
//*    IT IS ASSUMED THAT THE EXTLIB AND THE FDTLIB HAVE ALREADY BEEN
//*     ESTABLISHED.
//*********************************************************************
//*  RUN THE UIM
//*********************************************************************
//TSOTMP1  EXEC  PGM=IKJEFT01,REGION=2048K
//STEPLIB DD   DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR 1
//        DD   DSN=PRDUCT.COBTST1.COB2LIB,DISP=SHR
//        DD   DSN=DVR110.EXITLIB,DISP=SHR
//FDTLIB DD DSN=DVR110.FDTLIB,DISP=SHR
//EXTLIB DD DSN=DVR110.EXTLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSDUMP  DD  SYSOUT=A
//SYSOUT  DD SYSOUT=A
//SYSABOUT  DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//DSNTRACE DD SYSOUT=A
//SYSTSIN   DD *
DSN SYSTEM(DS3)
RUN PROGRAM(DVRU0000)  -
    PLAN(DB2REC)   -
 LIBRARY('DVR110.NEWUIM') - 2
 PARM('DEBUG=1,EXITLANG=COBOL')
END
```

Figure 80 (Part 1 of 2). JCL for calling the sample DB2 GDI Record exit routine

```
/*
//* U11
//DEMJCS  DD DATA,DLM=$$
*CD
*EO
$$
//DXTIN      DD   DATA,DLM=$$
/*                                                                  */
/*   DXTFILE DEFINITION FOR DB2 DATA THAT WILL BE ACCESSED BY       */
/*   A GDI RECORD EXIT.                                             */
CREATE
 DXTFILE NAME=TDEPT, ACCESS=GDI, GDIEXIT=DVRXOGRX, GDIXTYPE=RECORD,
    DETAIL=EXIT;
/*  THE VIEW IS BASED ON DXTFILE INFORMATION OBTAINED FROM THE GDI */
/*  RECORD EXIT.                                                   */
CREATE
 DXTVIEW NAME=TDEPT, DXTFILE=TDEPT, FIELDS= *;
SUBMIT EXTID=DB2REC,
     DBS=SQLDS,
     CD=JCS,
     JCS=DEMJCS
EXTRACT INTO TABLE1
SELECT * FROM TDEPT; 3
$$
//*  STEP TO RUN THE DEM
//TSOTMP  EXEC PGM=IKJEFT01,REGION=2048K
//STEPLIB  DD  DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR 1
//         DD  DSN=PRDUCT.COBTST1.COB2LIB,DISP=SHR
//         DD  DSN=DVR110.EXITLIB,DISP=SHR
//FDTLIB   DD  DSN=DVR110.FDTLIB,DISP=SHR
//EXTLIB   DD  DSN=DVR110.EXTLIB,DISP=SHR
//SYSTSPRT DD  SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSABOUT DD  SYSOUT=A
//SYSDUMP  DD  SYSOUT=A
//SYSOUT1  DD  SYSOUT=A
//SYSPRINT DD  SYSOUT=A
//DSNTRACE DD  SYSOUT=A
//SYSTSIN  DD  *
DSN SYSTEM(DS3)
RUN PROGRAM(DVRX0000)  -
    PLAN(DB2REC)   -
 LIBRARY('DVR110.NEWDEM') - 2
 PARM ('NONDLI')
END
/*
//DXTIN  DD DATA,DLM=$$
INITDEM NAME=IS01,DEBUG=1,EXITLANG=COBOL;
USE DXTFILE=TDEPT;
$$
/*
```

Figure 80 (Part 2 of 2). JCL for calling the sample DB2 GDI Record exit routine

1 The STEPLIBs are the DB2 product data set (PRDUCT.DSN220.DSNLOAD), the COBOL libraries data set (PRDUCT.COBTST1.COB2LIB), and the data set containing the GDI exit routine (DVR110.EXITLIB).

2 The libraries for the UIM and DEM are DVR110.NEWUIM and DVR110.NEWDEM respectively. These are the output data sets from the jobs that linked the UIM and DEM to the COBOL language modules.

3 Data is extracted from a DB2 table named TDEPT.

Calling a GDI Select exit routine: The sample GDI Select exit routine, DVRXOGSX, uses the plan name DB2SEL, assigned in the bind job. The UIM and DEM are called under the TSO Terminal Monitor Program in background mode.

The SQL SELECT statement is passed directly to the exit through the GSCSTMT field in DVRXCSCC. DVRXCSCC is the communication control block for GDI Select exit routines.

The JCL shown in Figure 81 calls the GDI Select exit routine.

```
//D250CS01 JOB  MSGLEVEL=(1,1),USER=&user,PASSWORD=&passwd
//*
//*  THIS JCL IS USED TO TEST THE COBOL SAMPLE DB2 SELECT EXIT
//*
//*   SETUP REQUIRED:
//*          EXIT MUST BE BOUND TO THE DB2 DATABASE
//*          THE DEM AND UIM COMPONENTS HAVE BEEN LINK-EDITED WITH
//*           THE NEEDED COBOL LANGUAGE MODULES.
//*          DB2 TABLE TDEPT MUST EXIST
//*          DB2 MUST BE RUNNING
//*   EXPECTED RESULTS:
//*          NORMAL EXTRACT FROM DATA.  ALL ZERO RETURN CODES
//*
//*********************************************************************
//*
//*   IT IS ASSUMED THAT THE EXTLIB AND FDTLIB HAVE ALREADY
//*    BEEN ESTABLISHED.
//*
//*********************************************************************
//*  RUN THE UIM
//*********************************************************************
//TSOTMP  EXEC  PGM=IKJEFT01,REGION=2048K
//STEPLIB DD   DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR 1
//        DD   DSN=PRDUCT.COBTST1.COB2LIB,DISP=SHR
//        DD   DSN=DVR110.EXITLIB,DISP=SHR
//FDTLIB DD DSN=DVR110.FDTLIB,DISP=SHR
//EXTLIB DD DSN=DVR110.EXTLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSDUMP  DD  SYSOUT=A
//SYSOUT  DD SYSOUT=A
//SYSABOUT  DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//DSNTRACE DD SYSOUT=A
//SYSTSIN   DD *
DSN SYSTEM(DS3)
RUN PROGRAM(DVRU0000)  -
    PLAN(DB2SEL)   -
 LIBRARY('DVR110.NEWUIM') - 2
 PARM('DEBUG=1,EXITLANG=COBOL')
END
/*
//* U11
//DEMJCS  DD DATA,DLM=$$
*CD
*EO
$$
```

Figure 81 (Part 1 of 2). JCL for calling the sample DB2 GDI Select exit routine

```
//SYSIN      DD   DATA,DLM=$$
/*                                                          */
/*   DXTFILE DEFINITION FOR DB2 DATA THAT WILL BE ACCESSED  */
/*   BY A GDI SELECT EXIT.  EXIT RETURNS A DATA DESCRIPTION */
CREATE
 DXTFILE NAME=TDEPT, ACCESS=GDI, GDIEXIT=DVRXOGSX, GDIXTYPE=SELECT;
SUBMIT EXTID=DB2SEL,
     DBS=SQLDS,
     CD=JCS,
     SELFILE=TDEPT,
     JCS=DEMJCS
EXTRACT
SELECT * FROM TDEPT; 3
$$
//TSOTMP1  EXEC  PGM=IKJEFT01,REGION=2048K
//STEPLIB DD   DSN=PRDUCT.DSN220.DSNLOAD,DISP=SHR 1
//        DD   DSN=PRDUCT.COBTST1.COB2LIB,DISP=SHR
//        DD   DSN=DVR110.EXITLIB,DISP=SHR
//FDTLIB DD DSN=DVR110.FDTLIB,DISP=SHR
//EXTLIB DD DSN=DVR110.EXTLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSABEND DD  SYSOUT=A
//SYSUDUMP DD  SYSOUT=A
//SYSABOUT  DD SYSOUT=A
//SYSDUMP  DD  SYSOUT=A
//SYSOUT1 DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//DSNTRACE DD SYSOUT=A
//SYSTSIN   DD *
DSN SYSTEM(DS3)
RUN PROGRAM(DVRX0000)  -
    PLAN(DB2SEL)   -
 LIBRARY('DVR110.NEWDEM') - 2
 PARM ('NONDLI')
END
/*
//DXTIN  DD DATA,DLM=$$
INITDEM NAME=IS01,DEBUG=1,EXITLANG=COBOL;
USE DXTFILE=TDEPT;
$$
/*
```

Figure 81 (Part 2 of 2). JCL for calling the sample DB2 GDI Select exit routine

1 The STEPLIBs are the DB2 product data set (PRDUCT.DSN220.DSNLOAD), the COBOL libraries data set (PRDUCT.COBTST1.COB2LIB), and the data set containing the GDI exit routine (DVR110.EXITLIB).

2 The libraries for the UIM and DEM are DVR110.NEWUIM and DVR110.NEWDEM respectively, which are the output data sets from the jobs that linked the UIM and DEM to the COBOL language modules.

3 Data is extracted from a DB2 table named TDEPT.

Example 3. Calling non-reusable database management systems (DBMSs)

Using certain non-IBM DBMSs with DataRefresher can cause a conflict in DataRefresher's normal operating environment. The following situations can cause difficulties:

- The DBMS module must be called from the JCL EXEC statement first. The DBMS module, in turn, calls DataRefresher.

- GDI exit routines which access the DBMS data must be link-edited with the DBMS module to use the services of the DBMS.
- The DBMS module is not reusable. In this case, MVS does not recognize that a usable copy of the GDI exit routine load module is in memory. Any attempt by DataRefresher to access the GDI exit routine using standard MVS services (LOAD and LINK, for example) causes a new copy of the entire load module to be loaded into memory. The newly loaded copy is not initialized and, therefore, the GDI exit routine is not able to use the services that the DBMS provides.

To solve these problems, control must be passed to a GDI exit routine that is already in memory, without causing a new copy of its load module to be loaded. The memory address of the GDI exit routine must be made available to DataRefresher.

This example shows how the DBMS module can pass the address of an exit address routine (EAR) to DataRefresher. When DataRefresher gives the name of a GDI exit routine to the EAR, the EAR can pass to DataRefresher the current address of that GDI exit routine.

When the DBMS calls DataRefresher

The DBMS must be called from the EXEC statement in the JCL. After its initialization, control must be passed to DataRefresher.

The code that calls DataRefresher from the DBMS module must observe the following conventions. These conventions are followed in the examples included in this chapter with code written in Assembler, but the code can be written in other languages.

When DataRefresher is called, register 1 must point to:

- A one word parameter list that points to a half-word in memory containing the length of the parameter string, followed by the parameter string itself. If no parameter string is specified, the length of the parameter string is set to zero.
- The address of the EAR.

The end-of-parameter-list indicator (the high-order bit of the address) must be turned off in the first address in the list, and turned on in the second (last) address in the list.

Figure 82 shows the structure of the parameter list when DataRefresher is called in a normal case.

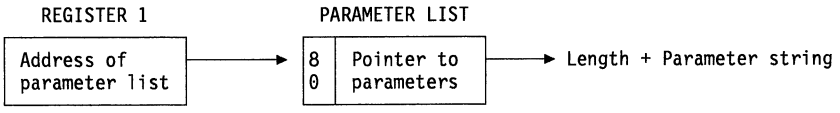

Figure 82. Parameter list in normal case

Figure 83 shows the structure of the parameter list in a non-reusable DBMS.

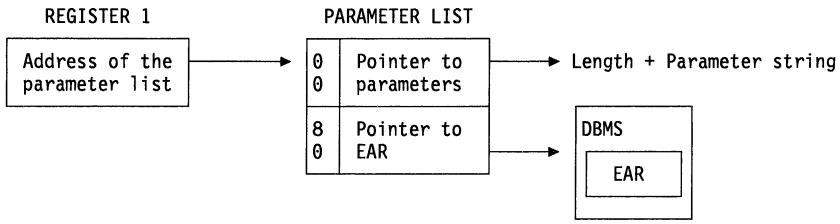

Figure 83. Parameter list in case of non-reusable DBMS

For the DEM, the top pointer either points to a parameter string that contains NONDLI, or to a list of PSBs. Because this pointer would always point to a NONDLI string, a join of IMS and DBMS data is not possible.

When DataRefresher detects this second address in the parameter list, it saves the address in COMEARA in the DVRXCCOM control block. The next section shows how DataRefresher uses this address.

Sample DataRefresher call routine

The code shown in Figure 84 calls the UIM:

```
CALLUIM  CSECT
*  Housekeeping
         STM      14,12,12(13)
         BALR     12,0
         USING    *,12
         LA       11,SAVEAREA
         ST       13,4(11)
         ST       11,8(13)
         LR       13,11
*  Get set to call DataRefresher's UIM component
         L        15,0(1)           Get address of parameter from MVS
         ST       15,PARMLIST       Save it in new place
         LA       15,PARMLIST       Get address of 1st word in list
         NI       0(15),255-X'80'   Turn off end-of-list indicator
         LA       15,4(15)          Bump to next word
         OI       0(15),X'80'       Turn on end-of-list indicator
*  Load the DataRefresher UIM module
         LA       2,MODNAME         Get address of UIM module
         LOAD     EPLOC=(2)
         LR       15,0              Save entry address in reg 15
         LA       1,PARMLIST        Point at parameter list
         BALR     14,15             Call DataRefresher
*  Return from DataRefresher -- return to caller
         L        13,4(13)          Restore caller's save area
         L        14,12(13)         Restore return address
         LM       0,12,20(13)       Restore caller's other regs,
*                                   leaving DataRefresher return code in R15
         BR       14                Return to caller
MODNAME  DC       CL8'DVRU0000'     Name of DataRefresher UIM module
PARMLIST DS       F                 Address passed by MVS
EARADDR  DC       A(ADDREAR)        Address of EAR address
ADDREAR  DC       V(EAR)            Address of Exit Address Routine
SAVEAREA DS       18F
         END      CALLUIM
```

Figure 84. Code for calling the UIM

This code assumes that the PARMs passed to DataRefresher are specified on the EXEC statement that calls the DBMS, and that these are the only parameters specified. If this is not the case, then this routine must include the PARMs as a con-

stant and ensure that the address at PARMLIST points to them, as shown in Figure 85 on page 123.

```
PARMLIST DC   A(DXTPARMS)     Address of PARM info for DataRefresher
EARADDR  DC   A(ADDREAR)      Address of EAR address
DXTPARMS DS   0H              PARM info for DataRefresher
         DC   H'26'           Actual length of parameter info
         DC   CL26'EXITLANG=COBOL,GDI=(.....)'
ADDREAR  DC   V(EAR)          Address of Exit Address Routine
```

Figure 85. PARMLIST for calling the UIM

This routine calls the UIM. To call the DEM, the value for MODNAME must be changed to DVRX0000, and the parameters, if included, must be changed to NONDLI with a length of six.

When DataRefresher calls a GDI exit routine

When DataRefresher calls a GDI exit routine, it checks the COMEARA field in the DVRXCCOM control block to see if an address has been specified. If the field is zero, processing continues normally. If a usable copy of the exit routine is not already loaded in memory, MVS loads the exit routine, then returns the address to DataRefresher. DataRefresher saves this address in a table. The GDI exit routine is called by branching to the address found in this table.

If the COMEARA field is not zero, DataRefresher sets up register 1 to point to a two-word parameter list, and branches to the address of the EAR, as shown in Figure 86.

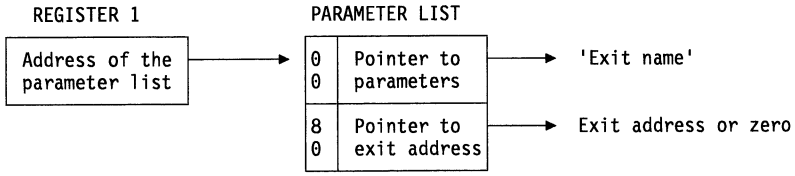

Figure 86. DataRefresher branches to the address of the EAR

This two-word parameter list contains a pointer to an eight-byte character string that includes the name of the GDI exit routine. The parameter list also contains a pointer to a word in which the EAR must store the current memory address of the GDI exit routine. DataRefresher initializes this word to all zeros.

If the EAR cannot determine the current memory address of the exit routine, it should return an address of all zeros. DataRefresher proceeds as if the COMEARA field were zero, and continues processing.

Figure 87 shows how DataRefresher and a GDI exit routine normally interact.

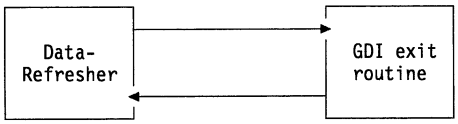

Figure 87. How DataRefresher normally interacts with a GDI exit routine

Normally, DataRefresher loads the exit routine and then branches.

Figure 88 shows the interaction between DataRefresher, the GDI exit routine and the EAR in a non-reusable DBMS

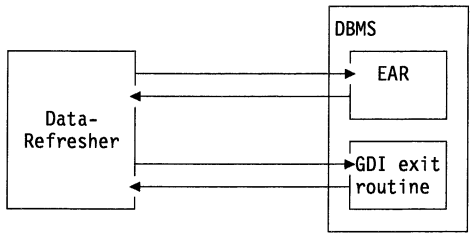

Figure 88. How DataRefresher and a GDI exit routine interact with a non-reusable DBMS

With a non-reusable DBMS, DataRefresher goes to the EAR and branches back. DataRefresher then loads the exit routine and branches back again.

Sample exit address routine

Figure 89 shows sample code for an EAR.

```
EAR      CSECT
         USING    *,15
         STM      3,4,SAVEAREA      Save registers
         LA       3,TBLSTART        Point at start of table
         LA       4,TBLLAST         Get address of last entry
CMPLOOP  DS       0H
         CR       3,4               Are we at end of table?
         BE       LOOPEND           Yes, terminate the loop
         CLC      0(8,3),0(1)       Is this the desired name?
         BE       LOOPEND           Yes, terminate the loop
         LA       3,12(3)           Bump to next table entry
         B        CMPLOOP           .. and repeat the loop
LOOPEND  DS       0H
         L        4,8(3)            Load the exit address
         ST       4,8(1)            .. and pass it back to
                                       DataRefresher
         LM       3,4,SAVEAREA      Restore registers
         BR       14                .. and return
*
SAVEAREA DS       2F                Save area
*********************************************************************
* * *      Table of Exit Names and Addresses                   * * *
*********************************************************************
TBLSTART DS       0F                Start of table
         DC       CL8'EXRX02  '     Name of first exit
         DC       V(EXRX02)         Address of first exit
         DC       CL8'GWSELXP '     Name of next exit
         DC       V(GWSELXP)        Address of next exit
TBLLAST  DS       0H
         DC       8CL1' '           End of table indicator
         DC       F'0'              Indication that name not found
TBLEND   EQU      *                 END OF TABLE
         END      EAR
```

Figure 89. Code for an exit address routine

Chapter 8. Writing Generic Output Interface (GOI) exit routines

Generic Output Interface (GOI) exit routines provide you with the facility to manipulate data after it has been extracted by DataRefresher. You can use GOI exit routines to convert data into a user-defined format, write it to a file, or immediately load it into a database.

A different GOI exit routine must be written for each different type of target.

Task overview

Writing and using GOI exit routines with DataRefresher consists of:

1. Determining what you want your GOI exit routine to do with the data DataRefresher has extracted.
   - For information on the functions that GOI exit routines can perform, see "Generic Output Interface exit routines" on page 6.
2. Writing your GOI exit routine.

   For information on:
   - How GOI exit routines interact with the control block, see "Using the GOI exit routine control block" on page 127.
   - Using the sample GOI exit routines provided with DataRefresher, see "Writing GOI exit routines using sample exit routines" on page 128.
3. *If your GOI exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. Compiling and link-editing your GOI exit routine.
   - For information on compiling and link-editing GOI exit routines, see "Compiling and link-editing your exit routines" on page 22.
5. *If you are including your GOI exit routine in extracts submitted via DataRefresher OS/2,* registering your exit routine.
   - For information on registering your GOI exit routine, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
6. Identifying your GOI exit routine to DataRefresher on your SUBMIT command.
   - For information on specifying GOI exit routines on extract requests, see "Identifying your GOI exit routine to DataRefresher" on page 128.

Calling GOI exit routines

Identify your GOI exit routine to DataRefresher on the SUBMIT command of your extract request, using the GOIEXIT keyword. When the DEM calls a GOI exit routine, it passes the extracted data to the exit routine for processing, one row at a time. If the ORDER BY keyword has been specified on your SUBMIT command, the GOI exit routine is called after the data has been re-sequenced.

The DEM makes up to five different types of calls to a GOI exit routine:

- DESCRIBE
- OPEN
- CD
- PUT
- CLOSE

Figure 90 shows how DataRefresher calls your GOI exit routine.

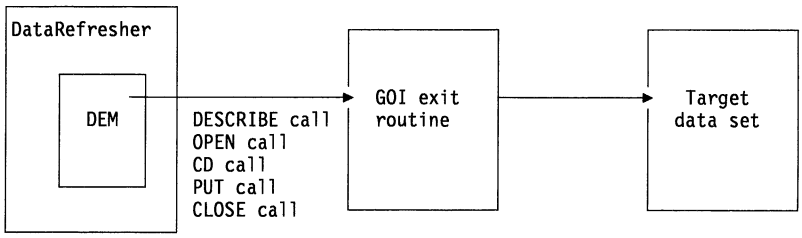

Figure 90. How DataRefresher calls a GOI exit routine

DESCRIBE The DEM describes an input row of data to the GOI exit routine. The exit routine determines whether the DEM can continue processing the extract request, and passes back a return code. The exit routine performs any initialization that needs to be done to process the call.

OPEN The exit routine can perform all initialization processing for this extract, for example:

- Opening target data sets
- Preparing a target database
- Initializing target fields
- Acquiring main storage through a GETMAIN macro

The exit routine determines whether the DEM can continue processing the extract request, and passes back a return code.

CD The DEM makes a CD call to the exit routine only if you have generated a load control deck or column descriptors for this extract request. You must have specified the CD keyword on the SUBMIT command, and the INTO keyword on the EXTRACT statement. For information on DEM output, including data in IXF, see "Working with control statements" on page 129.

The DEM issues a series of CD calls until it reaches the last statement in the load control deck or column descriptors. On a CD call, the DEM passes a line from the load control deck or column descriptors to the exit routine. The exit routine processes the statements, determines whether the DEM can continue processing the extract

request, and passes back a return code. When the last statement is reached, DataRefresher sends a signal to the exit routine by setting the end-of-data indicator in the GOIEOCD field of the control block.

PUT The DEM issues a series of PUT calls until it reaches the last row of data. On a PUT call, the DEM passes a row of data to the exit routine. The exit routine processes the row of data, and writes the output to the target database or file. The exit routine determines whether the DEM can continue processing the extract request, and passes back a return code.

CLOSE The exit routine cleans up the remnants of processing and returns control to the DEM, which performs final processing.

When the DEM calls your GOI exit routine, it passes as parameters the addresses of the:

- Control block
- Input buffer containing:
  - The control deck or column descriptors on CD calls
  - Input data on PUT calls
- Global work area

The control block and the global work area are both passed during all calls by the DEM to a GOI exit routine. The input buffer is only passed on CD and PUT calls.

As in a standard module linkage, your GOI exit routine returns to the DEM using an address passed to the exit routine in register 14. This address is passed to the GOI exit routine on entry.

Using the GOI exit routine control block

DataRefresher communicates with your GOI exit routine through a control block. A copy of the GOI control block is automatically associated with each GOI exit routine. There is a different GOI exit routine control block for each programming language. The control blocks are contained in the DVR110.DVRINTER library, as shown in Table 31.

Table 31. GOI exit routine control blocks

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
GOI exit routine	DVRXCGOP	DVRXCGOC	DVRXCGOA

Your GOI exit routine can modify the following fields in the control block:

- Return code field (GOIRETCD)
- Message field (GOIMSG)
- Two flag fields (GOIENTRD and GOIINCTL), which you can use for tracing and debugging

Writing GOI exit routines using sample exit routines

Included with DataRefresher are sample GOI exit routines in PL/I, COBOL and Assembler. These sample exit routines are contained in the DVR110.DVRINTER library. You can copy or print the sample exit routines and modify them to perform the functions you need. These sample GOI exit routines are called:

DVRXYGOI PL/I GOI exit routine
DVRXOGOI COBOL GOI exit routine
DVRXAGOI Assembler GOI exit routine

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

Examples of how GOI exit routines can be used are contained in "GOI exit routine example" on page 131.

Identifying your GOI exit routine to DataRefresher

You identify your GOI exit routine to DataRefresher on the SUBMIT command by specifying the name of the exit routine on the GOIEXIT keyword. Figure 91 shows an example of a SUBMIT command that specifies the GOIEXIT keyword.

```
SUBMIT    EXTID=EXT03,
          DBS=DB2,
          FORMAT=SOURCE,   1
          GOIEXIT=GOIXSUB  2
```

Figure 91. Extract request specifying a GOI exit routine

Figure 92 shows an example of a SUBMIT command that specifies the CD, EXTDATA and GOIEXIT keywords.

```
SUBMIT   EXTID=EXT04,
         DBS=SQLDS,
         FORMAT=SOURCE,    1
         GOIEXIT=GOIXSUB,  2
         CD=EXTDATA,     3
         EXTDATA=OUTDD   4

EXTRACT  INTO TESTTAB (COL1,COL2,COL3,COL4,COL5,COL6,COL7,COL8) 5
```

Figure 92. Extract request specifying a GOI exit routine and load control deck

1 FORMAT=SOURCE indicates that the data is passed to the exit routine in source format. FORMAT=EBCDIC would reformat extracted data into EBCDIC format.

2 GOIEXIT=GOIXSUB identifies your exit routine to the DEM. When this extract request is run, the DEM calls the exit routine GOIXSUB and passes the data to it, one row at a time.

3 CD=EXTDATA causes the DEM to generate a load control deck. In this example, the load control deck is passed on a CD call, one line at a time.

4 EXTDATA=OUTDD identifies the ddname of a data set to which the extracted data can be written. This information is passed to the exit routine in the GOIDDNAM field of the control block.

5 The INTO statement identifies the name of the relational database table and the table columns into which the extracted data is loaded. The DEM uses this information to generate the load control deck.

The GOI keyword on the INITDEM command may be used to pass parameters to a GOI exit routine while running the DEM. Figure 93 shows an example of an INITDEM command with a GOI keyword.

```
INITDEM   NAME=DEMA,
          RUNMODE=TERM,
          GOI='DDTARG'  6
```

Figure 93. INITDEM command

6 GOI='DDTARG' indicates a target data set for a GOI exit routine. You can use this keyword to pass information to the GOI exit routine at run time. For example, you could include this keyword if you were running a persistent extract request, and you wanted to send extract output to a target data set on a once-off basis. The GOI keyword is optional on the INITDEM command.

For detailed information on the SUBMIT and INITDEM commands, see the *DataRefresher Command Reference*.

Working with control statements

DataRefresher can generate a load control deck or column descriptors along with its data output, and pass them to your GOI exit routine. The DEM indicates which type of control deck statements are being passed to the exit routine by assigning a value to the GOICDTYP field in the control block. The following types of control deck statements can be passed:

- Create table
- Delete from table
- Load table
- IXF header record
- IXF table record
- IXF column descriptor record

To output data in IXF, you must include the CD keyword in your SUBMIT command. If the DEM passes IXF records or column descriptors to a GOI exit routine, the DEM generates a four-byte field that precedes each record or column descriptor. This field is formatted `nn00`, where `nn`=record length.

For an example of a SUBMIT command that includes control statements, see "Identifying your GOI exit routine to DataRefresher" on page 128. For detailed information on generating control deck statements and creating IXF file definitions, see the *DataRefresher Command Reference*.

GOI exit routine return codes

GOI exit routines communicate with DataRefresher via return codes placed in the GOIRETCD field of the control block. Table 32 explains the meaning of the return codes for each type of call to a GOI exit routine.

Table 32. GOI exit routine return codes

Type of Call	RC = 0	RC = 4	RC = 8
DESCRIBE	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
OPEN	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
CD	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
PUT	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
CLOSE	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	

If the exit routine returns a code that is not listed above, DataRefresher cannot interpret the code and discontinues processing of the batch of extract requests.

Handling errors during GOI exit routine processing

DataRefresher may issue a CLOSE call to a GOI exit routine before the exit routine has completed its processing, for example, if the:

- Extract request has been cancelled
- DEM has encountered an error while extracting data
- GOI exit routine has encountered an error while processing data, and returned a code of 8 to the DEM

When the DEM issues a CLOSE call in these situations, it sends a signal to the exit routine in the GOISTAT field of the control block. This signal tells the GOI exit routine that some additional cleanup procedures may be necessary.

To handle these situations, you can write your exit routine, for example, to:

- Undo any changes made to the target data set or file
- Retain any changes made to the target data set or file
- Set a checkpoint at the last row of output, to indicate where the exit routine is to continue processing

Working with batched extract requests

Batching your extract requests can save time and resource utilization costs at your site. All exit routines (except GDI Select exit routines) can process batched extract requests. GOI exit routines, however, are called separately for each extract request in the batch that specifies that exit routine.

To benefit most from batching extract requests with GOI routines:

- Write your GOI exit routines to be **reentrant**. A reentrant exit routine can be used concurrently by more than one program, thus saving time and storage space.

  For information on writing reentrant exit routines, see "Making your exit routines reentrant" on page 25.

- For OS PL/I exit routines that write to physical sequential data sets, specify file definitions for each data set in the batch.

For information on batching extract requests, see the *DataRefresher Administration Guide*.

GOI exit routine example

The sample GOI exit routines provided with DataRefresher converts extracted data into a 'Delimited' format. The output file produced, when transferred to a workstation, is suitable for loading into a database or other application package that accepts 'Delimited' format files for Load or Import.

Note: It is assumed that the conversion to ASCII and the insertion of Carriage Return / Linefeed (CRLF) and End Of File (EOF) characters will be performed by any file transfer utility used.

Identifying your GOI exit routine to DataRefresher on your SUBMIT command

Use the UIM to define your:

- DXTPSB or DXTFILE as for a normal extract.
- Extract Request to DataRefresher. This involves supplying the following SUBMIT / EXTRACT command keywords:

```
SUBMIT  EXTID=your_extid,
        FORMAT=EBCDIC,
        GOIEXIT=exitname
EXTRACT etc ;
```

Figure 94. Necessary SUBMIT / EXTRACT command keywords

Where: FORMAT=EBCDIC is required to tell DataRefresher to convert internal numeric data items to external format.

exitname is the name of the language version you are using: DVRXYGOI, DVRXOGOI or DVRXAGOI.

Run the DEM to perform the Extract Request, the INITDEM command used requires the following keywords:

```
INITDEM NAME=your_dem,
        EXITLANG=lang,
        GOI='xy';
```

Figure 95. Necessary INITDEM command keywords

Notes:

1. To indicate that your exit routine is not the ASM: specify:

   `EXITLANG=lang`

2. Replace *lang* with PLI for OS/VS PLI, COBOL for COBOL II, or LE for LE/370 PLI or COBOL compiled modules.

3. GOI='xy' specifies the parameters to be passed to the GOI exit. The overrides for the default Column and Character Delimiters are:

   - *x* is the override Column Delimiter and the default value is a comma (,).
   - *y* is the override Character Field Delimiter and the default value is a double quote (").

   For example, to use a single quote (') as the Column Delimiter, you specify:

   `GOI=','''`

   The parameter itself is within quotes and a comma is specified even though it is the default. The override values are positional.

   Note: You need a DD statement for the output file produced. The DDNAME should be DRFOUT1 and the file should be Fixed Blocked, LRECL of 80.

Chapter 9. Writing Map Capture exit routines

Map Capture exit routines gather and save information about extract requests from the UIM and the DEM. The information provided by Map Capture exit routines can be used to perform validation of your extract requests. There is no limit to the number of Map Capture exit routines you can write.

You can also use a Map Capture exit routine to help you define propagation requests for IBM's DataPropagator NonRelational. For more information on using DataRefresher with DataPropagator NonRelational, see the *DataPropagator NonRelational Customization Guide.*

Task overview

Writing and using Map Capture exit routines with DataRefresher consists of:

1. Determining what functions you want your Map Capture exit routine to perform.
   - For information on the functions that Map Capture exit routines can perform, see "Map Capture exit routines" on page 6.
2. Writing your Map Capture exit routine.

   For information on:
   - How Map Capture exit routines interact with the control block, see "How to use the Map Capture exit routine control block" on page 135.
   - Using the sample exit routines provided with DataRefresher, see "Writing Map Capture exit routines" on page 135.
3. *If your Map Capture exit routine is written in OS PL/I or VS COBOL II,* preparing the DataRefresher language environment.
   - For information on using non-Assembler exit routines, see "Setting up DataRefresher to use programming languages" on page 14 and "Preparing the language load module for OS PL/I and VS COBOL II" on page 17.
4. *If your Map Capture exit routine will access DB2,* precompiling and binding your exit routine.
   - For information on preparing your Map Capture exit routines for DB2 applications, see "Precompiling and binding exit routines for DB2 applications" on page 21.
5. Compiling and link-editing your Map Capture exit routine.
   - For information on compiling and link-editing your Map Capture exit routines, see "Compiling and link-editing your exit routines" on page 22.
6. *If you are using exit routines in extract requests prepared using DataRefresher OS/2,* registering your exit routines and specifying the exit to use when you create a source folder.

   For information on how to:
   - Register your exit routines, see "Registering your exit routines for use with DataRefresher OS/2" on page 23.
   - Specify which exit routine to use when creating a source folder, see the *DataRefresher OS/2 User's Guide.*

7. *If you are using exit routines in extract requests prepared using the DataRefresher dialogs or the system editor,* identifying your exit routines to DataRefresheron the SUBMIT command of your extract request.

   For information on:

   - The syntax of the SUBMIT command, see the *DataRefresher Command Reference.*
   - Specifying a Map Capture exit routine on your extract requests, see "Identifying your Map Capture exit routine to DataRefresher" on page 136.

Calling Map Capture exit routines

Identify your Map Capture exit routine to DataRefresher on the SUBMIT command of your extract request.

When the UIM processes an extract request that specifies a Map Capture exit routine, the UIM calls the exit routine prior to storing a valid extract request in the EXTLIB. When the DEM processes an extract request that specifies a Map Capture exit routine, the DEM calls the exit routine before extracting the data.

When the UIM calls the exit routine, information about the files, PCBs, views, and extract request is supplied in the interface to the exit routine. When the DEM calls the exit routine, only information about the extract and files or PCBs about to be run is supplied in the interface to the exit routine. Therefore, the exit routine must save information about the files and extract requests during the UIM call for use during the subsequent DEM call.

Before DataRefresher calls your Map Capture exit routine, it copies the following information into the Map Capture exit routine control block:

- File information and PCBs about the source data
- SUBMIT, EXTRACT, SELECT, FROM, and WHERE statements from the extract request

When the UIM calls your Map Capture exit routine, it passes as parameters the addresses of the:

- Control block
- TSO command processor parameter list (CPPL), or 0 if there is no CPPL

When the DEM calls your Map Capture exit routine, it passes the address of the Map Capture exit routine control block. As in a standard module linkage, your exit routine returns to the DEM using an address passed to the exit routine in register 14. This address is passed to the Map Capture exit routine on entry.

The MXCCALL field in the control block tells the Map Capture exit routine what type of call is being made to it (UIM or DEM call).

How to use the Map Capture exit routine control block

Map Capture exit routines communicate with DataRefresher through a control block. A copy of the Map Capture Communication Area (MCCA) control block is associated automatically with each Map Capture exit routine. There is a control block for each programming language. These sample control blocks are contained in the DVR110.DVRINTER library, as shown in Table 33.

Table 33. Map Capture exit routine control blocks

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Map Capture exit routine	DVRXCMXP	DVRXCMCC	DVRXCMXA

Your Map Capture exit routine can modify the following fields in the control block:

- Return code field (MXCRETC)
- Reason code field (MXCRSCND)
- A scratchpad area of 1024-bytes (MXCSCRT1) in which the Map Capture exit routine can save addresses and other information that it uses
- The 64-byte work areas:MXCPSCRT for the source file, MXCSSCRT for each segment entry, MXCFSCRT for each field entry
- Work area of 14-bytes for each qualifier entry (MXCQSCRT)
- Message field (MXCMESG) for up to 10 messages
- Target table entry (MXCTRGTN)
- The flag fields (MXCENTRD and MXCINCTL) that you can use for tracing and debugging

Writing Map Capture exit routines

Included with DataRefresher are sample Map Capture exit routines in PL/I and COBOL. Map Capture exit routines can also be written in Assembler. You can copy or print the sample exit routines and modify them to perform the functions you need. The following sample exit routines are distributed in the DVR110.DVRINTER library:

DVRXYMAP PL/I Map Capture exit routine
DVRXOMAP COBOL Map Capture exit routine

For information on copying and printing the sample exit routines provided with DataRefresher, see Appendix C, "DataRefresher sample exit routines and control blocks" on page 147.

For an example of how a Map Capture exit routine can be used, see "Map Capture exit routine example" on page 136.

Identifying your Map Capture exit routine to DataRefresher

Use the MAPEXIT keyword on the SUBMIT command in your extract request to identify your Map Capture exit routine to DataRefresher.

Note: The MAPUPARM keyword of the SUBMIT statement can be used to pass information that may be required by the exit routine, such as authorization information. Figure 96 shows the SUBMIT command for an extract request with a Map Capture exit routine called DVRXOMAP.

```
SUBMIT EXTID=EXT03,
DBS=DB2,
MAPEXIT=DVRXOMAP,
JCS=DVREDDJC, CD=JCS
EXTRACT
  INTO Q.STAFF  (NAME, ID)
  SELECT ID, COMM
  FROM PSEQSTAFF;
```

Figure 96. Extract request

DataRefresher Command Reference. contains detailed information on the syntax of the SUBMIT command.

Map Capture exit routine return codes

The Map Capture exit routine communicates with DataRefresher via return codes placed in the MXCRETC field of the control block. If your Map Capture exit routine returns a code other than 0, you will receive an error message. Table 34 shows the Map Capture exit routine return code values.

Table 34. Map Capture exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
8	To the UIM, request will not be put in EXTLIB. To the DEM, request will not be executed. To both, processing continues.
16	DataRefresher should terminate the UIM and DEM execution

Map Capture exit routine example

The exit routine used in this example is the sample COBOL Map Capture exit routine that is contained as member DVRXOMAP in the DVR110.DVRINTER library. This exit routine verifies that an extract request loading data into DB2 on the same system has the correct:

- Table name
- Column names
- Map Capture types

The exit routine checks that the extract request corresponds with the target table.

The sample Map Capture exit routine performs verification before the extract request is put in the EXTLIB once the SUBMIT command has been processed. It verifies the source column names against the target names and types (numbers to character, for example) to make sure that they are compatible. The interaction

between DataRefresher, a Map Capture exit routine and a DB2 database is shown in Figure 97 on page 137.

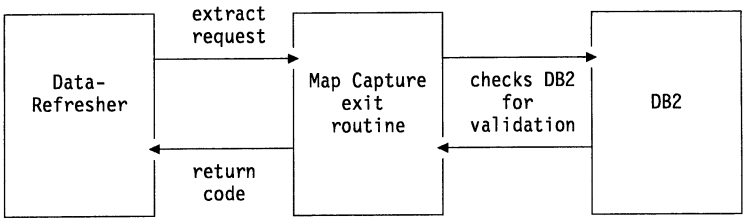

Figure 97. Extract request validation performed by Map Capture exit routine

As shown in Figure 97, DataRefresher passes the extract request information to the Map Capture exit routine for validation. This information is passed in the Map Capture Communication Area control block. The Map Capture exit routine accesses the DB2 catalog to validate the information, and passes back a return code to DataRefresher.

The sample exit routine requires that the:

- DataRefresher is linked to the proper COBOL run time library to allow the execution of a COBOL exit routine
- The DB2 table Q.STAFF is available in DB2 prior to defining the extract request
- The exit is precompiled and bound to the DB2 database
- The sample Map Capture exit routine, DVRXOMAP, is compiled and link-edited in the DVR110.DXT.EXITLIB for execution

Figure 98 is the DXTFILE definition used in this example.

```
CREATE DXTFILE NAME=PSEQSTAFF, ACCESS=PS, FREQ=20,
 DESC='EMPLOYEE DATA'
SEGMENT NAME=STAFFSEG,
 DESC='EMPLOYEE STATISTICS'
 FIELD NAME= ID,      START= 5,  TYPE=H, BYTES= 2, DESC='EMP ID     '
 FIELD NAME= NAME,    START= 7,  TYPE=C, BYTES= 9, DESC='EMP NAME  '
 FIELD NAME= DEPT,    START=16,  TYPE=H, BYTES= 2, DESC='DEPT NUM  '
 FIELD NAME= JOB,     START=18,  TYPE=C, BYTES= 5, DESC='JOB NAME  '
 FIELD NAME= YEARS,   START=23,  TYPE=B, BYTES= 1, DESC='YRS WORKED '
 FIELD NAME= SALARY  ,START=24,  TYPE=P, BYTES= 4, SCALE=2
 FIELD NAME= COMM    ,START=28,  TYPE=P, BYTES= 8, SCALE=2
 FIELD NAME= NUM_HIST,START=32,  TYPE=B, BYTES= 1

SEGMENT NAME=HISTSEG, FORMAT=FI, BYTES=24, OCCURS=NUM_HIST,
          START=NUM_HIST+1, PARENT=STAFFSEG,
          DESC='SEGMENT FOR HISTORY RECORDS'
 FIELD NAME= DATE,    START= 1,  TYPE=A, BYTES=10
 FIELD NAME= PASTJOB, START=11,  TYPE=C, BYTES= 5
 FIELD NAME= YRSWKED, START=16,  TYPE=B, BYTES= 1
 FIELD NAME= SSAL,    START=17,  TYPE=P, BYTES= 4, SCALE=2
 FIELD NAME= ESAL,    START=21,  TYPE=P, BYTES= 4, SCALE=2
    ;
```

Figure 98. DXTFILE definition (PSEQSTAFF)

Figure 99 shows the definition of the DXTVIEW.

```
CREATE DXTVIEW NAME=PSEQSTAFF,DXTFILE=PSEQSTAFF,SEGMENT=HISTSEG,
        FIELDS=*;
```

Figure 99. DXTVIEW definition (PSEQSTAFF)

Figure 100 on page 138 contains an example of an extract request that returns with an error condition code.

```
SUBMIT EXTID=EXT03,
DBS=DB2,
MAPEXIT=DVRXOMAP,
JCS=DVREDDJC, CD=JCS
EXTRACT
  INTO Q.STAFF  (NAME, ID)
  SELECT ID, COMM
  FROM PSEQSTAFF;
```

Figure 100. First extract request

The expected results from this extract request are:

- Map Capture types for field ID and column NAME are incompatible
- Null indicators for field COMM and column ID are incompatible

The extract request fails with return code 8 due to a data type error. Warnings are returned in the diagnostic messages to indicate conversion and nulls errors. Included in the output are the following diagnostic and completion messages:

```
DVRH0501
DATA TYPES ARE INCOMPATIBLE BETWEEN DATAREFRESHER AND THE DB2 CATALOG.

DVRH0501
THE DATAREFRESHER DEFINITION FOR FIELD 'ID' IS H .

DVRH0501
THE DB2 DEFINITION FOR COLUMN 'NAME' IS VARCHAR .

DVRH0501
DB2 TRUNCATION OR CONVERSION ERRORS MAY OCCUR ON THE LOAD.

DVRH0501
THE DATAREFRESHER LENGTH FOR FIELD 'COMM' IS 00008.

DVRH0501
THE DB2 LENGTH FOR COLUMN 'ID' IS 00002.

DVRH0501
DATAREFRESHER SAYS NULLS ALLOWED FOR COLUMN 'ID',DB2 SAYS NO.
```

Figure 101. Output from first extract request

Figure 102 contains an example of a successful extract request.

```
SUBMIT EXTID=EXT04,
DBS=DB2,
MAPEXIT=DVRXOMAP,
JCS=DVREDDJC, CD=JCS
EXTRACT
  INTO Q.STAFF  (SALARY,NAME, COMM)
  SELECT SALARY, NAME, COMM
  FROM PSEQSTAFF;
```

Figure 102. Second extract request

This extract request produces a warning that the column lengths specified for COMM in the extract request are incompatible with the column lengths for COMM in DB2. Only warning messages are issued, so the request is successful. The following diagnostic and completion messages are included in the output.

```
DVRH0501
THE DATAREFRESHER LENGTH FOR FIELD 'COMM' IS 00008.

DVRH0501
THE DB2 LENGTH FOR COLUMN 'COMM' IS 00007.
```

Figure 103. Output from second extract request

The exit routine accesses the DB2 system catalog to validate the target table and columns specified on the DataRefresher INTO keyword.

Since the exit routine calls DB2 to read the catalog during the UIM call, the UIM is called as a TSO batch program (IKJEFT01). The UIM call names the DB2 plan created during the bind step. The UIM processes the data descriptions and the extract request. Once the UIM has validated the extract request, the UIM calls the Map Capture exit routine.

During the DEM call, the exit routine does not call DB2. The DEM is called using the standard DEM JCL. The TSO batch program is not needed for this step, as the sample exit routine does not call DB2 during the DEM step.

Appendix A. Exit routine task overview

This appendix summarizes the tasks that exit routines can perform.

Converting nonstandard data

If you want to	Use a
Convert date/time data	Date/Time Conversion exit routine
Convert bit data	User Data Type exit routine
Convert unsigned packed numeric data	User Data Type exit routine
Convert 3-byte binary data	User Data Type exit routine
Convert encoded data (such as a 2-byte state ID that you want expanded)	User Data Type exit routine
Convert numeric data stored as character data	User Data Type exit routine
Convert specific fields in source data	User Data Type exit routine
Change data types	User Data Type exit routine
Change data lengths	User Data Type exit routine
Suppress source fields	Data exit routine
Create new fields for extraction	Data exit routine
Create several logical records or segments from one physical source record or segment	Data exit routine
Create one logical record or segment from several physical sources or segments	Data exit routine
Alter or combine data	Data exit routine
Summarize data	Data exit routine
Decompress or decrypt data	Data exit routine

Accessing data from unsupported sources

If you want to	Use a
Extract data from a non-IBM DBMS with an SQL interface	GDI Select exit routine
Extract data from a non-IBM DBMS without an SQL interface	GDI Record exit routine
Perform a two-stage extract for DB2 data	GDI Select exit routine
Join data from a range of sources, such as DB2, IMS, non-IBM DBMS, physical sequential, VSAM and IXF data	GDI Record exit routine

Monitoring DEM use

If you want to	Use an
Record information about extracts by specific users	Accounting exit routine
Screen users according to account information	Accounting exit routine
Restrict DEM use to authorized users	Accounting exit routine
Divide resource utilization costs between accounts or users	Accounting exit routine

Overriding information coded on extract requests

If you want to	Use an
Change the priority of extract requests	Accounting exit routine
Change the output limits of extract requests	Accounting exit routine

Working with and validating extract request information

If you want to	Use a
Save information about extract requests for IBM DataPropagator NonRelational processing	Map Capture exit routine
Save information about extract requests for processing by other programs	Map Capture exit routine
Check for consistency between selected fields and corresponding target columns from the INTO statement of your extract request	Map Capture exit routine

Working with extracted data

If you want to	Use a
Send data to an otherwise unsupported target	GOI exit routine
Summarize extracted data	GOI exit routine

Appendix B. Exit routine return codes

This appendix contains the meanings of the return codes for each type of exit routine.

Accounting exit routine return codes

Your Accounting exit routine communicates with DataRefresher via return codes placed in the ACXRETC field of the control block. Table 35 contains an explanation of the return code for each type of call to an Accounting exit routine.

Table 35. Accounting exit routine return codes

Type of Call	RC = 0	RC = 4	RC = 8	RC = 12
VALIDATE	Normal completion	Extract request not included in EXTLIB	Extract request not included in EXTLIB	UIM prevented from running
OPEN	Normal completion	Normal completion	Normal completion	DEM prevented from running
BEGIN-BATCH	Normal completion	Batch will be executed but not all extract requests included	Batch will not be executed	DEM prevented from running
END-BATCH	Normal completion	Normal completion	Normal completion	DEM prevented from running
CLOSE	Normal completion	Normal completion	Normal completion	DEM prevented from running

If the exit routine returns a code that is not listed above, DataRefresher cannot interpret the code and discontinues processing of the batch of extract requests.

Data exit routine return codes

Your Data exit routine communicates to DataRefresher using return codes in the DAXRETC field of the control block. Table 36 contains an explanation of the return codes.

Table 36. Data exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
4	Successful completion; DataRefresher should get next occurrence of source record or segment from the exit routine instead of from the file or database
8	DataRefresher should ignore data; data does not qualify for extraction
12	Error; terminate batch
16	Error; terminate DEM
Other	Error; terminate batch

If the exit routine returns a code that is not listed, DataRefresher cannot interpret the code and discontinues processing the batch of extract requests.

User data type exit routine return codes

Your User Data Type exit routine communicates with DataRefresher using return codes placed in the UDTXRETC field of the control block. Table 37 contains an explanation of the return codes.

Table 37. User Data Type exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
Other	Output will be based on the FLDERR keyword of the SUBMIT command

Date/Time Conversion exit routine return codes

Your Date/Time Conversion exit routine communicates with DataRefresher via return codes placed in the CXXCVNRC field of the control block.

Table 38. Date/Time Conversion exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
Other	Error

If the Date/Time Conversion exit routine returns a code other than 0, you will receive an error message. DataRefresher continues processing, but produces no output for the field that produced the error.

GDI exit routine return codes

DataRefresher communicates with your GDI exit routine via return codes that the GDI exit places in the data communication control block. This field is GSCRTNCD for GDI Select exit routines and GRCRTNCD for GDI Record exit routines.

Table 39 explains the meaning of each return code.

Table 39. GDI exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
4	End of data reached (on FETCH call); not enough room for field/column entries (on DESCRIBE call)
8	Error; terminate function or extract request
12	Error; terminate DataRefresher
16	Error; terminate DataRefresher
Other	Error; terminate DataRefresher

If your GDI exit routine indicates a return code of 8, 12, or 16 after a FETCH or REPOSITN call, DataRefresher immediately issues a CLOSE call to clean up the remnants of processing. However, if your GDI exit routine indicates a return code of 8, 12, or 16 after another type of call, DataRefresher expects the exit routine to do the cleanup.

Although DataRefresher responds to return codes 12 and 16 in the same way, your GDI exit routine could use these two codes to rank the severity of errors and then set different (error) reason codes.

DataRefresher can also communicate with the GDI exit routine via optional reason codes and messages that you define and the exit routine sets in the data communication control block. DataRefresher inserts the codes and messages returned by the GDI exit routine into the messages that DataRefresher issues. See *DataRefresher Messages and Codes* for the text of these DataRefresher messages.

GOI exit routine return codes

GOI exit routines communicate with DataRefresher via return codes placed in the GOIRETCD field of the control block. Table 40 explains the meaning of each return code for each type of call to a GOI exit routine.

Table 40. GOI exit routine return codes

Type of Call	RC = 0	RC = 4	RC = 8
DESCRIBE	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
OPEN	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
CD	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
PUT	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	Error; terminate function or extract request
CLOSE	Successful completion; proceed normally	Successful completion; processing continues, warnings issued	

If the exit routine returns a code that is not listed above, DataRefresher cannot interpret the code and discontinues processing of the batch of extract requests.

Map Capture exit routine return codes

Your Map Capture exit routine communicates with DataRefresher via return codes placed in the MXCRETC field of the control block. Table 41 explains the meaning of each return code.

Table 41. Map Capture exit routine return codes

Return Code	Meaning
0	Successful completion; proceed normally
8	To the UIM, request will not be put in EXTLIB. To the DEM, request will not be executed. To both, processing continues.
16	Error; terminate function or extract request

Appendix C. DataRefresher sample exit routines and control blocks

DataRefresher provides sample control blocks and sample exit routines for your use when writing your own exit routines. These sample exit routines and control blocks are contained in the DVR110.DVRINTER library. To copy or print a specific sample, you can use ISPF facilities, IEBPTPCH, or local facilities.

This appendix contains the names and locations of these control blocks and exit routines.

Control blocks

DataRefresher provides a set of control blocks for each supported programming language (PL/I, and COBOL and Assembler). These control blocks are shipped with DataRefresher for use in writing your exit routines. The control blocks for each programming language are contained in the DVR110.DVRINTER library.

DataRefresher uses the control blocks identified in Table 42 to communicate with the various exit routines. Only the control blocks identified in this table should be used to request or receive services of DataRefresher.

Product Sensitive Programming Interface

Table 42. Control blocks provided with DataRefresher

Exit Routine	PL/I Control Block	COBOL Control Block	Assembler Control Block
Accounting exit	DVRACXP	DVRACXC	DVRACXA
Data exit	DVRDAXP	DVRDAXC	DVRDAXA
Date/Time Conversion exit	DVRXCCXP	DVRXCCXC	DVRXCCXA
GDI Record exit: Data Communication Data Description	DVRXCRCP DVRXCRDP	DVRXCRCC DVRXCRDC	DVRXCRCA DVRXCRDA
GDI Select exit: Data Communication Data Description	DVRXCSCP DVRXCSDP	DVRXCSCC DVRXCSDC	DVRXCSCA DVRXCSDA
Map Capture exit	DVRXCMXP	DVRXCMCC	DVRXCMXA
User Data Type exit	DVRXCUTP	DVRXCUTC	DVRXCUTA
GOI exit	DVRXCGOP	DVRXCGOC	DVRXCGOA

End of Product Sensitive Programming Interface

Sample exit routines

Table 43 lists the sample exit routines contained in the DVR110.DVRINTER library.

Table 43. DataRefresher sample exit routines in the DVR110.DVRINTER library

Exit Routine	PL/I	COBOL	Assembler
Accounting exit	DVRXYACX	DVRXOACX	DVRXAACX
Data exit - conversion	DVRXYDPM	DVRXODPM	DVRXADPM
Data exit - summarization	DVRXYXDS	DVRXOXDS	DVRXAXDS
Date/Time Conversion exit	DVRXYTD	DVRXOTD	DVRXATD
GDI Record exit	DVRXYHIX	DVRXOGRX	
GDI Select exit		DVRXOGSX	DVRXAGSX
Map Capture exit	DVRXYMAP	DVRXOMAP	
User Data Type exit	DVRXYXUT	DVRXOXUT DVRXOXVC	DVRXAXUT
GOI exit	DVRXYGOI	DVRXCGOI	DVRXAGOI

Terms and abbreviations

This glossary defines terms and abbreviations as they are used in the DataRefresher library. Entries often include further information about how the term applies specifically to DataRefresher.

abend. Abnormal end of a task.

access control list (ACL). Defines the access rights to the associated data source in DataRefresher.

access level. The level of authority a user has when using a protected DataRefresher resource or command set.

Accounting exit routine. A user-written routine used for charging resources to individual users. This routine is started at the beginning and the end of the extract request execution cycle. It can be used to change output limits, change priorities of the DEM, and account for the resources used by the DEM on behalf of individual extract requests. This routine can be coded in Assembler, PL/I, or COBOL.

ACL. access control list.

Administrative Dialogs. A series of menus and displays that help a user create and submit data descriptions and extract requests, maintain the profiles for DataRefresher Dialogs and JCL for submitting data descriptions and extract requests, and administer End User Dialogs.

Advanced Program-to-Program Communications (APPC). The communication protocol (LU 6.2) that is used by DataRefresher.

alias. An alternate name for a member of a partitioned data set or for a name of a field described in a data description.

APAR. authorized program analysis report.

APPC. Advanced Program-to-Program Communications.

Application System (AS). An IBM* integrated decision support program that helps provide business planning, graphics, project control and management, statistical data analysis, and other functions.

AS. Application System.

asynchronous. Occurring without a regular or predictable time relationship.

ASCII. ANSI Standard Code for Information Interchange.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

batch message processing (BMP). An IMS/VS region where batch message processing occurs.

BMP. batch message processing.

Boolean expression. In DataRefresher, a conditional expression that evaluates to true or false to determine whether a particular unit of data is extracted. The expression may contain multiple conditions connected by the logical operators AND, OR, and NOT.

card-image input. Input that simulates punched card input (80 columns per record).

CCU. Consistency Check Utility - a DataPropagator* NonRelational feature used with the DataRefresher UIM when using a DataPropagator NonRelational map capture exit in the DataRefresher SUBMIT command.

CEEPIPI. Common Execution Environment Pre-Initialized Program Interface.

child segment. In a database, a segment that lies immediately below its parent segment. A child segment has only one parent segment.

CLIST. command list.

CMS. conversational monitor system.

code page. An assignment of graphic characters and control function meanings to all code points.

code point. A 1-byte code representing one of 256 potential characters.

command list (CLIST). A data set or a member of a partitioned data set containing TSO commands that run sequentially in response to the EXEC command.

---

* DB2, DataPropagator, CICS, Language Environment, SAA, AD/Cycle, VTAM, Systems Application Architecture, and SQL/DS are trademarks of the International Business Machines Corporation.

command string. A language construct that represents one step in a sequence of steps that produce a DataRefresher command.

containing segment. A parent segment that contains one or more internal segments.

control blocks. Storage areas used to hold control information.

conversation. An exclusive use of an LU-to-LU session by two transaction programs using the APPC. This is a short logical connection that lasts only for the duration of one complete transaction. (Contrast with *session*).

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities.

DAP. Dictionary Access Program.

database format. The format of the data prior to any segment preprocessing or data exit manipulation. (Contrast with *FDTLIB format*).

database manager. A program that controls the user's data, ensuring security and data integrity in a multiple user environment. Examples include Information Management System/Virtual Storage (IMS/VS), IBM DATABASE 2 (DB2*), and Structured Query Language/Data System (SQL/DS*).

Database Management System (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

data definition (DD) statement. A job control statement that describes the data sets associated with a specific job step.

data description. The description of a file, IMS/VS DL/I database, or any data source accessed by a user-written generic data interface (GDI) exit.

Data Dictionary. The IBM OS/VS DB/DC Data Dictionary is a central repository of information about data such as names, meaning, relationships to other data, origin, usage, and format.

data entry database (DEDB). An IMS/VS Fast Path database used to provide efficient access to large volumes of detailed data. Each DEDB can be partitioned, or divided into multiple "areas" for ease of access.

Data exit routine. A user-written routine to provide data verification. The routine can be used to process each record in source files and each segment in source databases. This routine can be coded in assembler, PL/I, or COBOL.

DataRefresher. An IBM program that extracts data from a source database or file and formats it for a target database or file.

Data Extract Manager (DEM). The DataRefresher program that extracts data from a VSAM file, a physical sequential file, an IMS/VS DL/I database, or any data source accessed by a generic data interface (GDI) exit.

data facility sort (DFSORT). An IBM product that works in conjunction with DataRefresher or other database products to sort data as it is being processed.

Data Language 1 (DL/I). The database management language for IMS/VS.

data propagation. The process of applying the changes to one set of data to the copy of that data in another database system.

Data Reformat Utility (DRU). This utility recombines 80-character record segments (used to transmit data from one system to another) into logical records.

Date/Time Conversion exit routine. A user-written routine to convert date/time data to ISO format. The routine can be used to process each field containing date/time data. It can be coded in assembler, PL/I, or COBOL.

DBCS. double-byte character set.

DBMS. Database Management System.

DB2. IBM DATABASE 2.

DD. data definition.

DEDB. data entry database.

DEM. Data Extract Manager.

DEM data source. The source data from which the DEM extracts data. (For example, an IMS/VS DL/I database, a VSAM or physical sequential data set, or data accessed by a user-written generic data interface (GDI) exit.)

DFSORT. data facility sort.

Dictionary Access Program (DAP). The DataRefresher program that generates descriptions of the files and nonrelational databases from which users extract data. The descriptions are taken from existing definitions in the IBM Data Dictionary.

DL/I. Data Language I.

double-byte character set (DBCS). A set of characters in which each character occupies 2 bytes. Languages such as Japanese, Chinese, and Korean that contain more symbols thatn can be represented by 256 code points require double-byte character sets. Entering, displaying, and printing DBCS characters requires special hardware and software support.

DRU. Data Reformat Utility.

DataRefresher dialogs. The DataRefresher programs that help users build and send data descriptions (Administrative Dialogs) and extract requests (End User Dialogs and Administrative Dialogs).

DXTFILE. A DataRefresher object stored in the FDTLIB that describes a SAM, VSAM, physical sequential, or other type of data set available by using the generic data interface.

DXTFILE description. Describes a physical sequential file, VSAM file, or any other data source accessed by a generic data interface (GDI) exit. (Describes the file organization and selected fields.)

DXTPCB. DataRefresher Program Communication Block.

DataRefresher Program Communication Block (DXTPCB). A DataRefresher object stored in the FDTLIB that describes an IMS/VS PCB to DataRefresher.

DataRefresher Program Specification Block (DXT PSB). A DataRefresher object stored in the FDTLIB that describes an IMS/VS PSB to DataRefresher.

DXTPSB. DataRefresher Program Specification Block.

DXTPSB description. For a given PCB, a description of the fields of interest to the user and the segments where they exist. Also included in the description of a segment are its length, its format (variable or fixed length), and the name of its parent. Included in the description for a field are its origin in its segment, its length, and its data characteristics.

DataRefresher user data type. Data in a format that DataRefresher does not directly support. A user-written conversion exit is required to convert data from the unsupported format into a format that DataRefresher supports.

DataRefresher user data type description. A description of the user-defined format to DataRefresher.

DXTVIEW. A DataRefresher object stored in the FDTLIB that defines which fields or segments the user may access. A DXTVIEW describes only those segments and fields in a single path of the hierarchy that the user can retrieve.

DXTVIEW description. Defines a DXTVIEW for a physical sequential or VSAM file, an IMS/VS DL/I database, or a data source accessed by a user-written generic data interface (GDI) exit.

EAR. Exit Address Routine.

EBCDIC. extended binary-coded decimal interchange code.

ECF. Enhanced Connectivity Facilities.

End User Dialogs. The DataRefresher program that lets a user build and submit extract requests through a series of panels.

end user table. A DataRefresher table generated to keep track of user IDs between systems.

Enhanced Connectivity Facilities (ECF). A set of programs designed to connect personal computers with host computers so that many host services and resources become available to personal computer users and application programmers.

entry-sequenced data set (ESDS). In VSAM, a file whose records are ordered by time of entry into the data set, and whose relative byte addresses cannot change. Records are retrieved and stored by sequential access, and new records are added at the end of the data set.

ESDS. entry-sequenced data set.

EXEC. A program consisting of a set of CP and CMS commands.

EXEC statement. An instruction within JCL/JCS that identifies the program to be run.

exit address routine. A user-written routine that, given the name of a GDI exit routine by DataRefresher, returns the current address of that GDI exit routine to DataRefresher.

exit routine. A user-written DataRefresher program. DataRefresher passes control to the exit routine for specialized processing.

EXTLIB. extract request library.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters.

extract request. A request to extract data from a source accessible by Data Refresher. Requires use of the SUBMIT and EXTRACT commands.

extract request library (EXTLIB). A VSAM key-sequenced data set (KSDS) that holds extract requests.

Fast Path. The IMS/VS function that supports applications requiring data availability and fast processing of simple data structures. Although Fast Path has its own databases and message processing, it is an integral part of IMS/VS.

FDTLIB. file description table library.

FDTLIB format. The format of a segment after a data exit transforms it, but prior to manipulation by any user data type exit during the extraction process. (Contrast with *database format*).

FDTLIB Migration Utility. A program used to migrate data descriptions from all prior releases of FDTLIB to the current release of FDTLIB.

field definition time. The time at which a user data type is specified in a FIELD statement of a CREATE DXTFILE or CREATE DXTPSB command.

file description table library (FDTLIB). A VSAM key-sequenced data set (KSDS) that holds the descriptions of all databases, files, and DataRefresher views available to the DEM and UIM.

file space. Used by shared file system (SFS), a logical space where a user's files are kept.

file pool. A set of minidisks managed by SFS.

fixed-length record. A record having the same length as all other records with which it is logically or physically associated. (Contrast with *variable-length record*)

full-screen editing. Editing at a display terminal which displays an entire screen of data at once and in which the user can access data through commands or by positioning a cursor.

GDI. generic data interface.

GDI Record exit routine. A user-written program that can access a non-IBM DBMS or data source that does not have an SQL interface, extract from a self-defining file, such as an IXF file, or join data from diverse data sources. DataRefresher does not pass SELECT statements to a GDI Record exit. However, DataRefresher can pass key values to a GDI Record exit. (Contrast with *GDI Select exit routine*).

GDI Select exit routine. A program that a user defines to access a non-IBM DBMS that has an SQL interface, or for help performing a two-stage extraction from a DB2 database. GDI select exits let a user submit an SQL-like SELECT statement to DataRefresher without previously storing field descriptions in the FDTLIB. (Contrast with *GDI record exit*).

General Data Extract feature. This DataRefresher feature (which includes the UIM and the DEM) lets you extract data from an IMS/VS database, or a physical sequential or VSAM file. If you use a generic data interface exit, you can extract data from IBM relational databases without using the REM or from other data sources not directly supported by DataRefresher.

generic data interface (GDI). An interface that accesses MVS databases and files not directly supported by DataRefresher. This source data is accessed via a GDI exit.

generic data interface (GDI) exit routine. A user-written routine that accesses MVS databases, VMS databases, and any files not directly supported by DataRefresher. There are two types of GDI exits: GDI Select exits and GDI Record exits.

generic output interface (GOI) exit routine. A user-written routine that receives extracted data and can be used to convert the data to a user-defined format or written to files not directly supported by DataRefresher.

GOI. generic output interface.

help panel. Information displayed when the user presses the HELP function key while using DataRefresher Dialogs.

hierarchical database. A tree-like, top-down arrangement of segments in a database, beginning at the top of the hierarchy with a root segment and proceeding downward to dependent segments, as in an IMS/VS DL/I database.

high-level language (HLL). A programming language that does not reflect the structure of any particular computer operating system.

High Speed Sequential Retrieval (HSSR). An IMS database tool that delivers efficient access performance to IMS data.

HLL. high-level language.

host. The primary or controlling computer in a multiple computer installation; in this case, the computer running DataRefresher. (Contrast with *remote*).

HSSR. High Speed Sequential Retrieval.

HUP. Hierarchical Update Program - a DataPropagator NonRelational feature used for propagating from a relational source, such as DB2, to a hierarchical target, such as IMS.

IBM Database 2* (DB2*). A program that provides a full-function relational database management system on MVS and supports access from MVS applications under IMS, CICS*, TSO, or batch environments.

IBM software distribution (ISD). The IBM division that distributes IBM programs.

IMS/VS. Information Management System/Virtual Storage.

Information Management System/Virtual Storage (IMS/VS). A database/data communication system capable of managing complex databases and networks.

INGRES* *. A relational database management system that is a product of Relational Technology, Inc.

installation verification procedure (IVP). An IBM program shipped with a product which verifies whether major segments of the product operate correctly following installation of the product.

Integration Exchange Format (IXF). IXF is a self-defining sequential file format providing character and source representation of data that helps applications exchange data.

Interactive System Productivity Facility (ISPF). A program that controls the execution of DataRefresher Dialogs.

internal segment. A repeating group of data within a parent segment; the data can be fixed or variable in length.

ISD. IBM software distribution.

ISPF. Interactive System Productivity Facility.

IVP. installation verification procedure.

IXF. Integration Exchange Format.

JCL. job control language.

JCS. job control statement(s).

JES2. Job Entry Subsystem 2.

JES3. Job Entry Subsystem 3.

job. A set of computer programs, files, and control statements that are sent to the operating system for processing.

job control language (JCL). A control language used to identify a job to an operating system and to describe the job's requirements. In DataRefresher, JCL is the control language used to describe required DataRefresher data resources and to run a Data Refresher job. A DataRefresher job stream includes JCL and Data Refresher commands.

job control statement(s) (JCS). JCS is a set of control statements that identify and describe a job to the operating system for routing and final processing of extracted data. JCS can include both JCL and extracted data. The JCS controls the DRU, any load utility, or any application that processes the extracted data.

Job Entry Subsystem 2 (JES2). An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In an installation with multiple processors, each JES2 processor runs independently. See also *Job Entry Subsystem 3*.

Job Entry Subsystem 3 (JES3). An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In an installation with several loosely coupled processors, JES3 lets the global processor exercise centralized control. See also *Job Entry Subsystem 2*.

job statement. The job control statement that identifies the beginning of a job. It contains such information as the name of the job, account number, and class and priority assigned to the job.

job step. A unit of work represented by running a single program that resides in the load library. A job can consist of one or more steps.

Julian date. A date format that contains the year in positions 1 and 2, and the day in positions 3 through 5. The day is represented as 1 through 366, right-adjusted, with zeros in any unused high-order positions.

Kanji feature. The DataRefresher program that establishes a Kanji-language environment for DataRefresher Dialogs users.

katakana. A character set of symbols used in one of the Japanese phonetic alphabets. The DataRefresher uppercase feature allows the dialogs to be viewed on terminals which support katakana.

key. (1) One or more characters used to identify the record and establish the order of the record within an indexed file. (2) In VSAM, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records.

key-sequenced data set (KSDS). A VSAM file whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by sequential access, and new records are inserted in key sequence by means of distributed free space.

keyword. (1) A part of a DataRefresher command parameter that has a specific meaning to that command

and is shown in uppercase letters in the syntax diagram. See also *parameter*.

KSDS. key-sequenced data set.

LE/370 (Language Environment/370). SAA* AD/Cycle* Language Environment*/370.

Language Environment/370. SAA AD/Cycle Lanaguage Environment/370. A program that allows one high level language to have imbedded calls to routines written in some other high level language.

load utility. A program that puts data into one or more tables in a table space or partition.

Logical Unit (LU). An interface through which a DataRefresher user accesses the SNA network.

logical unit name. The Virtual Telecommunications Access Method (VTAM*) logical unit resource name for initiating communications with the remote node.

LU. Logical Unit.

main storage databases (MSDB). An IMS/VS Fast Path database used to store and provide access to an installation's most frequently used data. The data in an MSDB is stored in segments. Each segment can be available to all terminals, or assigned to a specific terminal. To provide fast access and allow frequent update to this data, MSDBs reside in virtual storage during execution. MSDBs cannot be shared.

Map Capture Communication Area (MCCA). The control block used for communication between DataRefresher (either the UIM or the DEM) and a user-written map capture exit routine.

Map Capture exit routine. A user-written exit routine that can retrieve DataRefresher mapping information for all files and PSBs used during an extract request. This mapping information can be saved for later use, for example, for data propagation.

master index table (MIT). A table that contains administrative information for End User Dialogs. For example, the nickname table and the end user table.

MCCA. Map Capture Communication Area.

minimum segment (MINSEGM). The lowest segment necessary in a hierarchical path to qualify for extraction.

MINSEGM. minimum segment.

MIT. master index table.

MSDB. main storage databases.

MVG. Map Verification and Generation - a DataPropagator NonRelational feature used with the DataRefresher UIM when using a DataPropagator NonRelational map capture exit in the DataRefresher SUBMIT command.

Multiple Virtual Storage (MVS). An IBM operating system that is in an SAA environment.

MVS. Multiple Virtual Storage.

network job entry (NJE). Used in JES2. Allows selected jobs, in-stream (SYSIN) data sets, system output (SYSOUT) data sets, operator commands and messages, and job accounting information to be transmitted from one computer system to another.

network job interface (NJI). Used in JES3. Allows selected jobs, in-stream (SYSIN) data sets, system output (SYSOUT) data sets, operator commands and messages, and job accounting information to be transmitted from one computer system to another.

nickname. In DataRefresher End User Dialogs, a short, convenient name assigned to a specific node or subsystem and the JCL used to route requests to that system.

NJE. network job entry.

NJI. network job interface.

node entry. A name that defines the source and target systems; for example, the node entry in a JCL job control statement or CMSBATCH link.

null separator field. The 1-byte field (2 bytes for IXF output) that begins each data field in an extract output row. DataRefresher puts a hyphen in this field if the data field is null. It is called a separator because it visually separates the data columns in the extract output.

Online DataRefresher commands. A set of TSO REXX EXECs that let a user run DataRefresher commands in the TSO foreground; process UIM, DEM, and REM requests.

ORACLE**. A database management system that is a product of Oracle Corporation.

packed decimal data type. A data type in which each byte in the field except the right-most byte represents two numeric digits. The rightmost digit contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111.

panel. A predefined display image. It may be a menu, a data entry panel, or for information only.

parameter. A keyword, or variable, or a combination of keywords and variables used with a command to affect

its result. In DataRefresher command syntax, required parameters are displayed on the main path of the syntax and optional parameters are displayed below the main path. Default parameters are displayed above the main path of the syntax. See also *keyword*.

parent segment. A segment in a database that has one or more dependent segments below it in a hierarchy.

partial path. A hierarchical path without an occurrence for every segment.

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

PCB. Program Communication Block.

PDS. Partitioned Data Set.

persistent extract. An extract request that is retained in the EXTLIB after an extract runs.

physical segment. The smallest unit of accessible data in a database.

physical sequential file. A file in which records are processed in the order in which they occur in the file.

polling interval. The elapsed time between DEM searches of the EXTLIB. The DEM periodically searches for qualifying extract requests to process. This is used with long-running DEMs.

Program Communication Block (PCB). An object that describes a communication block to a program.

Program Specification Block (PSB). A set of statements naming the required databases, segments to access, and database modification options for a program. The PSB contains a given program communication block (PCB) for each database named, in the sequence used by the program.

Program Support Representative (PSR). An IBM appointed program support technician.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM.

proxy. An account identifier on a VMS operating system (a product of the Digital Equipment Corporation). The proxy provides IBM users with file access and default privileges on the VAX** computer system.

PSB. Program Specification Block.

PSR. Program Support Representative.

PTF. program temporary fix.

QMF. Query Management Facility.

QSAM. queued sequential access method.

Query Management Facility (QMF*). An interactive query product that lets you create reports and charts from relational data.

queued sequential access method (QSAM). A queue containing input data blocks that are awaiting processing or output data blocks that have been processed and are awaiting transfer to either auxiliary storage or to an output device.

RACF. Resource Access Control Facility.

relational database. A database that is organized and accessed according to relationships between data items. Contrast with *hierarchical database*.

relational database view. A relational database view is created by DB2 or SQL/DS and controls access to data on these relational databases.

Relational Data Extract Feature. A DataRefresher feature. The REM is used for extracting data from a DB2 or SQL/DS database.

Relational Extract Manager (REM). The DataRefresher program that extracts data from a DB2 or SQL/DS database.

REM. Relational Extract Manager.

REM data sources. Any DB2 or SQL/DS data accessed by the Relational Extract Manager (REM).

remote. Pertaining to a system, program, or device that is accessed through a telecommunication line. (Contrast with *host*)

remote file description table library (RFDTLIB). A library on a non-IBM computer containing all the DXTFILE and DXTVIEW definitions of data accessible to the non-IBM operating system that are candidates for extracts using DataRefresher.

This feature was used by DXT/D1 but is no longer supported.

Remote Spooling Communications Subsystem (RSCS). The licensed program that allows the VM system to fully participate in a network of SNA/non-SNA Network Job Entry (NJE) System nodes, SNA/non-SNA 3270* Information Display System printer nodes, and Bisync Remote Job Entry (RJE) nodes. This capability permits CMS users to transmit and receive spool files or messages to or from any defined node in the network.

repository. An organized group of information that supports business and data processing activities and provides a single point of control for the management and sharing of that information.

Resource Access Control Facility (RACF). An IBM program that provides for controlled access to system resources by identifying and verifying users to the system, authorizing access to DASD data sets, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected data sets.

RFDTLIB. remote file description table library.

root segment. In IMS/VS, the main segment of a database to which all other segments are related. This is the top of the hierarchy tree.

RSCS. Remote Spooling Communications Subsystem.

run mode. Identifies a DEM as either long-running or terminating. The run mode of a long-running DEM determines how long the DEM should run. When in long-running mode the DEM uses a polling interval.

RUP. Relational Utility Program - a DataPropagator NonRelational feature used with the DataRefresher UIM when using a DataPropagator NonRelational map capture exit in the DataRefresher SUBMIT command.

SAA. Systems Application Architecture*.

SAP. Structures Access Program.

session. A long logical connection that allows communication between two logical units using the APPC. (Contrast with *conversation*)

SFS. shared file system.

SFS Directory. The place where shared file system files are grouped — analogous to a minidisk.

shared file system (SFS). An extension of the CMS file system, that allows simultaneous sharing of CMS programs and data by multiple users and applications.

simple file. A file that contains only one record type. Contrast with *structured file*.

SMP. System Modification Program.

SQL. Structured Query Language.

SQL/DS. Structured Query Language/Data System.

structured file. A file that contains multiple record types or internal segments or both. (Contrast with *simple file*).

Structures Access Program (SAP). A DataRefresher program that employs user-specified data structures to generate DataRefresher data description statements, extract request statements, and a statement specifying the creation of a DB2 table to contain the extracted data.

Structured Query Language (SQL*). A language used to communicate with DB2 and SQL/DS.

Structured Query Language/Data System (SQL/DS*). The relational database management system that runs under VM.

synchronous. Occurring with a regular or predictable time relationship.

System Modification Program (SMP). The program used to install DataRefresher under MVS.

Systems Application Architecture* (SAA*). A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

table. A named collection of data consisting of a number of named vertical rows and a number of unordered horizontal rows which is under the control of a relational database manager.

time sharing option (TSO). An option on the operating system that provides interactive time sharing from a display station.

transaction. (1) A job or a job step. (2) In IMS, a specific set of input data that starts a specific processor job.

translation table. A table used by DataRefresher that provides replacement characters of one code page for characters of a different code page.

TSO. time sharing option.

TSO/E REXX. The implementation of the Systems Application Architecture (SAA) Procedures Language on the MVS system.

TSO foreground. The environment in which programs are swapped in and out of main storage to let terminal users share processing time.

2-stage extraction. The DataRefresher process for extracting data from a non-relational data source. First, the UIM validates and queues the request in EXTLIB; the DEM then executes the request, depending on the schedule that you have established for the DEM. By contrast, extract requests from relational data sources are immediately executed by the REM (one-stage).

UCF. Uppercase Feature.

UIC. user identification code.

UIM. User Input Manager.

Uppercase Feature (UCF). The DataRefresher program that lets users use the DataRefresher Dialogs on Japanese Katakana terminals (555x, 556x).

User Data Type exit routine. A user-written routine that transforms fields in a user-defined format into a data type supported by DataRefresher.

user identification code (UIC). A unique identifier for each user on the VMS system.

User Input Manager (UIM). The DataRefresher program that validates and enters file and database descriptions into the FDTLIB and validates and enters DEM extract requests into the EXTLIB.

value. Information assigned to a parameter associated with a command or keyword.

variable. A part of a DataRefresher command parameter that you supply and is displayed in lowercase letters in the syntax diagram.

variable-length record. A record having a length independent of the length of other records with which it is logically or physically associated. (Contrast with *fixed-length record*).

view. See *DXTVIEW, DXTVIEW description*, or *relational database view*.

Virtual Machine (VM). An IBM system that is part of an SAA environment.

Virtual Memory System (VMS)**. An operating system produced by Digital Equipment Corporation that is used on the VAX computer.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed- and variable-length records on direct access devices. A key field (key sequence) can organize the records in a data set or file logically, in the physical sequence in which they are written (entry sequence), or by relative-record number. DataRefresher does not support extraction from a VSAM relative record data set (RRDS).

VM. Virtual Machine.

VMS. Virtual Memory System.

VSAM. Virtual Storage Access Method.

Index

E

F

G

I

J

L

M

N

P

R

S

T

U

W

Communicating Your Comments to IBM

DataRefresher
Version 1
Exit Routines

Publication No. SH19-6998-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  +353 - 1 - 6614246
- If you prefer to send comments electronically, use this network ID:
  - IBM Mail Exchange: IEIBM3FL at IBMMAIL
  - Internet: IEIBM3FL@IBMMAIL.COM

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

DataRefresher
Version 1
Exit Routines

Publication No. SH19-6998-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH19-6998-00

IBM

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Software Solutions,
Information Development (IISL),
2 Burlington Road,
Dublin 4,
Ireland.

Fold and Tape | **Please do not staple** | Fold and Tape

Program Number: 5696-703

DataRefresher Version 1 Library

SH19-5000	Messages and Codes
LY19-6386	Diagnosis Guide
GH19-6993	An Introduction
SH19-6995	Administration Guide
SH19-6996	MVS and VM User's Guide
SH19-6997	OS/2 User's Guide
SH19-6998	Exit Routines
SH19-6999	Command Reference
GH19-9994	Licensed Program Specifications

SH19-6998-00