Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 422.
## Contents

**Preface** ................................................................. xii
- What is New in Version 3, Release 1 ........................ xii
- Product Changes ....................................................... xii
- Product Library Changes ............................................. xii
**Terms Used in This Book** ........................................... xiii
**What You Should Know** ................................................ xiii
**What is in This Book** ................................................... xiii
**How to Read the Syntax Diagrams** ................................. xiv

### Chapter 1. Introduction ........................................... 1
Segment, Field, and Propagation Exit Routines ....................... 1
- Segment Exit Routine .................................................. 3
- Field Exit Routine ...................................................... 3
- Propagation Exit Routine .............................................. 4
  - Propagation Exit Routine or IMS Data Capture Exit Routine .... 4
- Overview of RUP and Exit Routine Processing ...................... 5
- Overview of the HUP and Exit Routine Processing ................. 7
- Error Handling Logic of Exit Routines ............................. 9
- Exit Routine Relationship to DataRefresher ....................... 9
  - Segment and Field Exit Routines ................................. 9
  - Propagation Exit Routines ......................................... 10
- DB2 Data Capture Subexit Routine .................................. 10
- EKYRESLB Dynamic Allocation Exit Routine ..................... 11
- General Considerations for Exit Routines ......................... 11
  - A TSMF Callable Interface ....................................... 11
  - EMF Callable Interface ........................................... 12
  - User Asynchronous Programs .................................... 12
- Coding Exit Routines in High Level Languages ..................... 13
  - Preinitializing an HLL Environment ............................. 13
  - Specifying LE/370 Runtime Options .............................. 14
    - The //EKYLEOPT DD Statement ................................. 14
    - The TRAP Runtime Option ...................................... 14
  - LE/370 and DPROP Installation .................................. 14
  - Additional Requirements and Recommendations For COBOL ..... 15
  - Additional Requirements and Recommendations For PL/I ....... 15
  - Additional Requirements and Recommendations For C .......... 15

### Chapter 2. Segment Exit Routines ................................. 17
Providing Required Mapping Logic in Segment Exits .................. 19
- Mapping Logic for IMS Segments With No Internal Segments .... 19
  - IMS-to-DPROP Mapping ............................................ 19
  - DPROP-to-IMS Mapping ............................................ 19
- Mapping Logic for IMS Segments .................................... 20
  - IMS-to-DPROP Mapping ............................................ 21
  - DPROP-to-IMS Mapping ............................................ 21
- How To Write A Segment Exit Routine .............................. 23
  - Interface Control Block ......................................... 23
  - IMS DB Segment Buffer ........................................... 24
  - DPROP Segment Buffer ............................................ 25
    - Buffers and Variable-Length Segments ....................... 25
Figures

1. Sequence of Conversion by Segment and Field Exit Routines ........... 2
2. RUP Processing for Generalized Mapping .................................. 6
3. RUP Processing for User Mapping ........................................... 6
4. HUP Processing for Generalized Mapping Logic .......................... 8
5. HUP Processing for User Mapping ........................................... 9
6. Interface Control Block Parameters for Segment Exits .................... 23
7. Interface Control Block for a Segment Exit Routine ..................... 28
8. First Sample Segment Exit Routine (Assembler) ......................... 46
9. DBDGEN Definition ............................................................ 69
10. PSBGEN Definition ............................................................ 69
11. CREATE TABLE Statement .................................................. 70
12. CREATE DXTPSB Statement ................................................ 71
13. CREATE DXTVIEW Statement .............................................. 72
14. DataRefresher UIM SUBMIT Command and EXTRACT Statement ...... 72
15. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements .................................................. 73
16. Defining the PR in the MVG Input Tables .................................. 74
17. Second Sample Segment Exit Routine (COBOL) ......................... 77
18. DBDGEN Definition ............................................................ 88
19. PSBGEN Definition ............................................................ 88
20. CREATE TABLE Statements ................................................. 89
21. Using DataRefresher to Define the PR: CREATE DXTPSB .......... 90
22. Using DataRefresher to Define the PR: CREATE DXTVIEW ....... 91
23. Using DataRefresher to Define the PR: DataRefresher UIM SUBMIT Command and EXTRACT Statement ......................... 92
24. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements .................................................. 93
25. Defining the PR in the MVG Input Tables .................................. 95
26. Third Sample Segment Exit Routine (PL/I) ............................. 98
27. Interface Control Block Parameters for Field Exits .................... 112
28. Interface Control Block for a Field Exit Routine ....................... 115
29. Sample Field Exit Routine (Assembler) .................................. 128
30. DBDGEN Definition ............................................................ 138
31. PSBGEN Definition ............................................................ 138
32. CREATE TABLE Statement .................................................. 139
33. CREATE DATATYPE Definition ............................................ 139
34. CREATE DXTPSB Definition .............................................. 140
35. CREATE DXTVIEW Definition ............................................ 140
36. DataRefresher UIM SUBMIT Command and EXTRACT Statement ... 141
37. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements .................................................. 141
38. Defining the PR in the MVG Input Tables .................................. 143
39. Second Sample Field Exit Routine (COBOL) ......................... 144
40. Interface Control Block for a Propagation Exit Routine .............. 157
41. XPCB and XSDB Control Block Structures ........................... 164
42. Extended Program Communication Block (XPCB) ..................... 166
43. Extended Segment Data Block (XSDB) ................................... 169
44. Exit Routine Action Based on the XPCBPICAL Field Value .......... 170
45. Exit Routine Action Based on the XPCBPICAL, XPCBCALL, and XSDBPHP Field Values .................................................. 171

© Copyright IBM Corp. 1991,2001
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>46.</td>
<td>HEC, QWHS, QWHC, Table Description and Data Row Control Block Structures</td>
</tr>
<tr>
<td>47.</td>
<td>HUP Exit Communication Block</td>
</tr>
<tr>
<td>48.</td>
<td>Table Description and Data Row Control Blocks</td>
</tr>
<tr>
<td>49.</td>
<td>Values of QW0185ST and Their Meanings</td>
</tr>
<tr>
<td>50.</td>
<td>Overview of the Propagation Performed By the Exit Routine</td>
</tr>
<tr>
<td>51.</td>
<td>Mapping of IMS Source Fields to DB2 Target Columns</td>
</tr>
<tr>
<td>52.</td>
<td>First Sample Propagation Exit Routine (Assembler)</td>
</tr>
<tr>
<td>53.</td>
<td>DBDGEN Definition</td>
</tr>
<tr>
<td>54.</td>
<td>CREATE TABLE Statement</td>
</tr>
<tr>
<td>55.</td>
<td>CREATE DXTPSB Statement</td>
</tr>
<tr>
<td>56.</td>
<td>CREATE DXTVIEW Statement</td>
</tr>
<tr>
<td>57.</td>
<td>DataRefresher UIM SUBMIT Command and EXTRACT Statement</td>
</tr>
<tr>
<td>58.</td>
<td>Using DataRefresher For the Extract: INITDEM and USE DXTPSB Control Statements</td>
</tr>
<tr>
<td>59.</td>
<td>DSNTEP2 SQL Statements Required to Define the PR in the MVG Input Tables</td>
</tr>
<tr>
<td>60.</td>
<td>Overview of the Propagation Performed By the Exit Routine</td>
</tr>
<tr>
<td>61.</td>
<td>Mapping IMS Source Fields to DB2 Target Columns</td>
</tr>
<tr>
<td>62.</td>
<td>Second Sample Propagation Exit Routine (C)</td>
</tr>
<tr>
<td>63.</td>
<td>DBDGEN Definition</td>
</tr>
<tr>
<td>64.</td>
<td>CREATE TABLE Statement</td>
</tr>
<tr>
<td>65.</td>
<td>CREATE DXTPSB</td>
</tr>
<tr>
<td>66.</td>
<td>CREATE DXTVIEW Statement</td>
</tr>
<tr>
<td>67.</td>
<td>DataRefresher UIM SUBMIT Command and EXTRACT Statement</td>
</tr>
<tr>
<td>68.</td>
<td>Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements</td>
</tr>
<tr>
<td>69.</td>
<td>DSNTEP2 SQL Statements</td>
</tr>
<tr>
<td>70.</td>
<td>HEC, QWHS, QWHC, Table Description and Data Row Control Block Structures</td>
</tr>
<tr>
<td>71.</td>
<td>HUP Exit Communication Block</td>
</tr>
<tr>
<td>72.</td>
<td>Table Description and Data Row Control Blocks</td>
</tr>
<tr>
<td>73.</td>
<td>Values of QW0185ST and Their Meanings</td>
</tr>
<tr>
<td>74.</td>
<td>Sample DB2 Data Capture Subexit Routine (Assembler)</td>
</tr>
<tr>
<td>75.</td>
<td>DPROPGEN Definition</td>
</tr>
<tr>
<td>76.</td>
<td>CREATE TABLE Statement for Source Table</td>
</tr>
<tr>
<td>77.</td>
<td>CREATE TABLE Statement for Mirror Table</td>
</tr>
<tr>
<td>78.</td>
<td>Interface Control Block for EKYRESLB Dynamic Allocation Exit Routine</td>
</tr>
<tr>
<td>79.</td>
<td>Sample EKYRESLB Dynamic Allocation Exit</td>
</tr>
<tr>
<td>A 80.</td>
<td>Parameters passed to the TSMF callable interface</td>
</tr>
<tr>
<td>A 81.</td>
<td>TSMF Callable Interface, Declarations for PL/I</td>
</tr>
<tr>
<td>A 82.</td>
<td>TSMF Callable Interface, Call from a PL/I Program</td>
</tr>
<tr>
<td>A 83.</td>
<td>TSMF Callable Interface, Declarations for COBOL</td>
</tr>
<tr>
<td>A 84.</td>
<td>TSMF Callable Interface, Call from a COBOL Program</td>
</tr>
<tr>
<td>Q 85.</td>
<td>Parameters passed to the EMF callable interface</td>
</tr>
<tr>
<td>Q 86.</td>
<td>EMF Callable Interface, Declarations for COBOL</td>
</tr>
<tr>
<td>Q 87.</td>
<td>EMF Callable Interface, Call from a COBOL Program</td>
</tr>
<tr>
<td>A 88.</td>
<td>HR Asynchronous Propagation With the IMS Asynchronous Data Capture Function</td>
</tr>
<tr>
<td>A 89.</td>
<td>HR Asynchronous Propagation With a User-Written IMS Data Capture Exit Routine</td>
</tr>
<tr>
<td>A 90.</td>
<td>The XPCB and XSDB Control Block Structure</td>
</tr>
<tr>
<td>A 91.</td>
<td>Extended Program Communication Block (XPCB)</td>
</tr>
<tr>
<td>A 92.</td>
<td>Extended Segment Data block (XSDB)</td>
</tr>
<tr>
<td>Figure</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>93.</td>
<td>INQY ENVIRON Output Area</td>
</tr>
<tr>
<td>94.</td>
<td>Trace Request Block</td>
</tr>
<tr>
<td>95.</td>
<td>Example of Formatted Trace</td>
</tr>
<tr>
<td>96.</td>
<td>Trace Element Descriptor</td>
</tr>
<tr>
<td>97.</td>
<td>COBOL Interface Control Block for a Segment Exit Routine</td>
</tr>
<tr>
<td>98.</td>
<td>PL/I Interface Control Block for a Segment Exit Routine</td>
</tr>
<tr>
<td>99.</td>
<td>C Interface Control Block for a Segment Exit Routine</td>
</tr>
<tr>
<td>100.</td>
<td>COBOL Interface Control Block for a Field Exit Routine</td>
</tr>
<tr>
<td>101.</td>
<td>PL/I Interface Control Block for a Field Exit Routine</td>
</tr>
<tr>
<td>102.</td>
<td>C Interface Control Block for a Field Exit Routine</td>
</tr>
<tr>
<td>103.</td>
<td>COBOL Propagation Exit Interface</td>
</tr>
<tr>
<td>104.</td>
<td>COBOL DL/I Capture Interface</td>
</tr>
<tr>
<td>105.</td>
<td>COBOL HUP Exit Communication Block</td>
</tr>
<tr>
<td>106.</td>
<td>COBOL IFC Copyarea for IFCIDS 0185</td>
</tr>
<tr>
<td>107.</td>
<td>PL/I Propagation Exit Interface</td>
</tr>
<tr>
<td>108.</td>
<td>PL/I (RUP) DL/I Capture Interface</td>
</tr>
<tr>
<td>109.</td>
<td>PL/I HUP Exit Communication Block</td>
</tr>
<tr>
<td>110.</td>
<td>PL/I IFC Copyarea for IFCIDS 0185</td>
</tr>
<tr>
<td>111.</td>
<td>C Propagation Exit Interface</td>
</tr>
<tr>
<td>112.</td>
<td>C (RUP) DL/I Capture Interface</td>
</tr>
<tr>
<td>113.</td>
<td>C HUP Exit Communication Block</td>
</tr>
<tr>
<td>114.</td>
<td>C IFC Copyarea for IFCIDS 0185</td>
</tr>
</tbody>
</table>
Preface

This book explains how to customize IMS DataPropagator (IMS DPROP) and is intended for use by system programmers.

This softcopy book is available only in PDF and BookManager formats. This book is available on the z/OS Software Products Collection Kit, SK3T-4270. You can also get the most current versions of the PDF and BookManager formats by going to the IBM Data Management Tools Web site at www.ibm.com/software/data/db2imstools and linking to the Library page.

What is New in Version 3, Release 1

IMS DataPropagator (IMS DPROP) Version 3, Release 1 presents improvements to both the product and the product library.

This edition, which is available in softcopy format only, includes technical and editorial changes.

Product Changes

IMS DataPropagator V3.1 provides a new, MQSeries-based asynchronous (MQ-ASYNC) propagation of IMS database changes to DB2 tables. With MQ-ASYNC enterprises can implement both:

- Near Real Time Propagation - With Near Real Time propagation, the delay between the update of the IMS database and the update of the DB2 tables can often be as short as a couple of seconds.
- Point-In-Time Propagation - With Point-In-Time propagation, the data content of the DB2 tables matches the IMS database content at a previous clearly identified logical point in time. For example, an enterprise may decide that the content of the DB2 tables will match the following point in times: the logical end of a business day, the logical end of a business month, or the end of a specific IMS jobstream that updated the IMS databases.

Product Library Changes

The Version 3.1 library has been updated with information about MQSeries asynchronous propagation. There are now three Administrators Guides, one for each primary mode of propagation:

- **IMS DPROP Administrators Guide for MQSeries Asynchronous Propagation**
- **IMS DPROP Administrators Guide for Log Asynchronous Propagation**
- **IMS DPROP Administrators Guide for Synchronous Propagation**

There is also a new book, **IMS DataPropagator for z/OS: Concepts**, which provides a conceptual description of data propagation.

Special change indicators are used to identify information that is specific to LOG-ASYNC, MQ-ASYNC, and synchronous propagation:

- **Q** identifies information specific to MQ-ASYNC propagation.
- **A** identifies information specific to LOG-ASYNC propagation.
• S identifies information specific to synchronous propagation.

Terms Used in This Book

The following terms are synonymous in this book:

• File and data set
• DXT and DataRefresher

Unless a specific version or release is referenced, these terms refer to either of the following products:

– DXT Version 2 Release 5
– DataRefresher Version 1 or higher

• Databases that have been quiesced or set to READONLY status.

In all cases, these terms refer to either or both of the following:

– Any propagatable database, except for DEDBs, that has been set to READONLY status.
– DEDBs that have been taken offline with a /DBR command

References to DataRefresher and DXT in this book refer to only host activities.

This book assumes that you will use batch and command statements, not the DataRefresher workstation component.

Selector and Receiver (capitalized) refer to the IMS DPROP Selector and Receiver features. However, selector and receiver (not capitalized) refer to user-created functions.

IMS DPROP books use the term “child” instead of the term “dependent.” For example, IMS DPROP books use the terms “child table” and “child rows” instead of DB2 terms “dependent table” and “dependent rows.” The term “child” is used so that terms for IMS and DB2 are similar.

What You Should Know

This book assumes you understand what data propagation is and the business reasons for propagating data. Information on these topics is in IMS DPROP An Introduction.

This book also assumes you have a basic understanding of IMS, DB2, and DataRefresher concepts and functions.

What is in This Book

The Version 3.1 Customization Guide provides information on how to write exit routines for your IMS DPROP system. It contains sample segment, field, and propagation exit routines that you can use. It also describes how to design and develop programs required to implement asynchronous propagation. The chapters are as follows:

• Chapter 1, “Introduction” on page 1
• Chapter 2, “Segment Exit Routines” on page 17
How to Read the Syntax Diagrams

The following rules apply to the syntax diagrams used in this book:

Arrow symbols

Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

▶ Indicates the beginning of a statement.

▶▶ Indicates that the statement syntax is continued on the next line.

▶▶▶ Indicates that a statement is continued from the previous line.

▶▶▶▶ Indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the ▶ symbol and end with the ▶ symbol.

Conventions

- Keywords, their allowable synonyms, and reserved parameters, appear in uppercase for MVS and OS/2 platforms, and lowercase for UNIX platforms. These items must be entered exactly as shown.

- Variables appear in lowercase italics (for example, column-name). They represent user-defined parameters or suboptions.

- When entering commands, separate parameters and keywords by at least one blank if there is no intervening punctuation.

- Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as given.

- Footnotes are shown by a number in parentheses, for example, (1).

- A symbol indicates one blank position.

Required items

Required items appear on the horizontal line (the main path).
Optional Items
Optional items appear below the main path.

If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

Multiple required or optional items
If you can choose from two or more items, they appear vertically in a stack. If you must choose one of the items, one item of the stack appears on the main path.

If choosing one of the items is optional, the entire stack appears below the main path.

Repeatable items
An arrow returning to the left above the main line indicates that an item can be repeated.

If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can specify more than one of the choices in the stack.

Default keywords
IBM-supplied default keywords appear above the main path, and the remaining choices are shown below the main path. In the parameter list following the syntax diagram, the default choices are underlined.

IMS-specific syntax information
Fragments
Sometimes a diagram must be split into fragments. The fragments are represented by a letter or fragment name, set off like this: |A|. The fragment follows the end of the main diagram. The following example shows the use of a fragment.

```
|STATEMENT|item 1|item 2|A|
```

A:
```
|item 3|KEYWORD|item 5|
```
```
|item 4|
```

Substitution-block
Sometimes a set of several parameters is represented by a substitution-block such as <A>. For example, in the imaginary /VERB command you could enter /VERB LINE 1, /VERB EITHER LINE 1, or /VERB OR LINE 1.

```
/VERB |<A>|LINE|line#|
```

where <A> is:
```
|EITHER|
```
```
|OR|
```

Parameter endings
Parameters with number values end with the symbol ' # ', parameters that are names end with ' name', and parameters that can be generic end with ' * '.

```
/MSVERIFY |MSNAME|msname|SYSID|sysid#|
```

The MSNAME keyword in the example supports a name value and the SYSID keyword supports a number value.
Chapter 1. Introduction

This chapter introduces the routines you can use to customize IMS DPROP. These routines are:

- Segment exit routine
- Field exit routine
- Propagation exit routine
- DB2 Data Capture subexit routine
- EKYRESLB Dynamic Allocation exit routine
- Timestamp marker facility callable interface
- User Asynchronous programs

Information about coding the programs in high-level languages is also included. The rest of this book describes the programs in detail.

Segment, Field, and Propagation Exit Routines

If DPROP mapping and conversion capabilities do not meet your needs, you can use the following exit routines for special situations:

- Segment exit routines
- Field exit routines
- Propagation exit routines

These routines can be written in either Assembler language or one of the following high-level languages: COBOL, PL/I, and C. DPROP support for exit routines written in high-level languages requires Language Environment/370 (LE/370) Version 1 Release 2. See IMS DPROP An Introduction for a description of the software requirements for the LE/370 environment.

Segment and Field exit routines complement the generalized mapping logic of the RUP and HUP. They perform special data formatting that the RUP and HUP do not support. When called, a Segment exit routine reformats an entire IMS segment, while a Field exit routine reformats individual fields in a segment.

Figure 1 on page 2 illustrates the sequence in which the Segment exit routine, Field exit routines, and DPROP conversion routines are invoked by the RUP and HUP.
You may find that your mapping or propagation requirements cannot be handled by combining the generalized mapping logic of DPROP with Segment and Field exit routines. In this case, you may want to use a Propagation exit routine, which lets you substitute your own mapping logic for the generalized mapping logic of the RUP and HUP.

DPROP calls exit routines in both synchronous and LOG-ASYNC modes. During synchronous propagation, the RUP and HUP can call the exit routines from both IMS batch and dependent regions. For generalized mapping cases, Segment and Field exit routines are also called during execution of the Consistency Check utility (CCU) and DPROP DL/I Load utilities (DLU).

If you extract data with DataRefresher, the Segment and Field exit routines are also called by DataRefresher.
Segment Exit Routine

The Segment exit routine is generally used to map an IMS segment between an IMS database format that DPROP does not support and a DPROP-supported format.

The Segment exit routine can change the format and positions of data fields in an IMS segment. It cannot change the format and position of keys, including the concatenated key, nor can it change the format and position of any fields mapped to the primary key of the related DB2 row.

Segment exit routines can:

1. Map IMS segments that have fields with variable starting positions into a segment format where starting positions are fixed.
2. Clean up data, such as data stored in redefined areas of IMS segments.
3. Selectively suppress propagation based on selection criteria programmed into the exit routine. For IMS-to-DB2 propagation, it is preferable, where possible, to selectively suppress propagation by defining a WHERE clause during PR definition.

Segment exit routines used with PRTYPE=L (limited function) PRs are called only for IMS-to-DB2 propagation and must, therefore, support only IMS-to-DPROP mapping.

Segment exit routines used with PRTYPE=E (extended function) PRs must support both IMS-to-DPROP and DPROP-to-IMS mapping, even if the PRTYPE=E PR specifies MAPDIR=HR. This is because your Segment exit routine can be called during CCU and DLU processing to do DPROP-to-IMS mapping. The conversion done during DPROP-to-IMS mapping should be the opposite of the conversion done during IMS-to-DPROP mapping.

If you are using DataRefresher to extract IMS data, the Segment exit routine is also called by DataRefresher as a data type exit routine.

For additional information about the Segment exit routine, see Chapter 2, “Segment Exit Routines” on page 17. For information about data type exit routines, see the appropriate DataRefresher or DXT documentation.

Field Exit Routine

The Field exit routine is generally used to map a field between its IMS database format (referred to as a user format) and a DPROP-supported format.

Field exit routines are used:

1. For IMS segment fields that have special formats not supported by DPROP, and that cannot be converted by the DPROP conversion routines. Examples of such fields are:
   - Date and time formats other than USA, ISO, EUR, and JIS, which must be converted into a standard format
   - Unsigned, packed numeric fields
   - Encoded data, such as a two-byte state code that is to be expanded
When the format of the IMS field cannot be directly converted by DPROP to the format of the DB2 column, such as converting a character format to a numeric format, or converting a character field to a DBCS field.

To convert some values in an IMS field to a DB2 null value.

To change the contents or restructure the data in the field before storing it in the corresponding DB2 table.

To alter the contents of a key field.

When performing DB2-to-IMS propagation, to convert the value of a numeric DB2 column into a packed or zoned IMS field having a sign code other than the "preferred" sign codes X'C' and X'D'.

Field exit routines used with PRTYPE=L PRs are only called for IMS-to-DB2 propagation and must therefore only support user-to-DPROP mapping. They are called to convert an IMS field from your format in the IMS database to the format supported by and defined to DPROP.

Field exit routines used with PRTYPE=E PRs must support both user-to-DPROP mapping and DPROP-to-user mapping, even if the PRTYPE=E PR specifies MAPDIR=HR. This is because your Field exit routine can be called during CCU and DLU processing to do DPROP-to-user mapping. The conversion done during DPROP-to-user mapping should be the opposite of the conversion done during user-to-DPROP mapping.

If you are using DataRefresher to extract IMS data, the Field exit routine is also called by DataRefresher as a data type exit routine.

For additional information about the Field exit routine, see Chapter 3, "Field Exit Routines" on page 110. For information about data type exit routines, see the appropriate DataRefresher or DXT documentation.

**Propagation Exit Routine**

If the DPROP generalized mapping cases cannot be used for propagation, you can supply your own mapping in a Propagation exit routine. Propagation exit routines must provide all necessary mapping logic, build the SQL* calls needed for propagation to DB2, and build the IMS calls needed for propagation to IMS. Neither DataRefresher nor the DLU call Propagation exit routines during the extract/load phase.

For additional information about the Propagation exit routine, see Chapter 4, "Propagation Exit Routines" on page 153.

**Propagation Exit Routine or IMS Data Capture Exit Routine**

Using Propagation exit routines to propagate data from DPROP has some advantages over propagating data using an IMS Data Capture exit routine that you write. These advantages include:

- Propagation debugging support provided by DPROP
- Centralized error handling through the RUP and HUP
- Simplified operation of propagation since DPROP can be used to suspend and restart propagation
Overview of RUP and Exit Routine Processing

For each updated segment occurrence, the RUP is called once by the IMS Data Capture function. A particular segment type can be propagated by zero, one, or several PRs. The number of PRs can be zero if you changed the DBD with an EXIT= keyword, but have not yet generated PRs.

If the updated segment type is propagated by multiple PRs, the RUP will sequentially process these PRs within a single call by the IMS Data Capture function.

For each PR, the RUP checks the PR status to determine if the PR should be processed. Inactive PRs are not processed. Then the RUP determines if the PR specifies a generalized or user mapping case.

For a PR belonging to a generalized mapping case:

1. The RUP calls the optional Segment exit routine. The Segment exit routine converts the IMS segment from its IMS database format to the format supported by and defined to DPROP.

   For some PRs (for example, those defined with a WHERE clause, those propagating IMS segments that contain embedded structures, or those attempting to avoid unnecessary SQL updates by specifying AVU=Y), the Segment exit routine is called twice by the RUP during replace operations: once to convert the segment before replacement, and a second time to convert the segment after replacement.

2. For each field requiring it, the RUP calls the appropriate optional Field exit routine. The Field exit routine converts the field from its user format to the format supported by and defined to DPROP.

3. The RUP converts each field into its DB2 column format.

4. The RUP issues the propagating SQL statement by calling the appropriate SQL update module, which was generated when the PR was defined.

For a PR using a Propagation exit routine (user mapping), the RUP calls the Propagation exit routine. The Propagation exit routine is responsible for all required mapping, conversions, and propagating SQL statements.

Figure 2 on page 6 shows RUP processing for a generalized mapping case, including the relationship with Segment and Field exit routines. Figure 3 on page 6 shows RUP processing for user mapping with a Propagation exit routine.
Figure 2. RUP Processing for Generalized Mapping

Figure 3. RUP Processing for User Mapping
Overview of the HUP and Exit Routine Processing

The HUP runs as the IBM-supplied DB2 Data Capture exit routine. The HUP is called when the DB2 Data Capture function detects that an SQL update changes rows of tables that have been defined with the DATA CAPTURE parameter and when DB2 tracing for MONITOR CLASS(6) is active. You must also set the DB2 system parameter DPROP SUPPORT to 2 or 3, otherwise no call back to IMS occurs.

The HUP obtains all changed rows of these tables from DB2. Then, based on your PR definitions, the HUP determines whether and how the changed rows of a particular table should be propagated.

The HUP checks the PR status to determine if the PR should be processed. Inactive PRs are not processed. Then the HUP determines if the PR specifies a generalized or user mapping case.

For a PR belonging to a generalized mapping case:

1. The HUP converts each DB2 column into the DPROP-supported field format that you specified in the PR definition.

2. For each field requiring it, the HUP calls the appropriate optional Field exit routine. The Field exit routine converts the field from the format supported by and defined to DPROP into its user format.

3. Then the HUP builds the IMS segment search arguments (SSAs) required to access the target IMS database segment.

4. If the IMS target segment needs to be replaced or deleted, the HUP issues an IMS GHU (get hold unique) call to retrieve the segment.

   If the IMS target segment to be replaced is processed by a Segment exit routine, and some fields are not propagated, the HUP initially calls the Segment exit routine. The Segment exit routine must convert the retrieved IMS segment from its IMS database format into the format you defined to DPROP. The conversion done by your Segment exit routine should be the same as the conversion done during RUP calls for IMS-to-DB2 propagation. This processing is used to merge nonpropagated fields in the original IMS segment with the updated fields propagated from DB2.

5. If the target IMS segment will be replaced or inserted, the HUP builds the new segment image. If you have not specified use of a Segment exit routine, the segment image has the format of the IMS database segment. Otherwise, the segment image has the format you defined to DPROP.

   For IMS segments that do not contain propagated internal segments, the HUP builds the image of the IMS segment. In the other cases, the HUP builds the image of either the internal or containing segment.

6. The HUP calls the optional Segment exit routine.

   - If the IMS segment does not contain propagated internal segments, the Segment exit routine converts the IMS segment from the format supported by and defined to DPROP to the IMS database format. The conversion done by your exit routine should be the reverse of the mapping done during RUP calls for IMS-to-DB2 propagation.
If the IMS segment does contain propagated internal segments, the Segment exit routine must merge the internal/containing segment formatted by DPROP to the existing IMS segment that was previously retrieved by the HUP (see item 4 on page 7).

7. The HUP issues the DL/I update calls that propagate the DB2 change.

For a PR using a Propagation exit routine (user mapping), the HUP calls the Propagation exit routine. The Propagation exit routine does all required mapping, conversions, and propagation of DL/I update calls.

After propagation of the changed DB2 row, the HUP calls your optional DB2 Data Capture subexit routine.

Figure 4 shows HUP processing for a generalized mapping case, including the relationship with Field and Segment exit routines. Figure 5 on page 9 shows HUP processing for user mapping with a Propagation exit routine.

---

**Figure 4. HUP Processing for Generalized Mapping Logic**
Error Handling Logic of Exit Routines

The exit routines may encounter error situations. For example, a field defined as numeric may contain nonnumeric data. In this case, the exit routines should use the error handling logic of the RUP and HUP. This practice has the following advantages:

- The error option (ERROPT) in effect is used when the exit routine encounters an error.
- Errors are traced and placed on an audit trail for later review if desired.

To take advantage of the error handling logic of the RUP and HUP, your exit routine should:

- Signal propagation failures to RUP/HUP using a return code in the provided interface control block. Your exit routine should not issue an abend if you want to use the error handling logic of the RUP/HUP.
- Provide error or warning messages in the interface control block to help diagnose the problem. Your exit routine should not issue messages directly.

To change versions of an exit routine, the job step from which the exit routine is called must be stopped and restarted with the new version available.

Exit Routine Relationship to DataRefresher

This section describes the relationship between DataRefresher and the exit routines.

Segment and Field Exit Routines

DPROP calls Segment and Field exit routines during propagation. DataRefresher calls them during the extract/load when extracts are done by DataRefresher. This lets you have identical mapping for extracts done by DataRefresher and propagation done by DPROP. DataRefresher calls these routines data exits and data type exits, respectively.
There are some special restrictions and requirements for exit routines called by DPROP. For example, while the interface control blocks to the exit routines are identical for DPROP and DataRefresher, DPROP does not initialize all of the fields in the control blocks. Another example of these restrictions is that no SYSPRINT DCB is furnished to the exit routine by DPROP. The additional restrictions and requirements are discussed with each type of exit routine in that routine's chapter of this book.

**Propagation Exit Routines**

DataRefresher does not support Propagation exit routines. If you are using Propagation exit routines for user mapping, DataRefresher will not call your Propagation exit during the extract/load phase. If you determine that you can use the mapping capabilities of DataRefresher for the extract/load, the mapping logic of your Propagation exit routine must be compatible with that of DataRefresher. Otherwise, you write your own extract program providing the same mapping logic as your Propagation exit routine.

**DB2 Data Capture Subexit Routine**

If your installation requires that the HUP coexist with another generalized DB2 Data Capture exit routine, consider writing a DB2 Data Capture subexit routine. Instead of having two DB2 Data Capture exit routines (which is not supported by DB2), you would:

- Use the HUP as a DB2 Data Capture exit routine, and

- Define to DPROP the “other” generalized exit routine as a DB2 Data Capture subexit routine (its name is defined during DPROP installation).

The purpose of the subexit routine is usually not DB2-to-IMS propagation. Instead, its purpose is usually to:

- Propagate changed DB2 rows to other tables, or
- Perform other generalized functions, such as auditing changed DB2 rows.

DPROP calls your subexit routine when the DB2 Data Capture function calls the HUP. DPROP calls the subexit routine even if you have not defined a PR and even if propagation has been emergency stopped.

The HUP calls your subexit routine once for each changed row and gives it both the data and the description of the changed row. The HUP calls your subexit routine after processing of all DPROP PRs. However, your subexit routine is not called when the HUP issues a rollback of the unit of work or an abend. This is not a problem since, in this case, the SQL update can be considered nonexistent.

You can write your DB2 Changed Data Capture subexit routine in a high-level language, such as C, COBOL, and PL/I.

For additional information about the DB2 Data Capture Subexit routine, see Chapter 5, “DB2 Data Capture Subexit Routine” on page 259.
You can write an EKYRESLB Dynamic Allocation exit routine to dynamically allocate the APF-authorized library containing DPROP load modules. You can do this if the following methods of allocation are inappropriate for your installation:

- Allocation using an //EKYRESLB DD statement in the JCL of propagating job steps and DPROP utility job steps
- Dynamic allocation by DPROP to a data set name specified during DPROP installation.

Note that you cannot write the EKYRESLB Dynamic Allocation exit routine in a high-level language.

For more information about EKYRESLB Dynamic Allocation exit routine, see Chapter 6, “EKYRESLB Dynamic Allocation Exit Routine” on page 297.

**General Considerations for Exit Routines**

When called during synchronous propagation, the exit routines execute in the same environment as the propagating application program. The exit routines can issue the same IMS calls and SQL statements as the application. However, IMS and DB2 updates issued by the exit routines are not propagated. A possible exception to this would be IMS updates issued during DB2-to-IMS propagation; they can be propagated asynchronously if LOG-ASYNC propagation is based on the IMS Asynchronous Data Capture function. This is because IMS and DB2 calls issued from the IMS or DB2 Data Capture function do not result in recursive calls to Data Capture exit routines. To the IMS and DB2 Data Capture functions, DPROP’s exits appear to run as an extension to the RUP and HUP.

The exit routines must not perform functions incompatible with the environments in which they execute. For example, they should not write to MVS data sets from IMS message processing regions. The exit routines should also avoid using services that can impact the performance of propagating application programs. Examples could include the OPEN macro issued from IMS message processing regions.

DPROP Release 2 supports exit routines both written in Assembler language and with high-level language compilers supporting LE/370 Version 1 Release 2. Exit routines must receive and return control in AMODE 31, but their execution RMODE can be ANY. The addresses of parameters passed by DPROP to the exit routines are 31 bit, and the parameters are usually located above the 16MB line.

**TSMF Callable Interface**

When you are using DPROP LOG-ASYNC propagation, you must set an initial start time to run the Selector. Subsequently, the Selector can determine its own start and stop times. Alternatively, you can specify Group/database start times and group stop times for each Selector run. For details on Selector start and stop times, refer to the appropriate Administrators Guide for your propagation mode.

The timestamp marker facility (TSMF) allows you to specify timestamp markers (TSMs) to be used by the Selector for group/database start times and group stop times. The TSMF can be invoked as a batch job. It can also be invoked through a
callable interface to allow a user application to insert a stop TSM in the Selector
control file for a specified propagation group. The TSMF callable interface is
described in detail in Chapter 7, “TSMF Callable Interface” on page 314.

Note: The TSMF Callable Interface can only be used to create group stop times.

**EMF Callable Interface**

MQ-ASYNC supports both a Near Real Time Propagation and a Point-In-Time
Propagation.

For a Point-In-Time Propagation with MQ-ASYNC, you create Event Markers on the
Source System. Each Event Marker identifies a particular Source System Point In
Time. The Event Markers are transmitted in MQSeries messages, together with the
IMS DB Changes, in First-In-First-Out order to the Apply Programs on the Target
System.

An Apply Program can be instructed to stop its processing when it reads an
MQSeries message containing a specific Event Marker. When an Apply Program is
stopped in this way, the content of the target DB2 tables reflects the Source
System Point-in-Time that has been identified by the creation of the Event Marker.

Usually, the Event Markers are created by running the IMS DPROP 'Capture
System Utility (CUT)'. As an alternative, Event Markers can also be created by
application programs through use of a callable interface.

**User Asynchronous Programs**

IMS DataPropagator Version 3 implements LOG-ASYNC IMS-to-DB2 propagation.
You can implement user LOG-ASYNC IMS-to-DB2 propagation in one of the
following ways:

- Using the IMS Asynchronous Data Capture function. In this case, segment
  updates are written by this function to the IMS log. Later, programs you write
  (often called the “selector”) gather and select the changed data from the IMS
  log data sets. Another program you write (often called the *receiver*) reads the
  changed data and calls DPROP with it. Propagation is then done either by
  DPROP (if you used the generalized mapping cases) or by your Propagation
  exit routine (if you used user mapping). See Chapter 9, “User-Implemented
  Asynchronous Data Propagation (USER-ASYNC)” on page 322 for additional
  information.

- Using a Data Capture exit routine you write (often called the *sender*). In this
  case, segment updates are written by your routine to the:

  - IMS log
  - IMS full-function database
  - DEDB sequential dependent segments
  - MVS flat file

  Another program you write (often called the *receiver*) does the same
  processing as described in the preceding paragraph.

The LOG-ASYNC sender and receiver programs used in these two implementations
are described and illustrated in Chapter 9, “User-Implemented Asynchronous Data
Propagation (USER-ASYNC)” on page 322.
Coding Exit Routines in High Level Languages

All DPROP User exit routines can be written in Assembler. In addition, the following types of exit routines can be written in COBOL, PL/I, and C:

- Segment exit routines
- Field exit routines
- Propagation exit routines
- DB2 Data Capture subexit routine

DPROP support for user exit routines written in high-level languages (HLL) requires the following:

- LE/370 must be installed, and LE/370 modules must be available (through //STEPLIB, //JOBLIB, LINKLIB, or LPA concatenation) to the job steps where the exits are executed.
- Exit routines written in COBOL must be compiled with the SAA AD/Cycle COBOL/370 Version 1 Release 1 (or later releases) compiler.
- Exit routines written in PL/I must be compiled with the LE/370 Version 1 Release 1 (or later releases) compiler.
- Exit routines written in C must be compiled with the LE/370 Version 1 Release 1 (or later releases) compiler.

Preinitializing an HLL Environment

DPROP uses LE/370 support for preinitialization to call exit routines written in high-level languages. LE/370 preinitialization allows the HLL environment to be initialized once and to perform multiple executions of HLL exit routines using this preinitialized environment.

DPROP initialization triggers the preinitialization of the HLL environment. DPROP uses an interface to LE/370 that is similar to the CEEPIPI environment created with an INIT-SUB call. The resulting LE/370 enclave is used to execute all HLL DPROP exit routines.

Then, when one of your exit routines needs to be called, DPROP determines whether it was compiled with one of the previously listed HLL compilers, and proceeds as follows:

- If it was compiled, DPROP calls your exit routine through the CEEPIPI interface module. Your exit routines are called in the LE/370 enclave used for the HLL DPROP exit routines, as opposed to the LE/370 enclave of your propagating application programs. LE/370 treats your exit routines as subroutines, as opposed to main programs.
- If your exit routines were not compiled with one of the above compilers, DPROP assumes that they are Assembler exit routines, and calls them according to Assembler conventions.
- If LE/370 is installed and available through the //JOBLIB, //STEPLIB, linklist, or LPA concatenation, do not provide HLL exit routines compiled with compilers other than those listed above, because the results are unpredictable.
Specifying LE/370 Runtime Options

It is possible that both your exit routines and your applications will be coded in high-level languages. In most cases, each will use LE/370 runtime libraries and options. In such a situation, there are two enclaves: one for the exit, and one for the application. Note that the two enclaves operate under a different set of rules:

- The exit operates under the rules for a routine invoked through CEEPIPI preinitialization.
- The application operates under the LE/370 rules for a main routine.

Additionally, the enclaves are separate in terms of storage use. Each enclave has its own storage, based on installation defaults or overriding runtime options. For performance and storage use, ensure that each enclave has sufficient storage allocations.

The //EKYLEOPT DD Statement

DPROP allows you to specify LE/370 runtime options for the HLL DPROP exit routines' enclave. You can provide the runtime options in the //EKYLEOPT DD statements of the various DPROP job steps (described in IMS DPROP Reference). DPROP provides these runtime options to CEEPIPI during the INIT SUB call. Runtime options provided in //EKYLEOPT override the installation defaults that were defined in the CEEDOPT LE/370 module.

You cannot link runtime options with a particular exit, as you can with an application.

The TRAP Runtime Option

In an environment where both your exit routines and applications are coded in high-level languages, you should be aware of issues concerning the LE/370 TRAP runtime option. This runtime option is a combined ESTAE/ESPIE setting that you can set ON or OFF. If the TRAP option is set ON, a DPROP abend could be trapped by the LE/370 ESTAE/ESPIE mechanism. However, because LE/370 is not in control, a 4036 abend will likely result. In this case, rerun the failing situation with the TRAP option set to OFF to find the underlying abend.

For more information on LE/370, see OS/390 Language Environment Programming Guide. For more information on diagnosing DPROP problems, see IMS DPROP Diagnosis.

LE/370 and DPROP Installation

During DPROP installation, your DPROP system administrator can create a dummy, IEFBR14-type, CEEPIPI load module in your DPROP RESLIB. This is done if DPROP installation occurs before LE/370 installation. Creating the dummy CEEPIPI module prevents a large number of CSV003I messages informing you that the CEEPIPI module was not found. If the dummy CEEPIPI module was copied to the DPROP RESLIB during DPROP installation and you install LE/370 at a later time, do one of the following to enable DPROP to support LE/370:

- Delete the dummy, IEFBR14-type, CEEPIPI load module from the DPROP RESLIB. The LE/370 load modules must be available to DPROP (through //JOBLIB, //STEPLIB, LINKLIB, and LPA concatenation).
- Concatenate the load module library containing LE/370 modules ahead of the DPROP RESLIB in the //JOBLIB, //STEPLIB, LINKLIB or LPA concatenation.
Additional Requirements and Recommendations For COBOL

DPROP exit routines written in COBOL must be compiled with the RENT option.

Additional Requirements and Recommendations For PL/I

The PROCEDURE statement for a PL/I exit must include processing characteristics, as shown in the following example:

OPTIONS (FETCHABLE REENTRANT)

Refer to IBM SAA AD/Cycle PL/I MVS & VM Language Reference for more information.

The sample exit in “Third Sample Segment Exit Routine” on page 97 and the control blocks in Appendix B, “Sample Segment Exit Control Blocks” on page 352, Appendix C, “Sample Field Exit Control Blocks” on page 368, and Appendix D, “Sample Propagation Exit Control Blocks” on page 381 were coded with source code between columns 2 and 72. Column 1 is used for carriage control. Page ejects are inserted to make compiled sample listings more readable. To set this up in your exit routines, specify the MARGINS compiler option when compiling a PL/I exit, as follows:

MARGINS (2,72,1)

Refer to IBM SAA AD/Cycle PL/I MVS & VM Programming Guide for more information.

Additional Requirements and Recommendations For C

To establish correct linkage, all C language exits must include the following PRAGMA:

#PRAGMA LINKAGE (exitname, FETCHABLE)
Where:

EXITNAME

Is the name of the user exit (field, segment, or propagation).

The sample propagation exit routine in “Sample Exit Routine Source Code” on page 235 uses LE/370 callable services (CEETDLI). To call LE/370 callable services, add the following INCLUDE statement to your C language source code:

```
#include "LEAWI.H"
```

Refer to OS/390 Language Environment Programming Reference for additional information about LE/370 callable services.

Carriage control within the sample exit in “Sample Exit Routine Source Code” on page 235 and in the control blocks in Appendix B, “Sample Segment Exit Control Blocks” on page 352, Appendix C, “Sample Field Exit Control Blocks” on page 368, and Appendix D, “Sample Propagation Exit Control Blocks” on page 381 was forced using PRAGMA:

```
#pragma page(1)
```

Page ejects were inserted to make compiled sample listings more readable.
Chapter 2. Segment Exit Routines

The RUP and HUP call a Segment exit routine as part of DPROP’s generalized mapping logic processing. This exit routine is required for TYPE=E PRs that propagate IMS segments containing internal segments; it is optional for other PRs.

A Segment exit routine can be used to reformat or change the segment data during propagation. The RUP’s or HUP’s generalized mapping logic can take care of most situations, but if your data is stored in an unusual way or in some form that the RUP or HUP cannot handle, consider writing a Segment exit routine.

A Segment exit routine converts a segment between an IMS database format that DPROP does not support and the DPROP-supported format that you define in your PR. This is further referenced as:

- **IMS-to-DPROP mapping or normal call** when your exit routine is called to convert the segment from its IMS database format to the DPROP format. Calls to your exit routine for IMS-to-DPROP mapping are generated primarily by the RUP as part of IMS-to-DB2 propagation, and under some circumstances also by the HUP as part of DB2-to-IMS propagation.

- **DPROP-to-IMS mapping or reverse call** when your exit routine is called to convert the segment from its DPROP format to the IMS database format. Calls to your exit routine for DPROP-to-IMS mapping are only generated by the HUP as part of DB2-to-IMS propagation.

The conversion performed during DPROP-to-IMS mapping must be the reverse of the conversion performed during IMS-to-DPROP mapping.

Segment exit routines used with TYPE=L or TYPE=F PRs must support IMS-to-DPROP mapping; they do not need to support DPROP-to-IMS mapping.

Segment exit routines used with TYPE=E PRs must support both IMS-to-DPROP mapping and DPROP-to-IMS mapping, even if the TYPE=E PR specifies MAPDIR=HR. This is because the HUP may call your Segment exit routine and request DPROP-to-IMS mapping during CCU and DLU processing.

A Segment exit routine can be used to:

- Reorganize IMS segments whose fields have variable start positions into a format in which the fields have fixed start positions DPROP does not directly support fields with variable start positions.

- Clean up data stored in an unusual way, or reorganize it before propagation to DB2.

- Suppress the propagation of certain data changes. This subject is discussed in more detail in “Selective Suppression of Data Propagation” on page 44.

- Support propagation of IMS segments containing internal segments (mapping case 3).

The IMS-to-DPROP mapping logic of your Segment exit routine will typically perform one or more of the following functions:

- Artificially construct, in the internal segments, ID fields that uniquely identify each occurrence of the internal segment.
Artificially construct, in the containing IMS segment, a counter field that counts the number of occurrences of an internal segment type within the containing segment (for internal segments whose number of occurrences varies).

The DPROP-to-IMS mapping logic of your Segment exit routine must assemble the IMS segment as it is expected by your IMS applications. The assembly is performed from the containing segment and from multiple internal segments.

Your Segment exit routine does not need to distinguish between propagated and nonpropagated fields; it always receives a complete segment.

You can use a Segment exit routine to change the format, position, or content of fields in a segment before it is propagated to DB2 or stored in the IMS database. **Do not** change the format, position, or content of the segment's key, concatenated key, or any field that maps to the primary DB2 key. Changing these fields results in an error.

If you need to convert field formats that DPROP does not directly support, consider using a Field exit routine instead of (or in combination with) Segment exit routines. Field exit routines are described in Chapter 3, “Field Exit Routines” on page 110.

If you are using a Segment exit routine, the definitions of the field format and position that you provide to DPROP apply to the DPROP segment format. DPROP does not require definitions for the IMS database format of the segment.

All your exit routines can be written in Assembler, or in COBOL, PL/I, or C. DPROP support for exit routines written in high-level languages requires LE/370 Version 1 Release 2. For synchronous propagation, the RUP and HUP call your exits in both IMS batch and online dependent regions accessing DB2. For LOG-ASYNC propagation, the RUP calls your exit routines in an MVS batch environment. During user asynchronous propagation, depending on your implementation, the RUP calls your exit routines in IMS batch and dependent regions accessing DB2, or in a non-IMS DB2/TSO or CAF environment. The RUP and HUP also call your exits during execution of the CCU and DLU.

The DataRefresher term for segment exits is **data exits**. If you are using DataRefresher to extract the IMS data, DataRefresher calls your exit routines during extraction so that the mapping performed during extraction and data propagation is the same.

As shown in Figure 1 on page 2, your Segment exit routine is called by DPROP in the following contexts:

1. During HR propagation, the RUP first calls your Segment exit routine for IMS-to-DPROP mapping, immediately after the segment has been passed by the DL/I Data Capture.

   Your Segment exit routine must convert the segment from its IMS database format (as it is in the IMS database) to the DPROP format that you specified during PR definition.

   After calling your Segment and Field exit routines, the RUP converts the field formats that you specified in your PR definition to the format of the DB2 columns and issues SQL statements (INSERT, UPDATE, or DELETE) to update the DB2 table.
2. During RH propagation, the HUP first converts the format of the DB2 columns into the field format that you specified in the PR definition. Then it calls your optional Field exit routines.

The HUP then calls your Segment exit routine for DPROP-to-IMS mapping, just before performing the update of the IMS database. The Segment exit routine must convert the segment from its DPROP format to its IMS database format.

Your exit routine does not need to distinguish between propagated and nonpropagated fields; it always receives a complete segment from the HUP.

The HUP uses the following logic to provide a complete segment when only a subset of the fields are propagated: The HUP retrieves the existing IMS segment (if it exists) from the IMS database and calls your Segment exit routine to perform IMS-to-DPROP mapping of the existing IMS segment. If the IMS segment does not exist, the HUP initializes a segment in its DPROP format by setting nonpropagated fields to the default value associated with their data type (for example, zero or blank) or to binary zeroes (for space in the segment that was not explicitly defined to DPROP as fields). The HUP then merges the updated DB2 data with nonpropagated fields of the existing or initialized IMS segment; this results in a complete segment in its DPROP format.

Providing Required Mapping Logic in Segment Exits

The mapping logic provided by a Segment exit routine is usually straightforward, especially if the Segment exit routine does not support IMS segments containing internal segments that are propagated by mapping case 3 PRs. In this case, the Segment exit routine must convert the segment between its IMS database format (as it is in the IMS database) and its DPROP format (as defined to DPROP during PR definition).

Mapping Logic for IMS Segments With No Internal Segments

This section describes the mapping logic for IMS segments that do not contain internal segments.

**IMS-to-DPROP Mapping**

For IMS-to-DPROP mapping, when your exit routine is entered, a buffer contains the segment in its IMS format. Your exit routine must convert the segment to its DPROP format, and place it in another buffer.

**DPROP-to-IMS Mapping**

For DPROP-to-IMS mapping, when your segment routine is entered, a buffer contains the segment in its DPROP format, as it was mapped (according to the PR definition) by DPROP from the changed DB2 row. Your exit routine must convert the segment to its IMS format, and place it in another buffer.

The segment, in its DPROP format, is built by DPROP before calling your exit routine, as follows:

- For propagated fields, the value of the DB2 column is converted to the IMS field's DPROP format.

  If Field exit routines were defined, the Field exit routines are called to convert the fields from their DPROP format to their user format.
Nonpropagated IMS fields are initialized in the DPROP segment format as follows:

- For replace and delete operations, nonpropagated fields are initialized with their current value.
- For insert operations, nonpropagated fields are initialized to the default value associated with their data type (for example, zero or blank) or to binary zeroes (for space in the segment that was not explicitly defined to DPROP as fields).

Mapping Logic for IMS Segments

This section discusses mapping logic for IMS segments with internal segments.

When designing a Segment exit routine for IMS segments containing one or more internal segment types, consider the following:

1. Each internal segment type is propagated by a mapping case 3 PR to/from a different table.
   Fields of the IMS segment that are not located in any internal segment can be propagated by a mapping case 1 or 2 PR to/from another table, if performing DB2-to-IMS propagation, propagation of these other fields is required.
   Specify use of the same Segment exit routine when you define all these PRs. This is to avoid propagation failures resulting from inconsistent mapping. Consequently, your segment exit routine will typically be called during the processing of both mapping case 3 and mapping case 1 or 2 PRs.

2. The Segment exit routine is specified at the level of the IMS segment, not at the level of the internal segment. The output of the Segment exit routine is therefore an entire IMS segment, not an individual internal segment.

3. A Segment exit routine is required for the propagation of an IMS segment containing internal segments with TYPE=E PRs. It is optional for TYPE=L and TYPE=F PRs.

4. During IMS-to-DB2 propagation your segment exit routine will be called for IMS-to-DPROP mapping.
   During DB2-to-IMS propagation, your Segment exit routine is called primarily for DPROP-to-IMS mapping, and, in some circumstances, for IMS-to-DPROP mapping.

5. When called for IMS-to-DPROP mapping, your Segment exit routine always gets as input the entire IMS segment in its IMS format. Your Segment exit routine must then return the entire IMS segment in its DPROP format.

6. When called for DPROP-to-IMS mapping, your Segment exit routine must distinguish between two cases:
   a. Sometimes, your exit routine is called during the processing of a mapping case 3 PR propagating a table change to an occurrence of an internal segment. In this case your Segment exit routine gets both of the following as input:
      - The internal segment, in its DPROP format, as mapped by DPROP from the changed DB2 row
      - The entire IMS segment, in its IMS format as it exists in the database before propagation
Your Segment exit routine must then return the modified IMS segment, in its IMS format. This is done by merging the changed internal segment occurrence in the pre-existing IMS segment.

b. Other times, your exit routine is called during the processing of the mapping case 1 or 2 PR propagating a table change to the containing IMS segment. In this case your Segment exit routine gets both of the following as input:

- The IMS segment, in its DPROP format, as mapped by DPROP from the changed DB2 row
- The entire IMS segment (if it exists), in its IMS format as it exists in the database before the propagation

Your Segment exit routine must then return the modified IMS segment in its IMS format.

**IMS-to-DPROP Mapping**

For IMS-to-DPROP mapping, when your Segment exit routine is entered, a buffer contains the segment in its IMS format. Your exit routine must convert the segment to its DPROP format and place it in another buffer.

Make sure that the IMS-to-DPROP mapping logic of your exit routine creates a DPROP segment format that matches the PR definition. Be sure that:

- For each internal segment type defined as having a variable number of occurrences, the containing segment in its DPROP format has a count field. If such a count field does not exist in the IMS format, your exit routine must construct the count field in the DPROP format.
- Each internal segment type contains one or more ID fields that uniquely identify each occurrence of the internal segment type within its containing segment. If the ID fields do not exist in the IMS format, your exit routine must construct the ID fields in the DPROP format.
- The start position of the first occurrence of an internal segment type and the length of each internal segment occurrence exactly match the PR definitions.

**DPROP-to-IMS Mapping**

Sometimes, your Segment exit routine is called for the processing of a mapping case 3 PR propagating a table change to an internal segment. Other times, your Segment exit routine is called for the processing of the mapping case 1 or mapping case 2 PR propagating a table change to the containing IMS segment. Your Segment exit routine must provide logic for both types of calls (as explained in the information on page 36, your exit routine can distinguish between the two types of calls by testing the value provided by DPROP in the DAXSEGT field).

**Mapping logic when propagating to an internal segment**: This section discusses mapping logic when propagating to an internal segment with a mapping case 3 PR.

A mapping case 3 PR propagates a table change to an internal segment. When propagating this table change to IMS, DPROP provides the following information when entering your exit routine:
• The internal segment, in its DPROP format, as DPROP mapped it (according to the mapping case 3 PR definition) from the changed DB2 row. See below for a description of how DPROP builds it.

• The before-change IMS segment, in its IMS format, as it exists in the IMS database before propagation.

Your exit routine must merge (insert, delete, or replace) the changed internal segment occurrence into the existing IMS segment, and construct the changed IMS segment in its IMS format.

Before calling your exit routine, DPROP builds the internal segment that is provided as input to your exit routine. DPROP builds the segment as follows:

• For propagated fields, the value of the DB2 column is converted to the DPROP format of the IMS field.

  If Field exit routines were defined, the Field exit routines are called to convert the fields from their DPROP format to their user format.

• Nonpropagated IMS fields of the changed internal segment occurrence are initialized in the DPROP segment format as follows:
  – For replace and delete operations, they are initialized with their current value.
  – For insert operations, they are initialized with the default value associated with their data type (for example, zero or blank); or with binary zeroes (for space in the segment that was not explicitly defined to DPROP as fields).

**Mapping logic when propagating to containing segment:** This section discusses mapping logic when propagating to the containing segment with a mapping case 1 or mapping case 2 PR.

A mapping case 1 or mapping case 2 PR propagates a table change to a containing IMS segment. When called to propagate this table change to IMS, your exit routine receives the following information from DPROP upon entry:

• The containing IMS segment, in its DPROP format, as mapped by DPROP from the changed DB2 row according to the mapping case 1 or 2 PR definition. See below for a description of how DPROP builds it.

• The before-change IMS segment, in its IMS format, as it exists in the IMS database before propagation (it is provided only if the DB2 change is a replace or delete).

Your exit routine must return the new or changed IMS segment, in its IMS format, in another area.

The containing IMS segment, in its DPROP format, provided as input to your exit routine, is built as follows by DPROP before calling your exit routine:

• For fields that are propagated by the mapping case 1 or 2 PR (but not for fields propagated by mapping case 3 PRs), the value of the DB2 column is converted to the DPROP format of the IMS field.

  If Field exit routines were defined, the Field exit routines are called to convert the fields from their DPROP format to their user format.

• Fields that are not propagated by the mapping case 1 or 2 PR are initialized in the DPROP segment format as follows:
For replace and delete operations, they are initialized with their current value.

For insert operations, they are initialized with binary zeroes for fields located in internal segments and for space in the segment that was not explicitly defined to DPROP as fields. Other fields that are not propagated by the mapping case 1 or 2 PR are initialized with the default value associated with their data type (for example, zero or blank).

How To Write A Segment Exit Routine

This section describes some guidelines and requirements for writing a Segment exit routine to be used with DPROP. If DataRefresher uses your exit routine for data extraction, it must also conform to these requirements.

As mentioned above, your exit routine can be written in Assembler, COBOL, PL/I, or C when LE/370 Version 1 Release 2 is installed. When the RUP and HUP call your Segment exit routine, they pass the following four parameters to the exit:

- An Interface Control Block
- An IMS DB segment buffer
- A DPROP segment buffer
- A 64-byte anchor area

Note: When calling your exit routine for DPROP-to-IMS mapping, DPROP provides to your exit routine one additional segment buffer. This additional buffer contains the before-change and existing IMS segment in its IMS format. This additional buffer is not provided as a call parameter; instead, the buffer is pointed to by the DAXIDDSB field of the interface control block.

If your exit routine is written in Assembler, register 1 contains the address of the list of parameter addresses. This list is four fullwords long and contains the addresses of the parameters in the order listed above. If your exit routine is written in a high-level language supported by LE/370 Version 1 Release 2, then it must include the appropriate mapping definitions to access the four parameters being passed to it.

Interface Control Block

Figure 7 on page 28 shows the structure of the interface control block, EKYRCDAX, that is passed to your Segment exit routine. There is one interface control block per exit routine, lasting the duration of the exit in virtual storage. The following table lists:

- The fields most useful to your exit routine
- What the fields are used for
- Their displacement into the control block DSECT

<table>
<thead>
<tr>
<th>Field</th>
<th>Used For</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAXCALL</td>
<td>Call function</td>
<td>X'20'</td>
</tr>
<tr>
<td>DAXDBNM</td>
<td>Name of IMS database currently in use</td>
<td>X'9C'</td>
</tr>
<tr>
<td>DAXSEGM</td>
<td>Name of physical segment type</td>
<td>X'4C'</td>
</tr>
</tbody>
</table>
The interface control block has the same structure as the control block DataRefresher passes to its data exits. A more complete description of these fields is included in the copy of the control block DSECT shown in Figure 7 on page 28.

Of the fields listed above, the following can be changed by your Segment exit routine:

- DAXRETC
- DAXSMESG
- DAXSCRT1
- DAXENTRD
- DAXINCTL

Altering any of the other fields in the control block causes an error.

**IMS DB Segment Buffer**

The IMS DB Segment Buffer contains the segment in its IMS format.

- When performing IMS-to-DPROP mapping, DPROP or DataRefresher provides the IMS segment to your Segment exit routine in this buffer. Until now, there has been no processing of the segment, so it appears as it does in the IMS database.
Your Segment exit routine must not modify this buffer when it is called to perform IMS-to-DPROP mapping.

- When performing DPROP-to-IMS mapping, your Segment exit routine must provide the segment to DPROP in this buffer. The segment must be provided in its IMS format. This is the same IMS format as provided by DPROP to your exit routine when performing IMS-to-DPROP mapping.

  When called for DPROP-to-IMS mapping, this buffer is empty at entry to the Segment exit routine.

See “Buffers and Variable-Length Segments” for notes about variable length segment.

**DPROP Segment Buffer**

The DPROP Segment buffer contains the segment in the DPROP-supported format that you identified during your PR definition.

- *When performing IMS-to-DPROP mapping*, your exit must place the transformed segment into this buffer before returning to DPROP. When your exit routine returns, DPROP reads this buffer to get the transformed segment.

  You can return the segment in either fixed-length or variable-length format, depending on what you specified in your PR. This does not depend on whether the segment was fixed or variable in the IMS DB segment buffer. See “Buffers and Variable-Length Segments” for notes on variable-length segments.

- *When performing DPROP-to-IMS mapping*, the HUP provides in this buffer the segment to your Segment exit routine. This segment is in the DPROP-supported format that you specified during the PR definition; it is in either fixed-length or variable-length format.

  1. For an IMS segment containing imbedded structures, the segment is either the containing IMS segment or one of the internal segments whose name can be found in the DAXISEGM field of the interface control block. Refer to “DPROP-to-IMS Mapping” on page 21 for more details.

  2. For all other segment types, this buffer contains the complete IMS segment; the name in the DAXISEGM field is the same as the name of the physical segment type in DAXSEGEM.

This buffer always contains an entire internal or IMS segment that has:

- Propagated fields mapped from the changed DB2 table row
- Nonpropagated fields either mapped from an existing IMS segment image or set to their initial values

**Buffers and Variable-Length Segments**

- For variable-length IMS segments, the first two bytes contain the length field, followed by the segment data. The number in the length field includes the length of the segment data plus the two bytes of the length field itself. For example, if the segment data is 18 bytes long, the length field is set at 20 bytes, or X’14’.

To understand how DPROP sets the length of IMS segments during DB2-to-IMS propagation, refer to the appropriate *Administrators Guide* for your propagation mode.
• For variable-length internal segments, the length of the internal segment is not necessarily in the first two bytes. Instead, the length of a variable-length internal segment is provided by the HUP to your Segment exit routine in the DAXFLEN field. The HUP determines the length of the internal segment based on the PR definition. Remember that during PR definition, you specify the length of a variable-length internal segment on the NEXT=fieldname+n keyword of the SEGMENT statement of the DXTPCB (for PRs defined with DataRefresher), or the fieldname+n value of the NEXT column of the DPRISEG table (for PRs defined without DataRefresher).

Before-Change IMS DB Segment Buffer
The Before-Change IMS DB segment buffer exists only when the Segment exit routine is called to perform DPROP-to-IMS mapping and there is an existing segment in the IMS database. This is the case for:

• All segment types when the IMS update to be performed is a DLET or REPL
• Internal segment types for all types of updates

The Before-Change IMS DB segment buffer contains the IMS DB segment (in its IMS format) as it currently exists in the IMS database, before propagation of the DB2 change. This buffer is pointed to by the DAXIDDSB field of the interface control block.

Your Segment exit routine must not modify this buffer.

Although your exit always receives this buffer when there is an existing IMS segment image, it is only important when performing DPROP-to-IMS mapping of an IMS segment containing internal segments. In the other cases, it is recommended that your exit routine ignore this buffer.

See “Buffers and Variable-Length Segments” on page 25 for notes on variable-length segments.

64-Byte Anchor Area
DPROP gives you 64 bytes as a general-purpose storage area. Each exit routine has its own unique anchor area. You can use it for whatever you want. Initially, the area is set to all binary zeros, and DPROP (or DataRefresher if you are using it) never changes it again.

The anchor area exists in virtual storage, and remains yours for the duration of the exit, as follows:

• For IMS batch and BMP regions, the anchor area lasts for the duration of the application program.
• For MPP regions, the anchor area lasts for the duration of the IMS Program Controller Subtask. This can span multiple MPP executions.
• For CCU and DLU executions, the anchor area lasts for the duration of the job step.
• For LOG-ASYNC propagation and user asynchronous propagation, the anchor area lasts for the duration of the MVS task being used by the receiver program to call the RUP.
Interface Control Block DSECT

You can generate the following DSECT in your assembler exit routine by coding the EKYRCDAX macro statement. For HLL exit routines, you can include or copy one of the following members to map the Segment exit routine Interface Control Block:

- **EKYRCDXC** For exit routines written in COBOL
- **EKYRCDXP** For exit routines written in PL/I
- **EKYRCDXK** For exit routines written in C

Figure 7 on page 28 shows the interface control block, followed by detailed descriptions of its fields.
Figure 7 (Part 1 of 6). Interface Control Block for a Segment Exit Routine
Figure 7 (Part 2 of 6). Interface Control Block for a Segment Exit Routine
Figure 7 (Part 3 of 6). Interface Control Block for a Segment Exit Routine

**Figure 7 (Part 3 of 6). Interface Control Block for a Segment Exit Routine**

- `127+DAXKFBAD DS AL4` PTR TO SEGMENT'S FULLY
- `129+DAXKFBAD DS AL4` CONCAT KEY (IF DL/I).
- `130+DAXKFBAD DS AL4` IF CALLER IS DPROP:
- `131+DAXKFBAD DS AL4` - 0, IF 'NOKEY' HAS BEEN
- `132+DAXKFBAD DS AL4` SPECIFIED ON EXIT=
- `133+DAXKFBAD DS AL4` OF DBDGEN.
- `134+DAXKFBAD DS AL4`

- `135+DAXKFBAD DS AL4` LENGTH OF SEGMENT'S FULLY
- `136+DAXKFBAD DS AL4` CONCAT KEY (IF DL/I)
- `137+DAXKFBAD DS AL4` IF DPROP: 0, IF 'NOKEY' HAS BEEN
- `138+DAXKFBAD DS AL4` SPECIFIED ON EXIT=
- `139+DAXKFBAD DS AL4` OF DBDGEN.

- `140+DAXKFBAD DS AL4`

- `141+DAXKFBAD DS AL4` LENGTH OF IMS DB SEGMENT BUFFER
- `142+DAXKFBAD DS AL4`
- `143+DAXKFBAD DS AL4`

- `144+DAXKFBAD DS AL4` LENGTH OF DPROP SEGMENT BUFFER
- `145+DAXKFBAD DS AL4`

- `146+DAXKFBAD DS AL4` DAXSYSPR DS AL4 ONLY
- `147+DAXKFBAD DS AL4` POINTER TO SYSPRINT DCB (EXIT

- `148+DAXKFBAD DS AL4` MAY WISH TO RECORD INFORMATION
- `149+DAXKFBAD DS AL4` IN SYSPRINT VIA "PUT"
- `150+DAXKFBAD DS AL4` DCB FACTS: LRECL=121,
- `151+DAXKFBAD DS AL4` NO CARRIAGE CONTROL CHAR
- `152+DAXKFBAD DS AL4`

- `153+DAXKFBAD DS AL4` ENVIRONMENT SUBFIELDS
- `154+DAXKFBAD DS AL4` OPERATING SYSTEM:
- `155+DAXKFBAD DS AL4` =C'ESA ' IF MVS/ESA
- `156+DAXKFBAD DS AL4` =C'XA ' IF MVS/XA
- `157+DAXKFBAD DS AL4` =C'MVS ' IF MVS
- `158+DAXKFBAD DS AL4`

- `159+DAXKFBAD DS AL4` DB/DC ENVIRONMENT:
- `160+DAXKFBAD DS AL4` =C'BAT ' IF IMS BATCH/BMP
- `161+DAXKFBAD DS AL4` =C'MPP ' IF IMS MPP
- `162+DAXKFBAD DS AL4` =C'IFP ' IF FAST PATH
- `163+DAXKFBAD DS AL4` =C'CICS ' IF CICS
- `164+DAXKFBAD DS AL4` =C' ' IF NONE OF ABOVE.
- `165+DAXKFBAD DS AL4`

- `166+DAXKFBAD DS AL4` CALLING PROGRAM:
- `167+DAXKFBAD DS AL4` =C'DXT ' IF DataRefresher
- `168+DAXKFBAD DS AL4` =C'DPRS ' IF DPROP SYNCH PROP
- `169+DAXKFBAD DS AL4` =C'DPRA ' IF DPROP ASYNCH PROP
- `170+DAXKFBAD DS AL4` =C'DPRC ' IF DPROP CCU PROP
- `171+DAXKFBAD DS AL4` =C'DPRL ' IF DPROP DLU
- `172+DAXKFBAD DS AL4`

- `173+DAXKFBAD DS AL4` NAME OF THIS EXIT ROUTINE
- `174+DAXKFBAD DS AL4`

- `175+DAXKFBAD DS AL4` NAME OF IMS DATABASE
- `176+DAXKFBAD DS AL4` IF CALLER IS DPROP:
- `177+DAXKFBAD DS AL4` - NAME OF PHYSICAL DBD.
- `178+DAXKFBAD DS AL4` IF CALLER IS DXT:
- `179+DAXKFBAD DS AL4` - NAME OF USED DBD (CAN BE
- `180+DAXKFBAD DS AL4` NAME OF A PHYSICAL OR
- `181+DAXKFBAD DS AL4` LOGICAL DBD)
- `182+DAXKFBAD DS AL4`

- `183+DAXKFBAD DS AL4` RESERVED
- `184+DAXKFBAD DS AL4`

- `185+DAXKFBAD DS AL4` NUMBER OF DAXASEGS ARRAY
- `186+DAXKFBAD DS AL4` ELEMENTS CONTAINING
- `187+DAXKFBAD DS AL4` ANCESTOR SEGM INFORMATION
Figure 7 (Part 4 of 6). Interface Control Block for a Segment Exit Routine
Figure 7 (Part 5 of 6). Interface Control Block for a Segment Exit Routine
Interface Control Block Field Descriptions

The following list contains detailed descriptions of the fields in the interface control block. The primary descriptions given are for DPROP unless otherwise indicated. Additional descriptions are given for DataRefresher.

Some of the fields are not useful to your exit routine when DPROP calls it. These fields are for DataRefresher only, both in the interface control block and below.

**DAXTNAME**  Contains the constant `DVRXCDAX`, used to identify the control block in a storage dump.

**DAXCALL**  The call function that describes what action your exit routine must perform. This field can have the following values:

- **NO**  Normal (IMS-to-DPROP mapping). The exit routine is called to convert the segment from its IMS to its DPROP (or DataRefresher) format described in the PR definition. **NO** calls can be generated by both the RUP and HUP during propagation, and by DataRefresher DEM during extract.
RV  Reverse (DPROP-to-IMS mapping). The exit routine is called to convert the segment from the DPROP (or DataRefresher) format described in the PR definition to the IMS format. RV calls are generated only by the HUP.

ED  End of data (DataRefresher only). The exit routine is called to perform end of data summary functions. A DataRefresher user must request this function with a EODCALL=Y keyword on the SEGMENT statement of the DXTPSB.

RE  Return (DataRefresher only). The exit routine is called during data extract after returning with a return code of 4. DPROP does not support return code 4 and return calls. Your exit must not use return code 4 with DataRefresher if the extracted data is propagated using a generalized mapping case. This results in different mapping for the DataRefresher extract and propagation by DPROP, which causes inconsistencies in the propagated data.

DAXDATYP  Contains the constant DL, indicating that the data being mapped is an IMS segment.

DAXFIL  If called by DPROP, the name of the DBPCB used for the updating IMS call. This field is filled for both NO and RV calls. If called by DataRefresher, it contains the name of the DBPCB used for extract.

DAXPSB  If called by DPROP, the name of the PSB used for the program that modified the IMS data. This field is only specified for NO calls. If called by DataRefresher, it contains the name of the PSB used for DataRefresher DEM.

DAXDBNM  If called by DPROP, the name of the physical IMS DBD.

If called by DataRefresher, the name of the DBD referenced in the PCB of the PSB used for the DataRefresher DEM. In this case, the DBD can be either physical or logical. For DataRefresher users, if the exit routine is called for segments propagated by DPROP, it is recommended that you refer to physical DBDs in the PCB.

DAXSEGM  If called by DPROP, the name of the IMS segment type as specified in the physical IMS DBD.

If called by DataRefresher, it is the segment type found on the SEGMENT statement of the DXTPSB. For DataRefresher users, it is recommended that you specify the same segment names on the SEGMENT statements as in the physical IMS DBD.

DAXKFBAD  The address of the segment's fully concatenated key. Remember that your exit routine must not modify this key. The address can be zero if the segment has no fully concatenated key, or if the key was not supplied to the RUP (for example, if the NOKEY option was used in the EXIT= keyword of the DBD).

DAXKFBBLN  The length of the fully concatenated key. The length can be zero if the segment has no fully concatenated key, or if the key was not supplied to the RUP.

DAXDLEN  If called by DPROP for a NO call (IMS-to-DPROP mapping), the length of the IMS DB segment.

If called by DPROP for an RV call (DPROP-to-IMS mapping), this
field contains the length of the IMS DB segment buffer. For RV calls, this buffer contains the result of Segment exit routine processing; do not store a segment in the IMS DB segment buffer that is longer than the length specified in DAXDLEN. This can cause storage overlays and unpredictable results. If the segment in its IMS format is a variable-length segment, the segment exit must store the actual length of the segment in the first two bytes of the IMS segment buffer.

If called by DataRefresher, this field contains the length specified in the BYTES= keyword of the SEGMENT statement in the CREATE DXTPSB control statement. For variable length IMS DB segments, the actual length is found in the first two bytes of the buffer.

**DAXFLEN**

If called by DPROP for a NO call (IMS-to-DPROP mapping), the length of the DPROP segment buffer. For NO calls, this buffer contains the result from Segment exit routine processing; do not store a segment in the DPROP segment buffer that is longer than the length specified in DAXFLEN. This can cause storage overlays and unpredictable results. If the PR defines the segment in its DPROP format as a variable-length segment, the actual length of the segment must be stored by the segment exit in the first two bytes of the DPROP segment buffer.

If called by DPROP for an RV call (DPROP-to-IMS mapping), DAXFLEN contains the length of the segment in its DPROP format.

**DAXOPSYS**

Contains the constant **ESA**, indicating that the program is running in an MVS environment.

**DAXTRANS**

Contains a value describing the environment in which the exit routine is called. This field can have the following values:

- **BAT**  
  IMS Batch or BMP environment
- **MPP**  
  IMS MPP environment
- **IFP**  
  IMS Fast Path environment
- **CICS**  
  CICS environment

If the exit is called in an environment other than those listed above, the value consists of blanks.

**DAXPROGM**

Contains information about the calling program, either DPROP or DataRefresher. This field can have the following values:

- **DPRS**  
  Called by DPROP during synchronous propagation
- **DPRA**  
  Called by DPROP during LOG-ASYNC propagation and user asynchronous propagation
- **DPRC**  
  Called by DPROP during CCU execution
- **DPRL**  
  Called by DPROP during DLU execution
- **DataRefresher**  
  Called by DataRefresher

**DAXEXIT**

The load module name of the Segment exit routine.

**DAXDPRCT**

Contains a value describing the type of IMS or DB2 update performed. This field can contain the values **ISRT**, **REPL**, and **DLET** for the insert, replace, or delete of an IMS segment or DB2 row, respectively. The field is set only when DPROP calls the exit routine. If the exit is called for CCU or DLU processing, the value of the field is set to **ISRT** because the DPROP logic simulates an insert during CCU and DLU processing.
DAXREPL  This field is only set when the exit routine is called by DPROP during processing of a Replace. The field specifies whether the exit routine is being called to process the after-image (A) of the segment, or the before-image (B) of the segment.

DAXSEGT  This field describes which type of segment is being processed.
For NO calls (IMS-to-DPROP mapping), the possible values are:
U  The segment being processed is the IMS segment being updated.
A  The segment being processed is a physical ancestor of the IMS segment being updated. The value can be set to A when processing a PR propagating path-data located in an ancestor segment.

For RV calls (DPROP-to-IMS mapping), the possible values are:
U  The segment located in the DPROP segment buffer is the containing IMS segment.
I  The segment located in the DPROP segment buffer is an internal segment. The value can be set to I when processing mapping case 3 PRs. The name of the internal segment type being processed is located in DAXISEGM.

The field is set only when the exit routine is called by DPROP.

DAXPSUP  This field indicates whether the Segment exit routine can request suppression of data propagation during its current call. This field is set only when the exit routine is called by DPROP.
If your Segment exit routine is designed to support propagation suppression, it must test this field to determine if it can suppress propagation.
N  The exit routine cannot request suppression of data propagation.
Y  The exit routine can request suppression of data propagation.
This field is set to Y only if:
- The PR definition specified PROPSUP=Y.
- Other conditions are met (for example, if the current call of the exit is not for the before-image of a segment).

DAXISEGM  This field is only set for RV calls (DPROP-to-IMS mapping) and contains the name of the segment to be processed. For internal segments (mapping case 3), this is the name of an internal segment; for any other case, it contains the name of the physical IMS segment and is the same as in the DAXSEGM field above.

DAXIDDSSB  This field is only set for RV calls (DPROP-to-IMS mapping) and contains a pointer to a buffer, or zero. The buffer contains the before-change IMS DB segment (in its IMS format). The size of the before-change IMS DB segment is provided in DAXIDDSL.
The buffer must not be modified by your exit routine.
This pointer is only present if the type of update in DAXDPRCT is either REPL or DLET, or if the segment to be processed is a mapping case 3 internal segment.

The buffer pointed to by DAXIDDSB is only important when performing DPROP-to-IMS mapping of an IMS segment containing internal segments. In this case, your exit routine requires the following two inputs:

- The DPROP segment buffer. It contains either an internal segment (if it is the target table of an internal segment that has changed) or the IMS segment (in all other cases) in its DPROP format.
- The buffer pointed to by DAXIDDSB. It contains the before-change copy of the IMS segment, as stored in the IMS DB.

Your Segment exit routine must then use this input to assemble the after-change copy of the IMS segment (in its IMS format). The assembled IMS segment must be returned in the IMS DB segment Buffer.

If the IMS segment is variable length, the first two bytes contain the length field, followed by the segment data. The number in the length field includes the length of the segment data plus the two bytes of the length field itself.

**DAXIDDSL**  
The length of the segment in the before-change IMS DB segment buffer.

The next two fields are switches that can be useful for problem determination. DPROP and DataRefresher do not require your exit routine to set these fields. However, they can help you determine where a problem occurred if you have an ABEND. DPROP and DataRefresher set these fields to blanks before calling your exit routine for the first time.

**DAXENTRD**  
Exit-entered flag.

As you enter your exit routine, set this field to X. DPROP does not change this field again, so if a problem occurs, you can determine if your exit has been entered.

**DAXINCTL**  
Exit-in-control flag.

You can also set this field to X, indicating that your exit routine has control. When DPROP regains control, it resets this field to blank, so you can determine if your exit routine has control when an ABEND occurs.

The next two fields can be used along with the RUP's and HUP's error handling logic. For more information on return codes and error handling techniques, see “Return Codes and Error Handling Techniques” on page 39.

**DAXRETC**  
The return code that the exit routine provides when returning to its caller. This field is set to zero when the exit routine is called.

**DAXSMESG**  
User-provided error message. It is set to blanks when the exit routine is called. When the exit routine returns, if the field is not blank, DPROP or DataRefresher writes the contents of the field.
DPROP prefaces the message with the number EKYR980I or EKYR981E, and writes the message according to its usual error handling logic (for example, to the OS/VS console, trace data set, //EKYPRINT, or the Audit trail). DataRefresher prefaces the message with the number DVRA_50, (where _ is one of several possible digits) and writes the message to the //SYSPRINT data set.

**DAXSCRT1** An exit routine work space for your own use; for example, to save information across calls to the exit routine. Before the first call to your exit routine, DPROP initializes this space to binary zeros, and does not modify it again.

**Exit Routine Processing**

Using the information given above, your Segment exit routine can copy the propagated data from a buffer, transform it, and return it using another buffer.

When called for IMS-to-DPROP mapping (with NO in DAXCALL), your Segment exit routine can read the segment from the IMS DB segment buffer and return it in the DPROP segment buffer after transformation.

When called for DPROP-to-IMS mapping (with RV in DAXCALL), your Segment exit routine reads the segment from the DPROP segment buffer and returns it in the IMS DB segment buffer after transformation.

There are, however, some restrictions and guidelines to follow when developing your exit routine:

- When DPROP calls it, your exit routine always gets control in AMODE 31, and must return control in AMODE 31. Keywords DPROP passes to your exit are usually located above the 16MB line. The exit routine is loaded above or below the 16MB line, depending on the RMODE attribute of the exit load module.

  It is recommended that you code and link-edit your program as reentrant. To simplify programming, DPROP provides work spaces in your exit routines, in the interface control block, and the 64-byte anchor area.

- If your exit routine is written in Assembler language, DPROP uses standard OS/VS conventions when calling your exit routine.
  - Register 1 points to the parameter list described above.
  - Register 13 contains the address of a register save area.
  - Register 14 contains the return address.
  - Register 15 contains the entry point address of the exit routine.

  Upon entry, the exit routine must save the register contents into the save area that the caller provides. If your exit routine calls other routines that use standard MVS linkage conventions, it must also provide a save area of its own. The exit routine must return to its caller using normal OS/VS conventions after restoring the registers. A return code must be provided in the interface control block, not in register 15.

- Your Segment exit routine must never change the content or the displacement of the key field of the propagated IMS segment. Do not change the fully concatenated key, the address of which is in DAXKFBAD in the interface control block. and when called for DPROP-to-IMS mapping, your exit routine must **not** change the DPROP segment buffer,
When called for IMS-to-DPROP mapping, your exit routine must not change the IMS DB segment buffer, which contains a copy of the propagated segment.

If you map an IMS field that is not in the fully concatenated key to a column of the DB2 primary key, observe the following rules:

For a TYPE=E PR, your exit routine must not change the content or displacement of this field.

If the DPROP format of the segment is variable length, this field must be contained in the DPROP format of the segment that your exit routine returns to DPROP during IMS-to-DPROP mapping.

Because the exit routine for synchronous propagation runs in the same environment as the propagating application program, it can generate the same type of IMS calls and SQL statements as the application program. However, for LOG-ASYNC propagation and user asynchronous propagation using the TSO-Attach or CAF-Attach, the exit routines do not execute in an IMS environment, and cannot generate IMS calls. Therefore, it may be preferable to generate only SQL statements.

If your exit generates IMS calls, then use the AIB interface described in IMS/ESA Application Programming: DL/I Calls, which allows your exit routine to generate calls without the address of the IMS PCBs.

During synchronous propagation, IMS and DB2 update calls, made from within your exit routine, are not propagated synchronously (but can be propagated asynchronously, if you implement LOG-ASYNC propagation or user asynchronous propagation).

Exclude the PCBs your exit routine uses from the list passed to the application program upon entry. You can avoid changing the application program if you need to add PCBs that your exit routine uses exclusively. Refer to IMS/ESA Utilities Reference: System for more details.

A Segment exit routine must not perform functions that are not supported by the environment in which it is running. For example, an exit routine running in an MPP region must not WRITE to OS files; also, the exit routine must not generate STIMER macros in an IMS environment.

For performance reasons, your exit routine should generate static rather than dynamic SQL statements. Avoid using functions that have a detrimental impact on the performance of the propagating program (such as performing an OPEN and CLOSE on an OS/VS file each time the exit routine is called).

Return Codes and Error Handling Techniques

This section discusses how to return from your exit routine to DPROP, including return codes and error handling techniques.

Return Codes

The following list describes the return codes that you can set when returning from your Segment exit routine to the RUP or HUP. To set the return code, place it in the DAXRETC field in the interface control block. The RUP and HUP read this field when they regain control.

Returning with any code other than those on the list is considered an error and results in an ABEND, regardless of the error option (that is, even with ERROPT=IGNORE in effect).
0 Used for normal returns.

4 This return code is not supported by DPROP. Returning this code to the RUP or HUP causes it to ABEND. While DataRefresher supports this return code, your exit must not return this code to DataRefresher while processing a segment that is propagated by the DPROP generalized mapping logic. This can result in DataRefresher mapping during the extract that is not consistent with the DPROP mapping during propagation. This can result in propagation failures.

8 This return code causes DPROP to suppress propagation of the changed data segment. The exit must be specifically allowed to use this code during PR generation. The exit routine must not return with return code 8 when DAXPSUP was set to N (for example, the Segment exit routine is processing the before-image of a segment, or processing an ancestor of the changed IMS segment). For more information about suppressing data propagation, see “Selective Suppression of Data Propagation” on page 44.

If your exit routine uses this return code with DataRefresher, the current occurrence of the segment is not extracted.

12 DPROP interprets this return code as a failure indication. This prevents propagation of the changed data, and DPROP proceeds with its error logic.

If ERROPT=BACKOUT is in effect, for synchronous propagation, the RUP or HUP backs out the propagating application. If ERROPT=BACKOUT is in effect for LOG-ASYNC propagation, the Receiver terminates with an error message. For user asynchronous propagation, CCU or DLU execution, the RUP and HUP return to the caller with an error. DPROP uses its error reporting logic to write diagnosis information.

If ERROPT=IGNORE is in effect, the RUP and HUP do not perform propagation, and return to the caller without performing a backout and without providing any error indication to the caller. However, if this occurs during CCU or DLU execution, the RUP and HUP return to the CCU or DLU with an error. DPROP uses its error reporting logic to write diagnosis information.

If the exit returns to DataRefresher with this return code, DataRefresher terminates the extract requests currently being processed.

16 Return code 16 signals a severe error. DPROP does not propagate the changed data, but generates an ABEND. Returning this code to DataRefresher causes it to terminate the DEM.

Error Handling Techniques
When your exit routine encounters an error, it is strongly recommended that your exit routine take advantage of the standard error handling logic of DPROP. In the interface control block, you can supply a return code in DAXRETC, and an error message in DAXSMESG. You must not return an error message in DAXSMESG without providing an error return code (12 or 16), because this can create many console messages.

By supplying DPROP with an error return code and message, you gain many advantages. When an exit returns with an error return code, the RUP and HUP trace or snap the data and the control blocks involved in the interface. The exits are included in the standardized error handling scheme of DPROP. This scheme:

- Determines the difference between ERROPT=BACKOUT and ERROPT=IGNORE
Is different for propagation and CCU or DLU execution

Protects against sending too many messages to the MVS consoles

DPROP writes your error message using its standard message writing logic: WTO, trace data set (the IMS log, the //EKYLOG data set, or the //EKYTRACE data set), and AUDIT trail.

If the exit routine generates its own messages or ABENDs, the RUP and HUP cannot include the exit routine in their standardized error handling, or guard against sending numerous messages to the MVS consoles. Therefore, it is not recommended that your exit routine generate its own messages or ABENDs when an error occurs.

Saving Information Across Calls

You can save information across calls to the exit routine. Save the information either in the 64-byte anchor area or in the DAXSCRT1 field of the interface control block. If these areas are not large enough, you can generate a GETMAIN and save the address of the storage in either of these areas.

DPROP and DataRefresher treat the interface control block in slightly different ways. In DPROP, there is one interface control block per exit routine (lasting for the duration of the MVS task), while in DataRefresher, there is one interface control block per segment type (lasting for the duration of the extract request). If DataRefresher and DPROP call the exit routine, and are sensitive to this difference, you can use the anchor area to save information across calls. DPROP and DataRefresher handle this area the same way: there is one anchor area per exit, and it lasts for the duration of the exit in virtual storage.

Updating Your Segment Exit Routine

DPROP does not provide any online change logic to replace an existing load module copy of your segment exit routine with a new version of the load module. If you need to change your exit routine, then stop the affected IMS regions, DPROP asynchronous Receiver or any user asynchronous receiver programs before performing the change. A change of the exit routine without stopping the IMS regions or receiver programs can cause unpredictable results. For example, some MPP regions can use the new version of the exit routine, while other regions use the old version. After the change, you can restart the IMS regions.

Tracing Your Exit Routine

DPROP provides a trace facility that can assist you in detecting errors in your exit routines. You can activate the DPROP trace facility by providing a TRACE control statement in the //EKYIN data set of the job step where your exit routine runs. For synchronous propagation, you can also activate tracing by calling the SCU with a TRACE ON control statement.

If you include debug level 2 on the TRACE or TRACE ON statements, the trace output includes the changed IMS segment and the propagating SQL statements for HR propagation, or the changed DB2 row and the propagating DL/I call, including the IMS segment data, for RH propagation.

If you include debug level 4 on the TRACE or TRACE ON statements, each time the exit routine returns to DPROP, the trace output includes:
• The contents of the interface control block
• The IMS DB segment buffer
• The DPROP segment buffer
• The 64-Byte anchor area
• For DPROP-to-IMS mapping, the old image of the IMS segment (located in the buffer pointed to by DAXIDDSB)

This information is automatically included in the RUP or HUP trace information when a propagation failure occurs, even if you have not activated the DPROP trace.

If you include debug level 8 on the TRACE or TRACE ON statements, the trace output includes a record of each call to, and each return from, an exit routine.

Two other debugging aids, located in the interface control block, are:
• The exit-entered flag
• The exit-in-control flag

In a dump, these flags help you determine if your exit routine is in control at the time of a failure.

Differences Between Exit Routine Calls From DPROP or DataRefresher
This section summarizes the differences between calling your Segment exit routine from DPROP and calling it from DataRefresher.

• DPROP does not call the exit routine with ED or RE calls.

• DPROP does not support return code 4. If DataRefresher/DEM calls the exit routine for a segment that is propagated using generalized mapping logic, the exit routine must not return a return code 4 to DataRefresher/DEM.

• When DPROP calls the exit routine, there is one interface control block per exit routine, lasting for the duration of the IMS Program Controller MVS Subtask. When DataRefresher calls it, there is one interface control block per segment type, lasting for the duration of the extract request.

If your exit routine must save information across calls and is sensitive to this difference, you can use the 64-byte anchor area to save information. This area is treated the same by both DPROP and DataRefresher.

• The following fields in the interface control block are not set when DPROP calls the exit routine; a brief explanation of any consequences this has is included.
  
  **DAXPCBAD**  The exit routine cannot access the DB PCB used for the updating IMS calls.

  **DAXPCBLS**  The exit routine cannot access the list of DB PCBs. If the exit routine needs to issue IMS calls, it must use the IMS AIB interface.

  **DAXSYSPR**  The exit routine cannot write to the SYSPRINT file.

  **DAXASGNO**  This is the number of array elements in DAXASEGS.

  **DAXASEGS**  This is the array of names of ancestor segments.

  • The RV call is generated only by DPROP for DPROP-to-IMS transformation and is not used by DataRefresher.
- DAXDPRCT is only set when DPROP calls the exit routine.
- DAXISEGM is only set when DPROP calls the exit routine for DPROP-to-IMS mapping.
- DPROP support for exit routines written in high-level languages requires LE/370 Version 1 Release 2.

Refer to the appropriate DataRefresher or DXT documentation for information about DataRefresher trace facilities.

## Telling DPROP About Your Segment Exit Routine

This section discusses how to inform DPROP that you want to use a Segment exit routine. The procedure depends on how you are entering your PR.

### PRs Entered Through DataRefresher UIM

If you are entering the PR through DataRefresher UIM, you must provide the following keyword operands on the SEGMENT statement of the DXTPSB:

- Specify the load module name of the exit routine on the **EXIT**= keyword.
- Specify the fixed or maximum length of the segment, in its DPROP format, on the **XBYTES**= keyword.
- Specify whether the DPROP segment format is fixed with **FORMAT=F**, or variable with **FORMAT=V**.

You can also specify that your exit routine be allowed to suppress propagation of an update by returning a return code of 8. You specify this by coding a **PROPSUP=Y** value on the MVGUPARM keyword of the DataRefresher SUBMIT control statement. Specifying **PROPSUP=N** prohibits your exit routine from returning a return code of 8.

### PRs Entered Into the MVG Input Tables

If you are entering your PR information directly into the Mapping Verification and Generation (MVG) input tables, without using DataRefresher, you use the DPRISEG (or SEG) table to inform DPROP about your exit routine. The SEG table is one of the MVG input tables. There are three columns in the table that you must specify:

| SEGEXIT | The name of your Segment exit routine. It can be up to eight characters long. It must be alphanumeric, and begin with an alphabetic character. |
| SEGEXITL | The length, in bytes, of the segment in its DPROP format. The length must be specified as an integer. If the segment length is variable, use the maximum length. |
| SEGEXITF | The format of the segment in its DPROP format. If the segment is fixed length, place an **F** in this column. If the segment is variable length, place a **V** in this column. |

SEGEUIT and SEGEUITF describe the segment in the DPROP segment buffer and in the DPROP-supported format. This format is for IMS-to-DPROP mapping the output of the exit routine, and for DPROP-to-IMS mapping the input to your exit routine.
You must specify values for all three of these columns to use your exit routine. If either the segment length or the format is entered, the MVGU checks to make sure you have also entered the name of the exit routine.

To specify that your exit routine be allowed to suppress propagation of an update by returning a return code of 8, use the PROPSUP column of the DPRIPR MVG input table. Place a Y in the column to allow suppression. Place an N in the column to prohibit suppression.

**Selective Suppression of Data Propagation**

Your Segment exit routine can selectively suppress data propagation. Propagation is suppressed when your exit routine returns a return code of 8 to DPROP. This means that your exit can analyze the changed data segment and, based on your requirements, tell DPROP whether or not to propagate the change to your DB2 table or IMS database. For example, this can be used to suppress the propagation of IMS delete calls, turning your propagated copy into a kind of archive that contains data for longer periods than the source data. If you use a return code of 8, DPROP does not propagate the data, but continues with its normal processing.

This section describes how to set up selective suppression.

Your Segment exit routine must not return with return code 8 when DAXPSUP was set to N (for example, because the Segment exit routine is processing the before-image of a segment, or processing an ancestor of the changed IMS segment).

To indicate that you want to allow a return code of 8 to be used, you must specify the PROPSUP parameter as PROPSUP=Y. The default for this parameter is PROPSUP=N, which means that a return code of 8 is not allowed.

If you are using DataRefresher to code your PRs, specify this PROPSUP parameter in the MAPUPARM operand of the DataRefresher SUBMIT statement. If you are using the MVG input tables to code your PRs, the PROPSUP parameter is specified in the MVGIPR Table.

When you specify PROPSUP=Y, it is recorded in the mapping table. This can be useful for problem determination. If the database administrator (DBA) finds an inconsistency between IMS and DB2 data, the DBA can check the mapping table to see if it is caused by a return code of 8 from a Segment exit routine.

Be very careful when using selective suppression. The inconsistencies that it creates can result in future propagation failures. For example, an SQL INSERT can fail if the original DELETE statement was not propagated to DB2. Also, selective suppression can make the CCU useless, because the IMS and DB2 data are no longer consistent.

You can retain some of the usefulness of the CCU with the USE keyword in the CCU CHECK statement. If you are suppressing delete calls before they are propagated to your DB2 table, you can create a view of the DB2 table that excludes the undeleted rows during the CCU read phase. For more information, see *IMS DPROP Reference*.

For HR propagation, you can also selectively suppress propagation through definition of a WHERE clause during PR definition. If you can choose between specifying a WHERE clause and suppressing with a Segment exit routine, choose
the WHERE clause approach. Using the WHERE clause does not cause inconsistencies, and does not restrict the usefulness of the CCU.

Mapping case 2 propagates multiple segment types to or from one table. Suppression of propagation of the entity segment does not automatically suppress propagation of the extension segments (RH propagation of the delete of the entity segment is an exception; this also suppresses deletion of the extension segments). Therefore, if you provide a Segment exit routine that suppresses the propagation of the entity segment, you must also provide Segment exit routines that suppress the propagation of the extension segments. This is important for avoiding propagation failures.

First Sample Segment Exit Routine

Figure 8 on page 46 is an example of a Segment exit routine. The Segment exit routine transforms a segment between its IMS format and its DPROP format. The IMS format contains fields with variable start positions. In the DPROP format, all of the fields have fixed start positions.

In this example, the first two fields in the IMS format of the segment are fixed length and contain the segment key. The last three fields, however, are variable length, containing a last name, first name, and city. It is assumed that each of the variable length fields has a maximum length, and a variable start position within the segment in its IMS format.

When it receives a changed IMS data segment and is called for IMS-to-DPROP mapping, the exit routine transforms it into a DPROP-supported format, and returns the segment to the RUP for propagation to DB2.

When it is called for DPROP-to-IMS mapping, the exit routine transforms the DPROP format into the IMS format and returns the segment to the HUP for propagation to IMS.

The source code shown in Figure 8 on page 46 is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYESE1A. Following the source code are definitions related to the sample Segment exit routine.
** Figure 8 (Part 1 of 23). First Sample Segment Exit Routine (Assembler) **

---

* **START OF SPECIFICATIONS**

** **

* **MODULE NAME = EKYESE1A**

** **

* **DESCRIPTIVE NAME = SAMPLE 'SEGMENT USER EXIT ROUTINE'**

** **

* **STATUS: V1 R2 M**

** **

* **FUNCTION = EKYESE1A IS A SAMPLE DPROP**

** **

* **'SEGMENT USER EXIT ROUTINE' AND ILLUSTRATES**

** **

* **THE TRANSFORMATION OF A SEGMENT LAYOUT BETWEEN ITS:**

** **

* **- 'DL/I DB FORMAT'**

** **

* **- 'DPROP FORMAT'.**

** **

* **EKYESE1A ILLUSTRATES ONE OF THE MOST TYPICAL USAGE**

** **

* **OF DPROP SEGMENT USER EXITS: THE TRANSFORMATION OF:**

** **

* **- A VARIABLE LENGTH DL/I SEGMENT WITH FIELDS**

** **

* **HAVING VARIABLE START POSITIONS**

** **

* **INTO**

** **

* **- A SEGMENT LAYOUT WHERE ALL FIELDS HAVE A FIXED**

** **

* **START POSITION.**

** **

* **(DL/I SEGMENTS WITH FIELDS HAVING VARIABLE START**

** **

* **POSITIONS CAN BE SUPPORTED BY THE 'GENERALIZED**

** **

* **MAPPING LOGIC' OF DPROP V1R2, ONLY IF A SEGMENT**

** **

* **USER EXIT ROUTINE TRANSFORMS THE SEGMENT INTO A**

** **

* **FORMAT WHERE ALL FIELDS HAVE A FIXED START POSITION.**

** **

* **IF DB2 TO IMS OR TWO WAY PROPAGATION IS IN EFFECT,**

** **

* **THEN THE SEGMENT USER EXIT ROUTINE MUST ALSO BE ABLE**

** **

* **TO TRANSFORM SUCH A SEGMENT FROM A FORMAT WHERE ALL**

** **

* **FIELDS HAVE FIXED START POSITION (THE DPROP FORMAT)**

** **

* **TO A FORMAT WITH VARIABLE START POSITIONS (THE DL/I**

** **

* **SEGMENT FORMAT))**

** **

* **THIS SAMPLE ASSUMES THAT IN ITS DL/I DB FORMAT:**

** **

* **1) THE FIRST PORTION OF THE SEGMENT HAS A**

** **

* **FIXED FORMAT CONTAINING THE KEY OF THE**

** **

* **SEGMENT.**

** **

* **2) THE SECOND PORTION OF THE SEGMENT CONSISTS OF**

** **

* **THREE ADJACENT PAIR OF:**

** **

* **(LENGTH FIELD, VARIABLE LENGTH FIELD)**

** **

* **FOR THE FAMILY-NAME, FIRST-NAME, AND CITY.**

** **

* **WITH THE EXCEPTION OF THE FIRST PAIR OF**

** **

* **LENGTH FIELD AND VARIABLE LENGTH FIELD:**

** **

* **THESE PAIR OF LENGTH FIELDS AND VARIABLE LENGTH**

** **

* **FIELDS HAVE A VARIABLE START POSITION.**

** **

* **EACH VARIABLE LENGTH FIELD IS ASSUMED TO HAVE A**

** **

* **SPECIFIC MAXIMUM LENGTH.**

** **

* **THE FIGURE BELOW PROVIDES AN OVERVIEW OF**

** **

* **THE TRANSFORMATION PERFORMED BY THIS SAMPLE EXIT.**

** **

* **THE LEFT-HAND SIDE DESCRIBES THE SEGMENT SEG1 IN ITS**

** **

* **DL/I DB FORMAT. THE FIGURE PROVIDES FOR EACH FIELD**

** **

* **LOCATED IN THE SEGMENT:**

** **

* **- THE FIELD NAME**

** **

* **- THE FORMAT OF THE FIELD**

** **

* **'H' STANDS FOR 'HALFWORD BINARY' FORMAT.**

** **

* **'C' STANDS FOR 'FIXED LENGTH CHARACTER FORMAT'**

** **

* **'VC' STANDS FOR 'VARIABLE LENGTH CHARACTER FORMAT'**

** **

* **THE FIXED START POSITION OF THE FIELD (IF THE**

** **

* **FIELD HAS A FIXED START POSITION) OR 'V' IF**

** **

* **THE FIELD HAS A VARIABLE START POSITION.**
THE RIGHT-HAND SIDE DESCRIBES THE SEGMENT SEG1 IN ITS DPROP FORMAT. THE FIGURE PROVIDES FOR EACH FIELD THE FIELD NAME - THE FORMAT THE FIELD 'C' STANDS FOR 'FIXED LENGTH CHARACTER FORMAT' 'VC' STANDS FOR 'VARIABLE LENGTH CHARACTER FORMAT' THE FIXED START POSITION OF THE FIELD WITHIN THE DPROP FORMAT OF THE SEGMENT.

SEGMENT IN ITS
VARIABLE-LENGTH FIXED-LENGTH
DL/I DB FORMAT DPROP FORMAT

SEG1

FAMILY_L' H ' 11 'SEGILL' H ' 1
'KEYFLD1' C ' 3 ' -->'KEYFLD1' C ' 3
'KEYFLD2' C ' 5 ' -->'KEYFLD2' C ' 5
'FAMILY' VC ' 13 ' -->'FAMILY' VC ' 13
'FIRST_L' H ' 43 ' -->'FIRST_L' H ' 43
'FIRST' VC ' 45 ' -->'FIRST' VC ' 45
'CITY_L' H ' 65 ' -->'CITY_L' H ' 65
'CITY' VC ' 67 ' -->'CITY' VC ' 67

'FAMILY_L' H ' 11 ' -->'FAMILY_L' H ' 11
'FIRST_L' H ' 43 ' -->'FIRST_L' H ' 43
'CITY_L' H ' 65 ' -->'CITY_L' H ' 65
'CITY' VC ' 67 ' -->'CITY' VC ' 67

NOTES =

EKEYSEA IS CALLED:

- FOR TRANSFORMATION OF THE SEGMENT FROM ITS DL/I DB FORMAT WITH VARIABLE FIELD START POSITIONS TO ITS FORMAT SUPPORTED BY DXT/DPROP WITH FIXED FIELD START POSITIONS (NORMAL CALL TYPE INDICATED BY 'NO' IN DAXCALL FIELD OF THE DAX AREA):
- BY DXT (DURING EXTRACT OF THE DL/I DATA).
- BY DPROP DURING:
- SYNCH/ASYNCH IMS-TO-DB2 PROPAGATION
- SYNCH DB2-TO-IMS PROPAGATION
- CCU EXECUTION
- DLU EXECUTION

- FOR TRANSFORMATION OF THE SEGMENT FROM ITS FORMAT SUPPORTED BY DXT/DPROP WITH FIXED FIELD START POSITIONS TO ITS FORMAT ON THE DL/I DATABASE WITH VARIABLE FIELD START POSITIONS (REVERSE CALL TYPE INDICATED BY 'RV' IN DAXCALL FIELD OF DAX):
- BY DPROP DURING:
- SYNCH DB2-TO-IMS PROPAGATION
- CCU EXECUTION OF REPAIR FILE GENERATION
- DLU EXECUTION

- FOR TRANSFORMATION OF THE SEGMENT FROM ITS FORMAT SUPPORTED BY DXT/DPROP WITH FIXED FIELD START POSITIONS TO ITS FORMAT ON THE DL/I DATABASE WITH VARIABLE FIELD START POSITIONS (REVERSE CALL TYPE INDICATED BY 'RV' IN DAXCALL FIELD OF DAX):
- BY DPROP DURING:
- SYNCH DB2-TO-IMS PROPAGATION
- CCU EXECUTION OF REPAIR FILE GENERATION
- DLU EXECUTION

PLEASE REFER TO THE DSECTS TOWARDS THE BOTTOM OF THIS MODULE IN ORDER TO FIND ALL THE DETAILS ABOUT THE 'DL/I DB FORMAT' AND THE 'DPROP FORMAT' OF SEG1.

Figure 8 (Part 2 of 23). First Sample Segment Exit Routine (Assembler)
**Module Type** = PROCEDURE  
**Processor** = ASSEMBLER  
**Module Size** = APPROXIMATELY 1400 BYTES  
**Attributes** = REENTRANT  
**RMode** = ANY  
**AMode** = 31  

**Entry Point** = EKYESE1A  
**Purpose** = SEE FUNCTION  
**Linkage** = STANDARD OS/VS ASSEMBLER LINKAGE CONVENTIONS.  
**Return Codes** = 0  
**Exit-Normal** = STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.  
**Exit-Error** = STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.  
**Abend Code of EKYESE1A** = NONE  
**Abend Reason Codes** = NONE  
**Error Messages Issued by EKYESE1A**  
EKYESE01: CALL FUNCTION NOT SUPPORTED  
EKYESE31: UNSUPPORTED DBD OR SEGNAME  
EKYESE34: 3RD PARAMETER HAS INCORRECT LENGTH  
EKYESE35: INVALID SEGMENT LENGTH  
EKYESE4F: FAMILY FIELD DOES NOT FIT WITHIN SEGMENT  
EKYESE56: LENGTH OF FAMILY FIELD IS INVALID  
EKYESE58: FIRST-NAME FIELD DOES NOT FIT WITHIN SEGMENT  
EKYESE59: LENGTH OF FIRST-NAME FIELD IS INVALID  
EKYESE7C: CITY FIELD DOES NOT FIT WITHIN SEGMENT  
EKYESE8D: LENGTH OF CITY FIELD IS INVALID  

**Figure 8 (Part 3 of 23). First Sample Segment Exit Routine (Assembler)**
Figure 8 (Part 4 of 23). First Sample Segment Exit Routine (Assembler)
3) TRANSFORM THE SEGMENT: DL/I DB FORMAT ---> DPROP FORMAT

A) PROCESS THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT:

- VERIFY THAT THE DL/I FORMAT OF THE SEGMENT IS LONG ENOUGH TO CONTAIN THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT.
- MOVE THE FIELDS IN THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT INTO THE 'DPROP FORMAT' OF THE SEGMENT.

B) PROCESS THE SECOND, VARIABLE-FORMAT PORTION OF THE SEGMENT:

- INITIALIZE THE 'CURRENT POINTER' WITHIN THE DL/I FORMAT TO THE START OF THE VARIABLE-FORMAT PORTION.

- FOR EACH FIELD IN THE VARIABLE-FORMAT PORTION OF THE SEGMENT:
  -- VALIDATE THE FIELD LENGTH:
  --- FIELD LENGTH SHOULD BE POSITIVE
  --- FIELD LENGTH SHOULD NOT EXCEED A SPECIFIC MAXIMAL LENGTH.
  --- FIELD SHOULD BE TOTALLY WITHIN THE SEGMENT.
  -- COPY THE FIELD TO THE 'DPROP FORMAT' OF THE SEGMENT.

4) TRANSFORM THE SEGMENT: DPROP FORMAT ---> DL/I DB FORMAT

A) PROCESS THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT:

- VERIFY THAT THE DL/I FORMAT OF THE SEGMENT IS LONG ENOUGH TO CONTAIN THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT.
- MOVE THE FIELDS IN THE FIRST, FIXED-FORMAT PORTION OF THE SEGMENT INTO THE 'DPROP FORMAT' OF THE SEGMENT.

B) PROCESS THE SECOND, VARIABLE-FORMAT PORTION OF THE SEGMENT:

- INITIALIZE THE 'CURRENT POINTER' WITHIN THE DL/I FORMAT TO THE START OF THE VARIABLE-FORMAT PORTION.

- FOR EACH FIELD IN THE VARIABLE-FORMAT PORTION OF THE SEGMENT:
  -- VALIDATE THE FIELD LENGTH:
  --- FIELD LENGTH SHOULD BE POSITIVE
  --- FIELD LENGTH SHOULD NOT EXCEED A SPECIFIC MAXIMAL LENGTH.
  --- FIELD SHOULD BE TOTALLY WITHIN THE SEGMENT.
  -- COPY THE FIELD TO THE 'DPROP FORMAT' OF THE SEGMENT.

5) RETURN LOGIC

- RESTORE REGISTERS OF THE CALLER
- RETURN TO THE CALLER.

Figure 8 (Part 5 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 6 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 7 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 8 of 23). First Sample Segment Exit Routine (Assembler)
0000A8 D501 6000 C3F4 00000 003F4 530 CLC DL1_SEG1LL,=AL2(DL1_FIXEDDL) SEGMENT LARGE ENOUGH?
0000AE 4700 2C86 002B6 531 BNH INVSGL ...NO>>>THAT'S AN ERROR
0000B0 D201 7000 C3F6 00000 003F6 532 MVC DPR_SEG1LL,=AL2(DPR_SEG1L) SET LENGTH OF DPROP_SEG
0000B8 D201 7002 00002 00002 533 MVC DPR_KEYFLD1,DL1_KEYFLD1 MOVE KEYFLD1
0000BE 205 7004 00004 00004 534 MVC DPR_KEYFLD2,DL1_KEYFLD2 MOVE KEYFLD2

536 ************************************************************************************************
537 * INITIALIZE 'CURRENT POINTER' WITHIN DL/I FORMAT *
538 * *
539 * (IT IS REGISTER 2, WHICH IS USED AS CURRENT PTR *
540 * WITHIN DL/I FORMAT) *
541 ************************************************************************************************

0000C4 4120 600A 0000A 543 LA R2,DL1_SEGIVAR R2=CURRENT ADDR IN DL1 FMT

545 ************************************************************************************************
546 * FOR EACH FIELD IN THE VARIABLE-FORMAT PORTION OF THE *
547 * SEGMENT:
548 * - VALIDATE THE FIELD LENGTH:
549 * ---FIELD LENGTH SHOULD BE POSITIVE
550 * ---FIELD LENGTH SHOULD NOT EXCEED A SPECIFIC
551 * MAXIMAL LENGTH.
552 * ---FIELD SHOULD BE TOTALLY WITHIN THE SEGMENT.
553 * - COPY THE FIELD AND ITS LENGTH-FIELD TO THE
554 * DPROP-FORMAT OF THE SEGMENT
555 * *
556 ************************************************************************************************

560 ************************************************************************************************
561 * PROCESS FAMILY NAME FIELD *
562 ************************************************************************************************

564 *** CHECK LENGTH FIELD
565
0000CA 48F2 0000 00000 567 LH R15,0(R2) R15=LENGTH OF VC FIELD
0000CC 12FF 568 LTR R15,R15 LENGTH FIELD POSITIVE?
0000CE 4700 2CDE 002DE 569 BNP INVFAM2 ...NO>>>THAT'S AN ERROR
0000D2 55F0 C3BC 003BC 570 CL R15,-A(30) FAMILY FIELD LONGER THAN 30?
0000D6 205 7002 00002 571 BH INVFAM2 ...YES>>>THAT'S AN ERROR
572 *
573 *** CHECK THAT THE FIELD IS TOTALLY WITHIN THE SEG?
574
0000DA 4102 F002 00002 575 LA R0,2(R2,R15) R0=A(END OF VC FIELD)+1
0000DE 4810 6000 00000 576 LH R1,DL1_SEGILL R1=LENGTH OF SEGMENT
0000E2 4111 6000 00000 577 LA R1,DL1_SEGIL(R1) R1=A(END OF SEGMENT)+1
0000E6 1901 578 CR R0,R1 FLD TOTALLY WITHIN SEG?
0000EB 4720 2C2A 002CA 579 BH INVFAM1 ...NO>>>THAT'S AN ERROR
580 *
581 *** MOVE LENGTH FIELD INTO DPROP-FORMAT
582
0000EC 2D01 700A 2000 0000A 00000 583 MVC DPR_FAMILY_L,0(R2) MOVE LENGTH FIELD
584 *
585 *** MOVE VC FIELD INTO DPROP-FORMAT
586 *
0000F2 4102 0002 00002 587 LA R0,2(R2) R0=START FOR MVCL
0000F6 181F 588 LR R1,R15 R1=LENGTH FIR MVCL
0000FB B1F 03F8 589 ICM R1,B,'C' PADDING BLANK FOR MVCL
0000FC 4100 700C 0000C 590 LA R14,DPR_FAMILY R14=TARGET ADDRESS FOR MVCL
000100 41F0 001E 0001E 591 LA R15,30 R15=TARGET LENGTH FOR MVCL
000104 00ED 592 MVCL R14,R0 MOVE THAT FIELD

Figure 8 (Part 9 of 23). First Sample Segment Exit Routine (Assembler)
593 * 594 *** ADJUST 'CURRENT POINTER WITHIN DL/I DB FORMAT'
595 *
00106 1820
596 LR R2,R0 R2=START OF NEXT FIELD
597 *
598 *---------------------------------------------------------------------* 599 * PROCESS FIRST NAME FIELD * 600 *---------------------------------------------------------------------*
602 *
603 *** CHECK LENGTH FIELD
604 *
00108 48F2 0000
00000 605 LH R15,0(R2) R15=LENGTH OF VC FIELD
0010C 12FF
606 LTR R15,R15 LENGTH FIELD POSITIVE?
0010E 47D0 C306
00306 607 BNP INVFRST2 ...NO>>THATS AN ERROR
00112 55F0 C3C0
003C0 608 CL R15,=A(20) FIRST FIELD LONGER THAN 20?
00116 4720 C306
00306 609 BH INVFRST2 ...YES>>THATS AN ERROR
610 *
611 *** CHECK THAT THE FIELD IS TOTALLY WITHIN THE SEGM
612 *
0011A 4102 F002
00002 613 LA R0,2(R2,R15) R0=A(END OF VC FIELD)+1
0011E 4810 6000
00000 614 LH R1,DL1_SEG1LL R1=LENGTH OF SEGMENT
00122 4111 6000
00000 615 LA R1,DL1_SEG1(R1) R1=A(END OF SEGMENT)+1
00126 1901
616 CR R0,R1 FLD TOTALLY WITHIN SEG?
00128 4720 C2F2
002F2 617 LH R1,DL1_SEG1(R1) R1=A(END OF SEGMENT)+1
618 *
619 *** MOVE LENGTH FIELD INTO DPROP-FORMAT
620 *
0012C D201 702A 2000 0002A 00000
621 MVC DPR-FIRST_L,0(R2) MOVE LENGTH FIELD
622 *
623 *** MOVE VC FIELD INTO DPROP FORMAT
624 *
00132 4102 0002
00002 625 LA R0,2(R2) R0=START FOR MVCL
00136 181F
626 LR R1,R15 R1=LENGTH FOR MVCL
00138 BF18 C3F8
003F8 627 ICM R1,6,'C' ' PADDING BLANK FOR MVCL
0013C 41E0 702C
0002C 628 LA R14,DPR FIRST R14=TARGET ADDRESS FOR MVCL
00140 41F0 0014
00014 629 LA R15,20 R15=TARGET LENGTH FOR MVCL
00144 0EE0
630 MVCL R14,R0 MOVE THAT FIELD
631 *
632 *** ADJUST 'CURRENT POINTER WITHIN DL/I DB FORMAT'
633 *
00146 1820
634 LR R2,R0 R2=START OF NEXT FIELD
635 *
636 *---------------------------------------------------------------------* 637 * PROCESS CITY FIELD * 638 *---------------------------------------------------------------------*
640 *
641 *** CHECK LENGTH FIELD
642 *
00148 48F2 0000
00000 643 LH R15,0(R2) R15=LENGTH OF VC FIELD
0014C 12FF
644 LTR R15,R15 LENGTH FIELD POSITIVE?
0014E 47D0 C32E
0032E 645 BNP INVFRST2 ...NO>>THATS AN ERROR
00152 55F0 C3C4
003C4 646 CL R15,=A(35) CITY FIELD LONGER THAN 35?
00156 4720 C32E
0032E 647 BH INVFRST2 ...YES>>THATS AN ERROR
648 *
649 *** CHECK THAT THE FIELD IS TOTALLY WITHIN THE SEGM
650 *
0015A 4102 F002
00002 651 LA R0,2(R2,R15) R0=A(END OF VC FIELD)+1
0015E 4810 6000
00000 652 LH R1,DL1_SEG1LL R1=LENGTH OF SEGMENT
00162 4111 6000
00000 653 LA R1,DL1_SEG1(R1) R1=A(END OF SEGMENT)+1
00166 1901
654 CR R0,R1 FLD TOTALLY WITHIN SEG?
00168 4720 C31A
0031A 655 BH INVFRST1 ...NO>>THATS AN ERROR

Figure 8 (Part 10 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 11 of 23). First Sample Segment Exit Routine (Assembler)
**FOR EACH FIELD IN THE VARIABLE-FORMAT PORTION OF THE SEGMENT:**

- **VALIDATE IF THE FIELD IS REALLY PRESENT (MAY BE TRUNCATED IF "NULL" ON THE DB2 SIDE)**
- **VALIDATE THE FIELD LENGTH:**
  - **FIELD LENGTH SHOULD BE POSITIVE**
  - **FIELD LENGTH SHOULD NOT EXCEED A SPECIFIC MAXIMAL LENGTH.**
- **COPY THE FIELD AND ITS LENGTH-FIELD TO THE DL/I-FORMAT OF THE SEGMENT**

---

**PROCESS FAMILY NAME FIELD**

---

**CHECK LENGTH FIELD**

```
0001AE 17FF
0001B0 D503 5080 C300 00080 00300 736
0001B6 4740 C1D8 001DB 737
0001BA 48F0 700A 0008A 738
0001BE 12FF 739
0001C0 4740 C2DE 002DE 740
0001C4 59F0 C304 003D4 741
0001C8 4720 C2DE 002DE 742
0001CC 410F 000C 0000C 746
0001DD 5900 5080 00080 747
0001DA 4720 C2CA 002CA 748
0001DB 752
0001DB 40F0 2000 00000 753
0001DC 4120 2002 00002 757
0001E0 183F 758
0001E2 41E0 700C 0000C 759
0001E6 0E2E 760
```

**Store length field into DL/I-format**

```
0001AE 17FF
0001B0 D503 5080 C300 00080 00300 770
0001F0 4740 C212 00212 771
0001F4 48F0 702A 0002A 772
0001FB 12FF 773
0001FA 4740 C306 00306 774
0001FE 59F0 C3DC 003DC 775
000202 4720 C306 00306 776
```

---

**PROCESS FIRST NAME FIELD**

---

**CHECK LENGTH FIELD**

```
0001AE 17FF
0001B0 D503 5080 C300 00080 00300 770
0001F0 4740 C212 00212 771
0001F4 48F0 702A 0002A 772
0001FB 12FF 773
0001FA 4740 C306 00306 774
0001FE 59F0 C3DC 003DC 775
000202 4720 C306 00306 776
```

---

Figure 8 (Part 12 of 23). First Sample Segment Exit Routine (Assembler)
777 * CHECK THAT THE FIELD IS TOTALLY WITHIN THE SEG.
778 ***
000206 410F 002C 0002C 780 LA R0,DPR_FIRST-DPR_SEG1(R15) R0=DPR_FIRST+L'DPR_FIRST
00020A 5500 50B0 000B0 781 CL R0,DAXFLN FLD TOTALLY WITHIN SEG?
00020E 4720 C2F2 002F2 782 BH INVFRST1 ...NO>>>THAT'S AN ERROR
783 ***
784 ***
785 ***
786 ***
787 ***
788 ***
789 ***
790 ***
791 ***
792 ***
793 ***
794 ***
795 ***
796 ***
797 ***
798 ***
800 *
801 ***
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 ***
819 *
820 *
821 *
822 *
823 ***
824 *
825 *
826 *
827 *
828 *
829 *
830 ***
831 *
832 *
833 *
834 *
835 *
836 ***

---

Figure 8 (Part 13 of 23). First Sample Segment Exit Routine (Assembler)

---

58 Customization Guide
Figure 8 (Part 14 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 15 of 23). First Sample Segment Exit Routine (Assembler)
0003C0 00000023 969 =A(35)
0003B8 00000035 970 =A(DL1_CITY+L'DL1_CITY+DL1_SEG1)
0003C0 0000000A 971 =A(DPR_SEG1KEY+L'DPR_SEG1KEY+L'DPR_SEG1)
0003D0 0000000C 972 =A(DPR_CITY-DPR_SEG1)
0003D0 0000001E 973 =A(L'DPR_CITY)
0003D8 0000002C 974 =A(DPR_CITY-DPR_SEG1)
0003E0 00000042 976 =A(DPR_CITY-DPR_SEG1)
0003E4 00000023 977 =A(L'DPR_CITY)
0003E8 0000000C 978 =F'12'
0003EC 00000010 979 =F'16'
0003F0 DS06 980 =C'NO'
0003F2 D9E5 981 =C'RV'
0003F4 000A 982 =AL2(DL1_FIXEDL)
0003FE 0065 983 =AL2(DPR_SEG1)
0003F8 40 984 =C'
0003F9 C31D3D340C6E4D5 985 =CL55'CALL FUNCTION NOT SUPPORTED'
000430 E4052E2E40D7D0D9 986 =CL55'UNSUPPORTED DBD OR SEGNAME'
000467 F309C440D7C1D0C1 987 =CL55'3RD PARAMETER HAS INCORRECT LENGTH'
00046E C9055C5D3C9C440 988 =CL55'INVALID SEGMENT LENGTH'
0004D5 C61C4D3E840C6 989 =CL55'FAMILY FIELD DOES NOT FIT WITHIN SEGMENT'
00050C D3C505C7E3C840D6 990 =CL55'LENGTH OF FAMILY FIELD IS INVALID'
000543 C6C90DE2E360D5C1 991 =CL55'FIRST-NAME FIELD DOES NOT FIT WITHIN SEGMENT'
00057A D3C505C7E3C840D6 992 =CL55'LENGTH OF FIRST-NAME FIELD IS INVALID'
0005B1 C39E3E840C6C9C5 993 =CL55'CITY FIELD DOES NOT FIT WITHIN SEGMENT'
0005EB D3C505C7E3C840D6 994 =CL55'LENGTH OF CITY FIELD IS INVALID'

**Figure 8 (Part 16 of 23). First Sample Segment Exit Routine (Assembler)**
1025 *---------------------------------------------------------------------*

000000
1027 DL1_SEG1 DSECT ,
1028 *
1029 *** SEGMENT PORTION WITH FIXED-LENGTH FIELDS HAVING
1030 *** A FIXED START POSITION.
1031 *
000000
1032 DL1_SEG1LL DS H LENGTH OF SEGMENT
000002
1033 DL1_SEG1KEY DS OCL8 KEY FIELD
000002
1034 DL1_KEYFLD1 DS CL2 SUB-FIELD OF KEY OF SEG1
000004
1035 DL1_KEYFLD2 DS CL6 SUB-FIELD OF KEY OF SEG1
00000A
1036 DL1_FIXEDL EQU *-DL1_SEG1 LENGTH OF FIXED PORTION
1037 *
1038 *** START OF VARIABLE SEGMENT PORTION WITH CONTIGUOUS
1039 *** PAIRS OF:
1040 *** (LENGTH FIELD,VARIABLE-LENGTH FIELD).
1041 *** WITH THE EXCEPTION OF THE FIRST PAIR: ALL
1042 *** PAIRS HAVE A VARIABLE START POSITION (SINCE THE
1043 *** PAIRS ARE STORED ADJACENTLY IN ORDER TO CONSERVE
1044 *** DASD STORAGE IN THE DL/I DB).
1045 *
00000A
1046 DL1_SEG1VAR EQU *
1047 *
00000A
1048 DL1_FAMILY_L DS HL2 LENGTH OF FAMILY NAME
00000C
1049 DL1_FAMILY DS CL30 FAMILY NAME
1050 *
00002A
00000D
1051 ORG DL1_FAMILY+1
00000D
1052 DL1_FIRST_L DS HL2 LENGTH OF FIRST NAME
00000F
1053 DL1_FIRST DS CL20 FIRST NAME
1054 *
000023
000010
1055 ORG DL1_FIRST+1
000010
1056 DL1_CITY_L DS HL2 LENGTH OF CITY-NAME
000012
1057 DL1_CITY DS CL35 CITY-NAME
000035
000035
1058 ORG

1060 *---------------------------------------------------------------------*
1061 *
1062 *
1063 *
1064 *
1065 *
1066 *
1067 *
1068 *
1069 *
1070 *---------------------------------------------------------------------*

000000
1072 DPR_SEG1 DSECT ,
000000
1073 DPR_SEG1LL DS H SEGMENT LENGTH
000002
1074 DPR_SEG1KEY DS OCL8 KEY FIELD
000002
1075 DPR_KEYFLD1 DS CL2 SUB-FIELD OF KEY OF SEG1
000004
1076 DPR_KEYFLD2 DS CL6 SUB-FIELD OF KEY OF SEG1
1077 *
00000A
1078 DPR_FAMILY_L DS HL2 LENGTH OF FAMILY NAME
00000C
1079 DPR_FAMILY DS CL30 FAMILY NAME
1080 *
00002A
000002
1081 DPR_FIRST_L DS HL2 LENGTH OF FIRST NAME
00002C
1082 DPR_FIRST DS CL20 FIRST NAME
1083 *
000040
1084 DPR_CITY_L DS HL2 LENGTH OF CITY NAME
000042
1085 DPR_CITY DS CL35 CITY NAME
1086 *
000065
1087 DPR_SEG1L EQU *-DPR_SEG1 LENGTH OF SEGMENT

Figure 8 (Part 17 of 23). First Sample Segment Exit Routine (Assembler)
**DESCRIPTION OF ANCHOR AREA**

```
000000 1093 ANCHOR DSECT ,
000000 1094 ANCHOR_PTR DS F'0', PTR TO GETMAINED AREA
000004 1095 DS CL60 ' ' NOT USED
000007 1097 EKYRCDAX , EXIT INTERFACE CONTROL BLOCK
1098********** START OF CONTROL BLOCK SPECIFICATION **********/
1099++ 1100++ CONTROL BLOCK NAME:
1101++ 1102++ EKYRCDAX (DAX)
1103++ 1104++ DESCRIPTIVE NAME:
1105++ 1106++ DPROP SEGMENT EXIT INTERFACE BLOCK
1107++ 1108+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
1109++ 1110++ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992.
1111++ 1112++ ALL RIGHTS RESERVED.
1113++ 1114++ U.S. GOVERNMENT USERS RESTRICTED RIGHTS -
1115++ 1116++ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
1117++ 1118++ GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
1119++ 1120++ LICENSED MATERIALS - PROPERTY OF IBM.
1121++ 1122+ STATUS: V1 R2 M
1123++ 1124+ FUNCTION:
1125++ 1126++ THIS IS THE CONTROL BLOCK USED TO INTERFACE BETWEEN
1127++ 1128++ - DPROP OR DXT
1129++ 1130++ AND
1131++ 1132++ A USER'S SEGMENT EXIT ROUTINE (THESE USER
1133++ 1134++ EXIT ROUTINES ARE CALLED BY DXT 'USER DATA
1135++ 1136++ EXIT ROUTINES')
1137++ 1138++ THERE IS ONE DAX CONTROL BLOCK FOR EACH SEGMENT
1139++ 1140++ EXIT ROUTINE, LASTING FOR THE DURATION OF THE EXIT
1141++ 1142++ IN VIRTUAL STORAGE.
1143++ 1144++ FOR SYNCH PROPAGATION IN MPP REGIONS:
1145++ 1146++ - THIS IS THE DURATION OF THE IMS PROGRAM CONTROLLER
1147++ 1148++ SUBTASK.
1149++ 1150++ FOR SYNCH PROPAGATION IN BATCH/BMP REGIONS, FOR
1151++ 1152++ CCU AND DLU PROCESSING, AND FOR ASYNCH PROPAGATION
1153++ 1154++ (DEPENDING ON HOW ASYNCH PROPAGATION IS IMPLEMENTED):
1155++ 1156++ - THIS IS THE DURATION OF THE JOBSTEP.
1157++ 1158++ IMPORTANT NOTES:
1159++ 1160++ ==
1161++ 1162++ - SINCE THE SAME USER EXIT ROUTINE CAN BE INVOKED BOTH
1163++ 1164++ BY DPROP AND BY DXT: CHANGES TO THIS CONTROL BLOCK MUST
1165++ 1166++ BE COORDINATED BETWEEN DPROP DEVELOPMENT AND DXT
1167++ 1168++ DEVELOPMENT.
1169++ 1170++ - FIELDS MARKED IN THE COMMENT WITH '***DXT ONLY***'
1171++ 1172++ HAVE NO MEANING, WHEN THE SEGMENT USER EXIT
1173++ 1174++ ROUTINE IS INVOKED BY DPROP.
1175++ 1176++
```

Figure 8 (Part 18 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 19 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 20 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 21 of 23). First Sample Segment Exit Routine (Assembler)
Figure 8 (Part 22 of 23). First Sample Segment Exit Routine (Assembler)
1411++ - IF CALLER IS DPROP:
1412++ MSG WILL BE WRITTEN TO
1413++ VARIOUS DESTINATIONS ACCORDING
1414++ TO USUAL DPROP/RUP ERROR HANDLING
1415++ LOGIC IN MESSAGE EKYR9801 OR
1416++ EKYR081E.
1417++ - IF CALLER IS DXT:
1418++ TEXT OF MESSAGE WILL BE
1419++ WRITTEN TO
1420++ VSRCPR DATA SET IN MESSAGE
1421++ DFORM 50.
1422++ (UNICODE IS REPLACED
1423++ BY ONE OF SEVERAL DIGITS)
1424++ HAS EFFECT FOR ALL CALLS.
1425++
1426+DAXDPRPM DS CL24 STORAGE RESERVED FOR DATA EXIT
1427++
1428+DAXRSVD2 DS CL32 RESERVED FOR DXT USE
1429+DAXSCRT1 DS CL128 WORK SPACE (SCRATCHPAD)
1430++ MAY BE USED BY EXIT
1431++ ROUTINE AS DESIRED
1432++
1433+DAXEND EQU * END OF DAX DSECT
1434+DAXLEN EQU -*DAX LENGTH OF DAX DSECT
1435++++++++++++++++++++++++++++++++++++++
1436++
1437++ DAXANCRT DSECT ***DXT ONLY***
1438++ MAPS THE ARRAY ELEMENTS OF DAXASEGS
1439++
1440+++++++++++++++++++++++++++++++++++++++
1441+DAXANCRT DSECT , ***DXT ONLY***
1442+DAXASGNM DS CL8 ***DXT ONLY*** ANCESTOR SEGM NAME
1443++
1444+DAXASGAD DS AL4 ***DXT ONLY*** ANCESTOR SEGM ADDRESS
1445++
1447 ***REDEFINITION OF THE MESSAGE AREA LOCATED IN THE DAX***
1448 * REDEFINITION OF THE MESSAGE AREA LOCATED IN THE DAX *
1449+++++++++++++++++++++++++++++++++++++++
1450++
1451 DAX DSECT
1452+ ORG DAXSMESG
1453 MSGID DC CL8 ' ' MESSAGE ID
1454 MSGBL1 DC C' ' ONE BLANK
1455 MSGTXT DC CL55 ' ' TEXT
1456+END EKYESE1A

Figure 8 (Part 23 of 23). First Sample Segment Exit Routine (Assembler)

Definitions for the First Sample Segment Exit Routine

This section contains definitions associated with the first sample Segment exit routine. The following types of definitions are provided:

- IMS DBDGEN and PSBGEN definitions
- DB2 CREATE TABLE definitions
- DataRefresher definitions required to define the PR with DataRefresher and to extract the IMS data with DataRefresher
- SQL statements defining the PR without DataRefresher in the MVG input tables
DBDGEN Definitions

Figure 9 shows a DBDGEN definition for the Segment exit routine in Figure 8 on page 46.

```sql
DBD NAME=DB1,VERSION=V123456789,
ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC4mzqr!z!t,5,4),
EXIT=(EKYRUP00)
DATASET DD1=HDAM,SIZE=4096,DEVICE=3380
* SEGAM NAME=SEG1,PARENT=0,BYTES=(101,10)
FIELD NAME=(KEY,SEQ,U),BYTES=8,START=3
* DBDGEN
FINISH
END
```

*Figure 9. DBDGEN Definition*

**Note:** The EXIT= keyword of the DBD macro specifies that EKYRUP00 (the RUP) be called when a segment of this DBD is changed. This is required for synchronous data propagation.

PSBGEN Definitions

Figure 10 shows a PSBGEN definition for the Segment exit routine in Figure 8 on page 46.

```sql
PCB TYPE=DB,...
... SENSEG ...
PCB TYPE=DB,...
... SENSEG ...
PCBDPR1 PCB TYPE=DB,OBDDNAME=DB1,LIST=NO
KEYLEN=101,PROCLOPT=A
SENSEM NAME=SEG1
* PSBGEN PSBNAME=PSBDPR1
END
```

*Figure 10. PSBGEN Definition*

**Note:** The first two PCBs represent PCBs used by the application programs. The third PCB, PCBDPR1, is the PCB reserved for HUP usage.

CREATE TABLE Statement

Figure 11 on page 70 shows a CREATE TABLE statement for the segment exit routine in Figure 8 on page 46.
CREATE TABLE T096606.TABLE01
   (KEY1 CHAR(2) NOT NULL,
    KEY2 CHAR(6) NOT NULL,
    FAMILY VARCHAR(30) ,
    FIRST VARCHAR(20) ,
    CITY VARCHAR(35) ,
    PRIMARY KEY (KEY1, KEY2))
   DATA CAPTURE CHANGES
   IN 00096606.PROPTS;

CREATE UNIQUE INDEX XN01 ON TABLE01 (KEY1, KEY2)
   USING VCAT KOE ;

---

**Figure 11. CREATE TABLE Statement**

**Note:** The DATA CAPTURE CHANGES option of the create table command specifies that the DB2 Changed Data Capture exit (the HUP) be called when a row of this table is changed under IMS attach.

### Using DataRefresher to Define the PR

This section shows how to define the PR in Figure 8 on page 46 using DataRefresher.

**CREATE DXTPSB**

Figure 12 on page 71 shows a CREATE DXTPSB statement for the segment exit routine in Figure 8 on page 46.
CREATE DXTPSB  NAME=KOEPSB2

DXTPCB  NAME=DB1, DBNAME=DB1, DBACCESS=HDAM

SEGMENT  NAME=SEG1, PARENT=0, BYTES=101, DATAEXIT=EKYESE1A, XBYTES=101, FORMAT=V

FIELD  NAME = KEY ,
START = 3,
BYTES = 8,
SEQFLD = R

FIELD  NAME = KEY1,
TYPE = C,
START = 3,
BYTES = 2

FIELD  NAME = KEY2,
TYPE = C,
START = 5,
BYTES = 6

FIELD  NAME = LFAMILY,
TYPE = H,
START = 11,
BYTES = 2

FIELD  NAME = FAMILY,
TYPE = VC,
LFIELD = LFAMILY,
START = 13,
BYTES = 30

FIELD  NAME = LFIRST,
TYPE = H,
START = 43,
BYTES = 2

FIELD  NAME = FIRST,
LFIELD = LFIRST,
TYPE = VC,
START = 45,
BYTES = 20

FIELD  NAME = LCITY,
TYPE = H,
START = 65,
BYTES = 2

FIELD  NAME = CITY,
TYPE = VC,
LFIELD = LCITY,
START = 67,
BYTES = 35;

Figure 12. CREATE DXTPSB Statement

Notes:

1. Segment exit routine EKYESE1A is specified on the DATAEXIT= keyword of the SEGMENT statement of CREATE DXTPSB.

The SEGMENT statement also provides the following specifications:

- BYTES=101 specifies the maximum length of the segment in its IMS DB format.
- XBYTES=101 specifies the maximum length of the segment in its DPROP format.
- FORMAT=V specifies the segment has a variable length in its DPROP format.

2. The FIELD statements describe the fields as they appear in the DPROP format of the segment (as opposed to the segment in its IMS DB format).

All propagated fields need to be described in a FIELD statement.
3. The fields FAMILY, FIRST, and CITY are defined by TYPE=VC as variable-length character fields.

DataRefresher requires that each variable-length field have an associated length field. The length fields are described with their own FIELD statements. The LFIELD= keyword of a variable-length field must identify the name of the length field.

For example, this is illustrated in the FAMILY field. The LFIELD= keyword of the FAMILY field identifies LFAMILY as the length field of FAMILY.

The EXTRACT statement (see below) propagates the variable-length fields, but does not propagate the length fields.

**CREATE DXTVIEW**

Figure 13 shows a CREATE DXTVIEW statement for the Segment exit routine in Figure 8 on page 46.

```
CREATE DXTVIEW
FIGURE 13. CREATE DXTVIEW Statement
```

**DataRefresher UIM SUBMIT Command and EXTRACT Statement**

Figure 14 shows a DataRefresher UIM SUBMIT command and EXTRACT statement for the Segment exit routine in Figure 8 on page 46.

```
SUBMIT
EXTRACT
FIGURE 14. DataRefresher UIM SUBMIT Command and EXTRACT Statement
```
Notes:

1. The MAPEXIT= keyword of the SUBMIT control statement specifies EKYMCE00. This results in DataRefresher UIM calling the DPROP-provided Map Capture Exit EKYMCE00 during processing of the SUBMIT or EXTRACT. This is needed to allow DPROP to create the PR.

2. MAPUPARM= is used to provide the DPROP propagation keywords.

3. The EXTRACT statement describes to DataRefresher and DPROP which fields must be mapped to which columns.

The EXTRACT statement propagates the variable-length fields FAMILY, FIRST, and CITY; it does not propagate the length fields LFAMILY, LFIRST, and LCITY.

Using DataRefresher for the Extract

This section covers INITDEM and USE DXTPSB Control Statements. Figure 15 shows INITDEM and USE DXTPSB control statements for the Segment exit routine in Figure 8 on page 46.

```plaintext
INITDEM NAME=DEMPROD;
USE DXTPSB=KOEPSB2;
```

Figure 15. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements

Defining the PR in the MVG Input Tables

This section shows how to define the PR without using DataRefresher. Figure 16 on page 74 describes the DSNTEP2 SQL statements required to define the PR in the MVG input tables.

The following rows are inserted into the MVG input tables:

- One row is inserted into the DPRIPR table (the PR table).
  This row identifies the PR ID. By inserting an F into the PRTYPE column and a 1 into the MAPCASE column, you can set up the SQL statement so that the PR belongs to mapping case 1 of an extended-function PR.
- One row for the Entity segment Type SEG1 is inserted into the DPRISEG table (the SEG table).
  Because SEG1 is the root segment, no rows are inserted into DPRISEG for physical ancestors.

The row describing SEG1 provides the following column values:

- The nonblank value EKYESE1A in the SEGEXIT column. This specifies that the segment must be processed by the Segment exit routine EKYESE1A.
- The value 101 in the SEGEXITL column specifies the maximum length of the segment in its DPROP format.
- The value V in the SEGEXITF column specifies that the segment in its DPROP format has a variable length.
- One row is inserted into the DPRITAB table (the TAB table).
This row indicates that the target table is T096606.TABLE01.

- One row is inserted into the DPRIFLD table (the FLD table) for each propagated field.

The DPRIFLD rows describe the fields as they appear in the DPROP format of the segment (as opposed to the segment in its IMS DB format).

The fields FAMILY, FIRST, and CITY are defined by the VC value in the DATATYPE column as variable-length character fields.

DPROP requires two DPRIFLD rows for each variable length field:
- One row describes the variable-length field.
- The other row describes the length field.

The FAMILY field illustrates this. The row describing the variable-length field FAMILY identifies in the LENFIELD column the name of the length field, LFAMILY.

The row describing the length field LFAMILY has a blank value in the COLNAME column, because the length field is not propagated (only the variable-length field FAMILY is propagated).

```
DELETE FROM T096606.DPRIPR WHERE PRID = 'PR001';

INSERT INTO T096606.DPRIPR (PRID, USERID, PRTYPE, MAPCASE, MAPDIR, ERROPT, ACTION)
VALUES ('PR001', 'T096606','F', '1', 'TW', 'BACKOUT', 'REPL');

INSERT INTO T096606.DPRISEG (PRID, DBNAME, SEGNAME, ROLE, PCBLABEL, SEGEXIT, SEGEXITL, SEGEXITF)
VALUES ('PR001', 'DB1', 'SEG1', 'E', 'PCBDPR1', 'EKYESE1A', 1/zerodot1, 'V');

INSERT INTO T096606.DPRITAB (PRID, TABQUAL, TABNAME)
VALUES ('PR001', 'T096606', 'TABLE01');

INSERT INTO T096606.DPRIFLD (PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR001', 'DB1', 'SEG1', 'KEY1', 'T096606', 'TABLE01', 'KEY1', 'C', 3, 2);

INSERT INTO T096606.DPRIFLD (PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR001', 'DB1', 'SEG1', 'KEY2', 'T096606', 'TABLE01', 'KEY2', 'C', 5, 6);
```

Figure 16 (Part 1 of 2). Defining the PR in the MVG Input Tables
Second Sample Segment Exit Routine

Figure 17 on page 77 contains another example of a Segment exit routine. This example supports the propagation of an IMS segment containing internal segments propagated by a mapping case 3 PR.

When it receives a changed IMS data segment and is called for IMS-to-DPROP mapping, the exit routine transforms the segment into a DPROP-supported format. During this transformation process, the exit routine creates in each occurrence of the internal segment type an ID field. The ID field is required by DPROP and allows identification of each occurrence of the internal segment within its containing
IMS segment. The exit routine builds a counter field. The counter field describes how many internal segments are contained within a particular occurrence of the containing segment.

When it is called for DPROP-to-IMS mapping, the exit routine must build the IMS format of the segment. The Segment exit routine receives the following input:

- Either a changed occurrence of an internal segment (in its DPROP format), or the changed containing segment (in its DPROP format)
- The existing before-change image of the IMS segment in its IMS format

By combining information from this input, the segment exit routine builds the new after-change image of the IMS segment in its IMS format.

The source code in Figure 17 on page 77 is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYESE2C. Following the source code are definitions related to the sample Segment exit routine.
**MODULE NAME: EKYESE2C**

**DESCRIPTIVE NAME: SAMPLE SEGMENT EXIT COBOL ROUTINE**

**FUNCTION: EKYESE2C IS A SAMPLE DPROP SEGMENT EXIT ROUTINE**

WRITTEN IN COBOL AND USED FOR THE TRANSFORMATION OF A SEGMENT LAYOUT BETWEEN ITS:

- IMS FORMAT AND
- DPROP FORMAT.

EKYESE2C ILLUSTRATES ONE OF THE MOST TYPICAL USAGE OF DPROP SEGMENT USER EXITS: THE SUPPORT OF THE PROPAGATION OF AN IMS SEGMENT CONTAINING AN INTERNAL SEGMENT / REPEATING GROUP OF FIELDS.

THIS SAMPLE SEGMENT EXIT ROUTINE SUPPORTS TYPE=E PR'S AND IS THEREFORE CALLED BOTH FOR:

- IMS-TO-DPROP MAPPING (E.G. DURING IMS-TO-DB2 PROPAGATION; ALSO DURING DXT-EXTRACTS, CCU-PROCESSING AND DLU PROCESSING).
- DPROP-TO-IMS MAPPING (E.G. DURING DB2-TO-IMS PROPAGATION; ALSO DURING CCU PROCESSING AND DLU PROCESSING).

IN THIS EXAMPLE THE PROPAGATED IMS SEGMENT IS A BANK ACCOUNT SEGMENT. THE IMS SEGMENT CONSISTS OF THE FOLLOWING FIELDS:

- THE ACCOUNT-NBR (THIS IS THE KEY OF THE SEG)
- THE CUSTOMER-NAME
- A REPEATING GROUP OF FIELDS WITH THREE OCCURRENCES. EACH OCCURRENCE OF THE REPEATING GROUP CONTAINS INFORMATION ABOUT ONE TYPE OF CREDIT THAT THE BANK IS GRANTING. THIS INFORMATION IS:
  - THE CURRENT AMOUNT OF CREDIT GRANTED TO THE CUSTOMER/ACCOUNT
  - THE CREDIT LIMIT FOR THE CUSTOMER/ACCOUNT.

THE DATABASE ADMINISTRATOR WANTS TO HAVE A NORMALIZED DB2 TABLE DESIGN AND THEREFORE WANTS TO:

1) PROPAGATE THE ACCOUNT-NBR AND CUSTOMER-NAME TO/FROM THE TABLE CALLED "ACCOUNT":

- THIS IS DONE WITH A MAPPING-CASE-1 PR.

2) PROPAGATE THE INFORMATION RELATED TO THE DIFFERENT TYPES OF CREDITS (TOGETHER WITH THE ACCOUNT-NBR) TO/FROM ANOTHER TABLE CALLED "CREDIT":

- THIS IS DONE WITH A MAPPING-CASE-3 PR.

EACH OCCURRENCE OF THE CREDIT INFORMATION (THERE ARE 3 OF THEM) IS CONSIDERED TO BE AN OCCURRENCE OF AN INTERNAL SEGMENT AND IS PROPAGATED TO/FROM ONE ROW OF THE TABLE "CREDIT".

IN ORDER TO DISTINGUISH WITHIN THE CREDIT TABLE THE 3 TYPE OF CREDIT INFORMATION (AND IN ORDER TO HAVE A DB2 PRIMARY KEY), THE CREDIT TABLE DOES NOT ONLY CONTAIN AN ACCOUNT-NBR COLUMN AND THE CURRENT CREDIT AMOUNT AND LIMIT.

THE CREDIT TABLE CONTAINS ALSO A "TYPE" COLUMN WHICH IDENTIFIES THE TYPE OF CREDIT.
THE SAMPLE SEGMENT EXIT ROUTINE "EKYESE2C" PROVIDES LOGIC TO SUPPORT THE PROPAGATION OF THE IMS SEGMENT TO/FROM THE TABLES ACCOUNT AND CREDIT.

1) FOR IMS-TO-DPROP MAPPING, THE SAMPLE EXIT PROVIDES THE FOLLOWING FUNCTIONS WHEN BUILDING THE DPROP FORMAT OF THE SEGMENT:

- THE EXIT ROUTINE CREATES IN THE DPROP FORMAT AN ID-FIELD FOR EACH OCCURRENCE OF THE INTERNAL SEGMENT. THIS IS THE FIELD CALLED "TYPE".

  THIS ADDRESS THE DPROP REQUIREMENT THAT INTERNAL SEGMENTS HAVE AN "ID" FIELD IDENTIFYING UNIQUELY THE OCCURRENCES OF THE INTERNAL SEGMENTS WITHIN THE CONTAINING SEGMENT.

  IN THE DPROP FORMAT, EACH OCCURRENCE OF THE INTERNAL SEGMENT WILL CONSIST OF FOLLOWING FIELDS:
  - THE FIELD "TYPE" (THIS IS THE ID-FIELD CREATED BY THE EXIT)
  - THE FIELD "AMOUNT" (COPIED FROM THE IMS FORMAT OF THE SEGMENT)
  - THE FIELD "LIMIT" (COPIED FROM THE IMS FORMAT OF THE SEGMENT).

- THE EXIT ROUTINE CREATES IN THE DPROP FORMAT A COUNT FIELD. ITS VALUE IS THE NUMBER OF OCCURRENCES OF THE INTERNAL SEGMENT-TYPE WITHIN THE CONTAINING SEGMENT.

  NOTE THAT A COUNT FIELD IS REQUIRED BY DPROP FOR THE PROPAGATION OF INTERNAL SEGMENTS WITH TYPE=E PR'S.

2) FOR DPROP-TO-IMS MAPPING, THE SAMPLE EXIT DISTINGUISHES THE TWO FOLLOWING CASES:

A) IT IS CALLED DURING A REPLACE, DELETE, OR INSERT OF A ROW OF THE "CREDIT" TABLE.

  IN THIS CASE, THE EXIT ROUTINE GETS FOLLOWING TWO INPUTS FROM DPROP:
  - THE CHANGED OCCURRENCE OF THE INTERNAL SEGMENT IN ITS DPROP FORMAT.
    THIS INPUT HAS BEEN BUILT BY DPROP BY MAPPING THE CHANGED CREDIT ROW TO THE DPROP FORMAT OF THE INTERNAL SEGMENT.
  - THE EXISTING "BEFORE-CHANGE" IMS SEGMENT IN ITS IMS FORMAT.

  THE SEGMENT EXIT ROUTINE IS RESPONSIBLE BY COMBINING INFORMATION IN THESE TWO INPUTS TO BUILD THE NEW "AFTER-CHANGE" IMS SEGMENT IN ITS IMS FORMAT.

B) IT IS CALLED DURING A REPLACE, DELETE, OR INSERT OF A ROW OF THE "ACCOUNT TABLE".

  IN THIS CASE, THE EXIT ROUTINE GETS FOLLOWING TWO INPUTS FROM DPROP:
- THE CHANGED OCCURRENCE OF THE CONTAINING SEGMENT IN ITS DPROP FORMAT.
  THIS INPUT HAS BEEN BUILT BY DPROP BY MAPPING THE CHANGED ACCOUNT ROW TO THE DPROP FORMAT OF THE CONTAINING SEGMENT.

- THE EXISTING "BEFORE-CHANGE" IMS SEGMENT IN ITS IMS FORMAT (ONLY FOR REPLACES AND DELETES OF ROWS OF THE ACCOUNT TABLE).

THE SEGMENT EXIT ROUTINE IS RESPONSIBLE BY COMBINING INFORMATION IN THESE TWO INPUTS TO BUILD THE NEW "AFTER-CHANGE" IMS SEGMENT IN ITS IMS FORMAT.

/**

THE FIGURE BELOW DESCRIBES ON THE LEFT-HAND SIDE THE SEGMENT IN ITS IMS FORMAT AND ON THE RIGHT-HAND SIDE THE SEGMENT IN ITS DPROP FORMAT.

<table>
<thead>
<tr>
<th>IMS SEGMENT IN ITS IMS FORMAT</th>
<th>IMS SEGMENT IN ITS DPROP FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLD NAME</td>
<td>FLD</td>
</tr>
<tr>
<td>FMT</td>
<td>START</td>
</tr>
<tr>
<td>ACNT_NBR</td>
<td>C</td>
</tr>
<tr>
<td>NAME</td>
<td>C</td>
</tr>
<tr>
<td>COUNT</td>
<td>H</td>
</tr>
<tr>
<td>TYPE_1</td>
<td>P</td>
</tr>
<tr>
<td>AMOUNT_A</td>
<td>P</td>
</tr>
<tr>
<td>LIMIT_A</td>
<td>P</td>
</tr>
<tr>
<td>TYPE_2</td>
<td>P</td>
</tr>
<tr>
<td>AMOUNT_B</td>
<td>P</td>
</tr>
<tr>
<td>LIMIT_B</td>
<td>P</td>
</tr>
<tr>
<td>TYPE_3</td>
<td>P</td>
</tr>
<tr>
<td>AMOUNT_C</td>
<td>P</td>
</tr>
<tr>
<td>LIMIT_C</td>
<td>P</td>
</tr>
</tbody>
</table>

BOTH THE IMS FORMAT AND THE DPROP FORMAT OF THE IMS SEGMENT ARE DEFINED AS FIXED-LENGTH.

THE INTERNAL SEGMENT IS DEFINED TO DPROP AS FOLLOWS:

- IT HAS A VARIABLE NUMBER OF OCCURRENCES.
  (THE NUMBER OF OCCURRENCES IS IN THE COUNT FELD "COUNT" OF THE CONTAINING SEGMENT)
- THE FIRST OCCURRENCE STARTS AT A FIXED LOCATION WITHIN THE CONTAINING SEGMENT (START=33)
- IT HAS A FIXED LENGTH (15 BYTES)
- IT CONSISTS OF THE FOLLOWING FIELDS:
  THE 1-BYTE TYPE, 7-BYTES AMOUNT, AND 7-BYTES LIMIT

PLEASE REFER TO THE DSECTS TOWARDS THE BOTTOM OF THIS MODULE IN ORDER TO FIND ALL THE DETAILS ABOUT THE "IMS FORMAT" AND THE "DPROP FORMAT" OF THE SEGMENTS

Figure 17 (Part 3 of 11). Second Sample Segment Exit Routine (COBOL)
Following conventions are used to describe credit-info which do not exist:

- In the IMS format, a non-existing credit-info has a zero value in the field "limit".
- In the DPROP format, the count reflects the number of existing credit internal segments. Existing credit internal segments follow each other in the DPROP format of the IMS segment (non-existing internal segments are eliminated).
This must be so in order to conform to the way that internal segments are defined to DPROP and DXT.

Input:

1st parameter: address of DAX (DAX is the exit interface control block)
2nd parameter: address of segment in IMS format
3rd parameter: address of segment in DPROP format
4th parameter: address of anchor area preserved across calls to this exit.

Output:
The segment format transformation has been done.

Exit-Error=

- Return code = 12: Mapping problem / invalid data
- = 16: Should-not-occur errors
  - (Invalid call function, parameter area too small, invalid segment name).

Error messages issued by EKYESE2C

- EKYESE1E: Call function not supported
- EKYESE2E: Unsupported DBD or SEGNAME
- EKYESE3E: Unexpected length of IMS segment
- EKYESE4E: DPROP segment is too short
- EKYESE5E: IMS segment is too short
- EKYESE6E: Unexpected value in type column of credit table

Change Activity= None

-------------------- End of specifications --------------------------

-------------------- Logic of EKYESE2C -----------------------------

Main-line logic:

------------

1) Module Entry logic:

------------------------

- Set "module entered" and "module in control" flags into DAX.
- Verify that the exit is invoked to propagate the correct database and segment.
- Branch according to call function either for:
  - The processing of IMS-to-DPROP, or
  - The processing of DPROP-to-IMS.

Figure 17 (Part 4 of 11). Second Sample Segment Exit Routine (COBOL)
2) IMS-TO-DPROP FORMATTING

- Check length of segment in its IMS format and check that DPROP segment buffer is large enough.
- Move to DPROP format the accountnbr and the customer name.
- Initialize the number of internal segment occurrences to zero.
- For each non-zero limit in the IMS format:
  - Increase the occurrence counters by 1
  - Create in the DPROP buffer the id of the internal segment.
  - Move to the DPROP buffer the data of the internal segment.

Note: A limit with a zero value in the IMS format is considered to identify a "non-existing" credit information.

In the DPROP format there will be no occurrence of internal segments for these non-existing credits. As required by DPROP, the occurrences for the existing internal segments will follow each other.

3) DPROP-TO-IMS-FORMATING

- Check that IMS segment buffer is large enough.
- Initialize IMS segment buffer as follows:
  - If before-change image is provided by the caller, copy the before-change image to IMS buffer
  - Else initialize IMS buffer with proper initial values (zeros and blanks).
- If processing the change to the target of the containing segment:
  - Copy information of changed containing segment from DPROP buffer to IMS buffer.
- If processing the change to the target of an internal segment:
  - If processing a delete, set appropriate credit info to 0 in the IMS buffer
  - If processing a replace or insert, copy information of changed internal segment from DPROP buffer to IMS buffer.

Error logic

- Format an error message into DAX
- Set return code into DAX
- Return to the caller.

------------------------- END OF LOGIC --------------------------

Figure 17 (Part 5 of 11). Second Sample Segment Exit Routine (COBOL)
IDENTIFICATION DIVISION.
PROGRAM-ID. EKYESE2C.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

* 77  X1 PIC 9(8) COMP.
* INDEX FOR INTERNAL SEGMENTS IN IMS FORMAT
77  X2 PIC 9(8) COMP.
* INDEX FOR INTERNAL SEGMENTS IN DPROP FORMAT
77  IMSSEGL PIC 9(8) COMP VALUE +72.
* IMS SEGMENT LENGTH
77  DPRSEGL PIC 9(8) COMP VALUE +77.
* DPR SEGMENT LENGTH

*-----------------------------------------------------------------------*
* REDEFINITION OF THE MESSAGE AREA LOCATED IN THE DAX
*-----------------------------------------------------------------------*
01  MSGLINE.
  02  MSGID PIC X(11).
  02  MSGBL1 PIC X.
  02  MSGTXT PIC X(52).

*-----------------------------------------------------------------------*
* WORK AREA FOR THE IMS SEGMENT IN ITS DPROP-FORMAT
*-----------------------------------------------------------------------*
01  DPRSEG.
  02  DPRACNBR PIC X(9).
  02  DPRNAME PIC X(21).
  02  DPRCOUNT PIC 9(4) COMP.
  02  DPRINSEG OCCURS 3.
  03  DPRTYPE PIC 9 COMP-3.
  03  DPRAMOUN PIC 9(11)V99 COMP-3.
  03  DPRLIMIT PIC 9(11)V99 COMP-3.

*-----------------------------------------------------------------------*
* LINKAGE SECTION
*-----------------------------------------------------------------------*
LINKAGE SECTION.

*-----------------------------------------------------------------------*
* DESCRIPTION OF THE SEGMENT EXIT INTERFACE "DAX"
*-----------------------------------------------------------------------*
COPY EKYRCDXC.

*-----------------------------------------------------------------------*
* DESCRIPTION OF IMS SEGMENT IN ITS IMS FORMAT
*-----------------------------------------------------------------------*
01  IMSSEG.

Figure 17 (Part 6 of 11). Second Sample Segment Exit Routine (COBOL)
02 IMSACNBR PIC X(9).  * ACCOUNT NUMBER (KEY)
02 IMSNAME PIC X(21).  * NAME OF CUSTOMER
02 IMSINSEG OCCURS 3.  * 3 OCCURRENCES OF INTERNAL SEG
03 IMSAMOUN PIC 9(11)V99 COMP-3.  * CURRENT AMOUNT TYPE-A CREDIT
03 IMSLIMIT PIC 9(11)V99 COMP-3.  *
*---------------------------------------------------------------*
* THE THIRD PARAMETER CAN POINT TO DPRSEG OR TO DPRISEG        *
*---------------------------------------------------------------*
01 THIRDPARM PIC X(77).  *
01 CONTAINING REDEFINES THIRDPARM PIC X(32).  *
01 INTERNAL REDEFINES THIRDPARM PIC X(45).  *
*---------------------------------------------------------------*
* DSECT FOR THE BEFORE_CHANGE IMS IMAGE                         *
*---------------------------------------------------------------*
01 IMSBEFIM PIC X(72).  *
*---------------------------------------------------------------*
* PROCEDURE DIVISION                                               *
*---------------------------------------------------------------*
* PROCEDURE DIVISION USING DAX, IMSSEG, THIRDPARM.                *
*---------------------------------------------------------------*
* SET THE "EXIT ENTERED" AND "EXIT IN CONTROL" FLAGS.            *
*---------------------------------------------------------------*
* MOVE "X" TO DAXENTRD.
MOVE "X" TO DAXINCTL.
MOVE ZERO TO DAXRETC.
*---------------------------------------------------------------*
* VERIFY THAT THE EXIT IS CALLED TO FORMAT THE EXPECTED          *
* IMS DATABASE AND SEGMENT TYPE                                  *
*---------------------------------------------------------------*
* IF DAXDBNM NOT = "DB123"
   GO TO INVDBSEG.
   IF DAXSEGM NOT = "ACCOUNT"
      GO TO INVDBSEG.
*---------------------------------------------------------------*
* BRANCH ACCORDING TO CALL-FUNCTION                             *
*---------------------------------------------------------------*
* IF DAXCALL = "NO"
   GO TO IMSTDPR.
* IF DAXCALL = "RV"
   "REVERSE CALL" (DPROP TO IMS)
   GO TO INVCALL.
* IF DAXCALL = "NO"
   "NORMAL CALL" (IMS TO DPROP)
   GO TO IMSTDPR.
*}

Figure 17 (Part 7 of 11). Second Sample Segment Exit Routine (COBOL)
NORMAL CALL TO TRANSFORM THE SEGMENT FROM ITS IMS FORMAT INTO ITS DPROP FORMAT

IMSTDPR.

---------------------------------------------------------------
CHECK THE LENGTH OF SEGMENT IN ITS IMS FORMAT AND CHECK THAT THE DPROP BUFFER IS LARGE ENOUGH TO CONTAIN THE SEGMENT IN ITS DPROP FORMAT.

---------------------------------------------------------------

IF DAXDLEN NOT = IMSSEGL
  GO TO INVLENN1.
IF DAXFLEN < DPRSEGL
  GO TO INVLENN2.

---------------------------------------------------------------
MOVE THE ACCOUNT NUMBER AND CUSTOMER NAME TO DPROP FORMAT

---------------------------------------------------------------

MOVE IMSACNBR TO DPRACNBR.
MOVE IMSNAME TO DPRNAME.

---------------------------------------------------------------
INITIALIZE PROCESSING FOR THE THREE CREDITS:

---------------------------------------------------------------

MOVE ZERO TO DPRCOUNT.
MOVE ZERO TO X1, X2.
MOVE ZERO TO DPRTYPE (1), DPRTYPE (2), DPRTYPE (3).
MOVE ZERO TO DPRAMOUN (1), DPRAMOUN (2), DPRAMOUN (3).
MOVE ZERO TO DPRLIMIT (1), DPRLIMIT (2), DPRLIMIT (3).
PERFORM MOVECRED 3 TIMES.
MOVE DPRSEGL TO THIRDPARM.
RETURN IMS SEGMENT IN DPROP FORMAT
GO TO ENDPGM.

---------------------------------------------------------------
MOVE THE OCCURRENCE OF THE INTERNAL SEG FOR TYPE_A CREDITS.

---------------------------------------------------------------

MOVECRED.
ADD +1 TO X1.
INCREMENT INDEX FOR NEXT INT SEG
IF IMSLIMIT (X1) = ZERO
  NEXT SENTENCE
  SKIP IF THIS FIELD IS ZERO
ELSE
  ADD +1 TO X2
  MOVE X2 TO DPRCOUNT
INCREMENT COUNTER OF INTERNAL SEG
MOVE X1 TO DPRTYPE (X2)
MOVE IMSAMOUN (X1) TO DPRAMOUN (X2)
MOVE IMSLIMIT (X1) TO DPRLIMIT (X2).
SET ID, MOVE AMOUNT AND LIMIT
ENDMOVEC.

# REVERSE CALL TO TRANSFORM THE SEGMENT FROM ITS DPROP FORMAT INTO ITS IMS FORMAT
# *************************************************************************************************
DPRTIMS.

* CHECK THAT THE IMS BUFFER IS LARGE ENOUGH TO CONTAIN THE SEGMENT IN ITS IMS FORMAT.

* IF DAXDLEN < IMSSEGLEN
   GO TO INVLENN3.

* INITIALIZE THE AFTER_CHANGE IMS FORMAT AS FOLLOWS:
  * IF BEFORE-CHANGE IMAGE OF IMS SEGMENT HAS BEEN PROVIDED
    INIT THE AFTER-CHANGE IMAGE WITH BEFORE_CHANGE IMAGE
  * ELSE INIT THE AFTER-CHANGE IMAGE WITH PROPER INITIAL VALUES

* IF DAXIDDSB = NULL
   GO TO CALLRmzqr!z!t2mzqr!z!t.

* INITIALIZE AFTER-CHANGE IMAGE WITH BEFORE-CHANGE VALUES
  SET ADDRESS OF IMSBEFIM TO DAXIDDSB.
  ADDRESSING OF BEFORE_CHANGE IMAGE
  MOVE IMSBEFIM TO IMSSEG.
  MOVE BEFORE_CHANGE TO AFTER-CH.
  GO TO CALLR100.

* INITIALIZE AFTER-CHANGE IMAGE WITH PROPER INITIAL VALUES
  CALLRmzqr!z!t2mzqr!z!t.
  MOVE "mzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!t" TO IMSACNBR.
  MOVE SPACES TO IMSNAME.
  MOVE ZERO TO IMSAMOUN (1), IMSLIMIT (1).
  MOVE ZERO TO IMSAMOUN (2), IMSLIMIT (2).
  MOVE ZERO TO IMSAMOUN (3), IMSLIMIT (3).

* DETERMINE WHETHER WE ARE CALLED FOR A CHANGE TO THE ACCOUNT TABLE OR TO THE CREDIT TABLE.

* CALLR100.
  IF DAXSEGTI
     GO TO CALLR200.

* UPDATE OF INTERNAL SEGMENT

* EXIT ROUTINE IS CALLED FOR DPROP-TO-IMS MAPPING BECAUSE THE TARGET OF THE CONTAINING SEGMENT HAS CHANGED.
  WE WILL JUST MOVE INFORMATION FROM THE CONTAINING SEGMENT IN ITS DPROP FORMAT TO SEGMENT IN ITS IMS FORMAT

* MOVE CONTAINING TO DPRSEG.
  GET CONTAINING SEG IN DPROP FORMAT
  MOVE DPRACNBR TO IMSACNBR.
  MOVE DPRNAME TO IMSNAME.
  GO TO ENDPGM.

Figure 17 (Part 9 of 11). Second Sample Segment Exit Routine (COBOL)
EXIT ROUTINE IS CALLED FOR DPROP-TO-IMS MAPPING BECAUSE THE TARGET OF THE INTERNAL SEGMENT HAS CHANGED.

IF PROCESSING A DELETE, THE EXIT ROUTINE WILL ZERO THE APPROPRIATE AMOUNT AND LIMIT.

IF PROCESSING AN INSERT OR REPLACE, THE EXIT ROUTINE WILL COPY THE AMOUNT AND LIMIT FROM THE CHANGED INTERNAL SEGMENT TO THE IMS FORMAT OF THE SEGMENT.

- CALLR200.

* DETERMINE WHICH INTERNAL SEGMENT OCCURRENCE HAS CHANGED

* BRANCH DEPENDING ON THE TYPE OF UPDATE

* ERROR LOGIC:
  - BUILD IN THE INTERFACE CONTROL BLOCK AN ERROR MESSAGE
  - SET A RETURN CODE IN THE INTERFACE CONTROL BLOCK
  - RETURN TO CALLER OF THE EXIT

CALLR210.

IF DAXDPRCT NOT = "DLET"
  GO TO CALLR230.

*** A DELETE: ZERO CREDIT INFO IN IMS FORMAT

  MOVE 0 TO IMSAMOUN (X2).
  MOVE 0 TO IMSLIMIT (X2).
  GO TO ENDPGM.

*** INSERT OR REPLACE: COPY CHANGED CREDIT INFO INTO IMS FORMAT

CALLR230.

MOVE DPRAMOUN (1) TO IMSAMOUN (X2).
MOVE DPRLIMIT (1) TO IMSLIMIT (X2).
GO TO ENDPGM.

Figure 17 (Part 10 of 11). Second Sample Segment Exit Routine (COBOL)
INVDBSEG.
   MOVE "EKYESE2E" TO MSGID.
   MOVE SPACE TO MSGBL1.
   MOVE "UNSUPPORTED DBD OR SEGNAME" TO MSGTXT.
   GO TO INVRC16.
*
INVLENN1.
   MOVE "EKYESE3E" TO MSGID.
   MOVE SPACE TO MSGBL1.
   MOVE "UNEXPECTED LENGTH OF IMS SEGMENT" TO MSGTXT.
   GO TO INVRC16.
*
INVLENN2.
   MOVE "EKYESE4E" TO MSGID.
   MOVE SPACE TO MSGBL1.
   MOVE "DPROP SEGMENT BUFFER IS TOO SHORT" TO MSGTXT.
   GO TO INVRC16.
*
INVLENN3.
   MOVE "EKYESE6E" TO MSGID.
   MOVE SPACE TO MSGBL1.
   MOVE "IMS SEGMENT BUFFER IS TOO SHORT" TO MSGTXT.
   GO TO INVRC16.
*
INVTYPE.
   MOVE "EKYESE6E" TO MSGID.
   MOVE SPACE TO MSGBL1.
   MOVE "UNEXPECTED VALUE IN TYPE COLUMN OF CREDIT TABLE" TO MSGTXT.
   GO TO INVRC12.
*
INVRC12.
   MOVE MSGLINE TO DAXSMESG.
   MOVE 16 TO DAXRETC.
   * SET RETURN CODE 12 (ERROR)
   * GO TO ENDPGM.
*
INVRC16.
   MOVE MSGLINE TO DAXSMESG.
   MOVE 12 TO DAXRETC.
   * SET RETURN CODE 16 (SEVERE ERROR)
   * GO TO ENDPGM.
*
*---------------------------------------------------------------*
* RETURN TO CALLER OF THIS EXIT
*---------------------------------------------------------------*
* ENDPGM.
* GOBACK.
*---------------------------------------------------------------*

Figure 17 (Part 11 of 11). Second Sample Segment Exit Routine (COBOL)
Definitions for the Second Sample Segment Exit Routine

This section contains definitions associated with the second sample Segment exit routine. The following types of definitions are provided:

- IMS DBDGEN and PSBGEN definitions
- DB2 CREATE TABLE definitions
- DataRefresher definitions required to define the PR with DataRefresher and to extract the IMS data with DataRefresher
- SQL statements required to define the PR in the MVG Input Tables without DataRefresher

DBDGEN Definitions

Figure 18 show a DBDGEN definition for the Segment exit routine in Figure 17 on page 77.

```
DBD NAME=DB123,VERSION=V12, ACCESS=(HDAM,DSAM),RMNAME=(DFSHDC4,5,4), EXIT=(EKYRUP00)
DATASET DD1=HDAM,SIZE=496,DEVICE=3380
SEGMENT NAME=ACCOUNT,PARENT=9,BYTES=72
FIELD NAME=(ACNTNBR,SEQ,U),BYTES=9,START=1
```

Figure 18. DBDGEN Definition

Note: The EXIT= keyword of the DBD Macro specifies that EKYRUP00 (the RUP) be called when a segment of this DBD is changed. This is required for synchronous data propagation.

PSBGEN Definitions

Figure 19 shows a PSBGEN definition for the Segment exit routine in Figure 17 on page 77.

```
PCB TYPE=DB, ...
SENSEG ...
PCB TYPE=DB, ...
SENSEG ...
PCB TYPE=DB, DBDNAME=DB123, NAME=HUPPCB, KEYLEN=72, PROCOPT=A
SENSEG NAME=ACCOUNT
PSBGEN PSBNAME=PSBDPR3
END
```

Figure 19. PSBGEN Definition

Note: The first two PCBs represent PCBs used by the application programs. The third PCB, HUPPCB, is the PCB reserved for HUP usage.
CREATE TABLE Statements

Figure 20 contains the CREATE TABLE statements used to create the ACCOUNT table and the CREDIT table in Figure 17 on page 77.

The figure contains the CREATE UNIQUE INDEX statements required to create the indexes for the DB2 primary keys of the two tables.

```
CREATE TABLE ACCOUNT
    (ACT_NBR CHAR(9) NOT NULL,
    NAME CHAR(21) NOT NULL WITH DEFAULT,
    PRIMARY KEY (ACT_NBR))
DATA CAPTURE CHANGES
IN DU096606.PROPT1 ;

CREATE UNIQUE INDEX XN01 ON ACCOUNT
    (ACT_NBR)
    USING VCAT KOE ;

CREATE TABLE CREDIT
    (ACT_NBR CHAR(9) NOT NULL,
    TYPE DECIMAL (1) NOT NULL,
    AMOUNT DECIMAL (13,2) NOT NULL WITH DEFAULT,
    LIMIT DECIMAL (13,2) NOT NULL WITH DEFAULT,
    PRIMARY KEY (ACT_NBR,TYPE),
    FOREIGN KEY (ACT_NBR) REFERENCES ACCOUNT ON DELETE CASCADE)
DATA CAPTURE CHANGES
IN DU096606.PROPT2 ;

CREATE UNIQUE INDEX XN02 ON CREDIT
    (ACT_NBR, TYPE)
    USING VCAT KOE ;
```

Figure 20. CREATE TABLE Statements

**Note:** The DATA CAPTURE CHANGES option of the CREATE TABLE command specifies that the DB2 Changed Data Capture exit (the HUP) be called when a row of this table is changed under IMS attach. The FOREIGN KEY option is used if one-way DB2-to-IMS propagation or two-way propagation is implemented. The containing segment/internal segment relationship should be handled just as a parent/child segment is handled for setting up matching RIRs. In this example, the DELETE CASCADE option is used.

Using DataRefresher To Define the PR: CREATE DXTPSB

Figure 21 on page 90 shows a CREATE DXTPSB definition for the Segment exit routine in Figure 17 on page 77.
Figure 21. Using DataRefresher to Define the PR: CREATE DXTPSB

Notes:

1. The DXTPCB has two SEGMENT statements, which are followed by FIELD statements.
   - The first SEGMENT statement and its fields describe the containing IMS segment ACCOUNT in its DPROP format.
   - The second SEGMENT statement and its fields describe the internal segment type CREDIT in its DPROP format.

2. The Segment exit routine EKYESE2C is specified on the EXIT= keyword of the SEGMENT statement.
   The EXIT= keyword must be provided on the SEGMENT statement describing the containing segment, never on SEGMENT statements describing internal segments.
   The SEGMENT statement for the ACCOUNT segment also provides the following specifications:
   - BYTES=72 specifies the length of the segment in its IMS format.
   - XBYTES=77 specifies the length of the segment in its DPROP format.

3. The SEGMENT statement for the segment CREDIT describes the internal segment in its DPROP format.
   - FORMAT=FI specifies that the segment has a fixed length and is an internal segment.
   - OCCURS=COUNT specifies that the internal segment has a variable number of occurrences, and that the count field for the internal segment is the field called COUNT. The count field must be defined with a FIELD statement as a field of the containing segment (not as a field in the internal segment).
   - START=33 specifies that the internal segment starts at byte 33 of the containing segment (in its DPROP format).
   - BYTES=15 specifies that the internal segment has a length of 15 bytes.
Using DataRefresher to Define the PR: CREATE DXTVIEW

Figure 22 shows a CREATE DXTVIEW definition for the Segment exit routine in Figure 17 on page 77.

```
CREATE
  DXTVIEW NAME = VIEW CRED,
  DXTPSB = KOEPSB2,
  DXTPCB = PCB001,
  SEGMENT = CREDIT,
  MINSEG = CREDIT,
  FIELDS = * ;
```

Using DataRefresher To Define the PR

This section covers the DataRefresher UIM SUBMIT Command and EXTRACT Statement.

Figure 23 on page 92 contains two pairs of SUBMIT and EXTRACT statements to define the two PRs, PR1 and PR2.

PR1 propagates the fields ACT_NBR and NAME of the containing segment to the ACCOUNT table.

PR2 propagates the fields TYPE, AMOUNT, and LIMIT of the internal segment to the CREDIT table. PR2 propagates the key field ACT_NBR of the containing parent segment ACCOUNT to the CREDIT table.
Figure 23. Using DataRefresher to Define the PR: DataRefresher UIM SUBMIT Command and EXTRACT Statement

Notes:

1. The MAPEXIT= keyword of the SUBMIT control statement specifies EKYMCE00. This causes DataRefresher UIM to call the DPROP-provided Map Capture Exit EKYMCE00 during processing of the SUBMIT or EXTRACT. This is needed to allow DPROP to create the PR.

2. MAPUPARM= is used to provide the DPROP propagation keywords.

MAPCASE=1 defines PR1 as a mapping case 1 PR, and MAPCASE=3 defines PR2 as a mapping case 3 PR.

3. The EXTRACT statement describes to DataRefresher and DPROP which fields should be mapped to which columns.

The COUNT field is not propagated by either PR1 or PR2.
Using DataRefresher For the Extract

This section covers INITDEM and USE DXTPSB Control Statements. Figure 24 shows INITDEM and USE DXTPSB control statements for the Segment exit routine in Figure 17 on page 77.

INITDEM NAME=DEMPROD;
USE DXTPSB=KOEPSB2;

Figure 24. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements

Defining the PR in the MVG Input Tables

Figure 25 on page 95 describes the DSNTEP2 SQL statements required to define the two PRs, PR1 and PR2, in the MVG input tables.

PR1 propagates to the ACCOUNT table the fields ACT_NBR and NAME of the containing segment.

PR2 propagates to the CREDIT table the fields TYPE, AMOUNT and LIMIT of the internal segment. PR2 propagates to the CREDIT table the key field ACT_NBR of the containing parent segment ACCOUNT.

The following rows are inserted into the MVG input tables to define PR1:

- One row is inserted into the DPRIPR table (the PR table).
  
  This row identifies the PRID by inserting an E into the PRTYPE column and a 1 into the MAPCASE column. The SQL statement specifies that the PR belongs to mapping case 1 of an extended-function PR.

- One row for the entity segment type ACCOUNT is inserted into the DPRISSEG table (the SEG table).

Because ACCOUNT is the root segment, no rows are inserted into DPRISSEG for physical ancestors.

The row describing ACCOUNT provides the following column values:

  - Value E in the ROLE column specifies that the segment is the entity segment of the PR.
  - Nonblank value EKYESE2C in the SEGEXIT column specifies that the segment must be processed by the Segment exit routine EKYESE2C.
  - Value 77 in the SEGEXITL column specifies the length of the segment in its DPROP format.
  - Value F in the SEGEXITF column specifies that the segment in its DPROP format has a fixed length.

- One row is inserted into the DPRITAB table (the TAB table).
  
  This row specifies that the target table is T096606.ACCOUNT.

- One row is inserted into the DPRIFLD table (the FLD table) for each propagated field.

The DPRIFLD rows describe the fields as they appear in the DPROP format of the segment (as opposed to the segment in its IMS DB format).
The following rows are inserted into the MVG input tables to define PR2:

- One row is inserted into the DPRIPR table (the PR table).

  This row identifies the PRID by inserting an E into the PRTYPE column and a 3 into the MAPCASE column. The SQL statement specifies that the PR belongs to mapping case 3 of an extended-function PR.

- One row for the Containing segment Type ACCOUNT is inserted into the DPRISEG table (the SEG table).

  Because ACCOUNT is the root segment, no rows are inserted into DPRISEG for physical ancestors.

  The row describing ACCOUNT provides the following column values:
  - Value C in the ROLE column specifies that the segment is the containing segment of the mapping case 3 PR.
  - Nonblank value EKYESE2C in the SEGEXIT column specifies that the segment must be processed by the Segment exit routine EKYESE2C.
  - Value 77 in the SEGEXITL column specifies the length of the segment in its DPROP format.
  - Value F in the SEGEXITF column specifies that the segment in its DPROP format has a fixed length.

- One row for the internal segment type CREDIT is inserted into the DPRISEG table (the SEG table).

  The row describing CREDIT provides the following column values:
  - Value E in the ROLE column. This specifies that the segment is the Entity segment of the PR.
  - Blank value in the SEGEXIT column. This is because DPROP requires that the Segment exit routine be defined in the DPRISEG row of the containing segment (not in the DPRISEG row of the internal segment).
  - Value FI in the FORMAT column specifies that the segment has a fixed length and is an internal segment type.
  - Value COUNT in the OCCURS column specifies that the internal segment has a variable number of occurrences and that the number of occurrences is stored in the field COUNT.
  - Value 33 in the START column specifies that the first occurrence of the internal segment starts at location 33 within the containing segment (in its DPROP format).
  - Value 15 in the BYTES column specifies that the length of the internal segment type is 15 bytes.

- One row is inserted into the DPRITAB table (the TAB table).

  This row specifies that the target table is T096606.CREDIT.

- One row is inserted into the DPRIFLD table (the FLD table) for each propagated field. Another row is inserted into the DPRIFLD table for the COUNT field.

  The DPRIFLD rows describe the fields as they appear in the DPROP format of the segment (as opposed to the segment in its IMS DB format).
The row describing the field ACT_NBR has the value ACCOUNT in the SEGNAME column. This specifies that the ACT_NBR field is located in the containing segment ACCOUNT (not in the internal segment).

The row describing the field COUNT has a blank value in the COLNAME column, because the COUNT field is not propagated. The value ACCOUNT in the SEGNAME column specifies that the COUNT field is located in the containing segment ACCOUNT (not in the internal segment).

The row describing the fields TYPE, AMOUNT, and LIMIT COUNT have the value CREDIT in the SEGNAME column. This specifies that these fields are located in the internal segment CREDIT.

DELETE FROM T96606.DPRIPR WHERE PRID = 'PR1';

INSERT INTO T96606.DPRIPR
( PRID , USERID , PRTYPE, MAPCASE, MAPDIR, ERROPT , ACTION)
VALUES ('PR1' , 'T96606', 'E' , '1' , 'TW', 'BACKOUT', 'REPL') ;

INSERT INTO T96606.DPRISEG
( PRID , DBNAME , SEGNAME , ROLE , PCBLABEL, SEGEXIT , SEGEXITL, SEGEXITF )
VALUES ('PR1' , 'DB123' , 'ACCOUNT', 'E' , 'HUPPCB' , 'EYSEZC' , 77 , 'F') ;

INSERT INTO T96606.DPRITAB
( PRID , TABQUAL , TABNAME )
VALUES ('PR1', 'T96606', 'ACCOUNT') ;

INSERT INTO T96606.DPRIFLD
( PRID , DBNAME , SEGNAME , FLDNAME,
  TABQUAL , TABNAME , COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR1' , 'DB123' , 'ACCOUNT','ACT_NBR',
  'T96606' , 'ACCOUNT', 'ACT_NBR', 'C' , 1 , 9 ) ;

INSERT INTO T96606.DPRIFLD
( PRID , DBNAME , SEGNAME , FLDNAME,
  TABQUAL , TABNAME , COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR1' , 'DB123' , 'ACCOUNT', 'NAME',
  'T96606' , 'ACCOUNT', 'NAME' , 'C' , 10 , 21 ) ;

Figure 25 (Part 1 of 2). Defining the PR in the MVG Input Tables
DELETE FROM T096606.DPRIPR WHERE PRID = 'PR2' ;

INSERT INTO T096606.DPRIPR
(PRID, USERID, PRTYPE, MAPCASE, MAPDIR, ERROPT, ACTION)
VALUES ('PR2', 'T096606', 'E', '3', 'TW', 'BACKOUT', 'REPL') ;

INSERT INTO T096606.DPRISEG
(PRID, DBNAME, SEGNAME, ROLE, PCBLABEL, SEGEXIT, SEGEXITL, SEGEXITF)
VALUES ('PR2', 'DB123', 'ACCOUNT', 'C', 'HUPPCB', 'EKYESE2C', 77, 'F') ;

INSERT INTO T096606.DPRISEG
(PRID, DBNAME, SEGNAME, ROLE, PCBLABEL, SEGEXIT, SEGEXITL, SEGEXITF, FORMAT, OCCURS, START, BYTES)
VALUES ('PR2', 'DB123', 'ACCOUNT', 'E', '', ' ', mzqr!z!t, '', 'FI', 'COUNT', '33', 15) ;

INSERT INTO T096606.DPRITAB
(PRID, TABQUAL, TABNAME)
VALUES ('PR2', 'T096606', 'CREDIT') ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR2', 'DB123', 'ACCOUNT', 'ACT_NBR', 'T096606', 'CREDIT', 'ACT_NBR', 'C', 1, 9) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR2', 'DB123', 'ACCOUNT', 'COUNT', 'T096606', 'CREDIT', ' ', 'H', 31, 2) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR2', 'DB123', 'CREDIT', 'TYPE', 'T096606', 'CREDIT', 'TYPE', 'P', 1, 1, mzqr!z!t) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR2', 'DB123', 'CREDIT', 'AMOUNT', 'T096606', 'CREDIT', 'AMOUNT', 'P', 2, 7, 2) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR2', 'DB123', 'CREDIT', 'LIMIT', 'T096606', 'CREDIT', 'LIMIT', 'P', 9, 7, 2) ;

Figure 25 (Part 2 of 2). Defining the PR in the MVG Input Tables
Third Sample Segment Exit Routine

Figure 26 on page 98 contains an example of a Segment exit routine in PL/I. Its functions are the same as those for the exit routine in “Second Sample Segment Exit Routine” on page 75. For information about this routine, refer to “Second Sample Segment Exit Routine” on page 75.

The source code in Figure 26 on page 98 is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYESE2P. The definitions for this routine are the same as those for EKYESE2C, except that the exit name is different. Specifically, the \texttt{EXIT=EKYESE2C} in Figure 21 on page 90, and both occurrences of \texttt{EKYESE2C} in Figure 25 on page 95, are changed to \texttt{EKYESE2P}. The text that refers to EKYESE2C is also true for EKYESE2P. Refer to “Definitions for the Second Sample Segment Exit Routine” on page 88 for information about the definitions.
*PROCESS MAP(2,72,1);
EKYESE2P: PROCEDURE /* Sample Segment Exit Routine */
(DAX_PARM_PTR,
 IMSSEG_PARM_PTR,
 DPRSEG_PARM_PTR,
 USERAREA_PARM_PTR)
 OPTIONS (FETCHABLE REENTRANT);

******************************************************************************
*  Licensed Materials - Property of IBM
*  See Copyright Instructions
******************************************************************************

Module name: EKYESE2P
*  Descriptive name: Sample PL/I Segment Exit Routine
*  Function: The intent of this program is to provide a sample of
  a segment exit routine. This example is used for the
  transformation of a segment layout between its:
  - IMS format
  - DPROP format.
  EKYESE2P illustrates the usage of a DPROP segment exit to support
  the propagation of an IMS segment containing an internal
  segment / repeating group of fields.

This sample segment exit routine supports TYPE=E PR's and is
therefore called both for:
  - IMS-to-DPROP mapping
  (e.g. during IMS-to-DB2 propagation, also during
  DXT-extracts, CCU and DLU processing).
  - DPROP-to-IMS mapping
  (e.g. during DB2-to-IMS propagation, also during CCU and
  DLU processing).

In this example the propagated IMS segment is a bank account
segment. The IMS segment consists of the following fields:
  - The account number (this is the key of the segment).
  - The customer name.
  - A repeating group of fields with three occurrences.
    Each occurrence of the repeating group contains information
    about one type of credit that the bank is granting.
    This data is:
      - the current amount of credit granted to the
        customer/account.

Figure 26 (Part 1 of 12). Third Sample Segment Exit Routine (PL/I)
The database administrator wants to have a normalized DB2 table design and therefore wants to:

1) Propagate the account number and customer name to/from the table called "ACCOUNT". This is done with a mapping-case-1 propagation request.

2) Propagate the information akin to the different types of credits (together with the account number) to/from another table called "CREDIT".

This is done with a mapping-case-3 propagation request.

Each occurrence of the credit information (there are three of them) is considered to be an occurrence of an internal and is propagated to/from one row of the table "CREDIT".

To distinguish between the three types of credit information within the CREDIT table (and in order to have a DB2 primary key), the CREDIT table does not only contain an account number column and the current credit amount and limit. The CREDIT table also contains a "TYPE" column which identifies the type of credit.

The sample segment exit routine EKYESE2P provides logic to support the propagation of the IMS segment to/from the two tables "ACCOUNT" and "CREDIT".

1) For IMS-to-DPROP mapping the sample exit provides the following functions, when building the DPROP format of the segment:

- The exit routine creates in the DPROP-format an "ID" field for each occurrence of the internal segment. This is the field called "TYPE".

This addresses the DPROP requirement that internal segments have an "ID" field uniquely identifying the occurrences of the internal segments within the containing segment.

In the DPROP format, each occurrence of the internal segment will consist of the following fields:
- The field "TYPE" (this is the "ID" field created by the exit).
- The field "AMOUNT" (copied from the IMS format of the segment).
- The field "LIMIT" (copied from the IMS format of the segment).
- The exit routine creates in the DPROP-format a count field. Its value is the number of occurrences of the internal segment-type within the containing segment.

Figure 26 (Part 2 of 12). Third Sample Segment Exit Routine (PL/I)
Note that a count field is required by DPROP for the propagation of internal segments with TYPE=E PR's.

For DPROP-to-IMS mapping the sample exit differentiates between the two following cases:

a) It is called during a REPLACE, DELETE, or INSERT of a row of the "CREDIT" table.

Here, the exit routine gets the ensuing two inputs from DPROP:

- The changed occurrence of the internal segment in its DPROP format. This input has been built by DPROP by mapping the changed CREDIT row to the DPROP format of the internal segment.
- The existing "before-change" IMS segment in its IMS format.

By combining information from these two inputs the segment exit routine is responsible for building the new "after-change" IMS segment in its IMS format.

b) It is called during a REPLACE, DELETE, or INSERT of a row of the "ACCOUNT" table.

Here, this exit routine gets following two inputs from DPROP:

- The changed occurrence of the containing segment in its DPROP format. This input has been built by DPROP by mapping the changed ACCOUNT row to the DPROP format of the containing segment.
- The existing "before-change" IMS segment in its IMS format (only for REPLACES and DELETES of rows of the ACCOUNT table).

By combining information from these two inputs the segment exit routine is responsible for building the new "after-change" IMS segment in its IMS format.

The figure below describes on the left-hand side the segment in its IMS format and on the right-hand side the segment in its DPROP format.

---

Figure 26 (Part 3 of 12). Third Sample Segment Exit Routine (PL/I)
Both the IMS format and the DPROP format of the IMS segment are defined as fixed-length.

The internal segment is defined to DPROP as follows:

- It has a variable number of occurrences (the number of occurrences is in the count field "COUNT" of the containing segment).
- The first occurrence starts at a fixed location within the containing segment (start position = 33).
- It has a fixed length (15 bytes).
- It consists of the following fields:
  - TYPE (1 byte).
  - AMOUNT (7 bytes).
  - LIMIT (7 bytes).

Please refer to the DSECTS found later in this module to find all the details about the "IMS format" and the "DPROP format" of the segments.

The following conventions are used to describe credit information if it does not exist:

- In the IMS format, a non-existing credit-info has a zero value in the field "LIMIT".

- In the DPROP-format, the count reflects the number of existing CREDIT internal segments. Existing CREDIT internal segments follow each other in the DPROP format of the IMS segment (non-existing internal segments are eliminated). This must be so in order to conform to the way that internal segments are defined to DPROP and DXT.

Input: 1st parameter: Address of DAX (DAX is the exit interface control block).
2nd parameter: Address of segment in IMS format.
3rd parameter: Address of segment in DPROP format.
4th parameter: Address of anchor area preserved across calls to this exit.

Figure 26 (Part 4 of 12). Third Sample Segment Exit Routine (PL/I)
* Output: the segment format transformation has been completed.

* * *

* Exit-error:

* *

* Return code  = 12 - mapping problem / invalid data
* = 16 - should-not-occur errors (invalid call
* function, parameter area too small,
* invalid segment name).

* *

* Error messages issued by EKYESE2P:

* *
* EKYESE2P-1E: Call function not supported.
* EKYESE2P-2E: Unsupported DBD or segment name.
* EKYESE2P-3E: Unexpected length of IMS segment.
* EKYESE2P-4E: DPROP segment is too short.
* EKYESE2P-5E: IMS segment is too short.
* EKYESE2P-6E: Unexpected value in TYPE column of CREDIT
* table.

* *
* Change activity= none

* *

*************** End of Specifications ********************

1  ******************** Logic of EKYESE2P ********************

* *

* Main logic:

* *
* 1) Module entry logic:

* *
* - Set "module entered" and "module in control" flags into DAX.
* - Validate that the exit is invoked to propagate the correct
* database and segment.
* - Process according to call-function either for:
* - The processing of IMS-to-DPROP, or
* - The processing of DPROP-to-IMS

* *
* 2) IMS-to-DPROP formatting:

* *
* - Check length of segment in its IMS format and check that the
* size of the DPROP segment buffer is sufficient to contain
* the segment in its DPROP format.
* - Assign the account number and the customer name to the DPROP
* format.
* - Initialize the computation of internal segment occurrences
* to zero.
* - For each non-zero limit in the IMS format:
* - Increase the occurrence counters by 1.
* - Create the ID of the internal segment in the DPROP buffer.
* - Move the data of the internal segment to the DPROP buffer.


Figure 26 (Part 5 of 12). Third Sample Segment Exit Routine (PL/I)
Note: A limit with a zero value in the IMS format is deemed as identifying "non-existing" credit information. In the DPROP format there will be no occurrence of internal segments for these non-existing credits. As required by DPROP, the occurrences for the existing internal segments will follow each other.

3) DPROP-to-IMS formatting
- Check that the IMS segment buffer is large enough.
- Initialize the IMS segment buffer as follows:
  - If before-change image is provided by the caller, copy the before-change image to the IMS buffer,
  otherwise initialize the IMS buffer with the appropriate initial values (zeroes and blanks).
  - If processing a change to the target of a containing segment:
    - Copy information of the changed containing segment from the DPROP buffer to the IMS buffer.
  - If processing a change to the target of an internal segment:
    - If processing a DELETE, set the appropriate CREDIT information to zero in the IMS buffer.
    - If processing a REPLACE or INSERT, copy the information of the changed internal segment from the DPROP buffer to the IMS buffer.

Error Logic:
- Format an error message in the "DAX".
- Set a return code in the "DAX".
- Return to the caller.

************************************************************** End of logic summary *************************************/
%INCLUDE EKYRCDXP; /* DAX control block structure */
/* Description of IMS segment in its IMS format */
***************************************************************/

DECLARE IMSSEGBUFF CHAR(72) BASED(IMSSEG_POINTER);

DECLARE 1 IMSSEG BASED(IMSSEG_POINTER),
  2 IMSACNBR PIC'(8)9T', /* Account number (KEY) */
  2 IMSNAME CHAR(21), /* Name of customer */
  2 IMSINSEG(3), /* 3 occurrences of internal seg */
  3 IMSAMOUNT FIXED DEC(13), /* Current amount type-a credit */
  3 IMSLIMIT FIXED DEC(13);

Figure 26 (Part 6 of 12). Third Sample Segment Exit Routine (PL/I)
/* Description of IMS segment in its DPROP format */

DECLARE DPRSEGBUFF CHAR(77);

DECLARE 1 DPRSEG BASED(DPRSEG_POINTER),
2 DPRACNBR PIC '(8)9T', /* Account number (key) */
2 DPRNAME CHAR(21), /* Name of customer */
2 DPRCOUNT FIXED BIN(15), /* Internal segment occurrences count */
2 DPRINSEG(3), /* three occurrences of internal segment */
3 DPRTYPE FIXED DEC(1), /* ID */
3 DPRAMOUNT FIXED DEC(13), /* Current amount */
3 DPRLIMIT FIXED DEC(13); /* Limit amount */

/* Description of Internal segment in DPROP format */

DECLARE DPRI_SEG_POINTER POINTER;

DECLARE 1 DPRI_SEG BASED (DPRI_SEG_POINTER),
2 DPRI_TYPE FIXED DEC(1), /* ID */
2 DPRI_AMOUNT FIXED DEC(13), /* Current amount */
2 DPRI_LIMIT FIXED DEC(13); /* Limit amount */

/* Description of Containing segment in DPROP format */

DECLARE DPRC_SEG_POINTER POINTER;

DECLARE 1 DPRC_SEG BASED (DPRC_SEG_POINTER),
2 DPRC_ACNBR PIC '(8)9T', /* Account number (KEY) */
2 DPRC_NAME CHAR(21), /* Name of customer */
2 DPRC_COUNT FIXED BIN(15); /* Internal segment occurrences count */

/* Area for the before-change IMS image */

DECLARE IMSBEFIM_POINTER POINTER;

DECLARE IMSBEFIM CHAR(72) BASED(IMSBEFIM_POINTER);

/* Built-in functions and global variable declarations */

DECLARE DAX_PARM_PTR POINTER;
DECLARE IMSSEG_PARM_PTR POINTER;
DECLARE DPRSEG_PARM_PTR POINTER;
DECLARE USERAREA_PARM_PTR POINTER;
DECLARE DAX_POINTER POINTER;
DECLARE IMSSEG_POINTER POINTER;
DECLARE DPRSEG_POINTER POINTER;

Figure 26 (Part 7 of 12). Third Sample Segment Exit Routine (PL/I)
DECLARE I FIXED BIN(31);
DECLARE J FIXED BIN(31);
DECLARE ADDR BUILTIN;
DECLARE NULL BUILTIN;

/* Declarations for initialized variables */
DECLARE EKYESE1E FIXED BIN(31) INIT(1);
DECLARE EKYESE2E FIXED BIN(31) INIT(2);
DECLARE EKYESE3E FIXED BIN(31) INIT(3);
DECLARE EKYESE4E FIXED BIN(31) INIT(4);
DECLARE EKYESE5E FIXED BIN(31) INIT(5);
DECLARE EKYESE6E FIXED BIN(31) INIT(6);
DECLARE RETURN_CODE_12 FIXED BIN(31) INIT(12);
DECLARE RETURN_CODE_16 FIXED BIN(31) INIT(16);
DECLARE IMSSEGL FIXED BIN(31) INIT(72);
DECLARE DPRSEGL FIXED BIN(31) INIT(77);
DECLARE X CHAR(1) INIT('X');
DECLARE NO CHAR(2) INIT('NO');
DECLARE RV CHAR(2) INIT('RV');
DECLARE DB123 CHAR(8) INIT('DB123 ');
DECLARE ACCOUNT CHAR(8) INIT('ACCOUNT ');

/* Format error messages and place into DAX control block */
WRITE_ERROR_MESSAGE: PROCEDURE(MESSAGE_NUM,RETURN_CODE);

DECLARE MESSAGE_NUM FIXED BIN(31); /* Message identifier input */
DECLARE RETURN_CODE FIXED BIN(31); /* Return code input */

/* Buffer and structure for assignment of DAX message */
DECLARE MSGBUFF CHAR(64);
DECLARE MSG_PTR POINTER;
DECLARE 1 MSGLINE BASED(MSG_PTR),
2 MSGID CHAR(11),
2 MSGBL1 CHAR(1),
2 MSGTXT CHAR(52);

/* Error and informative message declarations */
DECLARE MESSAGE_ID(6) CHAR(11) INIT
('EKYESE2P-1E', 'EKYESE2P-2E', 'EKYESE2P-3E',
 'EKYESE2P-4E', 'EKYESE2P-5E', 'EKYESE2P-6E');

DECLARE MESSAGE_TEXT(6) CHAR(52) INIT
('Call function not supported. ',
 'Unsupported DBD or segname. ',
 'Unexpected length of IMS segment. ',
 'DPROP segment buffer is too short. ',
 'IMS segment buffer is too short. ',
 'Unexpected value in type column of credit table. ');
/* Format message and assign return code to DAX */
MSGBUFF = ' ';
/* Blank out message buffer */
MSG_PTR = ADDR(MSGBUFF);
msgid = MESSAGE_ID(MESSAGE_NUM);
msgtxt = MESSAGE_TEXT(MESSAGE_NUM);
DAXRETC = RETURN_CODE;
/* Return code into DAX */
DAXMSMG = MSGBUFF;
/* Formatted message to DAX */
END WRITE_ERROR_MESSAGE;

/* Normal call to transform the segment from its IMS format into its DPROP format */
IMS_TO_DPROP: PROCEDURE;

/* Establish addressability to DPROP format segment data area */
DPRSEG_POINTER = ADDR(DPRSEG_PARM_PTR);

/* Check the length of segment in its IMS format and check that the DPROP buffer is large enough to contain the segment in its DPROP format. */
IF DAXDLEN ¬= IMSSEGL THEN
  /* Unexpected length of IMS segment */
  DO;
    CALL WRITE_ERROR_MESSAGE(EKYESE3E,RETURN_CODE_16);
    GOTO FIN; /* Terminate */
END;
IF DAXFLEN < DPRSEGL THEN
  /* DPROP segment buffer is too short */
  DO;
    CALL WRITE_ERROR_MESSAGE(EKYESE4E,RETURN_CODE_16);
    GOTO FIN; /* Terminate */
END;

/* Assign the account number and customer name to DPROP format */
DPRACNBR = IMSACNBR;
DPRNAME = IMSNAME;

/* Initialize counter field to zero and initialize processing for the three credits */
DPRCOUNT = 0;
DPRINSEG = 0;
J = 0;

DO I = 1 TO 3;
  /* Move credits */
  IF IMSLIMIT(I) ¬= mzqr!z!t
    /* i.e. non zero limit */
    THEN DO;
      J = J + 1;
      DPRCOUNT = J; /* Increase occurrence counter */
    END;
END;

Figure 26 (Part 9 of 12). Third Sample Segment Exit Routine (PL/I)
DPRTYPE(J) = I; /* Assign ID of internal segment */
DPRAMOUNT(J) = IMSAMOUNT(I); /* Assign current amount */
DPRLIMIT(J) = IMSLIMIT(I); /* Assign limit amount */
END; /* IMSLIMIT(I) = 0 */
END; /* I */

END IMS_TONDPROP;
1/*************************************************************************/
/* Reverse call to transform the segment from its DPROP format into */
/* its IMS format */
/*************************************************************************/

DPROP_TO_IMS: PROCEDURE;
DECLARE INTERNAL CHAR(1) INIT('I');
DECLARE DLET CHAR(4) INIT('DLET');
DECLARE NINE_ZEROS PIC'(8)9T' INIT('0');
DECLARE BLANKS_21 CHAR(21) INIT(' '); /* Initialize the after change IMS format as follows: */
/* If before change image of IMS segment has not been provided, */
/* initialize the after change image with the appropriate */
/* initial values */
/* else */
/* initialize the after change image with before change image */
*************************************************************************/
IF DAXDLEN < IMSSEGL THEN /* IMS segment buffer is too short */
DO;
CALL WRITE_ERROR_MESSAGE(EKYESE5E,RETURN_CODE_16);
GOTO FIN; /* Terminate */
END;
*************************************************************************/

/*************************************************************************/
/* Check that the IMS buffer is large enough to contain the */
/* segment in its IMS format. */
*************************************************************************/
IF DAXDLEN < IMSSEGL THEN /* IMS segment buffer is too short */
DO;
CALL WRITE_ERROR_MESSAGE(EKYESE5E,RETURN_CODE_16);
GOTO FIN; /* Terminate */
END;
*************************************************************************/

/*************************************************************************/
/* Initialize the after change IMS format as follows: */
/* */
/* If before change image of IMS segment has not been provided, */
/* initialize the after change image with the appropriate */
/* initial values */
/* else */
/* initialize the after change image with before change image */
*************************************************************************/
IF DAXIDDSB = NULL /* Before-change image IS NOT provided */
THEN DO;
IMSACNBR = NINE_ZEROS; /* Initialize account number to zeros */
IMSNAME = BLANKS_21; /* Initialize customer name to blanks */
IMSMINSEG = 0; /* Initialize amounts/limits to zero */
END; /* DAXIDDSB is NULL */
ELSE DO; /* Before-change image IS provided */
IMSBEFIM POINTER = DAXIDDSB; /* Address of before-change image */
IMSSSEGBUFF = IMSBEFIM; /* Assign before-change to after-change */
END;
1 IF DAXSEG = INTERNAL /* value is "I" (i.e. internal segment) */
THEN
*************************************************************************/
/* Exit routine is called for DPROP-to-IMS mapping because the */
/* target of the internal segment has changed. */
/* */
/* When processing a DELETE - */
/* the exit routine will zero the appropriate amount and limit. */
/* When processing an INSERT or REPLACE - */
/* the exit routine will copy the amount and limit from the */
/* changed internal segment to the IMS format of the segment. */
*************************************************************************/

Figure 26 (Part 10 of 12). Third Sample Segment Exit Routine (PL/I)
DO;
DPRSEG_POINTER = ADDR(DPRSEGBUFF);
/****************************
* Establish addressability to DPROP format segment data area *
*----------------------------------------------------------------------
DPRSEG_POINTER = ADDR(DPRSEGBUFF);
DPRINSEG = DPRSEG_POINTER;
SELECT(DPRTYPE(i));
WHEN(1,2,3) /* Determine which internal segment */
DO; /* occurrence has changed */
J = DPRTYPE(i);
SELECT(DAXDPRCT);
WHEN(DLET) /* Processing a DELETE */
DO;
IMSAMOUNT(J) = 0; /* Zero appropriate amount */
IMSLIMIT(J) = 0; /* Zero appropriate limit */
END;
OTHERWISE /* Assume processing an INSERT or REPLACE */
DO;
IMSAMOUNT(J) = DPRAMOUNT(J); /* Copy internal amount */
IMSLIMIT(J) = DPLLIMIT(J); /* Copy internal limit */
END;
END; /* Select DAXDPRCT */
END; /* When DPRTYPE(i) = 1, 2 or 3 */
OTHERWISE /* Unexpected value in type column of credit tab */
DO;
CALL WRITE_ERROR_MESSAGE(EKYESE6E,RETURN_CODE_12);
GOTO FIN; /* Terminate */
END;
END; /* Select DPRTYPE */
END;
1 ELSE /" DAXSEG not = "I" (i.e. not an internal segment) */
/****************************
* Exit routine is called for DPROP-to-IMS mapping because *
* the target of the containing segment has changed. *
* We will just move information from the containing segment *
* in its DPROP format to segment in its IMS format. *
***************************************************************************/
DO;
/****************************
* Establish addressability to DPROP format segment data area *
***************************************************************************/
DPRC_SEG_POINTER = ADDR(DPRSEG_PARM_PTR);
IMSACNBR = DPRC_ACNBR;
IMSNAME = DPRC_NAME;
END;
END DPROP_TO_IMS;
1/****************************
* Main Routine *
***************************************************************************/
/****************************
* Establish addressability to DAX control block and IMS segment data *
***************************************************************************/
DAX_POINTER = ADDR(DAX_PARM_PTR);
IMSSSEG_POINTER = ADDR(IMSSSEG_PARM_PTR);
/****************************
* Set the "exit entered" and "exit in control" flags. *
***************************************************************************/
Figure 26 (Part 11 of 12). Third Sample Segment Exit Routine (PL/I)
DAXENTRD = X;
DAXINCTL = X;

/****************************************************************************
* Verify that the exit is called to format the expected IMS database and segment type *
****************************************************************************/
IF DAXDBNM ¬= DB123 THEN /* Unsupported DBD */
  DO;
    CALL WRITE_ERROR_MESSAGE(EKYESE2E,RETURN_CODE_16);
    GOTO FIN; /* Terminate */
  END;
ENDIF;

IF DAXSEGM ¬= ACCOUNT THEN /* Unsupported segname */
  DO;
    CALL WRITE_ERROR_MESSAGE(EKYESE2E,RETURN_CODE_16);
    GOTO FIN; /* Terminate */
  END;
ENDIF;

/****************************************************************************
* Process depending on call-function *
****************************************************************************/
SELECT (DAXCALL);

  WHEN (NO) CALL IMS_TO_DPROP; /* Normal call (IMS to DPROP) */
  WHEN (RV) CALL DPROP_TO_IMS; /* Reverse call (DPROP to IMS) */
  OTHERWISE /* Unsupported call function */
    DO;
      CALL WRITE_ERROR_MESSAGE(EKYESE1E,RETURN_CODE_16);
      GOTO FIN; /* Terminate */
    END;
END; /* Select DAXCALL */

FIN: /* End of processing */
END EKYESE2P;

Figure 26 (Part 12 of 12). Third Sample Segment Exit Routine (PL/I)
Chapter 3. Field Exit Routines

The RUP and HUP call the Segment and Field exit routines as part of DPROP's generalized mapping logic processing. These exit routines are optional and can be used to reformat or change data during propagation. The generalized mapping logic of the RUP or HUP can take care of most situations, but if your data is stored in an unusual way, or in some form that the RUP or HUP cannot handle, consider writing a Field exit routine to proceed.

A Field exit routine is generally used to convert an individual IMS data field between a user format that DPROP does not support, and a DPROP-supported format that you have defined in your PR. This is further referred to as:

- **User-to-DPROP mapping** when your exit routine is called to convert the field from its user format to the DPROP format. Calls to an exit routine for user-to-DPROP mapping are generated by the RUP as part of IMS-to-DB2 propagation.

- **DPROP-to-user mapping** when your exit routine is called to convert the field from its DPROP format to its user format. Calls to exit routine for DPROP-to-user mapping are generated by the HUP as part of DB2-to-IMS propagation.

Typical uses of Field exit routines include:

- Converting IMS fields that have special formats that DPROP does not directly support.

- Performing data conversions that are not supported by the DPROP data conversion routines.

- The following support for DATE and TIME formats in IMS fields:
  - Installation-specific (LOCAL) DATE and TIME formats
  - During RH-propagation, support of DATE and TIME formats other than those identified during DPROP installation

- Converting between some values in an IMS field and a DB2 NULL value.

- Cleaning up or reorganizing IMS data stored in an unusual way.

- If performing DB2-to-IMS propagation to convert the value of a numeric DB2 column into a packed or zoned IMS field, having a sign-code other than the "preferred" sign codes X’.C’ and X’.D’. (The period (.) represents the numeric digit that precedes the sign code.)

Field exit routines have many of the same characteristics as Segment exit routines.

Field exit routines used with TYPE=L or TYPE=F PRs are only called to perform HR-propagation and therefore support only user-to-DPROP mapping.

Field exit routines used with TYPE=E PRs support both user-to-DPROP mapping and DPROP-to-user mapping, even if the TYPE=E PR specifies MAPDIR=HR. This is because your Field exit routine can be called by the CCU and DLU. The conversion performed during DPROP-to-user mapping must be the reverse of the conversion performed during user-to-DPROP mapping.
1. During IMS-to-DB2 mapping, the RUP calls your Field exit routines immediately after your Segment exit routine, if you are using one. If you are not using a Segment exit routine, as soon as the RUP receives the changed data segment, it calls your Field exit routine.

Your Field exit routine must convert the field from its user format to the DPROP-supported format that you specified during the PR definition. The RUP calls a Field exit routine for each field that requires one according to your field definitions.

After calling your optional Segment exit routine and your optional Field exit routine, the RUP converts the field formats that you specified in your PR definition to the format of the DB2 columns.

2. During DB2-to-IMS mapping, the HUP converts the format of the DB2 columns into the field format that you specified in your PR definitions. Then the HUP calls your Field exit routine before your Segment exit routine, if you are using one.

Your Field exit routine must convert the field from the DPROP-supported format specified during PR definition to its user format. The HUP calls a Field exit routine for each field that, according to your PR definitions, can be processed by one.

Finally, after performing its own data conversion, calling your optional Field exit routines, and calling your optional Segment exit routines, the HUP updates the IMS database segment.

Like the Segment exit routines, your Field exit routines can be written in Assembler, or in COBOL, PL/I, or C. DPROP support for exit routines written in high-level languages requires LE/370 Version 1 Release 2.

For synchronous propagation, the RUP and HUP call your exits in both IMS batch and online dependent regions accessing DB2. For LOG-ASYNC propagation, the RUP calls your exit routines in an MVS batch environment. During user asynchronous propagation, depending on your implementation, the RUP calls your exit routines in IMS batch and dependent regions accessing DB2, or in a non-IMS DB2 TSO or CAF environment. DPROP also calls your exits during execution of the CCU and DLU.

DataRefresher calls Field exits User Data Type exits. If you are using DataRefresher to extract IMS data, your exit routines are called directly by DataRefresher during data extraction so that the mapping performed during extraction and data propagation are the same. DataRefresher also generates a definition call to your exit routine when you define the field on the CREATE DXTPSB statement.

---

**How To Write A Field Exit Routine**

This section describes some guidelines and requirements to follow when writing your Field exit routine. If your exit routine is used by DataRefresher during data extraction, it must follow these rules. See the appropriate DataRefresher or DXT documentation for information about DataRefresher requirements.

As with Segment exits, when the RUP or HUP calls your Field exit routine, the following four parameters are passed to your exit:
- An interface control block
- A user format buffer for the field in its user format
- A DPROP format buffer for the field in its DPROP format
- A 64-byte anchor area

If your exit routine is written in Assembler, register 1 contains the address of a list. This list is four fullwords long and contains the addresses of the parameters in the order listed above.

### Interface Control Block

Figure 28 on page 115 shows the structure of the interface control block, which is called EKYRCUDT and is passed to your Field exit routine. There is one interface control block per exit routine, lasting the duration of the exit in virtual storage. The following table lists:

- The fields most useful to your exit routine
- What the fields are used for
- Their displacement in the control block DSECT

<table>
<thead>
<tr>
<th>Field</th>
<th>Used For</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDTCALL</td>
<td>Call function, describing whether your exit routine is called either to perform user-to-DPROP mapping, DPROP-to-user mapping, or for a DataRefresher definition call.</td>
<td>X'20'</td>
</tr>
<tr>
<td>UDTPROGM</td>
<td>Name of the calling component</td>
<td>X'2C'</td>
</tr>
<tr>
<td>UDTSTYPE</td>
<td>User data type (data type of the field in its user format)</td>
<td>X'54'</td>
</tr>
<tr>
<td>UDTSBYTV</td>
<td>Number of bytes of field in its user format</td>
<td>X'58'</td>
</tr>
<tr>
<td>UDTSSCLV</td>
<td>Value of the scale of field in its user format</td>
<td>X'5E'</td>
</tr>
<tr>
<td>UDTTTYPE</td>
<td>DPROP data type (data type of the field in its DPROP format)</td>
<td>X'60'</td>
</tr>
<tr>
<td>UDTTBYTV</td>
<td>Number of bytes of field in its DPROP format</td>
<td>X'64'</td>
</tr>
<tr>
<td>UDTTSCLV</td>
<td>Value of the scale of field in its DPROP format</td>
<td>X'6A'</td>
</tr>
<tr>
<td>UDTXRETC</td>
<td>Return code that your exit routine provides</td>
<td>X'108'</td>
</tr>
<tr>
<td>UDTXMESG</td>
<td>exit routine message text</td>
<td>X'10C'</td>
</tr>
<tr>
<td>UTDSCRRT1</td>
<td>exit routine work space</td>
<td>X'6C'</td>
</tr>
<tr>
<td>UDTENTRD</td>
<td>Indicates the exit routine has been entered</td>
<td>X'104'</td>
</tr>
</tbody>
</table>
The interface control block has the same structure as the control block that DataRefresher passes to its User Data Type exits. A more complete description of these fields is included in the copy of the control block in Figure 28 on page 115.

When called for propagation, and for DataRefresher extraction, your exit routine must not change any fields in the control block except:

- UDTXRETC, UDTXMESSG, UDTSCRT1, UDTENTRD, UDTINCTL, and UDTNULLT
- UDTTBYTV (when performing user-to-DPROP mapping for a field having a VC or VG DPROP format)
- UDTSBYTV (when performing DPROP-to-user mapping)

Altering any of the other fields in the control block can cause unpredictable results.

When DataRefresher calls it for DEFINITION calls, your exit routine needs to update additional fields. Refer to the appropriate DataRefresher or DXT documentation for more details.

### User Format Buffer

The user format buffer contains the field in its User Format:

- When performing user-to-DPROP mapping, DataRefresher or DPROP provides the field to your field exit routine in this buffer. The field is still in its IMS format or in the format your Segment exit routine returns.

  Your field exit routine must not modify this buffer when called to perform user-to-DPROP mapping.

- When performing DPROP-to-user mapping, your field exit routine must provide the field to DPROP in this buffer. The field must be provided in its user format. This is the IMS format or the format your Segment exit routine expects.

  Do not place a field in the user format buffer that is longer than the fixed or maximum field length specified in UDTSBYTV. This causes storage overlays and unpredictable results.

### DPROP Format Buffer

The DPROP format buffer contains the field in the DPROP-supported format that you identified during your PR definition:

- When performing user-to-DPROP mapping, your exit routine must convert the field to the format you have defined in your PR, and place the converted field in the DPROP format buffer before returning to the RUP. The RUP reads the field from this buffer and then continues its normal processing as if the converted field were the original.
Do not place a field in the DPROP format buffer that is longer than the fixed or maximum field length specified in UDTTBYTV. This causes storage overlays and unpredictable results.

If the data type of the converted field is VC or VG, then the DPROP format buffer begins with the first data byte of the field (not with the field length). When converting to a VC or VG format, the actual field length (expressed in number of bytes) must be set by your exit routine in the UDTTBYTV field of the interface control block.

- When performing DB2-to-IMS mapping, the HUP provides the field for your Field exit routine in this buffer. The provided field is in the DPROP-supported format that you specified during PR definition. DATE and TIME fields are provided by the HUP in ISO format.

If the DPROP-supported data type is VC or VG, then the DPROP format buffer begins with the first data byte of the field (not with the field length). The actual field length (expressed in number of bytes) is provided by DPROP to your exit routine in the UDTTBYTV field of the interface control block.

Your field exit routine must not modify this buffer when being called to perform DPROP-to-user mapping.

64-Byte Anchor Area

The RUP or HUP gives you 64 bytes as a general storage area. Each exit routine has its own unique anchor area. You can use it for whatever you want. Initially, the area is set to all binary zeros, and is never changed again by DPROP (or DataRefresher if you are using it).

The anchor area exists in virtual storage, and remains yours for the duration of the exit. For IMS batch and BMP regions, the anchor area lasts for the duration of the application program. For MPP regions, the anchor area lasts for the duration of the IMS Program Controller Subtask. This can span multiple MPP executions. For CCU execution, the anchor area lasts for the duration of the job step. For asynchronous propagation, the anchor area lasts for the duration of the MVS task being used by the receiver program to call the RUP.

Interface Control Block DSECT

You can generate the following DSECT in your assembler exit routine by coding the EKYRCUDT macro statement. For high-level language exit routines, you can include or copy one of the following members to map the Field exit routine interface control block:

- **EKYRCUDC**: Exit routines written in COBOL
- **EKYRCUDP**: Exit routines written in PL/I
- **EKYRCUDK**: Exit routines written in C

The interface control block is shown in Figure 28 on page 115 followed by detailed descriptions of its fields.
Figure 28 (Part 1 of 4). Interface Control Block for a Field Exit Routine
Figure 28 (Part 2 of 4). Interface Control Block for a Field Exit Routine
131++  
132+UDTEXIT DS CL8 NAME OF THE USER EXIT  
133++  
134+UDTPCBLS DS AL4 ***DXT ONLY*** ADDRESS LIST OF ALL PCB  
135++ ADDRESSES IF DL/I ENVIRONMENT  
136++  
137+UDTDPRP1 DS CL24 ADDITIONAL WORK SPACE  
138++--------------------------------------------------------------------  
139++ THIS SECTION CONTAINS DATA PERTINENT TO FIELD IN ITS USER FORMAT  
140++--------------------------------------------------------------------  
141+UDTSTYPE DS CL2 USER DATA TYPE  
142+UDTSBYTI DS CL1 ***DXT ONLY*** LENGTH INDICATOR FOR USER FORMAT.  
143++ NOT USED BY DPROP. USED BY DXT.  
144++ 'N' - INDICATES LENGTH OF USER  
145++ 'V' - INDICATES LENGTH OF USER  
146++ FORMAT RESIDES WITH THE  
147++ DEFINITION.  
148++ FORMAT Varies, and MUST  
149++ BE RETURNED AT  
150++ "DEFINITION" TIME.  
151+ DS CL1 RESERVED FOR DXT USE  
152+UDTSBYTV DS H LENGTH OF FIELD IN USER FORMAT  
153++  
154+UDTSSCLI DS CL1 ***DXT ONLY*** SCALE INDICATOR FOR USER FORMAT  
155++ NOT USED BY DPROP. USED BY DXT.  
156++ 'N' - INDICATES VALUE OF SCALE  
157++ 'V' - INDICATES VALUE OF SCALE  
158++ OF USER FORMAT  
159++ RESIDES WITH THE  
160++ DEFINITION.  
161++ 'V' - INDICATES VALUE OF SCALE  
162++ OF USER FORMAT  
163++ Varies, and MUST  
164++ BE RETURNED AT  
165++ "DEFINITION" TIME.  
166+UDTSSCLV DS H VALUE OF SCALE IN USER FORMAT  
167++--------------------------------------------------------------------  
168++ THIS SECTION CONTAINS DATA PERTINENT TO FIELD IN ITS DPROP FORMAT  
169++--------------------------------------------------------------------  
170+UDTTTYPE DS CL2 DATA TYPE OF DPROP FORMAT  
171+UDTTBYTI DS CL1 ***DXT ONLY*** LENGTH INDICATOR FOR DPROP FORMAT  
172++ NOT USED BY DPROP. USED BY DXT:  
173++ 'N' - INDICATES LENGTH OF DPROP  
174++ 'V' - INDICATES LENGTH OF DPROP  
175++ FORMAT RESIDES WITH THE  
176++ DEFINITION.  
177++ 'V' - INDICATES LENGTH OF DPROP  
178++ FORMAT Varies, and MUST  
179++ BE RETURNED AT  
180++ "DEFINITION" TIME.  
181+UDTTBYTV DS H LENGTH OF FIELD IN DPROP FORMAT  
182+UDTSSCLI DS CL1 ***DXT ONLY*** SCALE INDICATOR FOR DPROP FORMAT  
183++ NOT USED BY DPROP. USED BY DXT.  
184++ 'N' - INDICATES VALUE OF SCALE  
185++ 'V' - INDICATES VALUE OF SCALE  
186++ RESIDES WITH THE  
187++ DEFINITION.  
188++ 'V' - INDICATES VALUE OF SCALE  
189++ OF DPROP FORMAT  
190++ Varies, and MUST  
191++ BE RETURNED AT  
192++ "DEFINITION" TIME.  
193+ DS CL3 RESERVED FOR DXT USE  
194+UDTTSSCLV DS H VALUE OF SCALE IN DPROP FORMAT

*Figure 28 (Part 3 of 4). Interface Control Block for a Field Exit Routine*
Figure 28 (Part 4 of 4). Interface Control Block for a Field Exit Routine

Interface Control Block Field Descriptions

The following list includes detailed descriptions of the fields in the interface control block. DPROP and DataRefresher descriptions are included. Some of the fields are not useful to your exit routine when DPROP calls it. These fields are described for DataRefresher only.

UDTTNAME  Contains the constant DVRXCUDT, used to identify the control block in a storage dump.

UDTXADDR  The virtual storage entry point address of the exit routine.

UDTCALL  The call function that describes what action your exit routine must perform. This field can have the following values:

ST  Source to target conversion (user-to-DPROP mapping).

The exit routine is called to convert the field from its user format to its DPROP format. The user format is the format in the IMS segment, or the format that your Segment exit routine provides, if one was called. The DPROP (or DataRefresher) format is defined either on the TRGTYPE= keyword of the DataRefresher CREATE DATATYPE statement, or in the FLDETYPE column of the DPRIFLD MVG input table.
Target to source conversion (DPROP-to-user mapping).

The exit routine is called to convert the field from its DPROP format to its user format. The user format is the format in the IMS segment, or the format your Segment exit routine expects, if one is called. The DPROP (or DataRefresher) format is defined either on the TRGTYPE= keyword of the DataRefresher CREATE DATATYPE statement, or in the FLDETYPE column of the DPRIFLD MVG input table.

Definition call (DataRefresher only).

DataRefresher calls the exit routine to complete or validate a field definition. If you provide all your mapping definitions through the MVG input tables, then your Field exit routine is never called with the DF call function.

The remaining descriptions are only for ST and TS calls. For more information on calls that only DataRefresher uses, refer to the appropriate DataRefresher or DXT documentation.

Contains the constant ESA, indicating that the program is running in an MVS environment.

Contains a label describing the environment in which the exit routine is called. This field can have the following values:

- **BAT**: IMS batch or BMP environment
- **MPP**: IMS MPP environment
- **IFP**: IMS Fast Path environment
- **CICS**: CICS environment

If the exit is called in an environment other than those listed above, the value consists of blanks.

Contains information about the calling program, either DPROP or DataRefresher. This field can have the following values:

- **DPRS**: Called by DPROP during synchronous propagation
- **DPRA**: Called by DPROP during LOG-ASYNC propagation or user asynchronous propagation
- **DPRC**: Called by DPROP during CCU execution
- **DPRL**: Called by DPROP during DLU execution
- **DataRefresher**: Called by DataRefresher

The load module name of the Field exit routine.

The next five fields describe the user format of the propagated field. The user format is the IMS DB format if you do not use Segment exit routines. If you do use a Segment exit routine, it is the format your Segment exit routine creates (HR-propagation) or expects (RH-propagation).

The user data type that was specified either on the SCRTYPE keyword of the DataRefresher CREATE DATATYPE statement, or in the DATATYPE column of the DPRIFLD MVG input table.

(DataRefresher only) Used for DF calls. Refer to the appropriate DataRefresher or DXT documentation for a complete description.
UDTSSCLI \textit{(DataRefresher only)} Used for DF calls. Refer to the appropriate DataRefresher or DXT documentation for a complete description.

UDTSSCLV The scale of the field in its user format.

The next five fields describe the field in its DPROP or DataRefresher format.

UDTTTYPE The DPROP or DataRefresher data type that is specified either on the TRGTYPE keyword of the DataRefresher CREATE DATATYPE statement, or in the FLDETYPE column of the DPRIFLD MVG input table.

UDTTBYTI \textit{(DataRefresher only)} used for DF calls. Refer to the appropriate DataRefresher or DXT documentation for a complete description.

UDTTBYTV The length (in bytes) of the field in its DPROP or DataRefresher format.

For graphic fields, the number of bytes is twice the number of DBCS characters. This is different from the usual DB2 convention that expresses the length of G and VG columns as the number of DBCS characters.

Observe the following rules about UDTTBYTV for user-to-DPROP mapping:

- On entry to your exit routine, UDTTBYTV was initialized by DPROP and DataRefresher to the maximum field length (for fields having a VC or VG data type in their DPROP format) or to the fixed length of the field (for fields having data types other than VC and VG in their DPROP format).

- When processing a field having in its DPROP format a VC or VG data type, your exit routine must return the actual length of the field in UDTTBYTV.

During DPROP-to-user mapping, UDTTBYTV is initialized on entry to your exit routine to the actual length of the field.

UDTTSCLI \textit{(DataRefresher only)} Used for DF calls. Refer to the appropriate DataRefresher or DXT documentation for a complete description.

UDTTSCLV The scale of the field in its DPROP or DataRefresher format.
The remaining fields can be changed in your Field exit routine.

**UDTSCRT1**  
An exit routine work space for your own use. Before the first call to your exit routine, DPROP initializes this space to binary zeros, and does not modify it again.

The next two fields are switches that can be useful for problem determination. DPROP and DataRefresher do not require your exit routine to set these fields. However, they can help you determine where a problem occurred if you have an ABEND. DPROP and DataRefresher set these fields to blanks before the first time your exit routine is called.

**UDTENTRD**  
Exit-entered flag. As you enter your exit routine, set this field to X. DPROP does not change this field again, so if a problem occurs, you can determine if your exit has been entered.

**UDTINCTL**  
Exit-in-control flag. You can also set this field to X, indicating that your exit routine has control. When DPROP regains control, it resets this field to blank, so you can determine if your exit routine has control when an ABEND occurs.

The next field supports conversion to or from a DB2 null value.

**UDTNULLT**  
Null value indicator. This field allows you to map the contents of the IMS field to a DB2 NULL value (user-to-DPROP mapping) or from a DB2 NULL value (DPROP-to-user mapping).

- When your exit routine is called with an ST Call Function (user-to-DPROP mapping), DPROP initializes this field to blanks. If your exit routine sets UDTNULLT to Y during an ST call, DPROP or DataRefresher assigns a NULL value to the target DB2 column. You must define the target column as containing a NULL value.
- When your exit routine is called with a TS Call Function (DPROP-to-user mapping), this field indicates whether or not the DB2 column contains a NULL value.

| Y | The DB2 column contains a NULL value. |
| N | The DB2 column does not contain a NULL value. |

The next two fields can be used along with the RUP's and HUP's error handling logic. For more information on return codes and error handling techniques, see “Return Codes and Error Handling Techniques” on page 123.

**UDTXRETC**  
The return code the exit routine provides when returning to its caller. This field is set to zero when the exit routine is called.

**UDTXMESG**  
User-provided error message. It is set to blanks when the exit routine is called. When the exit routine returns, if this field is not blank, DPROP or DataRefresher writes the contents of the field.

DPROP prefaces the message with the number EKYR970I or EKYR971E, and writes the message according to its usual error handling logic. DataRefresher prefaces the message and writes the message to the //SYSPRINT data set.

There is one exception to the above. If the exit routine is called during processing of the optional WHERE clause of the PR,
DPROP does not write error messages if the exit returns with a return code 0 or 4 in UDTXRETC.

Performing Data Conversions

The appropriate Administrators Guide for your propagation mode lists all the data conversions that are supported directly through the DPROP data conversion routines. If the IMS data field you want to propagate is in a format DPROP does not support, or if you want to perform a conversion that the DPROP data conversion routines do not support, your Field exit routine can perform the conversion. Examples of conversions that are not supported include binary integers to floating-point numbers, and time stamp to date or time formats.

Your exit routine does not need to convert a field directly into or from the format of the DB2 column. The DPROP data conversion routines are still called (if necessary) to complement the conversion done by your field exit routine. The RUP and HUP call the DPROP data conversion routines automatically if the DPROP format of the data field is different from the DB2 column format. Therefore, it is sufficient if your exit routine converts the data field between its user format and a DPROP-supported format.

Exit Routine Processing

When called for user-to-DPROP mapping, your Field exit routine can read the IMS field from the user format buffer and, using the information found in the interface control block, convert the field into the DPROP format you have defined in your PR. For more details on defining user and DPROP formats, see “Telling DPROP About Your Field Exit Routine” on page 125.

When called for DPROP-to-user mapping, your Field exit routine can read the field from the DPROP format buffer and, using the information found in the interface control block, convert the field into the user format.

Meanwhile, there are some restrictions and guidelines to follow when developing your exit routine.

- When DPROP calls it, your exit routine always gets control in AMODE 31, and must return control in AMODE 31. Parameters DPROP passes to your exit are usually located above the 16MB line. The exit routine is loaded above or below the 16MB line depending on the RMODE attribute of the exit load module.

  It is recommended that you code and link-edit your exit routine as reentrant.

  To simplify programming, DPROP provides work space for your exit routines, both in the interface control block and the 64-byte anchor area.

- If your exit routine is written in Assembler language, DPROP uses standard OS/VS conventions when calling your exit routine.
  - Register 1 points to the parameter list described above.
  - Register 13 contains the address of a register save area.
  - Register 14 contains the return address.
  - Register 15 contains the entry point address of the exit routine

  Upon entry, the exit routine must save the register contents into the save area that the caller provides. If your exit routine calls other routines that use standard MVS linkage conventions, it must also provide a save area of its own. The exit routine must return to its caller using normal OS/VS conventions after
restoring the registers. A return code must be provided in the interface control block, not in register 15.

- Your Field exit routine must check that the data returned to DPROP is valid. For example, it must make sure that a packed field contains a number in packed format. Conversions producing invalid formats can cause propagation or application failures. For example, during HR-propagation, SQL statements that the SQL update module generates, or conversions by DPROP data conversion routines, can be rejected. With RH-propagation, invalid conversion can result in application failures, when your IMS applications access the segments with the invalid data.

- When converting a field that is part of the IMS key or mapped to a primary DB2 key, DPROP cannot verify that the key is still unique after it is converted; you must check it.

- Because the exit routine for synchronous propagation runs in the same environment as the propagating application program, it can, if necessary, generate the same type of IMS calls and SQL statements as the application program. For LOG-ASYNC and user asynchronous propagation using either TSO-Attach or CAF-Attach, create only SQL statements, as the exit routines do not execute in an IMS environment, and cannot generate IMS calls.

If your exit generates IMS calls, use the AIB interface described in IMS/ESA Application Programming: DLI Calls, which allows your exit routine to generate calls without the address of the IMS PCBs.

During synchronous propagation, any changes you make to propagated data from within your exit routine are not propagated synchronously. However, the date can be propagated asynchronously, if you implement asynchronous propagation.

Exclude the PCBs your exit routine uses from the list passed to the application program upon entry. You can avoid changing the application program if you need to add PCBs for exclusive use by your exit routine. Refer to IMS/ESA Utilities Reference: System for more details.

- A Field exit routine must not perform functions that are not supported by the environment in which it is running. For example, an exit routine running in an MPP region must not write to OS files, and the exit routine must not generate STIMER macros in an IMS environment.

For performance reasons, your exit routine must generate static rather than dynamic SQL statements. Avoid using functions that have a detrimental impact on the performance of the propagating program, such as performing an OPEN and CLOSE on an OS/VS file each time the exit is called.

**Return Codes and Error Handling Techniques**

This section discusses how to return from your exit routine to the RUP and HUP, including return codes and error handling techniques.

**Return Codes**

You set the return code by placing it in the UDTXRETC field of the interface control block. The RUP and HUP read this field when they regain control. The valid return codes are 0 and 4. Returning any other code is an error and DPROP abends.

0 Normal return. DPROP or DataRefresher continues normal processing using the converted field value.
A failure occurred. DPROP uses its usual error handling logic. There is one exception to this: if the exit routine is called during processing of the optional WHERE clause of the PR, DPROP does not perform its error logic. The currently processed condition of the WHERE clause is considered to be false (or true if the operand of the condition is ¬=).

For synchronous propagation,

if ERROPT=BACKOUT is in effect, DPROP backs out the propagating application. For LOG-ASYNC propagation, if ERROPT=BACKOUT is in effect, the Receiver terminates with an error message. For user asynchronous propagation, CCU or DLU execution, the RUP and HUP return to their caller with an error. The RUP and HUP use their error reporting logic to write diagnosis information.

If ERROPT=IGNORE is in effect, the RUP and HUP do not perform propagation, and return to the caller without performing a backout and without providing any error indication to the caller. However, if this occurs during CCU or DLU execution, the RUP and HUP return to the CCU or DLU with an error. The RUP and HUP use their error reporting logic to write diagnosis information.

For DataRefresher, further processing is based on the FLDERR keyword of the DataRefresher SUBMIT control statement.

Error Handling Techniques
When you find an error in your exit routine, it is strongly recommended that your exit routine take advantage of the standard error handling logic of the RUP and HUP. In the interface control block, you can supply a return code in UDTXRETC, and an error message in UDTXMESG. You must not return an error message in UDTXMESG without providing an error return code, because this generates excessive console messages.

By supplying DPROP with an error return code and message, you gain many advantages. When an exit returns with an error return code, DPROP traces or snaps both the control blocks involved in the interface, and the data. The exits are included in the standardized error handling scheme of the RUP and HUP, which distinguishes between ERROPT=BACKOUT and ERROPT=IGNORE; this is different for propagation and CCU executions. It protects against excessive console messages. DPROP writes your error message using its standard message writing logic: WTO, trace data set (the IMS log, the //EKYTRACE data set, or the //EKYLOG data set), and Audit trail.

If the exit routine generates its own messages or ABENDs, the RUP and HUP cannot include the exit routine in their standardized error handling, and cannot guard against excessive messages on the MVS consoles. Therefore, it is recommended that your exit routine does not generate its own messages or ABENDs when an error occurs.

Saving Information Across Calls
You can save information across calls to the exit routine. You can save the information either in the 64-byte anchor area, or in the field UDTSCRT1 in the interface control block. If these areas are not large enough, generate a GETMAIN and save the address of the storage in either of these areas.
Updating Your Field Exit Routine

DPROP does not provide any online change logic to replace an existing load module copy of your Field Exit routine with a new version of the load module. If you need to change your exit routine, stop the affected IMS regions, DPROP asynchronous Receiver or user asynchronous receiver programs before performing the change. A change of the exit routine without stopping the IMS regions or receiver programs causes unpredictable results. For example, some MPP regions can use the new version of the exit routine, while other regions use the old version. After the change, you can restart the IMS regions.

Tracing Your Exit Routine

As described in “Tracing Your Exit Routine” on page 41, DPROP provides a trace facility to help you debug your exit routine. For a Field exit routine, DPROP includes in the trace output the user format buffer and DPROP format buffer, rather than the segment buffers.

Telling DPROP About Your Field Exit Routine

This section discusses how to inform DPROP that you want to use a Field exit routine during data propagation. To do this, you must provide DPROP with the name of your exit routine, the two-byte user data type, and the description of the DPROP-supported field format. How you proceed depends on how you enter your PR.

PRs Entered Through DataRefresher UIM

Defining the User Data Type

If you are using both DataRefresher and a Field exit routine, define a user data type before calling the exit routine.

Use the DataRefresher CREATE DATATYPE control statement to define the user data type and associate it with a Field exit routine. You define a user data type by assigning it a unique two-byte name using CREATE DATATYPE. You also specify the name of the exit routine on the control statement.

Usually, one CREATE DATATYPE control statement is used for each Field exit routine. But you can use multiple CREATE DATATYPE statements specifying multiple definitions for one exit routine. In this case, your exit routine is responsible for converting the multiple user data types.

You must provide the following keywords on the CREATE DATATYPE statement when calling DataRefresher:

EXIT=exitname
  The load module name of the Field exit routine.

SRCTYPE=xx
  A two-byte character value used to uniquely identify the user data type and associate it with the exit routine.

  The second character of the two-byte value cannot be blank, and the value cannot be VC or VG.
SRCBYTES=nn
The length in bytes of the field in its user format.

SRCScale=mm
Optional; the scale of the field in its user format.

TRGTYPE=yy
The data type of the DataRefresher or DPROP format.
It must be a data type that DataRefresher and DPROP support.

TRGBYTES=bb
The length in bytes of the field in its DataRefresher or DPROP format.
For variable-length character and graphic fields, specify the maximum length.

TRGSCALE=ss
The scale of the field in its DataRefresher or DPROP format.
This keyword is used only for packed decimal and zoned decimal data types.

**Requesting Exit Routine Use**
After defining a user data type, you can request the use of the associated Field exit routine. Specify if the exit routine must process a field by using the FIELD statement of the CREATE DXTPSB control statement.

Identify the user data type on the TYPE= keyword of the FIELD statement. This is the same data type specified earlier in the SRCTYPE= keyword of the CREATE DATATYPE control statement. Each FIELD statement identifying a user data type with the TYPE= keyword is processed by the exit routine.

DataRefresher calls the Field exit routine with a definition call each time it processes a FIELD statement identifying a user data type. During the definition call, the exit routine can validate or change the field definitions provided by the CREATE DATATYPE statement and the FIELD statement. Definition calls are generated when DataRefresher processes your CREATE DXTPSB control statement, not during the extract.

DataRefresher calls your Field exit routine during the extract; DPROP calls it during propagation. During these calls, your exit routine must convert the fields between the user data type and the data type supported by DataRefresher and DPROP.

For more information on DataRefresher calls, see the appropriate DataRefresher or DXT documentation.

**PRs Entered into the MVG Input Tables**
If you are entering your PR information directly into the MVG input tables, without using DataRefresher, you can use the DPRIFLD (or FLD) table to inform DPROP if your Field exit routine must process a particular field. The FLD table is one of the MVG input tables. Provide the information in the following columns:

FLDEXIT=exitname
The load module name of the Field exit routine.

The name must begin with an alphabetic character. If you insert blanks into the FLDEXIT column or leave the column blank, then the field described in the DPRIFLD row is not processed by your exit routine.
DATATYPE=xx
A two-byte character value used to uniquely identify the user data type, and to
associate the exit routine with this data type.

The second character of the two-byte value cannot be blank, and cannot be VC
or VG.

BYTES=nn
The length in bytes of the field in its user format.

SCALE=mm
Optional; the scale of the field in its user format.

FLDETYPE=yy
The data type of the DPROP format.

This value must be a data type that DPROP supports.

FLDEBYTE=bb
The length in bytes of the field in its DPROP format.

For variable-length character and graphic fields, specify the maximum length.

FLDESCAL=ss
The scale of the field in its DPROP format.

This keyword is used only for packed decimal and zoned decimal data types.

The MVGU validates all of these columns for a general mapping case. For a PR
entered into the MVG input tables, the Field exit routine is not called for a definition
call.

First Sample Field Exit Routine

The sample Field exit routine in Figure 29 on page 128 is an example of how to
convert the data type of an individual field. In this case, the exit routine converts a
bit string field into a character field (during user-to-DPROP mapping) and a
character field into a bit string (during DPROP-to-user mapping).

Specifically, during user-to-DPROP mapping, each bit is converted into a character
represented by 0 or 1 based on the value of the related bit. This can be useful to
convert bit control fields into individual flag character fields.

The source code in Figure 29 on page 128 is provided in the DPROP Sample
Source Library (EKYSAMP) under the member name EKYEFL1A. Following the
source code are definitions related to the sample Field exit routine.
**MODULE NAME: EKYEFL1A**

**DESCRIPTIVE NAME:** SAMPLE DPROP 'FIELD USER EXIT ROUTINE'

**STATUS:** V1 R2 M

**FUNCTION:**

The purpose of this program is to provide a sample structure for a 'FIELD USER EXIT ROUTINE'.

This example converts a bit string into a character string (or vice versa) with each bit represented by a character, to be set as '0' or '1' based on the value of the related bit (alternate representation might be 'T' for true and 'F' for false). This function could be useful for converting bit control fields to individual flag bytes.

In installations which combine usage of:

- **DXT (FOR THE ORIGINAL EXTRACT OF THE DL/I DATA).**
- **DPROP (FOR THE PROPAGATION OF THE DL/I DATA)***

The exit will be called both by DXT and DPROP.

DXT calls the exit:

- during DXT-UIM processing, with a 'DEFINITION CALL' in order to validate field definitions.
- during DXT-DIM processing, in order to map the DL/I data extract bit-strings into character-string.

DPROP calls the exit:

- for 'SOURCE-TO-TARGET' (ST) conversion, to map the bit-strings into character strings:
  - during DPROP CCU (CONSISTENCY CHECK UTILITY) conversion, to
  - during DPROP DLU (DL/I LOAD UTILITIES)

processing: - for 'SOURCE-TO-TARGET' calls, the source field is converted a bit a time into '0' or '1' characters in the target field. For example, the 2 byte character string 'AI' is hex 'C1F1' or '1100000111110001' in binary. It would be converted into the 16 byte character string '1100000111110001'. The length of the target field terminates processing. If the target length is more than source length times 8 the remaining right hand bytes are set to the character '0'.

---

*Figure 29 (Part 1 of 10). Sample Field Exit Routine (Assembler)*
- FOR 'TARGET-TO-SOURCE' CALLS, THE TARGET FIELD (IN THIS CASE INPUT TO THIS ROUTINE) WHICH MUST BE ALL CHARACTERS '0' OR '1', IS CONVERTED TO A BIT STRING WITH THOSE BITS ON, WHICH ARE '1' IN THE TARGET FIELD, WHEN USING THE ABOVE EXAMPLE, THE 16 BYTE CHARACTER FIELD WITH THE VALUE '1100000111110001' WOULD BE CONVERTED TO THE 2 BYTE BIT-STRING '1110000111110001' WHICH IS HEX 'C1F1' OR CHAR 'A1'. NOTE THAT VALUES OTHER THAN '0' AND '1' IN THE CHARACTER TARGET FIELD LEADS TO CONVERSION ERRORS DETECTED BY THIS FIELD EXIT ROUTINE.

PROCESSING DURING 'DEFINITION' CALLS ISSUED BY DXT-UIM:
- THE SOURCE LENGTH IS CHECKED AGAINST THE MAXIMUM SOURCE LENGTH(16).
- IF THE TARGET LENGTH HAS BEEN DEFINED ON THE DXT UIM 'CREATE DATATYPE' STATEMENT AS 'VARIES', THE EXIT SETS ITS VALUE TO 8 TIMES THE SOURCE LENGTH.
- IF THE TARGET LENGTH HAS BEEN SPECIFIED ON THE DXT UIM 'CREATE DATATYPE' STATEMENT: IT IS CHECKED AGAINST 8 TIMES THE MAXIMUM SOURCE LENGTH.
- TARGET DATA TYPE IS ENSURED TO BE 'C' AND TARGET SCALE ENSURED TO BE 'N'.

NOTE FOR INSTALLATIONS WHICH USE DPROP WITHOUT DXT: IF DPROP IS USED WITHOUT DXT, THE EXIT WILL NEVER BE INVOKED FOR A DEFINITION CALL (DEFINITION CALLS ARE NOT NECESSARY, SINCE THE USER PROVIDES ALL DEFINITIONS (I.E SOURCE LENGTH, TARGET LENGTH) IN THE DPROP 'MVG INPUT TABLES'.

PROCESSING DURING 'TARGET-TO-SOURCE' CALLS ISSUED BY DXT-DEM AND DPROP:
- THE DATA IN THE SOURCE BUFFER IS CONVERTED A BIT AT A TIME INTO A '1' OR '0' IN THE TARGET BUFFER. PROCESSING STOPS WHEN THE NUMBER OF BITS PROCESSED EQUALS THE VALUE PASSED AS THE TARGET LENGTH. IF THE TARGET LENGTH IS EXHAUSTED BEFORE THE TARGET, THE REMAINING RIGHT HAND TARGET BYTES ARE SET TO CHARACTER '0'.

INSTALLATIONS USING BOTH DXT AND DPROP WILL NOTICE THAT THE LOGIC OF EKYEFL1A IS THE SAME AS THE LOGIC OF THE DXT-PROVIDED SAMPLE EXIT ROUTINE DVRXAXUT, BUT IT IS ENHANCED BY THE 'TARGET-TO-SOURCE' CALL TYPE WHICH IS ISSUED ONLY BY DPROP.

SPECIFIC EXIT FUNCTIONS DEMONSTRATED BY THIS MODULE.
1. PROCESSING THE INVOCATION PARM LIST.
2. USING THE USER ANCHOR AREA.
3. IDENTIFYING THE REQUESTED FUNCTION.
4. UIM VALIDATION OF 'V' TYPE LENGTH FIELDS.
5. THE USE OF THE MESSAGE AREA.

Figure 29 (Part 2 of 10). Sample Field Exit Routine (Assembler)
123 /*
124 * INPUT: (PASSED AS PARAMETERS).
125 1. UDT - USER DATA TYPE INTERFACE CONTROL BLOCK.
126 2. SOURCE BUFFER - THE SOURCE USER DATA (N/A FOR DEFINE CALL).
127 3. TARGET BUFFER - TARGET AFTER CONVERSION (N/A FOR DEFINE).
128 4. USER ANCHOR AREA - A 64 BYTE AREA FOR USE BY THE EXIT.
129 */
130 /*
131 */
132 /**************************************************************************/
133 /*
134 */
135 /*
136 */
137 /*
138 */
139 /*
140 */
141 /*
142 */
143 /*
144 */
145 /*
146 */
147 /*
148 */
149 /*
150 */
151 /**************************************************************************/
152 /*
153 */
154 /*
155 */
156 /*
157 */
158 /*
159 */
160 /*
161 */
162 /**************************************************************************/
163 /*
164 */
165 /*
166 */
167 /*
168 */
169 /*
170 */
171 /*
172 */
173 /*
174 */
175 /*
176 */
177 /*
178 */
179 /*
180 */
181 /*
182 */
183 /*
184 */
185 /*
186 */
187 /*
188 */
189 /*
190 */

Figure 29 (Part 3 of 10). Sample Field Exit Routine (Assembler)
THE FIELD STATEMENT IN CREATE DXTPSB:

- TYPE = XX - RELATES THIS FIELD TO A DXT DATATYPE.
- BYTES = NN - THE SOURCE FIELD LENGTH.
- IF DEFINED AS 'VARIES' IN THE
  DATATYPE STATEMENT, IT MUST NOT
  EXCEED THE MAXIMUM FIELD LENGTH
  ALLOWED BY THE EXIT.
- IF NOT DEFINED AS 'VARIES',
  IT MUST EQUAL THE 'SRCBYTES' OPERAND IN THE DATATYPE STATEMENT.
- SCALE = MUST NOT BE SPECIFIED.

INFORMATION FOR INSTALLATIONS WHICH USE DPROP WITHOUT DXT.

THOSE INSTALLATIONS DEFINE THE DL/I-TO-DB2 MAPPING BY
PROVIDING MAPPING DEFINITIONS IN THE
DPROP 'MVG INPUT TABLES'.

USAGE OF THIS SAMPLE FIELD EXIT ROUTINE REQUIRES
FOLLOWING DEFINITIONS IN THE DPRIFLD TABLE:

INVOCATION OF A FIELD USER EXIT ROUTINE IS DEFINED BY PROVIDING SPECIFICATIONS IN
THAT ROW OF THE 'DPRIFLD' TABLE WHICH DESCRIBES
THE FIELD TO BE MAPPED.

COLUMNS OF THE DPRIFLD ROW SHOULD PROVIDE
FOLLOWING DEFINITIONS:
COLUMN OF COLUMN
DPRIFLD VALUE EXPLANATIONS

FLDEXIT = EKYEFL1A: THE EXIT LOAD MODULE NAME
DATATYPE = XX: A TWO CHARACTER DATA-TYPE ID.
BYTES = NNNN: THE SOURCE FIELD LENGTH
FLDETYPE = C: THE TARGET DATA-TYPE. MUST BE 'C'.
FLDEBYTE = MMMMM: THE TARGET FIELD LENGTH.
(MUST BE 8 TIMES THE SOURCE FIELD LENGTH).
SCALE =: SHOULD EITHER NOT BE PROVIDED OR
SHOULD BE SPECIFIED AS ZERO.
FLDESCAL =: SHOULD EITHER NOT BE PROVIDED OR
SHOULD BE SPECIFIED AS ZERO.

ENTRY REGISTERS
1 - A(PARAMETER LIST)
PARAMETER LIST = A(UDT)
A(SOURCE BUFFER)
A(TARGET BUFFER)
A(EXIT ANCHOR AREA)
13 - CALLER'S SAVE AREA
14 - CALLER'S RETURN ADDRESS
15 - ENTRY FOR THIS EXIT ROUTINE
Figure 29 (Part 5 of 10). Sample Field Exit Routine (Assembler)
Figure 29 (Part 6 of 10). Sample Field Exit Routine (Assembler)
Figure 29 (Part 7 of 10). Sample Field Exit Routine (Assembler)
Figure 29 (Part 8 of 10). Sample Field Exit Routine (Assembler)

```
5mzqr!z!t6 mkrtsl mkrtsl
5mzqr!z!t4 ... slmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtslmkrtsl
mzqr!z!tmzqr!z!tmzqr!z!t172 47Fmzqr!z!t Cmzqr!z!tEA mzqr!z!tmzqr!z!tmzqr!z!tEA 5mzqr!z!t2 B EXITRETN GO TO COMMON RETURN POINT
mzqr!z!tmzqr!z!tmzqr!z!t1BFF 5mzqr!z!t1 SR R15,R15 SET SUCCESSFUL RETURN CODE
mzqr!z!tmzqr!z!tmzqr!z!t16C 47Dmzqr!z!t C138 mzqr!z!tmzqr!z!t138 499 BNH TARGNEXT YES-BRANCH TO PROCESS NEXT BYTE
mzqr!z!tmzqr!z!tmzqr!z!t16A 1978 498 CR R7,R8 MORE SOURCE BYTES TO PROCESS?
```

```ml
463 *
464 + RECHECK SOURCE LENGTH - SHOULD ALWAYS BE GOOD, BUT.....
00011C 4880 B058 00058B 465 LH R8,UDTSBYTV GET NUMBER OF SOURCE BYTES
000120 4980 C248 00248 466 CH R8,ZEROLGT IS SOURCE LENGTH SPECIFIED
000124 4780 C1C8 001C8 467 BE ERRSLGT1 NO-PERFORM ERROR FUNCTION
000128 4980 C244 00244 468 CH R8,MAXSRCCLG IS MAXIMUM SOURCE LENGTH EXCEEDED
00012C 4720 C1F2 001F2 469 BH ERRTLGT2 YES-PERFORM ERROR FUNCTION
470 *
000130 5875 0004 00004 471 L R7,(R5) GET ADDRESS OF SOURCE (OUTPUT) BUFFR
000134 1A87 472 AR R8,R7 GET ADDRESS OF SOURCE END +1
000136 0680 473 BCTR R8,0 BACK UP TO SOURCE END

475 -----------------------------*
476 + PROCESS NEXT BYTE (8 BITS) OF SOURCE *
477 -----------------------------*
000138 478 TARGNEXT DS 0H
000138 1722 479 XR R2,R2 PRESET ALL BITS TO ZERO
00013A 5810 C450 00450 480 L R1,=’00000080’ SETUP ‘OR’ REGISTER FOR HIGH BIT
481 +
00013E 482 TARGX10 DS 0H
00013E 190A 483 CR R9,R10 ALL BYTES OF TARGET PROCESSED?
000140 4720 C15A 0015A 484 BH TARGNX30 YES-SKIP CHECK OF TARGET BYTE
000144 95F0 9000 00000 485 CLI 0(R9),XDBITOFF IS THIS BYTE OFF/ZERO/FALSE?
000148 4780 C156 00156 486 BE TARGNX20 YES-VALUE IS OK
00014C 95F1 9000 00000 487 CLI 0(R9),XDBITON IS THIS BYTE ON/ONE/TRUE?
000150 4770 C21C 0021C 488 BNE ERRCONV1 NO-PERFORM CONVERSION ERROR FUNCTION
000154 1621 489 OR R2,R1 INDICATE THE ONE IN SOURCE BYTE
000156 4720 490 TARGNX20 DS 0H
000156 4190 9001 00001 491 LA R9,(R9),0 POINT NEXT BYTE IN TARGET BUFFER
00015A 4720 492 TARGX30 DS 0H
00015A 8A10 0001 00001 493 SRA R1,1 SHIFT ‘OR’ REG TO NEXT BIT POSITION
00015E 4770 C13E 0013E 494 BH TARGX10 AND PROCESS IF NOT ZERO
495 +
000162 4220 7000 00000 496 STC R2,(0,R7) STORE THE SOURCE BYTE
000166 4170 7001 00001 497 LA R7,(0,R7) POINT NEXT SOURCE BYTE IN BUFFER
00016A 1978 498 CR R7,R8 MORE SOURCE BYTES TO PROCESS?
00016C 47D0 C138 00138 499 BNH TARGNEXT YES-BRANCH TO PROCESS NEXT BYTE
500 +
000170 18FF 501 SR R15,R15 SET SUCCESSFUL RETURN CODE
000172 47F0 C0EA 000EA 502 B EXITRETN GO TO COMMON RETURN POINT
504 =========================================================================
505 + PROCESSING A DXT-UIM 'DEFINITION CALL'. *
506 +
507 + LETS PERFORM UIM VALIDATION *
508 =========================================================================
509 +
000176 4870 B058 00058 511 LH R7,UDTSBYTV GET NUMBER OF MAX SOURCE BYTES
00017A 4970 C248 00248 512 CH R7,ZEROLGT IS SOURCE LENGTH SPECIFIED
00017E 4780 C1C8 001C8 513 BE ERRSLGT1 NO-PERFORM ERROR FUNCTION
000182 4970 C244 00244 514 CH R7,MAXSRCCLG IS MAXIMUM SOURCE LENGTH EXCEEDED
000186 4720 C106 00106 515 BH ERRSLGT2
516 + SOURCE IS VALIDATED - PROCESS TARGET
00018A 4860 B064 00064 517 LH R6,UDTDBYTV GET NUMBER OF MAX TARGET BYTES
00018E 95E5 B062 00062 518 CLI UDTDBYTV,C’V’ IS TARGET LENGTH VARYING
000192 4770 C1A0 001A0 519 BNE UIMVAL60 NOT JUST DO VALIDATION
000196 1867 520 LR R6,R7 DUPLICATE SOURCE BYTES
000198 89E0 0003 00003 521 SLL R6,3 MULTIPLE BY 8 FOR NUMBER BITS
00019C 4606 B064 00064 522 STH R6,UDTDBYTV SET NUMBER OF MAX TARGET BYTES
0001A0 4733 UIMVAL60 DS 0H VALIDATE TARGET LENGTH
0001A0 4960 C248 00248 524 CH R6,ZEROLGT IS TARGET LENGTH NON ZERO
0001A4 4780 C1E4 001E4 525 BE ERRTLGT1 NO-PERFORM ERROR FUNCTION
0001AB 4960 C246 00246 526 CH R6,MAXTARLG IS MAXIMUM TARGET LENGTH EXCEEDED
0001AC 4720 C1F2 001F2 527 BH ERRTLGT2
528 +
```
Customization Guide

Figure 29 (Part 9 of 10). Sample Field Exit Routine (Assembler)
Figure 29 (Part 10 of 10). Sample Field Exit Routine (Assembler)
Definitions for the First Sample Field Exit Routine

This section contains definitions associated with the sample Field exit routine. The following types of definitions are provided:

- IMS DBDGEN and PSBGEN definitions
- DB2 CREATE TABLE definitions
- DataRefresher definitions required to define the PR with DXT and to extract the IMS data with DataRefresher
- SQL statements required to define the PR in the MVG input tables without DataRefresher

DBDGEN Definitions

Figure 30 shows a DBDGEN definition for the Field exit routine in Figure 29 on page 128.

```
DBD NAME=DB1,ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC4mzqr!z!t,5,4), C
    EXIT=(EKYRUP/zerodot/zerodot)
DATASET DD1=HDAM,SIZE=4mzqr!z!t96,DEVICE=338m
mkrtsl
    SEGm NAME=SEG1,PARENT=mzqr!z!t,BYTES=12mzqr!z!t
mkrtsl
    FIELD NAME=(FLD1,SEQ,U),START=3,BYTES=2
DBDGEN FINISH END
```

Figure 30. DBDGEN Definition

Note: The EXIT= keyword of the DBD macro specifies that EKYRUP00 (the RUP) be called when a segment of this DBD is changed. This is required for synchronous HR propagation with DPROP.

PSBGEN Definitions

Figure 31 shows a PSBGEN definition for the Field exit routine in Figure 29 on page 128.

```
PCB TYPE=DB,DBDNAME=DB1,NAME=HUPPCB, C
    KEYLEN=12mzqr!z!t,PROCOPT=A
SENSEG NAME=SEG1
    PSBGEN PSBNAME=PSBDPR2
END
```

Figure 31. PSBGEN Definition

CREATE TABLE Statement

Figure 32 on page 139 shows a CREATE TABLE definition for the Field exit routine in Figure 29 on page 128.
CREATE TABLE TABLE01
  (COL1 CHAR(6) NOT NULL,
   COLB SMALLINT NOT NULL WITH DEFAULT,
   COLH SMALLINT NOT NULL WITH DEFAULT,
   COLC CHAR(32) NOT NULL WITH DEFAULT,
   PRIMARY KEY (COL1))
DATA CAPTURE CHANGES
IN DU096606.PROPTS ;
CREATE UNIQUE INDEX XN01 ON TABLE01 (COL1)
  USING VCAT KOE ;

Figure 32. CREATE TABLE Statement

Note: The DATA CAPTURE CHANGES clause specifies that the changed DB2 rows are captured and that the DB2CDCEX routine (the HUP) is called when a row of this table is changed. This is required for synchronous RH-propagation with DPROP.

Using DataRefresher to Define the PR

This section shows how to use DataRefresher to define the PR for the Field exit routine in Figure 29 on page 128.

CREATE DATATYPE

Figure 33 shows a CREATE DATATYPE definition for the Field exit routine in Figure 29 on page 128.

CREATE DATATYPE SRCTYPE=AA, EXIT=EKYEFL1A,
SRCBYTES=VARIES,
TRGTYPE=C,
TRGBYTES=VARIES;

Figure 33. CREATE DATATYPE Definition

The CREATE DATATYPE command provides the following information:

- It creates a user data type called AA and associates the Field exit routine EKYEFL1A with this user data type.
  The Field exit routine, EKYEFL1A, is called to reformat each field defined in a DXTPSB with a TYPE=AA keyword.

- SRCBYTES=VARIES means that the length of the fields (in their user format) with a user data type AA can have different BYTES values coded in the FIELD statements of the CREATE DXTPSB control statement.

- TRGTYPE=C means that the Field exit routine reformats the fields between the user data type and a character data type.

- TRGBYTES=VARIES means that the length of the fields (in their DPROP-supported and DXT-supported character format) are established by the definition call generated by DataRefresher UIM when it processes a FIELD statement with this user data type.
CREATE DXTPSB

Figure 34 shows a CREATE DXTPSB definition for the Field exit routine in Figure 29 on page 128.

CREATE DXTPSB NAME=KOEPSB2

   DXTPCB   NAME=PCBO01, DBACCESS=HDAM, DBNAME=DB1
         SEGMENT NAME=SEG1, PARENT=PCBmzqr!z!tmzqr!z!t1, BYTES=120
            FIELD NAME=FLD1 , START=3, BYTES=2, SEQFLD=R
            FIELD NAME=FLDB , START=5, BYTES=1, TYPE=B
            FIELD NAME=FLDH , START=6, BYTES=2, TYPE=H
            FIELD NAME=FLDC , START=9, BYTES=4, TYPE=AA

Figure 34. CREATE DXTPSB Definition

Notes:

1. The Field FLDC is defined as having a data type AA. When DataRefresher UIM processes this field statement, it calls the Field exit routine EKYEFL1A associated with the user data type AA for a definition call.

DataRefresher UIM also calls EKYEFL1A during the extract; DPROP calls it during propagation to reformat the field FLDC between its user data type AA and its character format.

2. The length of the FLDC in its user format is defined on the BYTES= keyword as four bytes.

   The length of the field in its DPROP format is set by the Field exit routine during the definition call that DataRefresher UIM generates.

CREATE DXTVIEW

Figure 35 shows a CREATE DXTVIEW definition for the Field exit routine in Figure 29 on page 128.

CREATE DXTVIEW NAME = VIEW011,
            DXTPSB = KOEPSB2,
            DXTPCB = PCBO01,
            SEGMENT = SEG1,
            MINSEG = SEG1,
            FIELDS = *

Figure 35. CREATE DXTVIEW Definition

DataRefresher UIM SUBMIT Command and EXTRACT Statement

Figure 36 on page 141 shows a DataRefresher UIM SUBMIT command and EXTRACT statement for the Field exit routine in Figure 29 on page 128.
SUBMIT  EXTID=PRO01,
            NODE=NODEX,
            USERID=T096606,
            CD=JCS,
            JCS=DDJCS01,
            FORMAT=SOURCE,
            MAPEXIT=EKYMCE00,
            MAPUPARM="PRTYPE=E,
                        MAPDIR=TW,
                        MAPCASE=1,
                        ACTION=REPL,
                        ERROPT=BACKOUT,
                        PCBLABEL=HUPPB"

EXTRACT
    INTO TABLE01 (COL1 NOT NULL,
                  COLB NOT NULL WITH DEFAULT,
                  COLH NOT NULL WITH DEFAULT,
                  COLC NOT NULL WITH DEFAULT)
    SELECT FLD1,
          FLD2,
          FLDH,
          FLD0
    FROM VIEW011 ;

Figure 36. DataRefresher UIM SUBMIT Command and EXTRACT Statement

Notes:

1. The MAPEXIT= keyword of the SUBMIT control statement specifies
   EKYMCE00. This results in DataRefresher UIM calling the DPROP Map
   Capture Exit EKYMCE00 during the processing of the SUBMIT or EXTRACT.
   This is needed to allow DPROP to create the PR.

2. The EXTRACT statement informs DataRefresher and DPROP which fields must
   be mapped to which columns. The EXTRACT statement indicates, for
   example, that the field FLDC must be mapped to column COLC.

Using DataRefresher for the Extract

This section covers INITDEM and USE DXTPSB Control Statements.

Figure 37 shows INITDEM and USE DXTPSB control statements for the Field exit
routine shown in Figure 35 on page 140.

INITDEM NAME=DXTPROD;
USE DXTPSB=KOEPSB2;

Figure 37. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control
Statements

Defining the PR in the MVG Input Tables

Figure 38 on page 143 describes the SQL statements required to define the PR in
the MVG input tables.

The following rows are inserted into the MVG input tables:

- One row is inserted into the DPRIPR table (the PR table).
This row identifies the PRID. By inserting an E into the PRTYPE column and a 1 into the MAPCASE column, the SQL statement indicates that the PR belongs to mapping case 1 of an extended-function PR.

- One row for the entity segment type SEG1 is inserted into the DPRISEG table (the SEG table).
  Because SEG1 is the root segment, no rows are inserted into DPRISEG for physical ancestors.

- One row is inserted into the DPRITAB table (the TAB table).
  This row indicates that the target table is T096606.TABLE01.

- One row is inserted into the DPRIFLD table (the FLD table) for each propagated field.
  The DPRIFLD row for the field FLDC has the value EKYEFL1A in the FLDEXIT column. This indicates that the Field is processed by the Field exit routine EKYEFL1A. The value AA in the DATATYPE column is used to identify the user data type.

  For PR definitions entered into the MVG input tables, the Field exit routine is not called for a definition call. Therefore, you must provide in the DPRIFLD row a complete definition of the field in its user and DPROP format. Accordingly, the row describing FLDC contains, in the BYTES column, the length of the field in its user format and, in the FLDEBYTE column, the length of the field in its DPROP format.
DELETE FROM T096606.DPRIPR WHERE PRID = 'PR001' ;

INSERT INTO T096606.DPRIPR
(PRID, USERID, PRTYPE, MAPCASE, MAPDIR, ERROPT, ACTION)
VALUES ('PR001', 'T096606', 'E', '1', 'TW', 'BACKOUT', 'REPL') ;

INSERT INTO T096606.DPRISEG
(PRID, DBNAME, SEGNAME, ROLE, PCBLABEL, VALUES ('PR001', 'DB1', 'SEG1', 'E', 'HUPPCB') ;

INSERT INTO T096606.DPRITAB
(PRID, TABQUAL, TABNAME )
VALUES ('PR001', 'T096606', 'TABLE01') ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR001', 'DB1', 'SEG1', 'FLD1', 'T096606', 'TABLE01', 'COL1', 'C ', 3 , 2 ) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR001', 'DB1', 'SEG1', 'FLD2', 'T096606', 'TABLE01', 'COL2', ' B ', 5 , 1 ) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES)
VALUES ('PR001', 'DB1', 'SEG1', 'FLD3', 'T096606', 'TABLE01', 'COL3', ' H ', 6 , 2 ) ;

INSERT INTO T096606.DPRIFLD
(PRID, DBNAME, SEGNAME, FLDNAME, TABQUAL, TABNAME, COLNAME, DATATYPE, POSITION, BYTES, FLDEXIT, FLDTYPE, FLDBYTE)
VALUES ('PR001', 'DB1', 'SEG1', 'FLDC', 'T096606', 'TABLE01', 'COLC', 'AA', 9 , 4 , 'EKYEFL1A', 'C ', 32 ) ;

COMMIT;

Figure 38. Defining the PR in the MVG Input Tables

Second Sample Field Exit Routine

Figure 39 on page 144 contains an example of a field exit routine in COBOL. Its functions are the same as those for the exit routine in “First Sample Field Exit Routine” on page 127. For information about this routine, refer to “First Sample Field Exit Routine” on page 127.

The source code in Figure 39 on page 144 is provided in the DPROP Sample Source Library (EKYSRC) under the member name EKYEFL1C. The definitions for this routine are the same as those for EKYEFL1A, except that the exit name is
different. Specifically, the EXIT=EKYEFL1A in Figure 33 on page 139, and EKYEFL1a in Figure 38, are changed to EKYEFL1C. The text that refers to EKYEFL1A is also true for EKYEFL1C. Refer to "Definitions for the First Sample Field Exit Routine" on page 138 for information about the definitions.

**Figure 39 (Part 1 of 9). Second Sample Field Exit Routine (COBOL)**
Chapter 3. Field Exit Routines

Figure 39 (Part 2 of 9). Second Sample Field Exit Routine (COBOL)
Figure 39 (Part 3 of 9). Second Sample Field Exit Routine (COBOL)
018000+ 01820000
018200+/ 01830000
01830000+********************************************************** 01840000
01840000+ INFORMATION FOR INSTALLATIONS WHICH USE DPROP WITHOUT DXT. 01850000
01850000+ ---------------------------------------------------------- 01860000
01860000+ THESE INSTALLATIONS DEFINE THE DL/I-TO-DB2 AND VICE-VERSA 01870000
01870000+ MAPPING BY PROVIDING MAPPING DEFINITIONS IN THE DPROP 01880000
01880000+ 'MVG INPUT TABLES'. 01890000
01890000+ USAGE OF THIS SAMPLE FIELD EXIT ROUTINE REQUIRES FOLLOWING 01900000
01900000+ DEFINITIONS IN THE DPRIFLD TABLE: 01910000
01910000+ INVOCATION OF A FIELD EXIT ROUTINE IS DEFINED BOTH 01920000
01920000+ BY SPECIFICATIONS IN THAT ROW OF THE 'DPRIFLD' TABLE 01930000
01930000+ WHICH DESCRIBES THE FIELD TO BE MAPPED. 01940000
01940000+ COLUMNS OF THE DPRIFLD ROW SHOULD PROVIDE FOLLOWING 01950000
01950000+ DEFINITIONS: 01960000
02000000+ COLUMN OF COLUMN 02010000
02010000+ DPRIFLD VALUE EXPLANATIONS 02020000
02020000+ ------------------------------------------------------------------------------------------------------------------ 02030000
02030000+ FLDEXIT = EKYEFLIC: THE EXIT LOAD MODULE NAME 02040000
02040000+ DATATYPE = XX : A TWO CHARACTER DATA-TYPE ID. 02050000
02050000+ BYTES = NNNN : THE SOURCE FIELD LENGTH 02060000
02060000+ FLDTYPE = C : THE TARGET DATA-TYPE MUST BE 'C '. 02070000
02070000+ FLDBYTE = MPPP : THE TARGET FIELD LENGTH 02080000
02080000+ (MUST BE & TIMES THE SOURCE 02090000
02090000+ FIELD LENGTH). 02100000
02100000+ SCALE = : SHOULD EITHER NOT BE PROVIDED OR 02110000
02110000+ SHOULD BE SPECIFIED AS ZERO. 02120000
02120000+ FLDSCAL = : SHOULD EITHER NOT BE PROVIDED OR 02130000
02130000+ SHOULD BE SPECIFIED AS ZERO. 02140000
02140000+ ------------------------------------------------------------------------------------------------------------------ 02150000
02150000+ END OF SPECIFICATIONS ************************************************* 02160000
02160000+/ 02170000
02170000+ IDENTIFICATION DIVISION. 02180000
02180000+ PROGRAM-ID. EKYEFLIC. 02190000
02190000+ ENVIRONMENT DIVISION. 02200000
02200000+ DATA DIVISION. 02210000
02210000+ WORKING-STORAGE SECTION. 02220000
02220000+ XBITON PICTURE X USAGE DISPLAY VALUE "I". 02230000
02230000+ XBITOFF PICTURE X USAGE DISPLAY VALUE "O". 02240000
02240000+ MAXSRCLG PICTURE S9999 USAGE COMPUTATIONAL VALUE 016. 02250000
02250000+ MAXTARLG PICTURE S9999 USAGE COMPUTATIONAL VALUE 0128. 02260000
02260000+ XTVALUE PICTURE S9999 USAGE COMPUTATIONAL. 02270000
02270000+/ 02280000
02280000+ EMESSAGE. 02290000
02290000+ EMSG0000. 02300000
02300000+ FILLER PICTURE X(16) 02310000
02310000+ VALUE "EXIT-EKYEFLIC - ". 02320000
02320000+ FILLER PICTURE X(44) 02330000
02330000+ VALUE "SOURCE LENGTH NOT SPECIFIED - REQUIRED. ". 02340000
02340000+ FILLER PICTURE X(04) 02350000
02350000+ VALUE " ". 02360000
02360000+ EMSG0010. 02370000
02370000+ FILLER PICTURE X(16) 02380000
02380000+ VALUE "EXIT-EKYEFLIC - ". 02390000
02390000+ FILLER PICTURE X(44) 02400000
02400000+ VALUE "SOURCE LENGTH EXCEEDS MAXIMUM ALLOWED. ". 02410000

Figure 39 (Part 4 of 9). Second Sample Field Exit Routine (COBOL)
Figure 39 (Part 5 of 9). Second Sample Field Exit Routine (COBOL)

024100  03  FILLER  PICTURE X(04)
024200  VALUE " ".
024300  02  EMSG020.
024400  03  FILLER  PICTURE X(16)
024500  VALUE "EXIT=EKYEFLEC - ".
024600  03  FILLER  PICTURE X(44)
024700  VALUE "TARGET LENGTH NOT SPECIFIED - REQUIRED. ".
024800  03  FILLER  PICTURE X(04)
024900  VALUE " ".
025000  02  EMSG030.
025100  03  FILLER  PICTURE X(16)
025200  VALUE "EXIT=EKYEFLEC - ".
025300  03  FILLER  PICTURE X(44)
025400  VALUE "TARGET LENGTH EXCEEDS MAXIMUM ALLOWED. ".
025500  03  FILLER  PICTURE X(04)
025600  VALUE " ".
025700  02  EMSG040.
025800  03  FILLER  PICTURE X(16)
025900  VALUE "EXIT=EKYEFLEC - ".
026000  03  FILLER  PICTURE X(44)
026100  VALUE "TARGET DATA TYPE MUST BE CHARACTER. ".
026200  03  FILLER  PICTURE X(04)
026300  VALUE " ".
026400  02  EMSG050.
026500  03  FILLER  PICTURE X(16)
026600  VALUE "EXIT=EKYEFLEC - ".
026700  03  FILLER  PICTURE X(44)
026800  VALUE "TARGET SCALE MUST NOT BE SPECIFIED. ".
026900  03  FILLER  PICTURE X(04)
027000  VALUE " ".
027100  02  EMSG000.
027200  03  FILLER  PICTURE X(16)
027300  VALUE "EXIT=EKYEFLEC - ".
027400  03  FILLER  PICTURE X(44)
027500  VALUE "DATA TYPE CALL FUNCTION CANNOT BE IDENTIFIED".
027600  03  FILLER  PICTURE X(04)
027700  VALUE " ".
027800  03  LINKAGE SECTION.
027900/*
028000  02  EMSG000.
028100  03  FILLER  PICTURE X(16)
028200  VALUE "EXIT=EKYEFLEC - ".
028300  03  FILLER  PICTURE X(44)
028400  VALUE "TARGET LENGTH EXCEEDS MAXIMUM ALLOWED. ".
028500  03  FILLER  PICTURE X(04)
028600  VALUE " ".
028700  03  COPY  EKRYCUDC.
028800  02  EMSG000.
028900  03  FILLER  PICTURE X(16)
028900  VALUE "EXIT=EKYEFLEC - ".
029000  03  FILLER  PICTURE X(44)
029100  VALUE "TARGET LENGTH NOT SPECIFIED - REQUIRED. ".
029200  03  FILLER  PICTURE X(04)
029300  VALUE " ".
029400  01  SRCFIELD.
030100   02 SRCBYTE     PICTURE X     USAGE DISPLAY OCCURS 16 TIMES.  02950000
030200+   02960000
030300+   02970000
030400+   THIS DESCRIBES THE TARGET FOR THE CONVERTED OUTPUT  * 02980000
030500+   ****************************************************** 02990000
030600+   ****************************************************** 03000000
030700+   01 TARFIELD.  03010000
030800+   02 TARBYTE     PICTURE X     USAGE DISPLAY OCCURS 128 TIMES.  03020000
030900+   ****************************************************** 03030000
031000+   ****************************************************** 03040000
031100+   THIS 64 BYTE USERAREA IS FOR THE EXCLUSIVE USE OF THIS  * 03050000
031200+   EXIT. ITS CONTENTS WILL BE PRESERVED BETWEEN CALLS.  * 03060000
031300+   IT IS INITIALIZED TO BINARY ZEROS.  * 03070000
031400+   ****************************************************** 03080000
031500+   01 USERAREA.  03100000
031700+   02 SPECAREA   PICTURE S99999 USAGE COMPUTATIONAL.  03110000
031800+   02 SPECARE2  PICTURE S99999 USAGE COMPUTATIONAL.  03120000
031900+   02 TARNUMBER  PICTURE S99999 USAGE COMPUTATIONAL.  03130000
032000+   02 TESTITX REDEFINES TARNUMBER.  03140000
032100+   03 TOPPART   PICTURE XXX USAGE DISPLAY.  03150000
032200+   03 TESTPART   PICTURE X     USAGE DISPLAY.  03160000
032300+   02 SCOUNT    PICTURE S9999 USAGE COMPUTATIONAL.  03170000
032400+   02 TCOUNT    PICTURE S9999 USAGE COMPUTATIONAL.  03180000
032500+   02 BCOUNT    PICTURE S9999 USAGE COMPUTATIONAL.  03190000
032600+   02 FUNCVALD  PICTURE X     USAGE DISPLAY.  03200000
032700+/*   ****************************************************** 03210000
032800+   ****************************************************** 03220000
032900+   02 SRCBYTE     PICTURE X     USAGE DISPLAY OCCURS 16 TIMES.  03230000
033000+   PROCEDURE DIVISION USING EXRCUDC  03240000
033100+   SRCFIELD      03250000
033200+   TARFIELD      03260000
033300+   USERAREA.  03270000
033400+   01 TARBYTE     PICTURE X     USAGE DISPLAY OCCURS 128 TIMES.  03280000
033500+   ****************************************************** 03290000
033600+   SET CONTROL FLAGS - EXIT ENTERED, EXIT IN CONTROL,  03300000
033700+   ****************************************************** 03310000
033800+   MOVE "X" TO UDENTRD.  03320000
033900+   MOVE "X" TO UDINCTL.  03330000
034000+   MOVE " " TO FUNCVALD.  03340000
034100+   01 TESTITX DEFINE CALL  03350000
034200+   SELECT THE REQUIRE PROCESSING ROUTINE BASED  03360000
034300+   ON CALL FUNCTION  03370000
034400+   01 SRCBYTE     PICTURE X     USAGE DISPLAY OCCURS 16 TIMES.  03380000
034500+   1. DXT-UIM DEFINE CALL  03390000
034600+   03400000
034700+ IF UDTCDEFN THEN  03410000
034800+   MOVE "X" TO FUNCVALD  03420000
034900+   PERFORM UIMVALDO THROUGH UIMVALXO.  03430000
035000+   03440000
035100+   2. DPROP/DataRefresher SOURCE TO TARGET  03450000
035200+   03460000
035300+ IF UDTCSRTG THEN  03470000
035400+   MOVE "X" TO FUNCVALD  03480000
035500+   PERFORM SRCTARTO THROUGH SRCARTX.  03490000
035600+   03500000
035700+   3. DPROP TARGET TO SOURCE  03510000
035800+   03520000
035900+ IF UDTCTGSR THEN  03530000
036000+   MOVE "X" TO FUNCVALD  03540000

Figure 39 (Part 6 of 9). Second Sample Field Exit Routine (COBOL)
Figure 39 (Part 7 of 9). Second Sample Field Exit Routine (COBOL)
PROCEDURE: SRCTART.

MOVE EMSG1 TO UDTXMESG.
ELSE
MOVE EMSG2 TO UDTXMESG.
END-ELSE.
GO TO TOTARGET.

Chapter 3. Field Exit Routines
048100* TO A '0' OR '1' BIT. * 04750000
048200* * 04760000
048300*** IN THE FOLLOWING PROCESS, THE 'TARGET' IS THE SENDING **** 04770000
048400*** FIELD AND THE 'SOURCE' IS THE RECEIVING FIELD **** 04780000
048500* + 04790000
048600***************************************************************************** 04800000
048700* 04810000
048800 TARTSRC0. 04820000
048900* 04830000
049000 MOVE ZERO TO UDXRETC. 04840000
049100 MOVE ZERO TO TCOUNT. 04850000
049200 MOVE ZERO TO SCOUNT. 04860000
049300+ 04870000
049400*** PROCESS FIRST OR NEXT 'SOURCE' BYTE. 04880000
049500+ 04890000
049600 TARTSRC1. 04900000
049700 MOVE 256 TO XTVALUE 04910000
049800 MOVE ZERO TO TARNUMBER. 04920000
049900 MOVE ZERO TO BCOUNT. 04930000
050000 ADD 1 TO SCOUNT 04940000
050100+ 04950000
050200*** WHEN ALL 'SOURCE' BYTES ARE FILLED, THEN STOP 04960000
050300*** ELSE, INITIALIZE THE 'SOURCE' BYTE TO ZERO. 04970000
050400+ 04980000
050500 IF SCOUNT IS GREATER THAN UDTSBYTV THEN 04990000
050600 GO TO TARTSRCX 05000000
050700 ELSE 05010000
050800 MOVE TESTPART TO SRCBYTE(SCOUNT). 05020000
050900+ 05030000
051000*** SET NEXT 'SOURCE' BIT TO 0 OR TO 1 DEPENDING IF 05040000
051100*** THE CORRESPONDING 'TARGET' BYTE IS '0' OR '1' 05050000
051200+ 05060000
051300 TARTSRC2. 05070000
051400 COMPUTE XTVALUE = XTVALUE / 2. 05080000
051500 ADD 1 TO BCOUNT. 05090000
051600 ADD 1 TO TCOUNT. 05100000
051700+ 05110000
051800 IF TCOUNT IS GREATER THAN UDTTBYTV THEN 05120000
051900 GO TO TARTSRC1. 05130000
052000+ 05140000
052100 IF TARBYTE(TCOUNT) = "1" 05150000
052200 ADD XTVALUE TO TARNUMBER. 05160000
052300+ 05170000
052400 IF BCOUNT EQUAL TO 8 THEN 05180000
052500 MOVE TESTPART TO SRCBYTE(SCOUNT) 05190000
052600 GO TO TARTSRC1 05200000
052700 ELSE 05210000
052800 GO TO TARTSRC2. 05220000
052900 TARTSRCX. 05230000
053000+ 05240000
053300***************************************************************************** 05250000

Figure 39 (Part 9 of 9). Second Sample Field Exit Routine (COBOL)
Chapter 4. Propagation Exit Routines

If the generalized mapping cases are not flexible enough for your needs, you can use a Propagation exit routine. This type of exit routine supplies all its own mapping logic and propagating SQL or DL/I calls. DPROP calls the exit routine, which retains many of the DPROP support functions. This is the advantage a Propagation exit routine has over an IMS Data Capture exit routine (as described in IMS/ESA Customization Guide), or a DB2 Data Capture exit routine. These DPROP-supported functions are discussed below.

If you have specified the use of a Propagation exit routine for a particular PR, DPROP calls your exit routine as soon as it receives the changed data. DPROP does not use any of its own mapping logic; instead, it relies on your exit routine to perform any data transformations you need and to propagate the data to the DB2 table or IMS database.

Your exit routine can be written in Assembler, or in COBOL, PL/I, or C. The DPROP support for exit routines written in HLLs requires LE/370 Version 1 Release 2.

For synchronous propagation, DPROP calls your exits in both IMS batch and online dependent regions accessing DB2. For LOG-ASYNC propagation the RUP calls your exit routines in an MVS batch environment using CAF attach to DB2. For user asynchronous propagation, depending on your implementation, the RUP can call your exit routine in IMS batch and dependent regions accessing DB2, or in a non-IMS DB2 TSO or DB2 CAF environment.

Propagation exit routines differ from Segment and Field exit routines, in that DataRefresher does not call Propagation exit routines during data extraction. In some cases, you can use DataRefresher's more powerful mapping capabilities to extract and load the data. Otherwise, you must write your own programs to extract the IMS data. Loading the DB2 tables can then be done either by creating an input data set for the DB2 Load Utility, or by inserting the DB2 rows with SQL statements; this takes more time.

Propagation exit routines differ from Segment and Field exit routines, in that the DPROP DLU does not call Propagation exit routines. Data propagated by Propagation exit routines can be passed, using sequential files, to the DLU. See IMS DPROP Reference for more information.

To avoid propagation failures, the mapping performed during the extract and load must be compatible with the mapping that your Propagation exit routine performs.

Environment Considerations for a Propagation Exit Routine

In Synchronous propagation mode, your Propagation exit routine can be called by the RUP (when the propagation direction is HR) or by the HUP (when the propagation direction is RH). Because the RUP and the HUP run as extensions of IMS mixed mode applications your Propagation exit routine runs as an IMS mixed mode application. This allows your Propagation exit routine to issue both DL/I calls and SQL calls, but you must link edit your Propagation exit routine with the DB2 language interface for IMS Attach.
In LOG-ASYNC propagation mode, your Propagation exit routine can only be called by the RUP (propagation direction is always HR). The RUP is called by the Receiver which runs as an MVS application with a CAF Attach to DB2. This means that your Propagation exit routine can only issue SQL calls. In this case, you must link edit your Propagation exit routine with the DB2 language interface for CAF Attach.

In User Asynchronous propagation mode, your Propagation exit routine can only be called by the RUP (propagation direction is always HR). The RUP is called by your own user-written receiver programs which can run either as an IMS application, a TSO application or as an MVS application with CAF Attach, depending on how you design it. If you design your own user-written receiver programs to run as an IMS mixed-mode application, then you can issue both DL/I calls and SQL calls from your Propagation exit routine.

It is recommended in all of the above cases that you code and link-edit your Propagation exit routine as reentrant. You must also link-edit your Propagation exit routine with the DPROP Trace Module EKYR410X.

How To Write A Propagation Exit Routine

Because you supply your own mapping logic and SQL or DL/I calls, DPROP is very flexible regarding the structure of your Propagation exit routine. You can even propagate data changes to more than one DB2 table. DPROP does not impose or check rules for the mapping of keys or referential integrity relationships (RIRs). DPROP also does not support the CCU and DLU. Mapping and verifying data propagation is left up to you.

Before discussing the development of your exit routine, the next section briefly lists which functions DPROP supports when using a Propagation exit.

Supported DPROP Functions

As mentioned above, DPROP does not impose or check rules for the mapping of keys or RIRs. Also, Propagation exits do not support the use of the CCU and the DLU.

However, DPROP still supports the following features when you use a Propagation exit routine:

- DPROP-provided tracing support
- DPROP-provided Audit support
- Standardized error handling
- Orderly suspension of propagation
- Activation or deactivation of PRs
- Emergency stops of all propagating activities
- The PROP OFF //EKYIN control statement
- Protection against unintentional updates during IMS extract and DLU processing
- Propagation definitions recorded in the DPROP directory
- Optional DBD version checking (for HR-propagation)
Although you control the propagation of changed data, DPROP still provides some of the valuable functions available to generalized mapping cases.

Creating your own user mapping with a Propagation exit routine, instead of using an IMS Data Capture exit routine, or DB2 Data Capture exit routine, helps establish a common process for managing the data propagation environment for both generalized and user mapping cases.

**Propagation Exit Routine Interface**

When DPROP receives the changed data, it calls your Propagation exit routine.

1. The RUP calls your Propagation exit routine for IMS-to-DB2 mapping with an interface similar to the IMS Data Capture Exit interface. The following control blocks are passed:
   - The Propagation Interface Control Block (PIC)
   - The Extended Program Communication Block (XPCB)
   
     The XPCB is a control block that IMS defines; it describes the changed IMS data.

2. The HUP calls your Propagation exit routine for DB2-to-IMS mapping with the following control blocks:
   - The Propagation Interface Control Block (PIC)
   - The HUP Exit Communication Block (HEC)

   The HEC is a control block that DPROP defines; it contains pointers to areas that the DB2 Data Capture exit passes.

Register 1 points to a list that is two fullwords long, containing the addresses of these control blocks.
Propagation Interface Control Block (PIC)

There is one interface control block per exit routine, lasting for the duration of the exit in virtual storage.

You can generate the following DSECT in your assembler exit routine by coding the EKYRPCIC macro statement. For HLL exit routines, you can include or copy one of the following members to map the Propagation exit routine Interface Control Block:

- **EKYRCPCCC**: Exit routines written in COBOL
- **EKYRCPCCP**: Exit routines written in PL/I
- **EKYRPCCK**: Exit routines written in C

Figure 40 on page 157 shows the structure of the control block, and is followed by a detailed description of its fields.
1  EKYRCPIC
2++++++++++++++++++++++++++++++++++ START OF CONTROL BLOCK SPECIFICATION ++++++++
3++*/
4++ CONTROL BLOCK NAME: +/-
5++  EKYRCPIC (PIC) +/-
6++*/
7++ DESCRIPTIVE NAME: +/-
8++  DPROP PROPAGATION EXIT INTERFACE BLOCK +/-
9++*/
10++*/
11++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
12++*/
13++ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM". +/-
14++*/
15++ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. +/-
16++  ALL RIGHTS RESERVED. +/-
17++*/
18++ U.S. GOVERNMENT USERS RESTRICTED RIGHTS - +/-
19++ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY +/-
20++  GSA ADP SCHEDULE CONTRACT WITH IBM CORP. +/-
21++*/
22++ LICENSED MATERIALS - PROPERTY OF IBM. +/-
23++*/
24++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
25++*/
26++ STATUS: V1 R2 M0 +/-
27++*/
28++ FUNCTION: +/-
29++ THIS IS THE CONTROL BLOCK USED TO INTERFACE BETWEEN +/-
30++ - DPROP +/-
31++ AND +/-
32++ - A USER'S PROPAGATION EXIT ROUTINE +/-
33++*/
34++ THERE IS ONE PIC CB FOR EACH EXIT PROPAGATION +/-
35++ EXIT ROUTINE, LASTING FOR THE DURATION OF THE EXIT +/-
36++ IN VIRTUAL STORAGE. +/-
37++ FOR SYNC PROPAGATION IN MPP REGIONS: +/-
38++ - THIS IS THE DURATION OF THE IMS PROGRAM CONTROLLER +/-
39++ SUBTASK. +/-
40++ FOR SYNC PROPAGATION IN BATCH/BMP REGIONS, FOR +/-
41++ ASYNCH PROPAGATION, AND FOR CCU PROCESSING: +/-
42++ - THIS IS THE DURATION OF THE JOBSTEP. +/-
43++*/
44++ MODULE TYPE= MACRO +/-
45++  PROCESSOR= ASSEMBLER H +/-
46++*/
47++ INNER CONTROL BLOCKS: NONE +/-
48++*/
49++ MACROS USED FROM MACRO LIBRARY: NONE +/-
50++*/
51++ CHANGE ACTIVITY: +/-
52++  KMP0587 12/13/90 +/-
53++  KMP0660 02/08/91 COPYRIGHT INFORMATION +/-
54++*/
55++++++++++++++++++++++++++++++++++ END OF CONTROL BLOCK SPECIFICATION ++++++++

000000
57+PIC  DSECT
58++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
59++ THIS SECTION CONTAINS INFORMATION PROVIDED BY +*
60++ DPROP TO THE INVOKED EXIT AT ENTRY TO CALL. THIS +*
61++ SECTION MUST NOT BE MODIFIED BY THE EXIT +*
62++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Figure 40 (Part 1 of 4). Interface Control Block for a Propagation Exit Routine
Figure 40 (Part 2 of 4). Interface Control Block for a Propagation Exit Routine
Figure 40 (Part 3 of 4). Interface Control Block for a Propagation Exit Routine
### Interface Control Block Field Descriptions

The following is a detailed description of the control block fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PICEYE</strong></td>
<td>Contains the constant <code>EKYRCPIC</code>, and is used to identify the control block in a dump.</td>
</tr>
<tr>
<td><strong>PICEXIT</strong></td>
<td>The load module name of the exit routine.</td>
</tr>
<tr>
<td><strong>PICCALL</strong></td>
<td>The call function that DPROP sets to HR to indicate hierarchical-to-relational or to RH to indicate relational-to-hierarchical propagation.</td>
</tr>
<tr>
<td><strong>PICDBLEV</strong></td>
<td>Contains the DPROP trace debug level in effect. If the PICDBLV2 bit is on, it indicates that you want to trace the propagating SQL statements for HR-propagation, and the propagating IMS calls for RH-propagation. The exit routine can then call the DPROP trace module.</td>
</tr>
<tr>
<td><strong>PICPTD</strong></td>
<td>Address of an internal DPROP control block that the exit needs for calls to the DPROP trace module.</td>
</tr>
<tr>
<td><strong>PICPRID</strong></td>
<td>The ID of the PR.</td>
</tr>
<tr>
<td><strong>PICPRSET</strong></td>
<td>The Set ID of the PR.</td>
</tr>
<tr>
<td><strong>PICPRTST</strong></td>
<td>The PR time stamp, assigned when MVG processed the PR.</td>
</tr>
</tbody>
</table>
**PICOPSYS**  Set to ESA to define the operating system.

**PICTRANS**  Identifies the IMS region type in which the exit routine is called. This field is blank if the exit routine is called from outside an IMS region—for example, during LOG-ASYNC propagation or user asynchronous propagation.

**PICPROGM**  Describes the program calling the exit routine. Set to DPRS for synchronous propagation or DPRA for LOG-ASYNC propagation or user asynchronous propagation.

The next two fields are switches that are useful for problem determination. DPROP does not require your exit routine to set these fields. However, they can help you determine where a problem occurred if you have an ABEND. DPROP sets these fields to blanks before the first time your exit routine is called.

**PICENTRD**  When you enter your exit routine, set this field to X. DPROP does not change this field again, so if a problem occurs, you can determine if your exit has been entered.

**PICINCTL**  You must also set this field to X, indicating that your exit routine has control. When DPROP regains control, it resets this field to blanks, so you can determine if your exit routine has control when an ABEND occurs.

The next two fields can be used along with the RUP’s and HUP’s error handling logic. For more information on return codes and error handling techniques, see “Return Codes and Error Handling Techniques” on page 184.

**PICXRETC**  The return code that the exit routine provides when returning to its caller. This field is set to zero when the exit routine is called.

0  Propagation was successful.
4  SQL error. Use return code 4 only if the failing SQL statement used the SQL communication area SQLCA provided in the interface control block.
8  DL/I call error. Use return code 8 only if the failing DL/I call used the DL/I Application Interface Block (AIB) provided in the interface control block.
12  Propagation failure (not caused by SQL or DL/I error); unavailable resource problem.
16  Propagation failure (not caused by SQL or DL/I error); Not an unavailable resource problem.
20  Severe error; DPROP ABENDs.

**PICXMESG**  User-provided error message. It is set to blanks when the exit routine is called. When the exit routine returns, if the first eight bytes are not blank, DPROP writes the contents of the field as an error message with its usual error reporting logic. It is written as a four-line message with 70 bytes in each line. If the trailing lines contain only blanks, they are not written.

The message lines must have the following format:

- The first eight bytes of the first message line must be a message ID, beginning with a letter in the range J-Z (to avoid confusion with IBM-provided messages).
• The ninth character of the first message line must be blank.
• The remaining 61 bytes of the first message line, and the entire second, third, and fourth message lines, can all be used for your message text.

If your exit routine returns an error code to its caller, the following fields can be used to identify which data objects are associated with the error.

For HR-propagation:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICTABQ</td>
<td>Table name qualifier of the table involved in the error.</td>
</tr>
<tr>
<td>PICTABN</td>
<td>Unqualified table name of the table involved in the error.</td>
</tr>
</tbody>
</table>

For-RH propagation:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICDBN</td>
<td>DBDNAME of the IMS database involved in the error.</td>
</tr>
<tr>
<td>PICSEGN</td>
<td>Segment name of the segment involved in the error.</td>
</tr>
</tbody>
</table>

The following field is the work area for the exit routine.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICSWORK</td>
<td>The work area can be used to save information across calls to the exit routine. You can also use this field to hold the address of storage that the exit routine obtains the first time it gains control. DPROP initializes this field to binary zeros before the first call to the exit routine, and never changes this field again. The contents of this field are saved until an application ABENDs in an MPP or an IFP region, when MVS releases the storage. After the ABEND, DPROP again initializes this field to binary zeros. For these types of asynchronous propagation, the contents of this field are preserved until the end of the MVS task that the receiver program uses to call the RUP. The PIC, and therefore the work area, is associated with an exit name. When an exit routine is called for multiple segments, tables, or multiple PRs, the work area is the same.</td>
</tr>
</tbody>
</table>

SQLCA

This area is the SQL Communication Area, used for the SQL statements your exit routine executes. It is recommended that all SQL statements that your Propagation exit routine generates use this SQL communication area.

If your exit routine encounters an SQL error and returns with a return code of 4, DPROP uses the contents of this area to determine which type of SQL error occurred and to provide detailed error messages.

DFSAIB

This area is the DL/I Application Interface Block (AIB) used for the DL/I calls your exit routine executes. It is recommended that all DL/I calls that your Propagation exit routine generates use this AIB.

If your exit routine encounters a DL/I error and returns with a return code of 8, DPROP uses the contents of this area to determine which type of DL/I error occurred and to provide detailed error messages.
Interface for HR Propagation

This section describes the interface used for HR-propagation. If your exit routine must not support HR-propagation, then you can skip this section and continue with the section “Interface for RH-Propagation” on page 171.

Interfaces between the RUP and your Propagation exit routine are the XPCB and the Extended Segment Data Block (XSDB). These are control blocks that the IMS Data Capture function defines; they are used to describe the changed IMS data.

The XPCB is the second parameter passed to your Propagation exit routine when the RUP calls it. It is used to provide information about the changed data and to point to XSDBs. An XSDB points to, and describes, either a changed segment occurrence or a physical ancestor of a changed segment.

Your exit routine must not modify the XPCB, the XSDB, or the data pointed to by these control blocks.

Figure 41 on page 164 provides an overview of the interface defined through the XPCB and XSDBs.
As shown in the numbered sections of the figure, the interface consists of:

1. One XPCB control block that provides a description of the changed data and contains various pointers.
2. A pointer to the fully concatenated key of the changed segment.

**Figure 41. XPCB and XSDB Control Block Structures**
3. A pointer to the XSDB control block describing the changed segment. This XSDB points to the data of the changed segment.

4. For Replace operations, a pointer to an XSDB describing the segment before it was replaced. The XSDB also points to the data of the before-image of the segment.

5. A pointer to the first XSDB in a chain of XSDBs for the hierarchical ancestors of the changed segment. The chain is in descending hierarchical order, with each XSDB pointing to the segment data of the segment and the next XSDB in descending order.

6. A pointer to the DBD version ID.

7. A pointer to the DB PCB.

8. A pointer to an area containing the output of an implied IMS INQY ENVIRON call.

The XPCB and XSDB Control Blocks
You can generate the following DSECTs in your assembler exit routine by coding the EKYRCDL1 macro statement. For HLL exit routines, you can include or copy one of the following members to map the XPCB and XSDB Control Blocks:

- **EKYRCDLC**  Exit routines written in COBOL
- **EKYRCDLP**  Exit routines written in PL/I
- **EKYRCDLK**  Exit routines written in C

**XPCB DSECT**
The XPCB control block is shown in Figure 42 on page 166 followed by a detailed description of those fields that are most useful to your exit routine.
The fields you need to use are:

**XPCBDBD**  
The physical database name.

**XPCBVERA**  
A pointer to a variable-length character string that identifies the DBD version. Unless the character string is set from the DBD VERSION= keyword, it is the time stamp of the DBDGEN. The first two bytes contain the length of the string followed by the string itself.

**XPCBSEG**  
The name of the updated physical segment type.
XPCBCALL Depending on the IMS call function, this field contains one of the following values:

REPL The IMS application generated a Replace call.
ISRT The IMS application generated an Insert call.
DLET The IMS application generated a Delete call.
CASC The IMS application generated a Delete call that resulted in a cascading delete of the IMS segment being processed by the current call of the Propagation exit routine.

The following value can be provided when logical parent segment types have an IMS Logical delete rule, and are involved in a unidirectional logical relationship. The value is encountered both for the logical parent segment type, and for its physical ancestors.

DLP The IMS application generated a Delete call that resulted in a delete from the logical path. This value is provided as a result of deleting the last logical child of a logical parent that was no longer accessible through a physical path (the logical parent segment was only accessible through its logical path). When the delete is completed, the logical parent segment is no longer accessible, either through logical or physical paths.

Refer to IMS/ESA Customization Guide for more information on this field.

XPCBPCALL The physical update function. This differs from the IMS call function and from the content of XPCBCALL. For example, when an application inserts a concatenated logical parent or child that was deleted on the same path, IMS performs a physical replace of the logical parent instead of an insert.

The logic of your Propagation exit routine depends on the combination of values in XPCBCALL and XPCBPCALL. Refer to “The XPCBCALL, XPCBPCALL, and XSDBPHP Fields” on page 170 for examples of valid logic.

XPCBPCALL can have the following values:

REPL A segment is replaced.
ISRT A segment is inserted.
DLET A segment is deleted. If the segment is involved in a logical relationship, it is no longer accessible by either its physical or logical paths.

The following two values can be provided when you have an IMS delete rule of LOGICAL with a unidirectional logical relationship. The values can be provided for both the logical parent segment type and its physical ancestors. For more information, see the appropriate Administrators Guide for your propagation mode.

DLPP A segment has been deleted from the physical path of the current segment. The current segment is still accessible from its logical path.
REIN  The reinsert of a segment that was no longer accessible from its physical path, but accessible through a logical path.

For more information on the XPCBPCALL, refer to *IMS/ESA Customization Guide*.

**XPCBINQA**  Address of the output of an IMS INQY ENVIRON call. An implied IMS INQY call is done before calling the exit routine. Therefore, the information returned to an application program after an INQY call is available to the exit routine without having to generate the call. This information includes the PSBNAME, RECOVERY TOKEN, PCB LIST, and so forth. You can use this information to augment the data in the exit routine control blocks. See *IMS/ESA Application Programming: DL/I Calls* for more details about the INQY ENVIRON call.

**XPCBCKEYL**  The length of the fully concatenated key. This field is zero if the fully concatenated key is not provided (for example, if the EXIT keyword of the DBD specifies the NOKEY data option).

**XPCBCKEYA**  The address of the fully concatenated key. This field is zero if the fully concatenated key is not provided (for example, if the EXIT keyword of the DBD specifies the NOKEY data option).

**XPCBXSDBD**  Address of the XSDB control block for the changed segment data. This field is zero if the XSDB is not provided (for example, if the EXIT keyword of the DBD specifies the NODATA data option).

**XPCBXSDBB**  Address of the XSDB control block for the *before-image* of a replaced segment. This field is zero if the XSDB is not provided (for example, if the EXIT keyword of the DBD specifies the NODATA data option, or if the IMS change is not a replace).

**XPCBXSDBP**  Pointer to the first XSDB on the descending hierarchic chain. This field is zero if the chain of XSDBs is zero (for example, if the EXIT keyword of the DBD specifies the NOPATH option, or if the changed segment is a root segment).

The XPCBRC, XPCBRSNC, and XPCBEXIWP fields are reserved for RUP use. Your exit routine must not modify them.
XSDB DSECT
The XSDB control block is shown in Figure 43, followed by a detailed description of those fields that are most useful to your exit routine.

XSDB Field Descriptions
The fields of the XSDB that you are likely to need are:

**XSDBNXSDB** If the XSDB describes path data, this field contains the address of the next XSDB. The XPCB points to the first XSDB, but there is more than one XSDB for path data. They are in hierarchical, top-down sequence. In this case, the XSDBs are chained together, with the last pointer set to zero to indicate the end of the chain.

If the XSDB does not describe path data, this field contains a zero.

**XSDBSEG** The physical segment name.

**XSDBPHP** Accessibility through the physical path.

This field describes whether a segment is accessible through its physical path. The field can have the following values:

- **Y** (Yes) the segment is accessible through its physical path.
- **N** (No) the segment is not accessible through its physical path.

This field is set to **Y**, unless you have an IMS *logical* delete rule for logical parent segment types. It can be set to **N** for such logical parents and their physical ancestors. Refer to *IMS/ESA Customization Guide* for more information about this field.

**XSDBSEGLV** The segment level in the database.
### The XPCBPCALL, XPCBCALL, and XSDBPHP Fields

If your Propagation exit routine does not need to support logical parent segments and their physical ancestors having a LOGICAL IMS delete rule and involved in a unidirectional IMS logical relationship, then you need to test only the value of the XPCBPCALL field. In this case, the logic of a Propagation exit routine performing a simple mapping can be summarized in the following table:

<table>
<thead>
<tr>
<th>XPCBPCALL</th>
<th>Meaning</th>
<th>Exit Routine Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPL</td>
<td>A segment is replaced</td>
<td>Propagate with SQL UPDATE statements</td>
</tr>
<tr>
<td>ISRT</td>
<td>A segment is inserted</td>
<td>Propagate with an SQL INSERT</td>
</tr>
<tr>
<td>DLET</td>
<td>A segment is deleted</td>
<td>Propagate with an SQL DELETE</td>
</tr>
</tbody>
</table>

More complex mapping (for example, mapping similar to generalized mapping case 2) propagates the ISRT of an extension segment with an SQL UPDATE statement.

The logic of your propagation exit routine becomes more complex if it needs to support a logical parent segment or one of its physical ancestors having a LOGICAL IMS delete rule and involved in a unidirectional IMS logical relationship. In this case, you first need to decide how the delete of the logical parent (or its physical ancestors) is propagated. You can do this in two ways:

1. Delete the DB2 target row as soon as the segment gets deleted on its physical path (even if the logical parent segment still has logical children and remains accessible through a logical path).
2. Delete the DB2 target row only when the segment gets both physically and logically deleted.

The sample Propagation exit routine illustrates the logic supporting the first choice. Its logic is summarized in Figure 45. For the various combinations of XPCBPCALL, XPCBCALL, and XSDBPHP field values, the table in the figure describes the action taken by the sample exit routine. When taking the described actions, the exit routine does not need to check if the updated segment is involved in logical relationships. A dash (-) in a column of the table below means that a test of that value is not performed in the sample exit routine for the combination of values in that row.
**Figure 45. Exit Routine Action Based on the XPCBPCALL, XPCBCALL, and XSDBPHP Field Values**

<table>
<thead>
<tr>
<th>XPCBPCALL</th>
<th>XPCBCALL</th>
<th>XSDBPHP</th>
<th>Meaning</th>
<th>Exit Routine Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPL</td>
<td>-</td>
<td>Y</td>
<td>A segment accessible through its physical path is replaced.</td>
<td>Propagate with an SQL UPDATE.</td>
</tr>
<tr>
<td>REPL</td>
<td>-</td>
<td>N</td>
<td>A segment not accessible through its physical path is replaced through its logical path.</td>
<td>Ignored by exit routine.</td>
</tr>
<tr>
<td>ISRT</td>
<td>-</td>
<td>-</td>
<td>A segment is inserted.</td>
<td>Propagate with an SQL INSERT.</td>
</tr>
<tr>
<td>REIN</td>
<td>-</td>
<td>-</td>
<td>A segment previously physically deleted (but still accessible through its logical path) is physically reinserted.</td>
<td>Propagate with an SQL INSERT.</td>
</tr>
<tr>
<td>DLET</td>
<td>DLET or CASC</td>
<td>-</td>
<td>A segment is physically deleted (if involved in a logical relationship, it is neither accessible through the logical path nor through the physical path).</td>
<td>Propagate with an SQL DELETE.</td>
</tr>
<tr>
<td>DLPP</td>
<td>DLET or CASC</td>
<td>-</td>
<td>A segment is physically deleted, but it remains accessible through a logical path.</td>
<td>Propagate with an SQL DELETE.</td>
</tr>
<tr>
<td>DLET</td>
<td>DLLP</td>
<td>-</td>
<td>A segment previously physically deleted is now also being logically deleted.</td>
<td>Ignored by exit routine.</td>
</tr>
</tbody>
</table>

**Interface for RH-Propagation**

The following section describes the interface used for RH-propagation. If your exit routine must not support RH-propagation, then you can skip this section and continue with the section “Exit Routine Processing” on page 182.

The HUP Exit Communication Block (HEC) is the second parameter passed to your Propagation exit routine when the HUP calls your routine. It provides the pointers to the areas received from the DB2 Data Capture (DB2CDC). These areas describe and contain the captured changed data, and are listed below:

- **QWHC**: Is the DB2 Instrumentation Facility standard header mapped by DSNDQWHC.
- **QWHS**: Is the DB2 Instrumentation Facility correlation data mapped by DSNDQWHS.
- **CDCDD**: Contains the Data Capture table description and is mapped by the QW0185 DSECT within DSNDQW02.
**CDCDA**

Contains the Data Capture data row and is also mapped by the QW0185 DSECT within DSNDQW02.

For inserts and deletes, there is one data row with the data of the inserted or deleted row. For updates, there is one data row containing the after-image and one data row with the before-image of the updated row.

Your exit routine must not modify the HEC or the data pointed to by this control block.

Figure 46 provides an overview of the interface defined through the HEC.

---

Figure 46. HEC, QWHS, QWHC, Table Description and Data Row Control Block Structures

As shown in the numbered sections of the figure, the interface consists of:

1. One HEC control block that provides various pointers.
2. A pointer to the DB2 Instrumentation Facility standard header data that contains specific DB2 information based on the active trace.
3. A pointer to the DB2 Instrumentation Facility correlation data header containing information about correlation and authorization.
4. A pointer to the Data Capture table description of the changed table and its columns.

5. A pointer to the Data Capture Data (data row) record containing the **after** image of the captured row. For SQL INSERT and DELETE, this is the only data row passed to your exit routine.

6. A pointer to the Data Capture Data (data row) record containing the **before** image of the captured row. This data row is only present for update operations.

7. A field containing the reason code returned by DB2 for the generated IFI call to retrieve the captured data. See *DB2 Messages and Codes* for a description of IFI reason codes.

**The HEC Control Block**

You can generate the following DSECT in your assembler exit routine by coding the EKYHCHEC macro statement. For HLL exit routines, you can include or copy one of the following members to map the HUP Exit Communication Block:

- **EKYHCHCC** Exit routines written in COBOL
- **EKYHCHCP** Exit routines written in PL/I
- **EKYHCHCK** Exit routines written in C
**HUP Exit Communication Block**

**FUNCTION:**

This is the control block used to pass information got by `DPROP` from the DB2 CHANGED DATA CAPTURE exit (using IFI calls) to the propagation exit routine and/or the DB2 CHANGED DATA CAPTURE subexit routine.

**HEC** is build for each exit call new and does contain data to be retained between exit calls.

**MODULE TYPE:** MACRO

**PROCESSOR:** ASSEMBLER H

**INNER CONTROL BLOCKS:** None

**MACROS USED FROM MACRO LIBRARY:** None

**CHANGE ACTIVITY:**

**STATUS:** V1 R2 M0

---

**START OF CONTROL BLOCK SPECIFICATION**

1 EKYHCHEC
2************************** START OF CONTROL BLOCK SPECIFICATION **************************
3*
4** CONTROL BLOCK NAME: **
5** EKYHCHEC (HEC) **
6*
7** DESCRIPTIVE NAME: **
8** DPROP HUP EXIT COMMUNICATION BLOCK **
9** = = = **
10*
11******************************************************************************

12*
13** THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
14*
15** 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. **
16** ALL RIGHTS RESERVED. **
17*
18** U.S. GOVERNMENT USERS RESTRICTED RIGHTS - **
19** USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY **
20** GSA ADP SCHEDULE CONTRACT WITH IBM CORP. **
21*
22** LICENSED MATERIALS - PROPERTY OF IBM. **
23*
24******************************************************************************

25*
26** STATUS: V1 R2 M0 **
27*
28** FUNCTION: **
29** THIS IS THE CONTROL BLOCK USED TO PASS INFORMATION GOT BY DPROP FROM THE DB2 CHANGED DATA CAPTURE EXIT **
30** (USING IFI CALLS) TO THE PROPAGATION EXIT ROUTINE AND/OR THE DB2 CHANGED DATA CAPTURE SUBEXIT ROUTINE. **
31*
32** THE HEC IS BUILD FOR EACH EXIT CALL NEW AND DOES **
33** CONTAIN DATA TO BE RETAINED BETWEEN EXIT CALLS. **
34*
35** MODULE TYPE= MACRO **
36** PROCESSOR= ASSEMBLER H **
37*
38** INNER CONTROL BLOCKS: NONE **
39*
40** MACROS USED FROM MACRO LIBRARY: NONE **
41*
42** CHANGE ACTIVITY: **
43*
44** STATUS: V1 R2 M0 **
45*
46******************************************************************************

---

**END OF CONTROL BLOCK SPECIFICATION**

**Figure 47 (Part 1 of 2). HUP Exit Communication Block**

---

174 Customization Guide
The QWHS and QWHC Control Blocks
The IFI standard header data and IFI correlation data are passed as received from the DB2 Instrumentation Facility.

DSNDQWHS: is the DB2 provided macro which maps the standard header data.
DSNDQWHC: is the DB2 provided macro which maps the correlation data.

Refer to DB2 Administration Guide for information about these control blocks.

The Table Description and Data Row Control Blocks
The Data Capture Table Description contains a description of the captured data. It is always present when the HUP calls your Propagation exit routine.

The Data Capture Data (data row) contains a row's data. When the HUP calls your Propagation exit routine, it passes one or two data row areas, depending on the type of SQL operation that caused the data to be captured:

- For INSERT and DELETE, there is only one data row that contains either the inserted or deleted row.
- For UPDATE, there are two data rows, one containing the image of the row before the update, and one after the update operation.
Both data rows have the same format and are described by the same Data Capture table description, which is passed to your exit routine.

The table description and data row are composed of a header common to both, and a data part which is different for each control block type:

- The header part describes the table, using its qualified table name and the time stamp of the table description. For the data row, it also contains the RBAs of log records, the operation code, and the operation code qualifier.
- The data part of the table description contains a description of the columns of the table. The description is similar to the SQLDA.
- The data part of the data row contains the row data, as described in the table description data part.

You can generate the following DSECT (provided by DB2) in your assembler exit routine by coding the DSNDQW02 macro statement. This macro contains the QW0185 DSECT that represents the mapping of the table description and data row control blocks that the DB2 Data Capture uses.

For HLL exit routines, you can include or copy one of the following members to map the table description and data row control blocks:

- **EKYHCQ2C** For exit routines written in COBOL
- **EKYHCQ2P** For exit routines written in PL/I
- **EKYHCQ2K** For exit routines written in C
DSNDQW02

+---------------------------------------------------------------+
| 4+   QW0185S IS WRITTEN FOR READS REQUESTS FOR IFCID 185.         |
| 5+   FOR IFCID 185, THE PRODUCT SECTION WILL PRECEDE THE DATA  |
| 6+   SECTION. A SINGLE READS REQUEST FOR IFCID 185 MAY RESULT IN |
| 7+   A SERIES OF 185 RECORDS. ONLY THE FIRST 185 RECORD IN SUCH  |
| 8+   A SERIES WILL CONTAIN A PRODUCT SECTION. IFCID 185 RECORDS |
| 9+   MAY BE BROKEN AT ANY POINT IN THE DATA. IT IS UP TO THE    |
| 10+  READER OF THE RECORD TO INTERPRET SPANNED IFCID 185 RECORDS.|
| 11+  |
| 12+  QW0185 CONTAINS A HEADER SECTION WHICH IS FOLLOWED BY A DATA |
| 13+  SECTION. THE DATA PORTION OF QW0185 BEGINS WITH FIELD      |
| 14+   - QW0185ID IF QW0185TP=S                                |
| 15+   OR                                                     |
| 16+   - QW0185DR IF QW0185TP=D                               |
+---------------------------------------------------------------+

**Figure 48 (Part 1 of 2). Table Description and Data Row Control Blocks**
### The Table Description and Data Row Header

The following describes the fields of the table description and data row header part in more detail:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QW0185LN</td>
<td>Length of total table description or data row (header and data).</td>
</tr>
<tr>
<td>QW0185TP</td>
<td>Contains the CDC control block type and is:</td>
</tr>
<tr>
<td></td>
<td><strong>S</strong> For the DB2CDC table description</td>
</tr>
<tr>
<td></td>
<td><strong>D</strong> For the DB2CDC data row</td>
</tr>
<tr>
<td>QW0185RC</td>
<td>Reason code describing errors for this table and used only for the data row. If a severe error was detected for this table, the HUP does not call your Propagation exit routine and enforce the rollback of the changes. Therefore, the only reason code that your Propagation exit routine must be able to handle, is the warning code X'00E60A0B'. This code indicates that although the date or time install option was specified as LOCAL, a date or time column value of the row has been returned in ISO format. The DB2 Data Capture never calls date and time exits.</td>
</tr>
</tbody>
</table>
QW0185QT  The qualified table name, which is composed by the table creator (QW0185CR) and table name (QW0185TB).

QW0185CR  Creator name (authorization ID), which is 8 bytes long and padded with blanks.

QW0185TB  Table name, which is 18 bytes long and padded on the right with blanks.

QW0185TS  Time stamp (internal format) of table description from the catalog.

QW0185TL  Time stamp (internal format) of log record within the log buffer CI. This field is present only in the data row (QW0185TP=D).

QW0185UR  RBA of the first log record for this unit of work. This field is present only in the data row (QW0185TP=D).

QW0185LR  RBA of log record of this data row. This field is present only in the data row (QW0185TP=D).

QW0185PC  Operation code describing the type of row image and the SQL operation that performed the data change. This field is present only in the data row (QW0185TP=D). The possible values of QW0185PC are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>Insert</td>
</tr>
<tr>
<td>UB</td>
<td>Update before-image</td>
</tr>
<tr>
<td>UA</td>
<td>Update after-image</td>
</tr>
<tr>
<td>DE</td>
<td>Delete</td>
</tr>
</tbody>
</table>

QW0185RI  Operation code qualifier present only in the data row (QW0185TP=D). This field is either blanks, or RI if the operation is a result of a referential constraint enforcement of a DELETE SET NULL or CASCADE operation.

The Table Description Data
The table description data portion contains a similar form of an SQLDA that describes the table. It is like the standard SQLDA external format, except for the field where you usually specify the address of the data area for a particular column. In the CDC table description this field is already set and contains the offset to the column within the data row data section, which is optionally prefixed by a null indicator variable.

The data portion of the table description consists of four variables, followed by an arbitrary number of occurrences of a sequence of five variables collectively called QW0185VR.

QW0185ID  An eye catcher for storage dumps containing CDCDD.

QW0185BC  The length of the table description data portion. It is (QW0185NO * 44) + 16.

QW0185NO  Total number of occurrences of QW0185VR.

QW0185LD  The number of columns described by occurrences of QW0185VR.

The following five variables are collectively called QW0185VR and occur QW0185NO times in the table description. Each occurrence of QW0185VR describes a column in the captured table.
QW0185ST  Tells the data type of the column and whether it has an associated indicator variable. For a description of the type codes, see Figure 49 on page 181.

QW0185LE  Defines the external length of a value of the column, as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Length attribute in bytes</td>
</tr>
<tr>
<td>Graphic</td>
<td>Length attribute in <strong>bytes</strong></td>
</tr>
<tr>
<td>Decimal</td>
<td>byte 1 = precision</td>
</tr>
<tr>
<td></td>
<td>byte 2 = scale</td>
</tr>
<tr>
<td>Float</td>
<td>4 (bytes) for single precision</td>
</tr>
<tr>
<td></td>
<td>8 (bytes) for double precision</td>
</tr>
<tr>
<td>Smallint</td>
<td>2 (bytes)</td>
</tr>
<tr>
<td>Integer</td>
<td>4 (bytes)</td>
</tr>
<tr>
<td>Date</td>
<td>10 (bytes) or LOCAL value</td>
</tr>
<tr>
<td>Time</td>
<td>8 (bytes) or LOCAL value</td>
</tr>
<tr>
<td>Time stamp</td>
<td>26 (bytes)</td>
</tr>
</tbody>
</table>

QW0185SD  Contains the CCSID (Coded Character Set Identifier) in bytes 3 and 4. It is a two-byte (unsigned) binary number that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages.

QW0185SI  Contains a flag byte and the offset of this column into the data row. The flag byte indicates if the column can be nullable or not. If the column value can be NULL, then the column data in the data row is prefixed by an indicator variable (2 bytes). The offset points to the null indicator variable instead of the data for the column; the data immediately follows the indicator and starts at offset + 2. The indicator variable is a two-byte field in the data row containing X'FFFF' (value -1) if the field is null, or X'0000' if the field contains data.

The format of the QW0185SI field is:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flag byte. If highest bit (bit 0) is on, then the column is prefixed with a null indicator variable, and the real data starts at offset + 2. The remaining bits are reserved.</td>
</tr>
<tr>
<td>2-4</td>
<td>Offset into the data, or indicator variable for this column. This offset must be added to the data row data portion address (QW0185DR) to compute the virtual storage address of the column data or indicator variable.</td>
</tr>
</tbody>
</table>

QW0185SN  Length of name (QW0185NL) and name of the column (QW0185CN).

QW0185NL  Contains the length of the column name.

QW0185CN  Contains the name of the column.
The table below lists values of the QW0185ST field of the table description and their meanings. There are two values for each data type. The first value means that the column does not have a null indicator and does not allow nulls; the second means the column has a null indicator and allows nulls. For more information about data types, refer to DB2 SQL Reference.

**Figure 49. Values of QW0185ST and Their Meanings**

<table>
<thead>
<tr>
<th>Values</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>384/385</td>
<td>Date</td>
</tr>
<tr>
<td>388/389</td>
<td>Time</td>
</tr>
<tr>
<td>392/393</td>
<td>Time stamp</td>
</tr>
<tr>
<td>448/449</td>
<td>Variable-length character string</td>
</tr>
<tr>
<td>452/453</td>
<td>Fixed-length character string</td>
</tr>
<tr>
<td>456/457</td>
<td>Long character string</td>
</tr>
<tr>
<td>460/461</td>
<td>Variable-length, optionally null terminated character string (C)</td>
</tr>
<tr>
<td>464/465</td>
<td>Variable-length graphic string</td>
</tr>
<tr>
<td>468/469</td>
<td>Fixed-length graphic string</td>
</tr>
<tr>
<td>472/473</td>
<td>Long graphic string</td>
</tr>
<tr>
<td>480/481</td>
<td>Floating point</td>
</tr>
<tr>
<td>484/485</td>
<td>Decimal</td>
</tr>
<tr>
<td>496/497</td>
<td>Large Integer</td>
</tr>
<tr>
<td>500/501</td>
<td>Small Integer</td>
</tr>
</tbody>
</table>

**The Data Row Data**

The data row data portion starts at label QW0185DR. It contains actual data mapped according to the table description, with DB2-calculated offsets into the data for each column.

SQL inserts (IN) and SQL deletes (DE) are passed as one row pointed to by HECCDCDA, a single image that contains all the columns in the table.

SQL updates are passed as two rows, an after-image (UA) pointed to by HECCDCDA, and a before-image (UB) pointed to by HECCDCDB. Both images contain all the columns of the table.

As applicable, the rules of the external form of a table description dictate how the following data items are handled:

- A string of fields, ordered as they were specified in the external form of a table description of the table, and in standard SQL external format.
- EDITPROCs and FIELDPROCs are called as in standard SQL. The returned data is as decoded by an EDITPROC or any FIELDPROCs that apply, the same as standard SQL.
- DBCS data is supported as in standard SQL.
- VARCHARs are padded to maximum length, but they contain the actual length in the first two bytes of the data.
Nulls are represented by an indicator variable (two bytes) that precedes the field, but this field is not included in the length.

Exit Routine Processing

Using the information in the control blocks described above (interface control block, XPCB, and XSDCB for HR-propagation, or interface control block, HEC, data description and data row for RH-propagation), you can propagate the changed data segment (pointed to by the XSDDB) or DB2 row (pointed to by the data row) in any way you choose. This section describes considerations for developing your Propagation exit routine.

Calling Your Exit Routine

DPROP loads your Propagation exit routine before its first call, and keeps it in virtual storage until the OS/VS task terminates. In MPP regions, this spans multiple MPP executions. Before calling your exit routine, the RUP or HUP reads the Propagation interface control block, checks the propagation status, and traces the changed IMS data or DB2 data.

DPROP uses standard OS/VS conventions when calling your exit routine.

Register 1 Points to the parameter list described above.
Register 13 Contains the address of a register save area.
Register 14 Contains the return address.
Register 15 Contains the entry point address of the exit routine.

Upon entering the exit routine, the register contents must be saved into the caller's save area. If your exit routine calls other routines that use standard MVS linkage conventions, it must also provide a save area of its own. The exit routine must return to its caller using normal OS/VS conventions after restoring the registers. A return code must be provided in the interface control block, not in register 15. Also, like the other exit routines, your Propagation exit routine gains control in AMODE 31, and must return control in AMODE 31.

For HR-propagation, Propagation exit routines can be called multiple times during one IMS call if the call updates more than one segment type, or if multiple PRs exist for one segment type. The number of calls, and the order in which they are made, depends on these conditions and the type of IMS update being made.

- During processing of an updating IMS call, IMS calls the RUP once for each occurrence of a modified segment type. For ISRT and REPL operations, the call sequence is top-down. For DLET operations, the call sequence is usually bottom-up. Refer to IMS/ESA Application Programming: DL/I Calls for more information on the call sequence.

- During one call, the RUP needs to process multiple PRs propagating the modified segment occurrence. The RUP processes the PRs sequentially.
  - The RUP calls a Propagation exit routine for each one of the following active PRs belonging to a user mapping case.
    1. If defining PRs with DataRefresher, for each PR identifying the modified segment type in the PROPSEGD keyword. The PROPSEGM keyword is part of the MAPUPARM keyword of the DataRefresher UIM SUBMIT control statement.
    2. If defining PRs in the MVG input tables, for each PR having a DPRISEG row identifying the modified segment type.
For details on defining a PR, see “Telling DPROP About Your Propagation Exit” on page 186.

- The RUP also processes each active PR belonging to a generalized mapping case that identifies the modified segment occurrence as an entity segment or as an extension segment.

For RH-propagation, Propagation exit routines can be called multiple times

- If you have multiple PRs propagating the same table, or
- During the processing of an SQL statement, if the statement updates or deletes more than one row.

The number of calls, and the order in which they are made, depends on the DB2 process sequence of the rows and is unpredictable for DPROP and the Propagation exit routine.

**Exit Routine Logic**

Your exit routine must supply all the mapping logic, SQL statements, and IMS calls necessary for propagating the changed data to DB2 or IMS. For performance reasons, it is recommended that your exit routine generate static SQL calls. Avoid using functions that have a detrimental effect on the performance of the propagating program (such as performing an OPEN and CLOSE on an MVS file each time the exit routine is called). It is also recommended that the Database Request Modules (DBRMs) of your Propagation exits be package bound. The DB2 plans created for the propagating application programs must then list the packages.

You can also propagate data changes to more than one DB2 table or IMS database. For more information, see “Propagating Data To More Than One DB2 Table” on page 188.

Because the exit routine for synchronous propagation runs in the same environment as the propagating application program, it can generate the same type of IMS calls and SQL statements that the application program can. For LOG-ASYNC and user asynchronous propagation using the TSO Attach or CAF Attach, the exit routines do not execute in an IMS environment, and cannot generate IMS calls. For asynchronous propagation, therefore, create only SQL statements.

If the exit generates SQL statements, then the DBRM of your Propagation exit routine must be included in the DB2 plans of those application programs which synchronously propagate the changed data. For both LOG-ASYNC and user asynchronous propagation, the DBRM must be included in the DB2 plan of the receiver program.

For RH-propagation, your exit probably generates IMS calls. Use the AIB interface described in *IMS/ESA Application Programming: DL/I Calls*, which allows your exit routine to generate calls without the address of the IMS PCBs.

During synchronous propagation, any changes you make to propagated data from within your exit routine are not propagated.

A Propagation exit routine must not perform functions that are not supported by the environment in which it is running. For example, an exit routine running in an MPP
region must not write to OS files, and the exit routine must not generate STIMER macros in an IMS environment.

It is recommended that you code and link-edit your program as reentrant. To simplify programming, DPROP provides a work space to your exit routine in the interface control block.

Return Codes and Error Handling Techniques

This section discusses how to return from your exit routine to DPROP, including return codes and a brief description of error handling techniques. For more information on how the RUP and HUP handle error situations, see the appropriate Administrators Guide for your propagation mode. First, though, remember that you must return control to the caller in AMODE 31, using the normal MVS conventions described in the previous section.

Return Codes

Below is a list of the return codes you can use when returning from your exit routine, including detailed descriptions of their meanings. The code must be returned in the PICXRETC field of the interface control block.

0  Used for normal returns.

4  Your exit routine must set return code of 4 when it encounters an SQL error code that it considers a propagation failure. If the SQL error code it encounters is considered a normal situation (not a propagation failure), your exit routine must use return code 0.

DPROP assumes that the SQLCA (located in the Propagation interface control block) was used to generate the last SQL statement, and that the last SQL statement was the one that failed. DB2 stores the type of SQL error in the SQLCA. DPROP then reads the SQLCA and, based on which type of error is indicated, proceeds with its usual error handling techniques. DPROP also uses the information in the SQLCA to write an error message describing the details of the error.

8  Your exit must set return code 8 when it encounters an IMS call error that it considers a propagation failure. If the IMS status code it encounters is considered a normal situation (not a propagation failure), your exit routine must use return code 0.

DPROP assumes that the AIB (located in the Propagation Interface Control Block) was used to generate the last IMS call, and that the last IMS call was the one that failed. IMS stores the status code in the failing PCB pointed to by the AIBRSA1 field of the AIB control block. DPROP then reads the AIB and PCB and, based on which type of error is indicated, proceeds with its usual error handling techniques. DPROP also uses the information in the AIB and PCB to write an error message describing the details of the error.

12 Your exit routine must set return code 12 if it encounters a propagation failure error that is not caused by an SQL error or IMS call error, and that DPROP considers as an unavailable resource problem. DPROP then executes its usual error handling techniques for unavailable resources.

16 This return code must be used for propagation failures that are not caused by an SQL error, an IMS call error, or an unavailable resource problem. DPROP again uses its usual error handling techniques for problems other than unavailable resources.
Your exit routine must set this return code if there is a severe error for which you want DPROP to ABEND, even if ERROPT=IGNORE is in effect.

Generating ABENDs from an exit routine is not recommended. Doing this results in loss of flexibility of DPROP's error handling techniques.

### Error Handling Techniques

When you encounter an error in your exit routine, it is strongly recommended that your exit routine take advantage of DPROP’s standard error handling logic. In the interface control block, you can supply a return code in PICXRETC, and an error message in PICXMESG. You must not return an error message in PICXMESG without providing an error return code, because this creates too many console messages.

By supplying DPROP with an error return code and message, you gain many advantages. When an exit returns with an error return code, DPROP traces or snaps the control blocks involved in the interface, and the data. The exits are included in DPROP’s standardized error handling techniques; they can differentiate between ERROPT=BACKOUT and ERROPT=IGNORE, and respond based on the type of error encountered; they protect against excessive console messages. DPROP writes your error message using its standard message writing logic: WTO, trace data set (the IMS log, the //EKYLOG data set, or the //EKYTRACE data set), and audit trail.

If the exit routine generates its own messages or ABENDs, DPROP cannot include the exit routine in its standardized error handling, and cannot guard against excessive console messages. Therefore, it is not recommended that your exit routine generate its own messages or ABENDs when an error occurs.

### Saving Information Across Calls

You can save information across calls to the exit routine. Save it in the PICSWORK field of the interface control block. If PICSWORK is not large enough, generate a GETMAIN and save the address of the storage in PICSWORK.

### Updating Your Propagation Exit Routine

DPROP does not provide any online change logic to replace an existing load module copy of your exit routine with a new version of the load module. If you need to change your exit routine, stop the affected IMS regions and any asynchronous receiver programs before performing the change. A change of the exit routine without stopping the IMS regions and receiver programs causes unpredictable results. For example, some MPP regions use the new version of the exit routine, while other regions use the old version. After the change, you can restart the IMS regions.

### Tracing Your Exit Routine

DPROP provides a trace facility that can assist you in detecting errors in your exit routines. DPROP creates trace output when it encounters propagation failures and when the user activates the trace facility.

You can activate the DPROP trace facility by providing a TRACE control statement in the //EKYIN data set of the job step where your exit routine runs. For synchronous propagation, you can also activate tracing by calling the SCU with a TRACE ON control statement.
If you include debug level 2 on the TRACE or TRACE ON statements, the trace output includes, for HR-propagation, the changed IMS segment, and, for RH-propagation, the changed DB2 row. Also, the PICDBLV2 bit of the interface control block is on when the exit routine is entered. When this bit is on, it is recommended that your exit routine also trace the propagating SQL statements for HR-propagation, or the propagating IMS calls for RH-propagation. See the appropriate Administrators Guide for your propagation mode for details on how to call the DPROP trace module directly from your exit routine.

If you include debug level 4 on the TRACE or TRACE ON statements, each time the exit routine returns to DPROP, the trace output includes:

For HR-propagation:
- The contents of the interface control block
- The XPCB and XSDBs
- The before replace image of changed segments
- The path data for the changed segment (if provided by the caller of the RUP)

For RH-propagation:
- The contents of the interface control block
- The HEC, QWHS, and QWHC
- The Data Capture Data Description
- The Data Capture Data area for the before- and after-image of the row.

If you include debug level 8 on the TRACE or TRACE ON statements, the trace output includes a record of each call to and each return from an exit routine.

Other useful debugging aids are the exit entered and exit in control flags in the interface control block. These flags help you determine if your exit routine is in control at the time of a failure.

---

Telling DPROP About Your Propagation Exit

This section describes how you can inform DPROP that you want to use a Propagation exit routine. During PR definition, specify which Propagation exit routines must be called when changes are made to specific IMS segment types or DB2 tables. The process you follow depends on whether or not you are creating your PRs using DataRefresher.

Creating a PR Using DataRefresher

Defining a PR that uses a Propagation exit routine is much the same as defining a PR used with the generalized mapping cases. The most significant difference is that, on the MAPUPARM operand of the DataRefresher SUBMIT statement, you must:
- Specify the PRTYPE parameter as PRTYPE=U.
- Give the load module name of the exit routine on the EXITNAME= parameter.
- Identify the list of the segment types propagated by the PR on the PROPSEGM= keyword.

This tells DPROP that you want to use a Propagation exit routine, which exit routine must be called, and which segment types and table are propagated.
For HR-propagation, one segment type is usually propagated by only one PR. However, one segment type can be propagated by multiple PRs, belonging to generalized and user mapping cases. If the segment type is specified on the PROPSEGM= keyword of more than one PR, the RUP calls your exit routine once for each associated PR.

For RH-propagation, one table is usually propagated by only one PR. However, one table can be propagated by multiple PRs, but they must all belong to user mapping cases.

Creating a PR Using the MVG Input Tables

This section discusses how to define a PR for a Propagation exit using the MVG Input Tables. The input into the tables is similar to that used for the generalized mapping cases. When specifying a Propagation exit routine, your PR must have at least one row in the PR table, one row in the DPRISEG (or SEG) table, and one row in the DPRITAB (or TAB) table.

In the PR table, you must specify the PRTYPE column as U. Also, specify the load module name of the exit routine using the EXITNAME column. When you define the PR for your exit routine, leave the MAPCASE column blank. The PROPSUP column is ignored.

For HR-propagation, you must include in the SEG table one row for each segment type that, when changed, is propagated by the Propagation exit routine associated with the PR being defined. When one of these segments is changed, the RUP calls the exit routine to propagate the segment.

Typically, one segment type is propagated by only one PR, and only one PR has a SEG row for that segment type. However, one segment type can be propagated by multiple PRs that belong to generalized and user mapping cases. If the segment type is specified on the SEG row of more than one PR, the RUP calls your exit routine once for each associated PR.

For RH-propagation, include in the TAB table one row for each table that, when changed, is propagated by the Propagation exit routine associated with the PR being defined.

Typically, one table is propagated by only one PR, and only one PR has a TAB row for that table. However, one table can be propagated by multiple PRs, belonging to user mapping cases. If the table is specified on the TAB row of more than one PR, the HUP calls your exit routine once for each associated PR.

The SEGEXIT, SEGEXITL, and SEGEXITF columns of the SEG row do not apply to user mapping cases, and are ignored; but they are copied to the SEG mapping table. Also, DPROP ignores the ROLE column, but still must be set to a value (P, E, or X) or blank.

In the TAB table, the columns are the same as those for the generalized mapping cases. Also, DPROP performs the same checks. The only difference is that, for a user mapping, you can specify more than one row in the table. For more information about multiple DB2 tables, see the next section.
You can also use the DPRIFLD (or FLD) table to provide information on the fields to be propagated to DB2. However, DPROP does not use the information in this table, and you are not required to provide it.

**Propagating Data To More Than One DB2 Table**

Using a Propagation exit routine, you can propagate your changed IMS data to more than one DB2 table. The SQL calls involved are created by you, but you must let DPROP know that more than one table is involved. You can only do this through the MVG input tables. To inform DPROP that you want to use more than one DB2 table, add one row in the MVG TAB table for each DB2 table that receives the data changes.

You can define PRs that propagate to multiple tables if you are defining them with the MVG input tables, but not with DataRefresher. However, with DataRefresher, you can define multiple PRs, each propagating the same data to another target DB2 table.

**Propagating Data To More Than One IMS Segment**

Using a Propagation exit routine, you can propagate your changed DB2 data to more than one IMS segment. The IMS calls involved are created by you, but you must let DPROP know that more than one database or segment is involved. You can do this using either DataRefresher or the MVG input tables. When using DataRefresher, you must use one DataRefresher SEGMENT statement for each segment to which you want to propagate the PR. If you use the MVG input tables, then add one row in the MVG SEG table for each IMS segment that receives the data changes.

**Binding the PR**

Use the name of the propagation exit as the member name when binding the PR.

---

**First Sample Propagation Exit Routine**

Figure 52 on page 190 shows the first example of a Propagation exit routine for HR-propagation only. This example shows you the basic principles for mapping a data change involving path data, although this is already supported by the generalized mapping case capabilities of DPROP Version 1 Release 2. The purpose of this sample exit is to illustrate typical aspects of the logic that a Propagation user exit needs to provide and how to call the DPROP trace module within such an exit routine.

In this case, the sample exit is mapping fields from an entity segment, and nonkey path data located in the segment's parent, to the target DB2 table.

Because this kind of mapping is supported by the DataRefresher mapping logic, the data extract in this case can be performed by DataRefresher.

**Mapping Performed By the Sample Exit Routine**

Figure 50 on page 189 illustrates the overview of the propagation done on IMS fields by the sample Propagation exit routine.
Figure 50. Overview of the Propagation Performed By the Exit Routine

Figure 51 shows the mapping of individual IMS source fields to the DB2 target columns.

<table>
<thead>
<tr>
<th>Segment Name</th>
<th>Field Name</th>
<th>Key attribute</th>
<th>Column Name</th>
<th>Column Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG1</td>
<td>SEG1KEY1</td>
<td>Key field</td>
<td>TAB2COL1</td>
<td>Part of primary Key</td>
</tr>
<tr>
<td>SEG1</td>
<td>SEG1DAT1</td>
<td></td>
<td>TAB2COL6</td>
<td></td>
</tr>
<tr>
<td>SEG1</td>
<td>SEG1DAT2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEG1</td>
<td>SEG1DAT3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEG2</td>
<td>SEG2KEY1</td>
<td>Key subfield</td>
<td>TAB2COL2</td>
<td>Part of primary Key</td>
</tr>
<tr>
<td>SEG2</td>
<td>SEG2KEY2</td>
<td>Key subfield</td>
<td>TAB2COL3</td>
<td>Part of primary Key</td>
</tr>
<tr>
<td>SEG2</td>
<td>SEG2DAT1</td>
<td></td>
<td>TAB2COL4</td>
<td></td>
</tr>
<tr>
<td>SEG2</td>
<td>SEG1DAT2</td>
<td></td>
<td>TAB2COL5</td>
<td></td>
</tr>
</tbody>
</table>

Sample Exit Routine Source Code

The example in Figure 52 on page 190 is intentionally simplified to emphasize the fundamental logic involved. Your Propagation exit routine will likely be more complex to meet your propagation requirements.

The source code below is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYEP1A. The following source code shows sample module EKYEP1A after the DB2 precompiler processed it.

Following the source code are definitions related to the sample Propagation exit routine.
**** START OF SPECIFICATIONS ****************************

** MODULE NAME = EKYEPR1A **

** DESCRIPTIVE NAME = SAMPLE 'PROPAGATION USER EXIT ROUTINE' **

** STATUS: V1 R2 MO **

** FUNCTION = EKYEPR1A IS A SAMPLE DPROP ****

** EKYEPR1A ILLUSTRATES TYPICAL ASPECTS OF THE LOGIC THAT ****

** THIS PARTICULAR SAMPLE EXIT ROUTINE PROPAGATES THE ****

** THE DL/I SOURCE FIELDS FOR THE PROPAGATION ARE ****

** NOTE THAT MAPPING INVOLVING 'PATH DATA' IS ****

** THE FIGURE BELOW PROVIDES AN OVERVIEW OF ****

** THE DL/I-TO-DB2 MAPPING PERFORMED BY THIS SAMPLE EXIT. ****

**---------------------------------------------------**

** ' DL/I WORLD ' ' DB2 WORLD ' **

**---------------------------------------------------**

**--------------------- **

**-------------------- **

**-------------------- **

**--------------------- **

Figure 52 (Part 1 of 40). First Sample Propagation Exit Routine (Assembler)
THE PROPAGATION OF A DL/I REPL OF SEG2 RESULTS IN:

A SQL UPDATE STATEMENT FOR THE THREE COLUMNS WHICH ARE NOT PART OF THE PRIMARY DB2 KEY OF TAB2.

Provides the values for the three columns which make up the primary DB2 key of TAB2.

THE PROPAGATION OF A DL/I ISRT OF SEG2 RESULTS IN:

A SQL INSERT STATEMENT OF A ROW INTO TAB2 WITH ALL 6 COLUMNS SHOWN IN THE ABOVE TABLE.

THE PROPAGATION OF A DL/I DLET OF SEG2 RESULTS IN:

A SQL DELETE STATEMENT OF A ROW INTO TAB2.

Provides the values for the three columns which make up the primary DB2 key of TAB2.

DISCLAIMERS:

1) THIS SAMPLE EXIT IS BY PURPOSE VERY SIMPLE, IN ORDER TO AVOID TO OBSCURE THE MOST ESSENTIAL ASPECTS OF THE LOGIC OF A PROPAGATION USER EXIT. IN REAL-LIFE, MOST PROPAGATION USER EXITS WILL BE MORE COMPLEX THAN THIS SAMPLE BECAUSE THEY MIGHT NEED TO PROVIDE LOGIC IN ORDER TO SUPPORT FOR EXAMPLE:

- FIELD FORMAT CONVERSION
- CONVERSION TO A DB2 'NULL' VALUE
- VARIABLE LENGTH SEGMENTS
- DL/I FIELDS HAVING A VARIABLE START POSITION WITHIN THE SEGMENT.

2) NOTE ALSO THAT THIS SAMPLE EXIT DOES NOT PROPAGATE 'PATH DATA' (I.E THE FIELD SEG1DAT1 OF SEGMENT SEG1) TO TAB2.

I.E.: THIS EXIT PROPAGATES THE PATH DATA LOCATED IN SEG1 ONLY WHEN A SEG2 SEGMENT IS BEING UPDATED. THIS EXIT DOES NOT PROPAGATE DATA LOCATED IN SEG1 WHEN A SEG1 SEGMENT IS BEING UPDATED.

IN REAL-LIFE THE USER HAS AT LEAST TWO OPTIONS TO PROPAGATE SUCH CHANGES:

A) HE CAN DEFINE FOR THE PROPAGATION OF SUCH CHANGES ANOTHER PR AND PROVIDE ANOTHER PROPAGATION USER EXIT ROUTINE TO PERFORM THE REQUIRED PROPAGATION.

OR:

B) HE CAN PERFORM THE PROPAGATION OF THESE CHANGES WITH THE SAME PR AND WITH THE SAME PROPAGATION USER EXIT ROUTINE AS THE PROPAGATION OF CHANGES TO SEG2.

HE SHOULD THEN EXPAND THE LOGIC OF EKYEPR1A IN ORDER TO INCLUDE PROPAGATING SQL UPDATE STATEMENTS IN ORDER TO PROPAGATE TO TAB2 DL/I REPL OF SEG1 WHICH RESULTS IN A CHANGE OF SEG1DAT1.

FOR THE PROPAGATION OF REPL AND ISRT OF SEG2, EKYEPR1A NEEDS DL/I DATA STORED IN:

- THE FULLY CONCATENATED KEY OF SEG2
- THE DATA OF SEG2
- THE DATA OF THE PARENT SEGMENT ('PATH DATA').

Figure 52 (Part 2 of 40). First Sample Propagation Exit Routine (Assembler)
FOR THE PROPAGATION OF DLET, EKYEPRIA NEEDS ONLY THOSE DL/I FIELDS WHICH ARE MAPPED TO THE COLUMNS OF THE DB2 PRIMARY KEY. ALL THESE DL/I FIELDS ARE LOCATED IN THE FULLY CONCATENATED KEY OF SEG2.

1) THEREFORE, EXIT= SPECIFICATIONS DURING DBDGEN SHOULD SPECIFY:

```
* EXIT=((EKYRUP00,KEY,PATH,DATA))
```

THESE SPECIFICATIONS ALLOW TO SATISFY THE EKYEPRIA DATA REQUIREMENTS FOR THE PROPAGATION OF REPL, ISRT AND DLET OPERATIONS.

2) ***IF*** THE TARGET DB2 TABLES ARE NOT INVOLVED IN REFERENTIAL INTEGRITY CONSTRAINTS ALLOWING TO USE THE DL/I DBDGEN 'NOCASCADE' OPTION, THEN PROPAGATION OF DL/I DLET REQUIRES THE DBDGEN OPTION OF 'CASCADE' ('CASCADE' IS A DBDGEN DEFAULT OPTION).

THE DL/I DBDGEN CASCADE OPTION:
- MUST SPECIFY (OR DEFAULT TO) THE 'KEY' SUBOPTION (BECAUSE EKYEPRIA NEEDS THE FULLY CONCATENATED KEY OF SEG2 TO PROPAGATE CASCADING DELETES OF SEG2),
- CAN SPECIFY THE 'NODATA' AND 'NOPATH' OPTIONS (BECAUSE EKYEPRIA NEEDS NEITHER SEG2 DATA NOR PATH DATA TO PROPAGATE DELETES OF SEG2).

IT IS ALSO OK TO TAKE THE DEFAULT CASCADE OPTIONS, WHICH ARE:

```
* (CASCADE,KEY,NODATA,NOPATH)
```

DEPENDENCIES ON LINKAGE EDITING

1) EKYEPRIA MUST BE LINK EDITED WITH THE 'RIGHT' DB2 LANGUAGE INTERFACE ROUTINE (DB2 HAS DIFFERENT LANGUAGE INTERFACE ROUTINES FOR EACH UNIQUE LANGUAGE INTERFACE ROUTINES FOR EACH UNIQUE IMS ENVIRONMENTS, ANOTHER FOR TSO ENVIRONMENTS, AND ANOTHER FOR CAF ENVIRONMENTS).

IF USING EKYEPRIA FOR DPROP ASYNCHRONOUS PROPAGATION OR USER ASYNCHRONOUS PROPAGATION USING A CAF ATTACH - THE INSTALLATION MUST LINK EKYEPRIA WITH THE DB2 LANGUAGE INTERFACE FOR THE CAF ATTACH.

IF USING EKYEPRIA FOR ASYNCH PROPAGATION IN AN IMS ENVIRONMENT:

IF USING EKYEPRIA FOR ASYNCH PROPAGATION IN A TSO ATTACH ENVIRONMENT:

Figure 52 (Part 3 of 40). First Sample Propagation Exit Routine (Assembler)
IF USING EKYEPR1A FOR SYNCHRONOUS PROPAGATION:

- THE INSTALLATION MUST LINK EKYEPR1A WITH THE DB2 LANGUAGE INTERFACE FOR THE IMS ATTACH.

2) EKYEPR1A MUST ALSO BE LINK EDITED WITH THE DPROP TRACE MODULE EKYR41X.

RESTRICTIONS = NONE

REGISTER CONVENTIONS=

R13 = ADDRESS OF SAVE AREA
R12 = MODULE BASE REGISTER
R11 = BAS REGISTER TO CALL SUBROUTINE
R10 = ADDRESS OF XPCB
R9 = ADDRESS OF PIC
R8 = ADDRESS OF XDDB
R7 = ADDRESS OF FULLY CONCATENATED KEY
R6 = ADDRESS OF SEGMENT DATA
R5 = ADDRESS OF PATH DATA
R4 = A(SQLDSECT) / A(TRB) / A(TED)

PATCH LABEL = - (NONE)

MODULE TYPE = PROCEDURE
PROCESSOR = ASSEMBLER
MODULE SIZE = APPROXIMATELY 3200 BYTES
ATTRIBUTES = REENTRANT
RMODE = ANY
AMODE = 31

ENTRY POINT = EKYEPR1A
PURPOSE = SEE FUNCTION
LINKAGE = STANDARD OS/VS ASSEMBLER LINKAGE CONVENTIONS.

INPUT : R1 = POINTING TO A STANDARD PARAMETER ADDRESS LIST.
1ST PARAMETER: ADDRESS OF PIC (PIC IS THE EXIT INTERFACE CONTROL BLOCK)
2ND PARAMETER: ADDRESS OF DL/I XPCB

OUTPUT : THE CHANGED DL/I SEGMENT HAS BEEN PROPAGATED

EXIT-NORMAL=
STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.
RETURN CODES = 0

EXIT-ERROR=
STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.
RETURN CODE = 4 : SQL ERROR
20: SEVERE ERRORS

ABEND-CODE OF EKYEPR1A = NONE
ABEND-REASON CODES = NONE

ERROR MESSAGES ISSUED BY EKYEPR1A
EKYEPR0E : PROPAGATION FAILURE FOR TABLE=XXXXXXXX
EKYEPR1E : FAILING SQL STATEMENT=XXXXX SQL ERROR CODE=XXXX
EKYEPR3E : UNEXPECTED DBD- OR SEGNAMe FOR EKYEPR1A
EKYEPR4E : DBDNAME=XXXX SEGNAME=XXXXXX FUNC=XXXX
EKYEPR5E : KEY OF SEG2 NOT PROVIDED BY DL/I CAPTURE
EKYEPR7E : DATA OF SEG2 NOT PROVIDED BY DL/I CAPTURE
EKYEPR8E : PATH DATA NOT PROVIDED BY DL/I CAPTURE
EKYEPR9E : UNEXPECTED CALL FUNCTION IN DL/I XPCB
EKYEPR40E : DBDNAME=XXXX SEGNAME=XXXXXX FUNC=XXXX

Figure 52 (Part 4 of 40). First Sample Propagation Exit Routine (Assembler)
EXTERNAL REFERENCES

ROUTINES: SQL LANGUAGE INTERFACE

DATA AREAS: SEE CONTROL BLOCKS

CONTROL BLOCKS = PIC INTERFACE CB FOR PROPAGATION EXIT

DATA AREAS = PIC INTERFACE CB FOR PROPAGATION EXIT

CONTROL BLOCKS = PIC INTERFACE CB FOR PROPAGATION EXIT

DESCRIPTION

TRB TRACE REQUEST BLOCK

TED TRACE ELEMENT DESCRIPTION

MACROS CODED IN MODULE=

SETTED - SET INFORMATION INTO A TED

MACROS USED FROM MACRO LIBRARY=

SAY - SAVE REGISTERS

GETMAIN - OS/VS GETMAIN

EKYRCPIC - INTERFACE CB FOR PROPAGATION EXIT

EKYRCDL1 - DL/I CAPTURE INTERFACE CONTROL BLOCKS

EKYTRB - TRACE REQUEST BLOCK

EKYTED - TRACE ELEMENT DESCRIPTOR

TABLES= NONE

INCLUDE CODE FROM LIBRARY= NONE

CHANGE ACTIVITY=

KMPS046: SUPPORT OF LOGICAL PARENT SEGMENTS HAVING

A 'LOGICAL' IMS DELETE RULE AND INVOLVED IN A UNIDIRECTIONAL LOGICAL RELATIONSHIP.

END OF SPECIFICATIONS

LOGIC OF EKYEPIA

MAIN LINE LOGIC:

1) MODULE ENTRY LOGIC:

- PROVIDE REGISTER EQUATES
- GENERATE A MODULE SAVEID
- SAVE REGISTERS AND ESTABLISH MODULE-BASE REGISTER
- LOAD ADDRESSES OF CALL PARAMETERS
- SET 'MODULE ENTERED' AND 'MODULE IN CONTROL' FLAGS INTO PIC.
- SET TABLE QUALIFIER AND TABLE NAME INTO PIC
- IF FIRST INVOCATION OF THE EXIT:
  - GETMAIN AN AREA CONTAINING AMONG OTHER
  - A MODULE SAVE AREA AND MODULE WORKSPACE.
  - SAVE ADDRESS OF GETMAINED AREA.
  - CLEAR THE GETMAINED AREA.
  - CHAIN MODULE SAVE AREA AND SAVE AREA OF CALLER.
- CHAIN MODULE SAVE AREA AND SAVE AREA OF CALLER.

2) VERIFY INFORMATION PROVIDED BY DL/I CAPTURE AND/OR DPROP

- VERIFY THAT THE EXIT IS INVOKED TO PROPAGATE THE RIGHT DBD/SEGNAME.
- VERIFY THAT DL/I CAPTURE PROVIDES THE FULLY CONCATENATED KEY OF THE SEGMENT
- FOR ISRT AND REPL OPERATIONS:
- VERIFY THAT DL/I CAPTURE PROVIDES:
  - THE SEGMENT DATA
  - PATH DATA.

3) BRANCH ACCORDING TO TYPE OF DL/I UPDATE OPERATION.

4) FOR A DL/I REPL:

- ISSUE A SQL UPDATE STATEMENT FOR A ROW WITH COLUMNS ORIGINATING FROM:
  - THE DATA PORTION OF SEG2
  - PATH DATA (I.E FROM THE DATA PORTION OF THE PARENT SEGMENT)
  - THE 'WHERE CLAUSE' OF THE UPDATE STATEMENT PROVIDES THE VALUES OF THE DB2 COLUMNS WHICH MAKES UP THE PRIMARY DB2 KEY.

- IF THE SQL UPDATE RESULTS IN AN ERROR OR WARNING:
  - B TO SQLERR ('SQL ERROR LOGIC').
- IF THE SQL UPDATE IS OK:
  - B TO TRACRET ('TRACE AND RETURN TO CALLER')

5) FOR A DL/I ISRT:

- ISSUE A SQL INSERT STATEMENT TO INSERT A ROW WITH COLUMNS ORIGINATING FROM:
  - THE FULLY CONCATENATED KEY OF SEG2
  - THE DATA PORTION OF SEG2
  - PATH DATA (I.E FROM THE DATA PORTION OF THE PARENT SEGMENT)

- IF THE SQL INSERT RESULTS IN AN ERROR OR WARNING:
  - B TO SQLERR ('SQL ERROR LOGIC').
- IF THE SQL INSERT IS OK:
  - B TO TRACRET ('TRACE AND RETURN TO CALLER')

6) FOR A DL/I DLET:

- ISSUE A SQL DELETE STATEMENT TO DELETE THE TARGET ROW.
  - THE 'WHERE CLAUSE' OF THE DELETE STATEMENT PROVIDES THE VALUES OF THE DB2 COLUMNS WHICH MAKES UP THE PRIMARY DB2 KEY.

- IF THE SQL DELETE RESULTS IN A WARNING OR AN ERROR OTHER THAN 'NOT FOUND':
  - B TO SQLERR ('SQL ERROR LOGIC').
- IF THE SQL DELETE RESULTS IN A 'NOT FOUND' AND THE DL/I DELETE WAS NOT A CASCADING DELETE
  - B TO SQLERR ('SQL ERROR LOGIC').

- IF THE SQL DELETE IS OK:
  - OR IF THE SQL DELETE RESULTS IN A 'NOT FOUND' AND THE DL/I DELETE WAS A CASCADING DELETE:
  - B TO TRACRET ('TRACE AND RETURN TO CALLER')

7) RETURN LOGIC

- IF THE USER REQUESTED A TRACING OF THE PROPAGATING SQL STATEMENTS:
Figure 52 (Part 7 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 8 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 9 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 10 of 40). First Sample Propagation Exit Routine (Assembler)
**Figure 52 (Part 1 of 40). First Sample Propagation Exit Routine (Assembler)**

```
000104 D503 A02C CAD4 0002C 00A04 655 CLC XPCBPCALL,=CL4'DLET' IS IT A DLET?
00010A 4780 C360 00360 656 BE DLET ...YES>>B
00010E D503 A02C CAD8 0002C 00A08 657 CLC XPCBPCALL,=CL4'DLPP' PHYSICAL-DELETE-ONLY OF A
000114 4780 C360 00360 658 BE DLET ...LOGICAL PARENT?
000118 4790 C52A 0052A 659 B INVCALL INVALID CALL FUNCTION
661 ******************************************************
662 ******************************************************
664 ****
665 *** DL/I SEGMENT HAS BEEN REPLACED: ***
666 **** THIS RESULTS IN A PROPAGATING 'SQL UPDATE' ***
667 **** OF THE TARGET D82 ROW. ****
668 ****
669 ******************************************************
670 ******************************************************
671 ******************************************************

00011C 673 REPL DS 0H
00011C 9E6B 801C 0001C 674 CLI XSDBPHP,XSDBPHPY RETURN, IF SEG NOT ACCES-
000120 4770 C464 00464 675 BNE RETURN SIBLE VIA PHYSICAL PATH.
000124 D207 D048 CA38 00048 00A38 677 MVC OPER,=CL8'UPDATE' IDENTIFY TYPE OF SQL OPERATION
680 * ISSUE A SQL UPDATE STATEMENT TO UPDATE THE TAB2 ROW *
681 ******************************************************

00012A 4140 D058 00058 683 LA R4,WORKSQL ESTABLISH ADDRESSABILITY
00012A 4140 D058 00058 684 USING SQLDSECT,R4 ...OF SQL DSECT
686 ***$$
687 * EXEC SQL UPDATE C
688 *TAB2 C
689 SET TAB2COL4 = :SEG2DAT1 , C
690 TAB2COL5 = :SEG2DAT2 , C
691 TAB2COL6 = :SEG1DAT1 C
692 WHERE TAB2COL1 = :FKC_SEG1KEY1 AND C
693 TAB2COL2 = :FKC_SEG2KEY1 AND C
694 TAB2COL3 = :FKC_SEG2KEY2

00012E 47F0 C14E 0014E 688 B **32
000132 0028B000001E 689 DC H'40',X'B000',H'30'
00013B E7404040404040 690 DC CLB'X',XLB'14E73CB4030FCAB4',H'1'
00014A 029300EA 691 DC H'659,234'
00014E D217 4004 C132 00004 00132 692 MVC SQLPPLEN(24),,+-28
000154 D203 4028 C14A 00028 0014A 693 MVC SQLSTNUM(4),,+-10
00015A 41F0 9310 00310 694 LA 15,SQLCA
00015E 50F0 401C 0001C 695 ST 15,SQLCODEP
000162 41F0 6008 00008 696 LA 15,SEG2DAT1
000166 50F0 4034 00034 697 ST 15,SQLPVARS+B
00016A D201 4030 CB56 00030 00856 698 MVC SQLPVARS+4(2),,X'01C4'
000170 D201 4032 CB58 00032 00858 699 MVC SQLPVARS+6(2),,H'8'
000176 1FFF 700 SLR 15,15
000178 50F0 403B 0003B 701 ST 15,SQLPVARS+12
00017C 41F0 6010 00010 702 LA 15,SEG2DAT2
000180 50F0 4040 00040 703 ST 15,SQLPVARS+20
000184 D201 403C CB56 0003C 00856 704 MVC SQLPVARS+16(2),,X'01C4'
000188 D201 403E CB58 0003E 00858 705 MVC SQLPVARS+18(2),,H'8'
000190 1FFF 706 SLR 15,15
000192 50F0 4044 00044 707 ST 15,SQLPVARS+24
000196 41F0 5005 00005 708 LA 15,SEG1DAT1
00019A 50F0 404C 0004C 709 ST 15,SQLPVARS+32
00019E D201 4048 CB56 00048 00856 710 MVC SQLPVARS+28(2),,X'01C4'
0001A2 D201 404A CB5A 0004A 0085A 711 MVC SQLPVARS+30(2),,H'7'
0001A6 1FFF 712 SLR 15,15
0001AC 50F0 4050 00050 713 ST 15,SQLPVARS+36
0001B0 41F0 7000 00000 714 LA 15,FCK_SEG1KEY1
```
Figure 52 (Part 12 of 40). First Sample Propagation Exit Routine (Assembler)
```assembly
mzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!t316 1FFF 828 SLR 15,15
mzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!t D2mzqr!z!t1 4mzqr!z!t6E CB5A mzqr!z!tmzqr!z!tmzqr!z!t6E mzqr!z!tmzqr!z!tB5A 827 MVC SQLPVARS+66(2),=H'7'
```

---

**Figure 52 (Part 13 of 40). First Sample Propagation Exit Routine (Assembler)**

---

**202 Customization Guide**
Figure 52 (Part 14 of 40). First Sample Propagation Exit Routine (Assembler)

Chapter 4. Propagation Exit Routines  203
Figure 52 (Part 15 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 16 of 40). First Sample Propagation Exit Routine (Assembler)
0004A2 4E20 D050 00050 0138 CVD R2,DLWL CONVERT SQL CODE TO DECIMAL
0004A6 F321 9115 D056 00115 00056 0139 UNPK MSGSQLC(3),DLWL+6(2) UNPACK SQL CODE
0004AC 96F0 9117 00117 0140 OI MSGSQLC>2,X'FO' FORCE PRINTABLE CHARACTER
0004B0 9240 9114 00114 0141 MVI MSGSQLCS,C' ' PRESET SIGN TO BLANKS
0004B4 1222 0142 LTR R2,R2
0004B6 4780 C4CA 004CA 0143 BZ SQLERR04 B, IF SQLCODE IS ZERO
0004BA 4740 C4C6 004C6 0144 BM SQLERR02 B, IF SQLCODE IS NEGATIVE
0004BE 924E 9114 00114 0145 MVI MSGSQLCS,C'+' SET '+' SIGN
0004C0 47FO C4CA 004CA 0146 B SQLERR04
0004C6 0147 SQLERR02 DS 0H
0004C6 9260 9114 00114 0148 MVI MSGSQLCS,C'-' SET '-' SIGN
0004CA 0149 SQLERR04 DS 0H

0004CA 4140 D118 00118 0156 LA R4,WRKTRB R4=A(TRACE REQUEST BLOCK)
0004CE 92DS 4034 00034 0157 USING TRB,R4
0004D2 4D80 C56C 0056C 0158 MVI TRBSOLI,TRBSOLN SET 'NOT A SOLICITED TRACE'
0004D6 47F0 C464 00464 0159 DROP R4
0004D8 0160 BAS R11,TRACE TRACE THE FAILED SQL STATEMENT
0004DE 0161 B RETURN RETURN TO CALLER

0162 *****************************************************
0163 0164 *****************************************************
0165 *****************************************************
0166 ***** ERRORS OTHER THEN SQL ERRORS:
0167 ***** - BUILD IN THE INTERFACE CONTROL BLOCK AN
0168 ***** ERROR MESSAGE CONTAINING:
0169 ***** - A B-BYTE MESSAGE ID
0170 ***** - A DESCRIPTION OF THE TYPE OF FAILURE
0171 ***** - THE DBDNAME, THE SEGMENT NAME, AND THE TYPE
0172 ***** OF DL/I UPDATE.
0173 ***** - SET A RETURN CODE IN THE INTERFACE CONTROL BLOCK
0174 ***** - RETURN TO CALLER OF THE EXIT
0175 *****
0176 *****
0177 *****************************************************
0178 *****************************************************
0179 *****************************************************

0004DA 0181 INVDBSEG DS 0H
0004DA D207 90A0 CA68 000A0 00A8 0182 MVC MSGOID,=CL8'EKYEPRIE'
0004E0 9240 90A8 000A8 0183 MVC MSGBL1,C''
0004E4 D226 90A9 CB6C 000A9 008C6 0184 MVC MSGTXTK(39),=C'UNEXPECTED DBD- OR SEGNAME FOR EKYEPR1A'
0004EA 47F0 C53E 0053E 0185 B ERRCOM

Figure 52 (Part 17 of 40). First Sample Propagation Exit Routine (Assembler)
Chapter 4. Propagation Exit Routines

Figure 52 (Part 18 of 40). First Sample Propagation Exit Routine (Assembler)
SETTED performs the following:
- It describes in the TED the element to be included in the trace.
- It stores the address of the TED into the call parameter list used to invoke the DPROP tracer.

Setted is invoked in one of the three following ways:
1) For a 'header-TED':
   SETTED NBR=...,TYPE=HEADER,TXT=......
2) For a 'sub-header TED':
   SETTED NBR=...,TYPE=SUBH,TXT=....
3) For a 'data-TED':
   SETTED NBR=...,TYPE=DATA,TXT=......,DATA=.....

The NBR= keyword operand is used to identify the relative number of the TED.

The TXT= keyword operand is used to provide the name of an assembler field containing the descriptive text associated with the TED.

The DATA= keyword operand is used to provide the name of an assembler field containing the data to be included in the trace.

Note that 'Setted' is not a general purpose macro and cannot be used 'as is' in user-programmed exit routines. Instead of using the SETTED macro 'as is' in other exit routines, programmers of the customer may use 'SETTED' as a model, which can help them develop their own macro which is adapted to their requirements.

---

**Figure 52 (Part 19 of 40). First Sample Propagation Exit Routine (Assembler)**

```assembly
1150 * 1151 * SETTED PERFORMS THE FOLLOWING: * 1152 * - IT DESCRIBES IN THE TED THE ELEMENT TO BE INCLUDED * 1153 * IN THE TRACE. * 1154 * - IT STORES THE ADDRESS OF THE TED INTO THE * 1155 * CALL PARAMETER LIST USED TO INVOKE THE DPROP TRACER. * 1156 * TO BE INCLUDED IN THE TRACE OUTPUT. * 1157 * 1158 * SETTED IS INVOKED IN ONE OF THE THREE FOLLOWING WAYS: * 1159 * 1) FOR A 'HEADER-TED': * 1160 * SETTED NBR=...,TYPE=HEADER,TXT=...... * 1161 * 2) FOR A 'SUB-HEADER TED': * 1162 * SETTED NBR=...,TYPE=SUBH,TXT=.... * 1163 * 3) FOR A 'DATA-TED': * 1164 * SETTED NBR=...,TYPE=DATA,TXT=......,DATA=..... * 1165 * 1166 * THE NBR= KEYWORD OPERAND IS USED TO IDENTIFY THE * 1167 * RELATIVE NUMBER OF THE TED. * 1168 * 1169 * THE TXT= KEYWORD OPERAND IS USED TO PROVIDE THE NAME * 1170 * OF AN ASSEMBLER FIELD CONTAINING THE DESCRIPTIVE TEXT * 1171 * ASSOCIATED WITH THE TED. * 1172 * 1173 * THE DATA= KEYWORD OPERAND IS USED TO PROVIDE THE NAME * 1174 * OF AN ASSEMBLER FIELD CONTAINING THE DATA TO BE INCLUDED * 1175 * IN THE TRACE. * 1176 * 1177 * NOTE THAT 'SETTED' IS NOT A GENERAL PURPOSE MACRO * 1178 * AND CAN NOT BE USED 'AS IS' IN USER-PROGRAMMED * 1179 * EXIT ROUTINES. INSTEAD OF USING THE SETTED MACRO 'AS IS' * 1180 * IN OTHER EXIT ROUTINES, PROGRAMMERS OF THE CUSTOMER MAY * 1181 * USE 'SETTED' AS A MODEL, WHICH CAN HELP THEM DEVELOP * 1182 * THEIR OWN MACRO WHICH IS ADAPTED TO THEIR REQUIREMENTS. * 1183 * 1184 *----------------------------------------------------------------------* 1185 *----------------------------------------------------------------------* 1186 **------- START OF SAMPLE 'SETTED' MACRO **-------* 1187 MACRO 1188 &LABEL SETTED &NBR=,&TYPE=,&TXT=,&DATA= 1189 .* 1190 *** GET ADDRESS OF TED AND STORE ITS ADDRESS INTO 1191 *** THE TRACE PARAMETER LIST 1192 .* 1193 AIF ('&NBR EQ 'N').NBROK CHECK THAT NBR= IS NUMERIC 1194 MNOTE 8,'VALUE OF NBR= KEYWORD OPERAND MUST BE NUMERIC' 1195 MEXIT 1196 .NBROK ANOP 1197 &LABEL LA R4,WORKTED+((&NBR-1)*TEDLEN) R4=A(TED) 1198 USING TED,R4 1199 ST R4,TRAPARML+(4*&NBR) SET A(TED) INTO PARMLIST 1200 .* 1201 *** SET EYE CATCHER INTO TED 1202 .* 1203 MVC TDEYE,=CL4'TED' SET EYE CATCHER INTO TED 1204 .* 1205 *** VALIDATE THE VALUE OF THE TYPE= KEYWORD OPERAND 1206 *** AND SET TYPE OF TED. 1207 .* 1208 AIF ('&TYPE' EQ 'HEADER').HDR 1209 AIF ('&TYPE' EQ 'SUBH').SUBH 1210 AIF ('&TYPE' EQ 'DATA').DATA 1211 MNOTE 8,'INVALID OR MISSING VALUE FOR TYPE= OPERAND' 1212 MEXIT 1213 .HDR MVI TEDTYPE,TEDTYPH SET 'THIS IS A HEADER-TED'
```

Figure 52 (Part 19 of 40). First Sample Propagation Exit Routine (Assembler)
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGO .TYPCOM .SUBH MVI TEDTYPE,TEDTYP SET 'THIS IS A SUBHEADER-TED'</td>
<td></td>
</tr>
<tr>
<td>AGO .TYPCOM .DATA MVI TEDTYPE,TEDTYPD SET 'THIS IS A DATA-TED'</td>
<td></td>
</tr>
<tr>
<td>AGO .TYPCOM .TYPCOM ANOP</td>
<td>CHECK THAT TXT= KEYWORD OPERAND HAS BEEN PROVIDED AND SET ADDRESS AND LENGTH OF TEXT INTO TED</td>
</tr>
<tr>
<td></td>
<td>AIF (T'&amp;TXT NE 'O').TXTOK B, IF TXT= NOT OMITTED</td>
</tr>
<tr>
<td></td>
<td>MNOTE B,'TXT= KEYWORD OPERAND VALUE IS MISSING'</td>
</tr>
<tr>
<td></td>
<td>MEXIT</td>
</tr>
<tr>
<td></td>
<td>LA R15,&amp;TXT R15=A(TEXT)</td>
</tr>
<tr>
<td></td>
<td>ST R15,TEDTXT STORE A(TEXT) INTO TED</td>
</tr>
<tr>
<td></td>
<td>MVC TEDTXTL,=A(L'&amp;TXT)</td>
</tr>
<tr>
<td></td>
<td>IF TYPE=DATA:</td>
</tr>
<tr>
<td></td>
<td>AIF ('&amp;TYPE' NE 'DATA').NOTDATA</td>
</tr>
<tr>
<td></td>
<td>AIF (T'&amp;DATA NE 'O').DATAOK B, IF DATA= NOT OMITTED</td>
</tr>
<tr>
<td></td>
<td>MNOTE B,'DATA= KEYWORD OPERAND VALUE IS MISSING'</td>
</tr>
<tr>
<td></td>
<td>MEXIT</td>
</tr>
<tr>
<td></td>
<td>LA R15,&amp;DATA R15=A(DATA)</td>
</tr>
<tr>
<td></td>
<td>ST R15,TEDMA STORE A(DATA) INTO TED</td>
</tr>
<tr>
<td></td>
<td>MVC TEDALEN,=A(L'&amp;DATA)</td>
</tr>
<tr>
<td></td>
<td>MVI TEDALIGN,TEDALIGL REQUEST 'LEFT ALIGNMENT'</td>
</tr>
<tr>
<td></td>
<td>END OF SAMPLE 'SETTED' MACRO</td>
</tr>
<tr>
<td></td>
<td>PROVIDE INFORMATION IN THE TRACE PARAMETER BLOCK (TRB)</td>
</tr>
<tr>
<td></td>
<td>PROVIDE INFORMATION IN A 'HEADER TED'</td>
</tr>
<tr>
<td></td>
<td>PROVIDE A TED FOR THE SQL CODE</td>
</tr>
<tr>
<td></td>
<td>LOGIC COMMON FOR THE TRACING OF INSERT/UPDATE/DELETE SQL STATEMENTS:</td>
</tr>
<tr>
<td></td>
<td>PROVIDE INFORMATION IN THE TRACE PARAMETER BLOCK</td>
</tr>
<tr>
<td></td>
<td>PROVIDE INFORMATION IN A 'HEADER TED'</td>
</tr>
<tr>
<td></td>
<td>PROVIDE A TED FOR THE SQL CODE</td>
</tr>
</tbody>
</table>

**Figure 52 (Part 20 of 40): First Sample Propagation Exit Routine (Assembler)**
Figure 52 (Part 21 of 40). First Sample Propagation Exit Routine (Assembler)
Chapter 4. Propagation Exit Routines

Figure 52 (Part 22 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 23 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 24 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 25 of 40). First Sample Propagation Exit Routine (Assembler)
1607 * SETTED NBR=3,TYPE=SUBL,TXT=TXTWH
1608 LA R4,WORKTED+((3-1)*TEDLEN) R4=A(TED)
1609+ USING TED,R4
1610+ ST R4,TRAPARML+(4mkrtsl6) SET A(TED) INTO PARMLIST
1611+ PROVIDE 4TH TED (DATA FOR 1ST COLUMN IN WHERE CLAUSE)
1612+ SETTED NBR=4,TYPE=DATA,TXT=TXTCOL1|DATA=FCK SEG1KEY1
1613+ LA R4,WORKTED+((4-1)*TEDLEN) R4=A(TED)
1614+ USING TED,R4
1615+ ST R4,TRAPARML+(4mkrtsl5) SET A(TED) INTO PARMLIST
1616+ PROVIDE 5TH TED (DATA FOR 2ND COLUMN IN WHERE CLAUSE)
1617+ SETTED NBR=5,TYPE=DATA,TXT=TXTCOL2|DATA=FCK SEG2KEY1
1618+ LA R4,WORKTED+((5-1)*TEDLEN) R4=A(TED)
1619+ USING TED,R4
1620+ ST R4,TRAPARML+(4mkrtsl4) SET A(TED) INTO PARMLIST
1621+ PROVIDE 6TH TED (DATA FOR 3RD COLUMN IN WHERE CLAUSE)
1622+ SETTED NBR=6,TYPE=DATA,TXT=TXTCOL3|DATA=FCK SEG3KEY2
1623+ LA R4,WORKTED+((6-1)*TEDLEN) R4=A(TED)
1624+ USING TED,R4
1625+ ST R4,TRAPARML+(4mkrtsl3) SET A(TED) INTO PARMLIST

Figure 52 (Part 26 of 40). First Sample Propagation Exit Routine (Assembler)
**Figure 52 (Part 27 of 40). First Sample Propagation Exit Routine (Assembler)**

```assembly
000982 9680 D100 00100 1670 OI TRATED6,X'80' SET VL BIT INTO TRACE PARMLIST
00098B 47F0 C98A 0098A 1671 B TRACEO GO TO COMMON TRACE LOGIC

1673 *----------------------------------------------------------------------------------------------------------------------------------*
1674 * CALL DPROP TRACE MODULE WITH THE PREVIOUSLY FORMATTED PARAMETER LIST. *
1675 *----------------------------------------------------------------------------------------------------------------------------------*

00098A
1678 TRACECO DS

00098A 4110 D0E8 000E8 1680 LA R1,TRAPARM R1=TRACE PARAMETER LIST FOR TRACER
00098E 50F0 C940 00840 1681 L R15.=V(ER4X10D) CALL
0009C2 D0EF 1682 BASR R14,R15 ...DPROP TRACER MODULE

0009C4
1684 BR R11 RETURN TO CALLER OF SUBROUTINE

1686 *----------------------------------------------------------------------------------------------------------------------------------*
1687 * TEXT USED FOR TRACING *
1688 *----------------------------------------------------------------------------------------------------------------------------------*

0009C6 C3D0D3E4D0D5E240 1690 TXTWH DC C' COLUMNS IN WHERE CLAUSE' TEXT FOR TRACE SUBHEADER
0009DD D7D9D671C7C1E3 1691 TXTPRC DC C' PROPAGATED COLUMNS' TEXT FOR TRACE SUBHEADER
0009EF E3C1C2F3C0D603F1 1692 TXTCO1 DC CL8'TAB2COL1' TEXT FOR TRACE
0009FF E3C1C2F3C0D603F2 1693 TXTCO2 DC CL8'TAB2COL2' TEXT FOR TRACE
000A07 E3C1C2F3C0D603F3 1694 TXTCO3 DC CL8'TAB2COL3' TEXT FOR TRACE
000A0F E3C1C2F3C0D603F5 1695 TXTCO4 DC CL8'TAB2COL4' TEXT FOR TRACE
000A17 E3C1C2F3C0D603F6 1696 TXTCO5 DC CL8'TAB2COL5' TEXT FOR TRACE

000A20
1699 LTORG

000A20 4048040404040040 1700 =CL8' ' 
000A2B C4C214040404040 1701 =CL8'DBI' 
000A30 E2C5C7F240D404040 1702 =CL8'SEG2' 
000A38 E4D7C41E3C54040 1703 =CL8'UPDATE' 
000A40 C9D6E2C5D9E34040 1704 =CL8'INSERT' 
000A48 C4C5D3E3C54040 1705 =CL8'DELETE' 
000A50 C5D2E8C5D7D9F0C5 1706 =CL8'EKYEPR1E' 
000A58 40AE00D340C50D9D 1707 =CL16' SQL ERROR CODE=' 
000A68 C5D2E8C5D7D9F1C5 1708 =CL8'EKYEPIE' 
000A70 C5D2E8C5D7D9F2C5 1709 =CL8'EKYEPR2E' 
000A78 D2C5E8406C640E2 1710 =CL8'KEY OF SEG2 NOT PROVIDED BY DL/I CAPTURE'
000A80 C5D2E8C5D7D9F3C5 1711 =CL8'EKYEPR3E' 
000A88 C5D2E8C5D7D9F4C5 1712 =CL8'EKYEPR4E' 
000A90 C5D2E8C5D7D9F5C5 1713 =CL8'EKYEPR5E' 
000A98 C4C2C4D5C1D4C57E 1714 =CL8'DBDNAME=' 
000A9C C5D2E8C5D7D9F1C1 1715 =CL8'EKYEPIA' 
000AC0 C5D2E8C5D7D9F1C1 1716 =CL8'ISRT' 
000ADD D9C5D703 1717 =CL4'REPL' 
000ADD D9C5C9D5 1718 =CL4'REIN' 
000ADD D4C3C5E3 1719 =CL4'DLET' 
000ADD D4C3D707 1720 =CL4'DLPP' 
000ADC 0000004C 1721 =F'76' 
000ADD E0000000D 1722 =XLA'00000000' 
000ADD E0000000B 1723 =V(DSMLH) 
000ADD E4C3D3D7 1724 =C'DLP' 
000ADD E4C3E0052B 1725 =F'40' 
000ADD E000000E4 1726 =F'100' 
000ADD F3C1E2C3 1727 =C'CASC' 
000ADD F3D9C240 1728 =CL4'TRB' 
000ADD F3C5C440 1729 =CL4'TED' 
000ADD F00000035 1730 =A(L'TRHEADER) 
000ADD F00000013 1731 =A(L'TXTSQLC) 
000ADD F00000017 1732 =A(L'TXTMH)

Figure 52 (Part 27 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 28 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 29 of 40). First Sample Propagation Exit Routine (Assembler)
**DESCRIPTION OF THE PARENT SEGMENT 'SEG1' OF 'SEG2'.**

The field SEG1DAT1 is 'PATH DATA' which needs to be propagated together with SEG2 data to the target DB2 table 'TAB2'.

**DESCRIPTION OF THE FULLY CONCATENATED KEY OF THE DL/I SEGMENT 'SEG2'.**

**DESCRIPTION/DECLARATION OF THE 'TAB2' TABLE**

```
EXEC SQL DECLARE TAB2 TABLE C
(TAB2COL1 CHAR(5) NOT NULL ,
TAB2COL2 CHAR(2) NOT NULL ,
TAB2COL3 CHAR(6) NOT NULL ,
TAB2COL4 CHAR(8) NOT NULL ,
TAB2COL5 CHAR(8) NOT NULL ,
TAB2COL6 CHAR(7) NOT NULL )
```
Figure 52 (Part 31 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 32 of 40). First Sample Propagation Exit Routine (Assembler)
2047**-----------------------------------------------*
2048** EXIT WORK AREA  *
2049**  *
2050** THE EXIT WORK AREA CAN BE USED TO SAVE  *
2051** INFORMATION ACROSS CALLS TO THE EXIT (E.G.  *
2052** TO SAVE THE ADDRESSES OF GETMAINED AREAS ACROSS  *
2053** CALLS TO THE EXIT.  *
2054**-----------------------------------------------*

000200  2056+ DS 0D
000200  2057+PICSWORK DC XL256'00' WORK AREA FOR THE EXIT
000300  2058+ DC XL16'00' RESERVED FOR DPROP

2060**-----------------------------------------------*
2061** SQL COMMUNICATION AREA (SQLCA).  *
2062**  *
2063** THE EXIT SHOULD USE THIS SQLCA FOR ITS SQL  *
2064** STATEMENTS.  *
2065**-----------------------------------------------*

000310  2067+SQLCA DS 0D
000310  2068+SQLCID DS CL8 ID
000318  2069+SQLABC DS F BYTE COUNT
00031C  2070+SQLCODE DS F RETURN CODE
000320  2071+SQLERRM DS H,CL70 ERROR MSG PARMS
000368  2072+SQLERRP DS CL8 IMPL DEPENDENT
000370  2073+SQLERRD DS 6F
000388  2074+SQLWARN DS 0C WARNING FLAGS
000388  2075+SQLWARN0 DS C'W' IF ANY
000389  2076+SQLWARN1 DS C'W' = WARNING
00038A  2077+SQLWARN2 DS C'W' = WARNING
00038B  2078+SQLWARN3 DS C'W' = WARNING
00038C  2079+SQLWARN4 DS C'W' = WARNING
00038D  2080+SQLWARN5 DS C'W' = WARNING
00038E  2081+SQLWARN6 DS C'W' = WARNING
00038F  2082+SQLWARN7 DS C'W' = WARNING
000390  2083+SQLEXT DS CL8
000398  2084+ DS 4F RESERVED

2086**-----------------------------------------------*
2087** DLI APPLICATION INTERFACE BLOCK (AIB)  *
2088**  *
2089** THE EXIT SHOULD USE THIS AIB FOR ITS DLI  *
2090** CALL BEFORE FIRST CALL, DPROP INITS  *
2091** AIBID, AIBLEN, AIBRSNM1 AND AIBSFUNC FIELDS.  *
2092**  *
2093**-----------------------------------------------*

0003A0  2095+PICAIB DS 0D AIB INITIALIZED BY DPROP
0003A8  2096+PIC_AIBID DS CLB'DFSAINB' EYECATCHER
0003B0  2097+PIC_AIBLEN DS F DFSAINB ALLOCATED LENGTH
0003B4  2098+PIC_AIBSFUNC DS CLB SUBFUNCTION CODE
0003BC  2099+PIC_AIBRSNM1 DS CLB RESOURCE NAME 1
0003C4  2100+PIC_AIBRSNM2 DS CLB RESOURCE NAME 2
0003CC  2101+ DS 2F RESERVED
0003D4  2102+PIC_AIBOALEN DS F OUTPUT AREA LENGTH (MAX)
0003D8  2103+PIC_AIBOAUSE DS F OUTPUT AREA LENGTH (USED)
0003DC  2104+ DS 2F RESERVED
0003E4  2105+ DS H RESERVED
0003E6  2106+ DS H RESERVED

Figure 52 (Part 33 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 34 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 35 of 40). First Sample Propagation Exit Routine (Assembler)
2244+DBPCB DSECT
2245+DBPCBNAME DS CL8 DBD NAME
2246+DBPCLEVEL DS CL2 LEVEL FEEDBACK
2247+DBPCBLEVEL2 DS CL2 STATUS CODES (RETURNED TO USER)
2248+DBPCBNAME DS CL4 PROCESSING OPTIONS
2249+DBPCBNAME F F PREFIX ADDRESS
2250+DBPCBNAME DS CL8 SEGMENT NAME FEEDBACK
2251+DBPCBNAME DS CL2 STATUS CODES (RETURNED TO USER)
2252+DBPCBNAME DS CL4 PROCESSING OPTIONS
2253+DBPCBNAME F F OR GSAM FEEDBACK AREA
2254+DBPCBNAME DS CL256 KEY FEEDBACK AREA
2255+DBPCBNAME DS F NO OF SENSITIVE SEGMENTS IN PCB
2256+DBPCBNAME EQU mkrtsl DBPCB SIZE OF PCB WITHOUT KEY FEEDBACK AREA
2257+DBPCBNAME DS CL256 KEY FEEDBACK AREA

2258+INQY CALL OUTPUT
2259+INQY CALL OUTPUT
2260+INQY CALL OUTPUT

2261+INQENV N DSECT
2262+INQENV I DS CL8 IMS IDENTIFIER
2263+INQENV R DS CL8 IMS RELEASE LEVEL
2264+INQENV R DS CL8 IMS RELEASE LEVEL

2265+INQENV R DS CL8 IMS RELEASE LEVEL

2266+INQENV R DS CL8 IMS RELEASE LEVEL

2267+INQENV R DS CL8 IMS RELEASE LEVEL

2268+INQENV R DS CL8 IMS RELEASE LEVEL

2269+INQENV R DS CL8 IMS RELEASE LEVEL

2270+INQENV R DS CL8 IMS RELEASE LEVEL

2271+INQENV R DS CL8 IMS RELEASE LEVEL

2272+INQENV R DS CL8 IMS RELEASE LEVEL

2273+INQENV R DS CL8 IMS RELEASE LEVEL

2274+INQENV R DS CL8 IMS RELEASE LEVEL

2275+INQENV R DS CL8 IMS RELEASE LEVEL

2276+INQENV R DS CL8 IMS RELEASE LEVEL

2277+INQENV R DS CL8 IMS RELEASE LEVEL

2278+INQENV R DS CL8 IMS RELEASE LEVEL

2279+INQENV R DS CL8 IMS RELEASE LEVEL

2280+INQENV R DS CL8 IMS RELEASE LEVEL

2281+INQENV R DS CL8 IMS RELEASE LEVEL

2282+INQENV R DS CL8 IMS RELEASE LEVEL

2283+INQENV R DS CL8 IMS RELEASE LEVEL

2284+INQENV R DS CL8 IMS RELEASE LEVEL

2285+INQENV R DS CL8 IMS RELEASE LEVEL

2286+INQENV R DS CL8 IMS RELEASE LEVEL

2287+INQENV R DS CL8 IMS RELEASE LEVEL

2288+INQENV R DS CL8 IMS RELEASE LEVEL

2289+INQENV R DS CL8 IMS RELEASE LEVEL

2290+INQENV R DS CL8 IMS RELEASE LEVEL

2291+INQENV R DS CL8 IMS RELEASE LEVEL

2292+INQENV R DS CL8 IMS RELEASE LEVEL

2293+INQENV R DS CL8 IMS RELEASE LEVEL

2294+INQENV R DS CL8 IMS RELEASE LEVEL

2295+INQENV R DS CL8 IMS RELEASE LEVEL

2296+INQENV R DS CL8 IMS RELEASE LEVEL
**Status Group Indicator:**

- ' ' - NO STATUS GROUP WAS INITIALIZED
- 'A' - INIT STATUS GROUPA CALL WAS ISSUED
- 'B' - INIT STATUS GROUPB CALL WAS ISSUED

**Highest Status Group ID**

- DS CL4

**Address of Recovery Token**

- DS A

**Address of Application PARM**

- DS A

**String Mapped by INQERTS**

- DS A

**Address of Application Parm**

- DS A

**Start of Recovery Token**

- EQU

**Length of Recovery Token**

- DS H

**Start of Application Parm String**

- EQU

**Application Parameter String DSECT**

**Application Interface Block - Application Interface Block**

- THE DFSAIB IS THE APPLICATION INTERFACE BLOCK PASSED TO IMS ON APPLICATION CALLS WHICH USE THE DFSAI B

**Use This Entry Point. Applications Which**

- USE THIS ENTRY POINT ARE EITHER ISSUING CALLS USING A PCB NAME INSTEAD OF A PCB ADDRESS, OR ARE ISSUING CALLS WHICH ARE NOT ASSOCIATED WITH A PCB.

**The DFSAIB Provides a Standard Mechanism for IMS and the Application to Exchange Information.**

**The DFSAIB is Allocated and Initialized by the Application Program. Individual DL/I Calls May Have Different Requirements For Required Input Fields.**

**At a Minimum, the Following Fields Must Be Initialized Prior to Issuing Any DL/I Call.**

- AIBID = CHARACTER STRING 'DFSAIB ' = STORAGE AREA.

**IMS Will Return a Return Code to the Application.**

**As a Reason Code, It May Be Returned as Required By Specific Calls.**

---

Figure 52 (Part 37 of 40). First Sample Propagation Exit Routine (Assembler)
Figure 52 (Part 38 of 40). First Sample Propagation Exit Routine (Assembler)
**Figure 52 (Part 39 of 40). First Sample Propagation Exit Routine (Assembler)**

```
2428** CHANGE ACTIVITY: *
2429** KMP0057 12/13/90 *
2430** *
2431********************** END OF CONTROL BLOCK SPECIFICATION **********************

000000 2435+TRB   DSECT
000000 E390C240 2436+TRBEYE DC C'TRB ' EYE-CATCHER
000004 00000000 2437+TRBPTD DC A(O) ADDRESS OF THE DPROP-PTD CONTROL BLOCK

2439*******
2440********** NAME OF OBJECTS ASSOCIATED WITH THE TRACE
2441**********

000008 400404040404040404040 2443+TRBTABQ DC CL8 ' TABLE-NAME QUALIFIER ASSOC. W. TRACE
000010 400404040404040404040 2444+TRBTABN DC CL18 ' TABLE-NAME ASSOCIATED WITH THE TRACE
000022 4040 2445+ TRBDBN DC CL8 ' DBD-NAME ASSOCIATED WITH THE TRACE
000024 400404040404040404040 2446+TRBDBN DC CL8 ' SEG-NAME ASSOCIATED WITH THE TRACE
00002C 400404040404040404040 2447+TRBDBN DC CL8 ' DBD-NAME ASSOCIATED WITH THE TRACE

2449*******
2450********** SOLICITED/UN SOLICITED INDICATION
2451**********

000034 40 2453+TRBSOLI DC CL1 ' SOLICITED TRACE
000035 000000000000000000 2454+TRBOLY EQU C'Y' ...Y: TRACE SOLICITED BY THE USER
000035 000000000000000000 2455+TRBSOLN EQU C'N' ...N: TRACE NOT SOLICITED BY THE USER

2457+ DC 13X'00' RESERVED/MUST BE ZERO
000040 2458+TRBEND EQU *
000040 2459+TRBLEN EQU **-TRB LENGTH OF ONE TRB
2461 EKYTED , TRACE ELEMENT DESCRIPTION
2462********************** START OF CONTROL BLOCK SPECIFICATION **********************
2463++ *
2464++ CONTROL BLOCK NAME:
2465++ EKYTED (TED)
2466++ *
2467++ DESCRIPTIVE NAME:
2468++ DPROP TRACE ELEMENT DESCRIPTOR (TED)
2469++ = = = *
2470++ *
2471********************** END OF CONTROL BLOCK SPECIFICATION **********************
2472++ *
2473++ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
2474++ *
2476++ ALL RIGHTS RESERVED.
2477++ *
2478++ U.S. GOVERNMENT USERS RESTRICTED RIGHTS -
2479++ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
2480++ GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
2481++ *
2482++ LICENSED MATERIALS - PROPERTY OF IBM.
2483++ *
2484********************** END OF CONTROL BLOCK SPECIFICATION **********************
2485++ *
2486++ STATUS: V1 R2 M0
2487++ *
2488++ FUNCTION:
2489++ WHEN INVOKING THE DPROP TRACE FUNCTION, THE CALLING
2490++ MODULE MUST PROVIDE ONE TED FOR EACH:
2491++ - TRACE-HEADER
2492++ - TRACE-SUBHEADER
2493++ - DATA-AREA
2494++ WHICH SHOULD BE TRACED/SNAPPED.
2495++ *
```
Figure 52 (Part 40 of 40). First Sample Propagation Exit Routine (Assembler)
Definitions For First Sample Propagation Exit

This section contains definitions associated with the first sample Propagation exit routine. It includes the following types of definitions:

- IMS DBDGEN and PSBGEN definitions
- DB2 CREATE TABLE definitions
- DataRefresher definitions required to define the PR DataRefresher and to extract the IMS data with DataRefresher
- SQL statements defining the PR without DataRefresher in the MVG input tables

DBDGEN Definitions

Figure 53 shows a DBDGEN definition for the sample Propagation exit routine in Figure 52 on page 190.

```c
DBD NAME=DB1,VERSION=V123456789,
   ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC4mzqr!z!t,5,4),
   EXIT=(EKYRUP00,KEY,PATH,DATA)
DATASET DD1=HDAM,SIZE=4mzqr!z!t96,DEVICE=338mzqr!z!t
   SEGM NAME=SEG1,PARENT=mzqr!z!t,BYTES=24
   FIELD NAME=(SEG1KEY,SEQ,U),BYTES=5,START=1
*   SEGM NAME=SEG2,PARENT=SEG1,BYTES=24
   FIELD NAME=(SEG2KEY,SEQ,U),BYTES=8,START=1
*   DBDGEN
FINISH
END
```

**Figure 53. DBDGEN Definition**

**Notes:**

1. The EXIT= keyword of the DBD macro specifies that EKYRUP00 (the RUP) be called when a segment of this DBD is changed. This is required for synchronous data propagation with DPROP.

2. The EXIT= keyword of the DBD statement requests the PATH data option. This is required for the mapping performed by this sample Propagation exit routine (because the Propagation exit routine maps nonkey, path data, from the parent segment).

CREATE TABLE Statement

Figure 54 on page 231 shows a CREATE TABLE statement for the sample Propagation exit routine in Figure 52 on page 190.
CREATE TABLE T96606.TAB2
(TAB2COL1 CHAR(5) NOT NULL,
TAB2COL2 CHAR(2) NOT NULL,
TAB2COL3 CHAR(6) NOT NULL,
TAB2COL4 CHAR(8) ,
TAB2COL5 CHAR(8) ,
TAB2COL6 CHAR(8) ,
PRIMARY KEY (TAB2COL1, TAB2COL2, TAB2COL3))
IN DUMO96606.PROPTS;

CREATE UNIQUE INDEX XN01 ON TAB2 (TAB2COL1, TAB2COL2, TAB2COL3)
USING VCAT KOE ;

Figure 54. CREATE TABLE Statement

Using DataRefresher to Define the PR

This section describes how you can use DataRefresher to define the PR for the sample Propagation exit routine in Figure 52 on page 190.

CREATE DXTPSB

Figure 55 shows a CREATE DXTPSB statement for the sample Propagation exit routine in Figure 52 on page 190.

CREATE DXTPSB NAME=KOEPSB

DXTPCB NAME=DB1, DBNAME=DB1, DBACCESS=HDAM

SEGMENT NAME=SEG1, PARENT=mzqr!z!t, BYTES=24

FIELD NAME=SEG1KEY1, START=1 , BYTES=5, SEQFLD=R
FIELD NAME=SEG1DAT1, START=6 , BYTES=7, TYPE=C
FIELD NAME=SEG1DAT2, START=13, BYTES=4, TYPE=C
FIELD NAME=SEG1DAT3, START=17, BYTES=8, TYPE=C

SEGMENT NAME=SEG2, PARENT=SEG1, BYTES=24

FIELD NAME=SEG2KEY1 START=1 , BYTES=8, TYPE=C
FIELD NAME=SEG2KEY2 , START=3 , BYTES=6, TYPE=C
FIELD NAME=SEG2DAT1 , START=9 , BYTES=8, TYPE=C
FIELD NAME=SEG2DAT2 , START=17, BYTES=8, TYPE=C ;

Figure 55. CREATE DXTPSB Statement

The Propagation exit routine does not map the key field of segment SEG2 to one DB2 column. Instead, the key field of SEG2 is mapped as two key subfields to two columns of the DB2 primary key. Therefore, the key field SEG2KEY is redefined by the two key subfields SEG2KEY1 and SEG2KEY2 that overlay SEG2KEY.

CREATE DXTVIEW

Figure 56 on page 232 shows a CREATE DXTVIEW statement for the sample Propagation exit routine in Figure 52 on page 190.
CREATE  DXTVIEW NAME  = VIEW011,  
                DXTPSB = K0EPSB,  
                DXTPCB = DB1,  
                SEGMENT = SEG2,  
                MINSEGM = SEG2,  
                FIELDS = * ;  

Figure 56. CREATE DXTVIEW Statement

DataRefresher UIM SUBMIT Command and EXTRACT Statement

Figure 57 shows a DataRefresher UIM SUBMIT command and EXTRACT statement for the Propagation exit routine in Figure 52 on page 190.

SUBMIT  EXTID=PR001,  
        NODE=NODEX,  
        USERID=T096606,  
        CD=JCS,  
        JCS=DDJCS01,  
        FORMAT=SOURCE,  
        MAPEXIT=EKYMCE00,  
        MAPUPARM='  
                      PRTYPE=U,  
                      MAPDIR=HR,  
                      ACTION=REPL,  
                      ERROPT=BACKOUT,  
                      EXITNAME=EKYEPR1A,  
                      PROPSEGM=(DB1/SEG2)';  

EXTRACT INTO T096606_TAB2 (TAB2COL1 NOT NULL,  
                             TAB2COL2 NOT NULL,  
                             TAB2COL3 NOT NULL,  
                             TAB2COL4,  
                             TAB2COL5,  
                             TAB2COL6)  
SELECT SEG1KEY1,  
       SEG2KEY1,  
       SEG2KEY2,  
       SEG2DAT1,  
       SEG2DAT2,  
       SEG1DAT1  
FROM VIEW011 ;  

Figure 57. DataRefresher UIM SUBMIT Command and EXTRACT Statement

Notes:

1. The MAPEXIT= keyword of the SUBMIT command specifies EKYMCE00. This causes DataRefresher UIM to call the DPROP-provided Map Capture Exit EKYMCE00 during the processing of the SUBMIT or EXTRACT. This is required to allow DPROP to create the PR.

2. PRTYPE=U (user mapping) must be specified, because the PR must be processed by a Propagation exit routine.

3. EXITNAME=EKYEPR1A specifies the name of the Propagation exit routine that performs the propagation for this PR.

4. PROPSEGM=DB1 or SEG2 identifies the segment types being propagated by this PR. As explained in the commentary for the source code of the EKYEPR1A, the sample exit routine propagates changes to the data of SEG2 (together with path data of SEG1). However, the sample exit routine does not propagate changes to the data of SEG1. Therefore, the PROPSEGM= keyword identifies only SEG2 as the segment being propagated.
5. The EXTRACT statement describes to DataRefresher which fields must be mapped to which columns during the data extract. These definitions are important for the extract but are not important for DPROP (because the mapping and propagation is not done by the generalized mapping logic of DPROP).

Using DataRefresher For the Extract

This section covers INITDEM and USE DXTPSB Control Statements. Figure 58 shows INITDEM and USE DXTPSB control statements for the Propagation exit routine in Figure 52 on page 190.

```
INITDEM NAME=BASILEUS;
USE DXTPSB=KOEPSB;
```

Figure 58. Using DataRefresher For the Extract: INITDEM and USE DXTPSB Control Statements

Defining the PR in the MVG Input Tables

Figure 59 on page 234 describes DSNTEP2 SQL statements required to define the PR in the MVG input tables.

The following rows are inserted into the MVG input tables:

- One row is inserted into the DPRIPR table (the PR table).
  
  This row identifies the PRID, indicates that the PRTYPE is U (user mapping), and provides in the EXITNAME column the name of the Propagation exit routine EKYEPR1A that performs the propagation for this PR.

- One row for each segment type being propagated by the PR and the Propagation exit routine is inserted into the DPRISEG table (the SEG table).
  
  As explained in the commentary of the source code of EKYEPR1A, the sample exit routine propagates changes to the data of SEG2 (together with path data of SEG1). However, the sample exit routine does not propagate changes to the data of SEG1. Therefore, only one row is inserted into the DPRISEG table, a row indicating that the PR is propagating SEG2.

- One row is inserted into the DPRITAB table (the TAB table).
  
  This row indicates that the target table is T096606.TAB2.

For PRTYPE=U, DPROP does not require that you insert any rows in the DPRIFLD table; this is why the example below does not insert any row in the DPRITAB table.
DELETE FROM T096606.DPRIPR WHERE PRID = 'PR001' ;

INSERT INTO T096606.DPRIPR
( PRID, USERID, PRTYPE, MAPCASE, MAPDIR, ERROPT, ACTION, EXITNAME )
VALUES ('PR001', 'T096606', 'U', ' ', 'HR', 'BACKOUT', 'REPL', 'EKYEPR1A') ;

INSERT INTO T096606.DPRISEG
( PRID, DBNAME, SEGNAME, ROLE )
VALUES ('PR001', 'DB1', 'SEG2', ' ' ) ;

INSERT INTO T096606.DPRITAB
( PRID, TABQUAL, TABNAME )
VALUES ('PR001', 'T096606', 'TAB2') ;

COMMIT;

Figure 59. DSNTEP2 SQL Statements Required to Define the PR in the MVG Input Tables

Second Sample Propagation Exit Routine

A second example of a Propagation exit routine written in an HLL is shown in Figure 62 on page 236.

This is a key range splitting example: the mapping is provided from two different segment types of two different databases. Both segments have the same structure and the same key construction, but each key is unique over both databases.

The first database contains the lower key range (000000 to 499999), and the second one contains the higher key range (500000 to 999999).

Each segment occurrence is mapped to a specific row of the propagated table.

Mapping Performed by the Sample Exit Routine

Figure 60 illustrates an overview of the propagation done by the sample Propagation exit routine.

Database IMS081

Database IMS082

Figure 61 shows the mapping of individual IMS source fields to the DB2 target columns and vice versa.
### Sample Exit Routine Source Code

The example in Figure 62 on page 236 is intentionally simplified to emphasize the fundamental logic involved. Your Propagation exit routine will likely be more complex to meet your propagation requirements.

The source code below is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYEPR2K. The following source code shows sample module EKYEPR2K after the DB2 precompiler processed it.

Following the source code are definitions related to the sample Propagation exit routine.
The purpose of this program is to provide a sample propagation exit routine. This is a key range splitting example, e.g. the mapping is provided from two different segment types of two different databases. Both segments have the same structure and the same key construction, but each key content is unique over both databases.

The first database contains the lower key range i.e. - "000000" to "499999".

The second one contains the higher key range i.e. - "500000" to "999999".

Each segment occurrence is mapped to one row of the propagated table.

The figure below provides an overview of the IMS-to-DB2 mapping performed by this sample propagation exit.

Figure 62 (Part 1 of 18). Second Sample Propagation Exit Routine (C)
Processing:

- Set "module entered" and "module in control" flags into PIC.
- Check function code to see if the module is called to perform IMS-to-DB2 propagation (HR) or to perform DB2-to-IMS propagation (RH).

Processing for IMS-to-DB2 propagation:

- Provide addressing for "EKYRCDLP".

Figure 62 (Part 2 of 18). Second Sample Propagation Exit Routine (C)
* - Set table qualifier and table name into PIC.
* - Verify information provided by DL/I capture and/or DPROP.
* - Verify that the exit is invoked to propagate the right
  * DBD/segname.
* - For ISRT and REPL operations:
  * Verify that DL/I capture provides the segment data.
* - For DLET operation:
  * Verify that DL/I capture provides the KFBA.
* - Branch according to type of IMS update operation:
* - For an IMS REPL:
  * Issue a SQL update statement for a row with columns
    * originating from SEG1 or SEG2.
  * If the SQL update results in an error or warning execute
    * the SQL error logic.
* - For an IMS ISRT:
  * Issue a SQL insert statement to insert a row with columns
    * originating from SEG1 or SEG2.
  * If the SQL insert results in an error or warning execute
    * the SQL error logic.
* - For an IMS DLET:
  * Issue a SQL delete statement to delete the target row.
  * If the SQL delete results in an error or a warning execute
    * the SQL error logic.
**************************************************************************************************************
#pragma page(1)
**************************************************************************************************************
* SQL error logic:
* - set return code of 4
* - copy the SQLCA used in this module to the "PIC" SQLCA
* - format an error message
* - return to the caller.

* Processing for DB2-to-IMS propagation:
* - provide addressing for "$EKYHCP" and other appropriate control
  * blocks
* - get the column data and move it to the IMS segment work area
* - build the SSA, init the AIB and set the correct function code
  * for the DL/I call
* - perform the following depending on the DB2 operation:

* Figure 62 (Part 3 of 18). Second Sample Propagation Exit Routine (C)
- for an INSERT call:
  - issue an IMS insert with the IMS segment work area
  - if the IMS insert results in an error or warning
    build the error message and set an 8 - return code
  - return to the caller

- for an UPDATE or DELETE call:
  - issue an IMS get hold unique
  - if the GHU results in an error or warning
    build the error message and set an 8 - return code
  - else issue an IMS REPL or DLET depending on the SQL operation
  - if the IMS call results in an error or warning
    build the error message and set an 8 - return code
  - return to the caller

Errors other than SQL errors:
- set return code of 4
- build an error message in the PIC
- return to caller of the exit

End of Logic Description

#include <leawi.h>
#include <ims.h>
#pragma linkage(ekyper2k,fetchable)
#include <stdlib.h>
#include <string.h>

typedef struct
{ short SQLPLLEN;
  short SQLFLAGS;
  short SQLCTYPE;
  char SQLPROGN[8];
  short SOLTIMES[4];
  short SQLSECTN;
  char *SQLCODEP;
  char *SQLVARM;
  char *SQLAPARM;
  short SQLSTNUM;
  short SQLSTYPE;
} SQLPLIST;

Figure 62 (Part 4 of 18). Second Sample Propagation Exit Routine (C)
typedef struct
    { short SQLTYPE;
    short SQLLEN;
    char +SQLADDR;
    char +SQLIND;
} SQLELTS;
typedef SQLELTS *SQLELTS_PTR;
char SQLTEMP[ 19 ];

EXEC SQL DECLARE TABLE
    (TABXCOL1 CHAR(6) NOT NULL,
    TABXCOL2 CHAR(7) NOT NULL,
    TABXCOL3 CHAR(4) NOT NULL,
    TABXCOL4 CHAR(8) NOT NULL)

EXEC SQL INCLUDE SQLCA

#ifndef SQLCODE
struct sqlca
    { unsigned char sqlcaid[8];
    long sqlcabc;
    long sqlcode;
    short sqlerrml;
    unsigned char sqlerrmc[70];
    unsigned char sqlerrp[8];
    long sqlerrd[6];
    unsigned char sqlwarn[11];
    unsigned char sqlstate[5];
    };
#define SQLCODE sqlca.sqlcode
#define SQLWARN0 sqlca.sqlwarn[0]
#define SQLWARN1 sqlca.sqlwarn[1]
#define SQLWARN2 sqlca.sqlwarn[2]
#define SQLWARN3 sqlca.sqlwarn[3]
#define SQLWARN4 sqlca.sqlwarn[4]
#define SQLWARN5 sqlca.sqlwarn[5]
#define SQLWARN6 sqlca.sqlwarn[6]
#define SQLWARN7 sqlca.sqlwarn[7]
#define SQLWARN8 sqlca.sqlwarn[8]
#define SQLWARN9 sqlca.sqlwarn[9]
#define SQLWARN10 sqlca.sqlwarn[10]
#define SQLSTATE sqlca.sqlstate
#endif
struct sqlca sqlca;

#pragma page(1)
/**************
 * Declare Host variables
 *************/

EXEC SQL BEGIN DECLARE SECTION

/*****

Figure 62 (Part 5 of 18). Second Sample Propagation Exit Routine (C)
Figure 62 (Part 6 of 18). Second Sample Propagation Exit Routine (C)
void lablnf (EKYRCPIC *);

#pragma page(1)
/*******************************************************************************/
 /* Declare global variables */
******************************************************************************/

long int x1, x2, x3, x4, ncount;
cchar opcode[6], wsqlcode[6], funccode[4],
    w7ckey[6];

PIXML1 msg11 = {"","","Propagation failure for table=\0"},
    msg21 = {"","",""},
    msg31 = {"","","Propagation failure for segment=\0"};

struct {
    cchar msgstxt2[22];
cchar msgstxt3[16];
cchar msgssqlc[4];
    } msg12 = {"Failing SQL statement="},
            {""},{" SQL error code="},{""};

struct {
    cchar msgotxt2[22];
cchar msgodbd[8];
cchar msgoseg[8];
cchar msgofunc[4];
    } msg22 = {"DBDNAME="},
            {""},{" SEGNAME="},{" "},{" FUNC="},{""};

struct {
    cchar msgitxt2[22];
cchar msgiarea[25];
    } msg32 = {"Failing IMS statement="};

char ssaisrt[9];

struct {
    cchar ssasegm[8],
        filler_1,
        ssakeynm[8],
        filler_2,
        ssavalue[6],
        filler_3;
    } ssa = {""},{"="},{"="},{""},
char segiarea[25];
char segoarea[25];

#pragma page(1)
/*******************************************************************************/
 /*Main function - ekyepr2k */
******************************************************************************/
void ekyepr2k (EKYRCPIC *ekyrcpic, void *secondcb)
{ /* XPCB *xpcb; */

Figure 62 (Part 7 of 18). Second Sample Propagation Exit Routine (C)
/* Set control flags - exit entered and in control */
      ekyrcpic->picentrd = 'X';
      ekyrcpic->picinctl = 'X';

/***************************************************************************/
* Check function code to determine if module is called
* to perform IMS-to-DB2 propagation (HR) or
* to perform DB2-to-IMS propagation (RH).
******************************************************************************/
      if (strncmp(ekyrcpic->piccall, "HR", 2) == mzqr!z!t)
        imstodbd2(ekyrcpic, secondcb);
      else
        if (strncmp(ekyrcpic->piccall, "RH", 2) == mzqr!z!t)
          db2toids(ekyrcpic, secondcb);
        else
          invdir(ekyrcpic); /* invalid propagation direction */
          return;
    } /* end of ekyepr2k */

#pragma page(1)
/***************************************************************************/
* R U P i s t h e c a l l e r
* M a i n IMS_to_DB2 processing
******************************************************************************/
void imstodbd2 (EKYRCPIC *ekyrcpic, XPCB *xpcb)
{
    strncpy(ekyrcpic->pictabq, "", 8);
    strncpy(ekyrcpic->pictabn, "TABX", 18);
    if (strncmp(xpcb->xpcbdbd, "DIVNTZ", 8) == mzqr!z!t)
        if (strncmp(xpcb->xpcbseg, "SEG1", 8) == mzqr!z!t)
            segok(ekyrcpic, xpcb);
        else
            invseg(ekyrcpic, xpcb); /* unexpected segment name */
    else
        if (strncmp(xpcb->xpcbdbd, "DHVNTZ", 8) == mzqr!z!t)
            if (strncmp(xpcb->xpcbseg, "SEG2", 8) == mzqr!z!t)
                segok(ekyrcpic, xpcb);
            else
                invseg(ekyrcpic, xpcb); /* unexpected segment name */
        else
            invseg(ekyrcpic, xpcb); /* unexpected DBD */
    return;
} /* end of imstodbd2 */

#pragma page(1)

void segok(EKYRCPIC *ekyrcpic, XPCB *xpcb)
{
    XSDB *xsdb = NULL;
    SEGI *segi = NULL;

Figure 62 (Part 8 of 18). Second Sample Propagation Exit Routine (C)
if (strncmp(xpcb->xpcbcall, "DLET", 4) == 0)
{
    strncpy(opcode, "DELETE", 6);
    db2dlet(ekyrcpic,xpcb);
}
else

    verify that the segment data is provided for "REPL" and "ISRT". 
    process according to the type of IMS update.

    if (xpcb->xpcbxsbd == NULL)
        datmis(ekyrcpic); /* data is missing (EKYEPR5E) */
    else xsdb = xpcb->xpcbxsbd;
    if (xsdb->xsdbsega == NULL)
        datmis(ekyrcpic); /* data is missing (EKYEPR5E) */
    else segi = xsdb->xsdbsega;
    if (strncmp(xpcb->xpcbcall, "REPL", 4) == 0)
    {
        strncpy(opcode, "UPDATE", 6);
        db2repl(ekyrcpic,xpcb,segi);
    }
    else
        if (strncmp(xpcb->xpcbcall, "ISRT", 4) == 0)
    {
        strncpy(opcode, "INSERT", 6);
        db2isrt(ekyrcpic,segi);
    }
    else
        invcal(ekyrcpic, xpcb);
}
return;
/* end of segok */

#pragma page(1)

/* IMS segment has been replaced. This results in a propagating SQL UPDATE of the target DB2 row. */

void db2repl(EKYRCPIC *ekyrcpic, XPCB *xpcb, SEGI *segi)
{
    strncpy(SEGIKEY, segi->segikey, 6);
    strncpy(SEGIDAT1, segi->segidat1, 7);
    strncpy(SEGIDAT2, segi->segidat2, 4);
    strncpy(SEGIDAT3, segi->segidat3, 8);

    EXEC SQL UPDATE TABX
        SET TABXCOL2 = :SEGIDAT1,
            TABXCOL3 = :SEGIDAT2,
            TABXCOL4 = :SEGIDAT3
    WHERE TABXCOL1 = :SEGIKEY

}$**$

Figure 62 (Part 9 of 18). Second Sample Propagation Exit Routine (C)
{SQLPLIST SQLPLIST2 =
 {40, -32768, 30, "EKYEP2ZK", 0, 0, 0, 0, 1, 0, 0, 0, 463, 234};
 SQLELTS_PTR SQLELTS_PTR2;
 struct
  { long SQLPVARS;
    char SQLPVELT[(sizeof(SQLELTS) - 4)];
  } SQLPVARS2;
 SQLELTS_PTR2 = (SQLELTS *) &SQLPVARS2.SQLPVELT;
 SQLELTS_PTR2->SQLTYPE = 460;
 SQLELTS_PTR2->SQLLEN = 8;
 SQLELTS_PTR2->SQLADDR = (char *) &SEGIDAT1);
 SQLELTS_PTR2->SQLIND = NULL;
 SQLELTS_PTR2 = SQLELTS_PTR2 + 1;
 SQLELTS_PTR2->SQLTYPE = 460;
 SQLELTS_PTR2->SQLLEN = 5;
 SQLELTS_PTR2->SQLADDR = (char *) &SEGIDAT2);
 SQLELTS_PTR2->SQLIND = NULL;
 SQLELTS_PTR2 = SQLELTS_PTR2 + 1;
 SQLELTS_PTR2->SQLTYPE = 460;
 SQLELTS_PTR2->SQLLEN = 9;
 SQLELTS_PTR2->SQLADDR = (char *) &SEGIDAT3);
 SQLELTS_PTR2->SQLIND = NULL;
 SQLELTS_PTR2 = SQLELTS_PTR2 + 1;
 SQLELTS_PTR2->SQLTYPE = 460;
 SQLELTS_PTR2->SQLLEN = 7;
 SQLELTS_PTR2->SQLADDR = (char *) &SEGIKEY);
 SQLELTS_PTR2->SQLIND = NULL;
 SQLPVARS2.SQLPVARS = 52;
 SQLPLIST2.SQLVPARM = (char *) &SQLPVARS2.SQLPVELT;
 SQLPLIST2.SQLCODEP = (char *) &sqlca;
 SQLPLIST2.SQLTIMES[0] = 0x14EA;
 SQLPLIST2.SQLTIMES[1] = 0x9521;
 SQLPLIST2.SQLTIMES[2] = 0x0570;
 SQLPLIST2.SQLTIMES[3] = 0x64A0;
 DSNHLI ( (unsigned int ) &SQLPLIST2);
}

db2check(ekyrcpic);
return;

} /* end of db2repl */

#pragma page(1)

void db2isrt(EKYRCIC *ekyrcpic, SEGI *segi) {
  strncpy( SEGIKEY, segi->segikey, 6);
  strncpy( SEGIDAT1, segi->segidat1, 7);
  strncpy( SEGIDAT2, segi->segidat2, 4);
}

Figure 62 (Part 10 of 18): Second Sample Propagation Exit Routine (C)
```c
strncpy( SEGIDAT3, segi->segidat3, 8);

/***$$
EXEC SQL INSERT INTO TABX
    (TABXCOL1,
     TABXCOL2,
     TABXCOL3,
     TABXCOL4)
VALUES
    (:SEGIKEY,
     :SEGIDAT1,
     :SEGIDAT2,
     :SEGIDAT3)
$$***/
{
SQLPLIST SQLPLIST3 =
    {4mzqr!z!t, -32768, 3mzqr!z!t, "EKYEPR2K", mzqr!z!t, mzqr!z!t, mzqr!z!t ,mzqr!z!t,
     2, mzqr!z!t, mzqr!z!t, mzqr!z!t, 486, 232};
SQLELTS_PTR SQLELTS_PTR3;
struct
    {
    long SQLPVARS;
    char SQLPVELT[(sizeof(SQLELTS) - 4)];
    } SQLPVARS3;
SQLELTS_PTR3 = (SQLELTS *) &SQLPVARS3.SQLPVELT;
SQLELTS_PTR3->SQLTYPE = 46mzqr!z!t;
SQLELTS_PTR3->SQLLEN = 7;
SQLELTS_PTR3->SQLADDR = (char *) &SEGIKEY);
SQLELTS_PTR3->SQLIND = NULL;
SQLELTS_PTR3 = SQLELTS_PTR3 + 1;
SQLELTS_PTR3->SQLTYPE = 460;
SQLELTS_PTR3->SQLLEN = 8;
SQLELTS_PTR3->SQLADDR = (char *) &SEGIDAT1);
SQLELTS_PTR3->SQLIND = NULL;
SQLELTS_PTR3 = SQLELTS_PTR3 + 1;
SQLELTS_PTR3->SQLTYPE = 460;
SQLELTS_PTR3->SQLLEN = 5;
SQLELTS_PTR3->SQLADDR = (char *) &SEGIDAT2);
SQLELTS_PTR3->SQLIND = NULL;
SQLELTS_PTR3 = SQLELTS_PTR3 + 1;
SQLELTS_PTR3->SQLTYPE = 460;
SQLELTS_PTR3->SQLLEN = 9;
SQLELTS_PTR3->SQLADDR = (char *) &SEGIDAT3);
SQLELTS_PTR3->SQLIND = NULL;
SQLPVARS3.SQLPVARS = 52;
SQLPLIST3.SQLVPARM = (char *) &SQLPVARS3.SQLPVARS;
SQLPLIST3.SQLCODEP = (char *) &sqlca;
SQLPLIST3.SQLTIMES[0] = 0x14EA;
SQLPLIST3.SQLTIMES[1] = 0x9521;
SQLPLIST3.SQLTIMES[2] = 0x0570;
SQLPLIST3.SQLTIMES[3] = 0x64A0;
DSNHLI ( (unsigned int ) &SQLPLIST3); }

db2check(ekyrcpic);
return;
} /* end of db2isrt */
```

Figure 62 (Part 11 of 18). Second Sample Propagation Exit Routine (C)
void db2dlet(EKYRCPIC *ekyrcpic, XPCB *xpcb) {
    if (xpcb->xpcbckeya == NULL)
        invkfb(ekyrcpic); /* IMS-to-DB2: KFBA is missing (EKYEPR7E) */
    else
        strncpy( SEGIKEY, xpcb->xpcbckeya, 6);
    
    EXEC SQL DELETE FROM TABX
            WHERE TABXCOL1 = :SEGIKEY

    SQLPLIST SQLPLIST4 =
        (40, -32768, 30, "EKYEPR2K", 0, 0, 0, 0,
         3, 0, 0, 0, 514, 233);
    SQLELTS_PTR SQLELTS_PTR4;
    struct
        { long SQLPVARS;
          char SQLPVELT[(sizeof(SQLELTS) + 1)];
        } SQLPVARS4;
    SQLELTS_PTR4 = (SQLELTS *) &SQLPVARS4.SQLPVELT;
    SQLELTS_PTR4->SQLTYPE = 46;
    SQLELTS_PTR4->SQLLEN = 7;
    SQLELTS_PTR4->SQLIND = NULL;
    SQLPVARS4.SQLPVARS = 16;
    SQLPLIST4.SQLVPARM = (char *) &SQLPVARS4.SQLPVARS;
    SQLPLIST4.SOCKDREP = (char *) &sqlca;
    SQLPLIST4.SQLTIMES[0] = 0x14EA;
    SQLPLIST4.SQLTIMES[1] = 0x9521;
    SQLPLIST4.SQLTIMES[2] = 0x0570;
    SQLPLIST4.SQLTIMES[3] = 0x64A0;
    DSNHLI ( (unsigned int *) &SQLPLIST4);
}

db2check(ekyrcpic);
    return;
} /* end of db2dlet */

void db2check( EKYRCPIC *ekyrcpic)
{
    strncpy (ekyrcpic->picsqlca.sqlcaid, sqlca.sqlcaid, 136);
    if ((SQLCODE != 0) || (SQLWARN0 == 'W')) sqlerr(ekyrcpic);
    return;
} /* end of db2check */
#pragma page(1)

void sqlerr(EKYRCPIC *ekyrcpic)
{
    unsigned int i, j;

    ekyrcpic->picxretc = 4;
    strncpy(msg11.picxmsgi, "EKYEPR1E", 8);
    i = strlen(msg11.picxmtxt) + 4;
    strncat(msg11.picxmtxt, "TABX", 4);
    memset(&msg11.picxmtxt[i], ' ', 61-i);
    strncpy(msg12.msgstxto, opcode, 6);
    i = abs(SQLCODE);
    for (j = 3; j > mzqr!z!t; j--,i/=1mzqr!z!t)
        wsqlcode[j] = i%1mzqr!z!t + 'mzqr!z!t';
    wsqlcode[mzqr!z!t] = (SQLCODE > mzqr!z!t)? '+':'-';
    strncpy(msg12.msgssqlc, wsqlcode, 4);
    strncpy(ekyrcpic->picxmesg.picxml11.picxmsgi, msg11.picxmsgi, 7mzqr!z!t);
    memset(&(ekyrcpic->picxmesg.picxml2), ' ', 7mzqr!z!t);
    strncpy(ekyrcpic->picxmesg.picxml2, msg12.msgstxt2, 48);
    return;
} /* end of sqlerr */

#pragma page(1)

void db2toims(EKYRCPIC *ekyrcpic, HEC *hec)
{
    long THREE = 3;
    long FOUR = 4;
    int i, j;
    QWmzqr!z!t185A = hec->heccdcdd;
    QWmzqr!z!t185b = hec->heccdcda;
    qw0185a = hec->heccdcdd;
    qw0185b = hec->heccdcda;

    for(x1 = mzqr!z!t; x1 < qw0185a->qw01851d; x1++)
    {
        x2 = qw0185a->qw0185vr[x1].qw0185x;
    }
}

Figure 62 (Part 13 of 18). Second Sample Propagation Exit Routine (C)
```c
/* Set output offset depending on the column name. */
if (strncmp(&(qwmzqr!z!t185a->qwmzqr!z!t185vr[x1].qwmzqr!z!t185cn[mzqr!z!t]),
        "TABXCOL1", 8) == 0 )
    x3 = 0;
else
    if (strncmp(&(qwmzqr!z!t185a->qwmzqr!z!t185vr[x1].qwmzqr!z!t185cn[mzqr!z!t]),
        "TABXCOL2", 8) == 0 )
        x3 = 6;
    else
    if (strncmp(&(qwmzqr!z!t185a->qwmzqr!z!t185vr[x1].qwmzqr!z!t185cn[mzqr!z!t]),
        "TABXCOL3", 8) == 0 )
       x3 = 13;
    else
    x3 = 17;

/* Move the content of the column, byte by byte. */
for(i = mzqr!z!t; i < qwmzqr!z!t185a->qwmzqr!z!t185vr[x1].qwmzqr!z!t185le; i++)
    segoarea[x3+i] = qwmzqr!z!t185b->qwmzqr!z!t185dr[x2+i];
}

#pragma page(1)

/* Initialize the AIB and the SSAs */
memset(&(ekyrcpic->picaib.aibsfunc), ' ', 8);
memset(&(ekyrcpic->picaib.aibrsnm2), ' ', 8);
ekyrcpic->picaib.aiboalen =25;
strncpy(ssa.ssavalue, segoarea, 6);
if (strncmp(segoarea, "5mzqr!z!tmzqr!z!tmzqr!z!tmzqr!z!t", 6) < mzqr!z!t)
{
    strncpy(ssa.ssasegnm, "SEG1 ", 8);
    strncpy(ssa.ssakeynm, "SEG1KEY ", 8);
}
else
{
    strncpy(ssa.ssasegnm, "SEG2 ", 8);
    strncpy(ssa.ssakeynm, "SEG2KEY ", 8);
}

#pragma page(1)

/* Search the PCB label */
for (x4 = 0; x4 < hec->hecdbsln; x4++)
{
    if (strncmp(ssa.ssasegnm,
            hec->hecdbsla->hecdslds[x4].hecsegnm,
            4) == 0)
    {
        strncpy(ekyrcpic->picaib.aibrsmn1,
            hec->hecdbsla->hecdslds[x4].hecpcbnm, 8);
    }
}
```

Figure 62 (Part 14 of 18). Second Sample Propagation Exit Routine (C)
if (strncmp(qw0185b->qw0185pc, "IN", 2) == 0)
/* *************************************************************/
* IMS segment to be inserted.  *
/* *************************************************************/
{  
   strncpy(funcode, "ISRT", 4);
   strncpy(ssaisrt, ssa.ssasegm, 8);
   ssaisrt[8] = ' ';  
   CEETDLI (&FOUR, funcode,  
   ekyrcpic->picaib, segarea, ssaisrt);  
   if (ekyrcpic->picaib.aibretrn != mzqr)
      imserr(ekyrcpic);  
}  /* end INSERT call */

#pragma page(1)
else
if (strncmp(qw0185b->qw0185pc, "UA", 2) == 0)
/* *************************************************************/
* IMS segment is to be replaced.  *
/* *************************************************************/
{  
   strncpy(funcode, "GHU ", 4);
   CEETDLI (&FOUR, funcode, ekyrcpic->picaib, segiarea, ssa);  
   if (ekyrcpic->picaib.aibretrn != 0)
      imserr(ekyrcpic);  
      return;
   else  
   {  
      strncpy(funcode, "REPL", 4);
      CEETDLI (&THREE, funcode, ekyrcpic->picaib, segiarea);  
      if (ekyrcpic->picaib.aibretrn != 0)
         imserr(ekyrcpic);  
         return;
   }  
}  /* end UPDATE call */

#pragma page(1)
else
if (strncmp(qw0185b->qw0185pc, "DE", 2) == 0)
/* *************************************************************/
* IMS segment is to be deleted.  *
/* *************************************************************/
{  
   strncpy(funcode, "GHU ", 4);
   CEETDLI (&FOUR, funcode, ekyrcpic->picaib, segiarea, ssa);  
   if (ekyrcpic->picaib.aibretrn != 0)
      /* Propagation failure for segment (EKYEPR2E) */  
      imserr(ekyrcpic);  
      return;
}  /* end AIB return code not equal to zero for GHU call */
else  
{  
   strncpy(funcode, "DLET", 4);

Figure 62 (Part 15 of 18). Second Sample Propagation Exit Routine (C)
CEETDLI (&THREE, funccode, ekycrpc->picaib, segiarea);

if (ekycrpc->picaib.aiibretrn != mzqr!z!t)
{
  /* Propagation failure for segment (EKYEPR2E) */
  imserr(ekycrpc);
  return;
}

/* end AIB return code not zero for DLET call */

/* end AIB return code equal to zero for GHU call */

} /* end DELETE call */

/************************* **********
 * Invalid function call
 *************************

else invfun(ekycrpc); /* not INSERT, UPDATE or DELETE call */

return;

} /* SSASEGNN matches HECSEGNN */

} /* end for x4 loop */

lablnf(ekycrpc); /* PCB label not found */

return;

} /* end of db2toims */

#pragma page(1)

/************************* **********
 * IMS error. *
 *************************

void imserr(EKYRCPIC *ekycrpc)
{

  int i;

  ekycrpc->picxretc = 8;
  strncpy(msg31.picxmsgi, "EKYEPR2E", 8);
  i = strlen(msg31.picxmtxt) + 8;
  strncat( msg31.picxmtxt, ssa.ssasegnm, 8);
  memset( &(msg31.picxmtxt[i]), ' ', 61-i);

  strncpy(msg32.msgitxt2, funccode, 4);
  memset(&(ekycrpc->picxmesg.picxml2), ' ', 7mzqr!z!t);
  strncpy(ekycrpc->picxmesg.picxml2, msg32.msgitxt2, 26);

  return;
}

} /* end of imserr */

#pragma page(1)

/************************* **********
 * Invalid propagation direction (found in PICCALL) *
 *************************

void invdir(EKYRCPIC *ekycrpc)
{

  ekycrpc->picxretc = 16;
  strncpy(ekycrpc->picxmesg.picxmsgi, "EKYEPR3E", 8);
  ekycrpc->picxmesg.picxml2 = ' ';
  strncpy(ekycrpc->picxmesg.picxml2, "Invalid propagation direction in PICCALL ", 61);

  return;
}

} /* end of invdir */

Figure 62 (Part 16 of 18). Second Sample Propagation Exit Routine (C)
#pragma page(1)

IMS to DB2 - unexpected DBD or segment name.

```c
void invseg(EKYRCPIC *ekyrcpic, XPCB *xpcb)
{
    ekyrcpic->picxretc = 16;
    strncpy(ekyrcpic->picxmesg.picx11.picxmsgi, "EKYEPR4E", 8);
    ekyrcpic->picxmesg.picx11.picxmsgb = ' ';  
    strncpy(ekyrcpic->picxmesg.picx11.picxmtxt,  
        "IMS-to-DB2: Unexpected DBD or SEGNAME for EKYEPR2K ", 61);
    errcom(ekyrcpic, xpcb);
    return;
}
```

 IMS to DB2 - unexpected DBD or segment name.

```c
void invcal(EKYRCPIC *ekyrcpic, XPCB *xpcb)
{
    ekyrcpic->picxretc = 16;
    strncpy(ekyrcpic->picxmesg.picx11.picxmsgi, "EKYEPR6E", 8);
    ekyrcpic->picxmesg.picx11.picxmsgb = ' ';  
    strncpy(ekyrcpic->picxmesg.picx11.picxmtxt,  
        "IMS-to-DB2: Unexpected call function in IMS XPCB ", 61);
    errcom(ekyrcpic, xpcb);
    return;
}
```

Additional processing when unexpected DBD or segment encountered

```c
/* Additional processing when unexpected DBD or segment encountered */  
```

Figure 62 (Part 17 of 18). Second Sample Propagation Exit Routine (C)
void errcom(EKYRCPIC *ekyrcpic, XPCB *xpcb)
{
    strncmp(msg22.msgodbdb, xpcb->xpcbdbd, 8);
    strncmp(msg22.msgoseg, xpcb->xpcbseg, 8);
    strncmp(msg22.msgofunc, xpcb->xpcbcall, 4);
    memset(&((ekyrcpic->picxmesg.picxml2), ' ', 7);
    strncpy(ekyrcpic->picxmesg.picxml2, msg22.msgotxt2, 43);
    return;
} /* end of errcom */

#pragma page(1)
/*@begin of include 12*/

void invkfb(EKYRCPIC *ekyrcpic)
{
    ekyrcpic->picxretc = 16;
    strncpy(ekyrcpic->picxmesg.picxml2.picxmsgi, "EKYEPR7E", 8);
    ekyrcpic->picxmesg.picxml2.picxmsgb = ' ';
    strncpy(ekyrcpic->picxmesg.picxml2.picxmtxt,
            "IMS-to-DB2: KFBA is missing for REPL call ", 61);

    return;
} /* end of invkfb */

#pragma page(1)
/*@begin of include 13*/

void invfun(EKYRCPIC *ekyrcpic)
{
    ekyrcpic->picxretc = 16;
    strncpy(ekyrcpic->picxmesg.picxml2.picxmsgi, "EKYEPR8E", 8);
    ekyrcpic->picxmesg.picxml2.picxmsgb = ' ';
    strncpy(ekyrcpic->picxmesg.picxml2.picxmtxt,
            "DB2-to-IMS: Invalid call function in the HEC ", 61);

    return;
} /* end of invfun */

#pragma page(1)
/*@begin of include 14*/

void lablnf(EKYRCPIC *ekyrcpic)
{
    ekyrcpic->picxretc = 16;
    strncpy(ekyrcpic->picxmesg.picxml2.picxmsgi, "EKYEPR9E", 8);
    ekyrcpic->picxmesg.picxml2.picxmsgb = ' ';
    strncpy(ekyrcpic->picxmesg.picxml2.picxmtxt,
            "DB2-to-IMS: PCBLABEL not found ", 61);

    return;
} /* end of lablnf */

/* end of program */

Figure 62 (Part 18 of 18). Second Sample Propagation Exit Routine (C)
Definitions for Second Sample Propagation Exit

This section contains definitions associated with the second sample Propagation exit routine. The following types of definitions are provided:

- IMS DBDGEN and PSBGEN definitions
- DB2 CREATE TABLE definitions
- DataRefresher definitions required to define the PR DataRefresher and to extract the IMS data with DataRefresher
- SQL statements defining the PR without DataRefresher in the MVG input tables

DBDGEN Definitions

Figure 63 shows a DBDGEN definition for the Propagation exit routine in Figure 62 on page 236.

```
* *** DESCRIPTION OF THE FIRST DBD
*  DBD NAME=IMSDB1,VERSION=V123456789,
  ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC456789,5,4),
  EXIT=(EKYRUP00)
  DATASET DDI=IMSDB1,SIZE=4096,DEVICE=3380
*  SEGM NAME=SEG1,PARENT=0,BYTES=25
  FIELD NAME=(SEG1KEY,SEQ,U),BYTES=6,START=1
* *** DESCRIPTION OF THE SECOND DBD
*  DBD NAME=IMSDB2,VERSION=V123456789,
  ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC456789,5,4),
  EXIT=(EKYRUP00)
  DATASET DDI=IMSDB2,SIZE=4096,DEVICE=3380
*  SEGM NAME=SEG2,PARENT=0,BYTES=25
  FIELD NAME=(SEG2KEY,SEQ,U),BYTES=6,START=1
*  DBDGEN
  FINISH
  END
```

Figure 63. DBDGEN Definition

Note: The EXIT= keyword of the DBD macros specify that EKYRUP00 (the RUP) be called when a segment of these DBDs is changed. This is required for synchronous data propagation with DPROP.

CREATE TABLE Statement

Figure 64 on page 255 shows a CREATE TABLE statement for the Propagation exit routine in Figure 62 on page 236.
CREATE TABLE T096606.TABX
(TABXCOL1 CHAR(6) NOT NULL,
 TABXCOL2 CHAR(7) ,
 TABXCOL3 CHAR(4) ,
 TABXCOL4 CHAR(8) ,
 PRIMARY KEY (TABXCOL1))
DATA CAPTURE CHANGES
IN DU096606.PROPTS;
CREATE UNIQUE INDEX XN01 ON TABX (TABXCOL1)
USING VCAT KOE ;

Figure 64. CREATE TABLE Statement

Note:  The DATA CAPTURE CHANGES option of the CREATE TABLE command specifies that the DB2 Changed Data Capture exit (the HUP) be called when a row of this table is changed under IMS attach.

Using DataRefresher to Define the PR
This section shows how can use DataRefresher to define the PR for the Propagation exit routine in Figure 62 on page 236.

CREATE DXTPSB
Figure 65 shows a CREATE DXTPSB statement for the Propagation exit routine in Figure 62 on page 236.

CREATE DXTPSB NAME=KOEPSB
  DXTPCB NAME=DECADIX1, DBNAME=IMSDB1, DBACCESS=HDAM
  SEGMENT NAME=SEG1, PARENT=mzqr!z!t, BYTES=25
      FIELD NAME=SEG1KEY, START=1, BYTES=6, SEQFLD=R
      FIELD NAME=SEG1DAT1, START=7, BYTES=7, TYPE=C
      FIELD NAME=SEG1DAT2, START=14, BYTES=4, TYPE=C
      FIELD NAME=SEG1DAT3, START=18, BYTES=8, TYPE=C
    DXTPCB NAME=DECADIX2, DBNAME=IMSDB2, DBACCESS=HDAM
    SEGMENT NAME=SEG2, PARENT=mzqr!z!t, BYTES=25
        FIELD NAME=SEG2KEY, START=1, BYTES=6, SEQFLD=R
        FIELD NAME=SEG2DAT1, START=7, BYTES=7, TYPE=C
        FIELD NAME=SEG2DAT2, START=14, BYTES=4, TYPE=C
        FIELD NAME=SEG2DAT3, START=18, BYTES=8, TYPE=C
;
Figure 65. CREATE DXTPSB

The DXTPXB contains two DXTPCBs, each referring to a particular database.

CREATE DXTVIEW
Figure 66 on page 256 shows a CREATE DXTVIEW statement for the Propagation exit routine in Figure 62 on page 236.
CREATE DXTVIEW NAME = VIEW01,
    DXTPSB = KOEPSB,
    DXTPCB = DECADIX1,
    SEGMENT = SEG1,
    MINSEGM = SEG1,
    FIELDS = * ;

CREATE DXTVIEW NAME = VIEW02,
    DXTPSB = KOEPSB,
    DXTPCB = DECADIX2,
    SEGMENT = SEG2,
    MINSEGM = SEG2,
    FIELDS = * ;

Figure 66. CREATE DXTVIEW Statement

DataRefresher UIM SUBMIT Command and EXTRACT Statement
Figure 67 shows a DataRefresher UIM SUBMIT command and EXTRACT statement for the Propagation exit routine in Figure 62 on page 236.

SUBMIT EXTID=PROPmzqr!z!t1,
    NODE=NODEX,
    USERID=Tmzqr!z!t966mzqr!z!t6,
    CD=JCS,
    JCS=DDJCSmzqr!z!t1,
    FORMAT=SOURCE,
    MAPEXIT=EKYMCEBO,
    MAPUPARM='PRTYPE=U,
    MAPDIR=TW,
    ACTION=REPL,
    ERROPT=BACKOUT,
    EXITNAME=EKYEPR2K,
    PROPSEGM=(IMSDB1/SEG1,IMSDB2/SEG2)' 

EXTRACT INTO T096606.TABX (TABXCOL1 NOT NULL, 
    TABXCOL2 ,
    TABXCOL3 ,
    TABXCOL4 )
SELECT SEGIKEY, 
    SEGI1DAT1,
    SEGI1DAT2,
    SEGI1DAT3
    FROM VIEW01, VIEW02 ;

SUBMIT EXTID=EXTRmzqr!z!t2,
    NODE=NODEX,
    USERID=Tmzqr!z!t966mzqr!z!t6,
    CD=JCS,
    JCS=DDJCSmzqr!z!t1,
    USERDECK='RESUME(YES)',
    FORMAT=SOURCE 

EXTRACT INTO T096606.TABX (TABXCOL1 NOT NULL, 
    TABXCOL2 ,
    TABXCOL3 ,
    TABXCOL4 )
SELECT SEG2KEY, 
    SEG2DAT1,
    SEG2DAT2,
    SEG2DAT3
    FROM VIEW02 ;
Notes:

1. It is necessary to provide two DataRefresher extract requests (ER) to extract the complete data by the DEM, but only the first extract request becomes a propagation request (PR) for DPROP.

In the figure above, the first ER is the PR used by DPROP, and the second ER is required only to extract the data from the second database.

The following description refers only to the propagation request (PROP01).

2. The MAPEXIT= keyword of the SUBMIT command specifies EKYMCE00. This causes DataRefresher UIM to call the DPROP-provided Map Capture Exit EKYMCE00 during the processing of the SUBMIT/EXTRACT. This is required to allow DPROP to create the PR.

3. PRTYPE=U (user mapping) must be specified, because the PR should be processed by a Propagation exit routine.

4. EXITNAME=EKYEPR2K specifies the name of the Propagation exit routine which will perform the propagation for this PR.

5. PROPSEGM=(IMSDB1/SEG1, IMSDB2/SEG2) identifies the segment types and their respective databases being propagated by this PR.

6. FROM VIEW01, VIEW02 identifies the views for the two databases that this PR propagates.

7. The EXTRACT statement describes to DataRefresher which fields should be mapped to which columns during the data extract. These definitions are important for the extract but are not important for DPROP because the mapping and propagation is not done by the generalized mapping logic of DPROP.

8. There is no PCBLABEL provided in the MAPUPARM operand. DPROP needs two different PCB labels (two different databases). The two PCB labels needed by the HUP to perform DB2-to-IMS propagation are the names of the DXTPCBs provided at DXTPSB coding (DECADIX1 and DECADIX2). These names are picked up by DPROP and are passed in the HEC to the Propagation exit routine.

Using DataRefresher for the Extract

This section covers INITDEM and USE DXTPSB Control Statements. Figure 68 shows a INITDEM and USE DXTPSB control statements for the Propagation exit routine in Figure 62 on page 236.

```
INITDEM NAME=BASILEUS;
USE DXTPSB=KOEPSB;
```

Figure 68. Using DataRefresher for the Extract: INITDEM and USE DXTPSB Control Statements

Defining the PR in the MVG Input Tables

Figure 69 on page 258 describes DSNTEP2 SQL statements required to define the PR in the MVG input tables.

The following rows are inserted into the MVG input tables:

- One row is inserted into the DPRIPR table (the PR table).
This row identifies the PRID, indicates that the PRTYPE is U (user mapping), and provides in the EXITNAME column the name of the Propagation exit routine, EKYEPR2K, which performs the propagation for this PR.

- One row for each segment type being propagated by the PR and the Propagation exit routine is inserted into the DPRISEG table (the SEG table).
  As explained in the commentary of the source code of EKYEPR2K, the sample exit routine propagates changes to segment SEG1 of database IMSDB1 or to segment SEG2 of database IMSDB2 depending on the key content.

- One row is inserted into the DPRITAB table (the TAB table).

  This row indicates that the target table is T096606.TABX.

For PRTYPE=U, DPROP does not require that you insert any rows in the DPRIFLD table; this is why the example below does not insert any rows in the DPRIFLD table.

```sql
DELETE FROM T096606.DPRIPR WHERE PRID = 'PROP1' ;

INSERT INTO T096606.DPRIPR
( PRID, USERID, PRTYPE, MAPCASE, MAPDIR,
  ERROPT, ACTION, EXITNAME)
VALUES ('PROP1','T096606','U', ' ', 'TW', 'BACKOUT','REPL', 'EKYEPR2K') ;

INSERT INTO T096606.DPRISEG
( PRID, DBNAME, SEGNAME, ROLE, PCBLABEL)
VALUES ('PROP1','IMSDB1', 'SEG1', ' ', 'DECADIX1') ;

INSERT INTO T096606.DPRISEG
( PRID, DBNAME, SEGNAME, ROLE, PCBLABEL)
VALUES ('PROP1','IMSDB2', 'SEG2', ' ', 'DECADIX2') ;

INSERT INTO T096606.DPRITAB
( PRID, TABQUAL, TABNAME)
VALUES ('PROP1','T096606', 'TABX') ;

COMMIT;
```

*Figure 69. DSNTEP2 SQL Statements*
You will need to write a DB2 Data Capture subexit routine if your installation needs the HUP to coexist with another DB2 Data Capture exit routine. Instead of having two DB2 Data Capture exit routines (which is not supported by DB2), you will:

- Use the HUP as a DB2 Data Capture exit routine, and
- Define to DPROP the other generalized exit routine as a DB2 Data Capture subexit routine (definition is done during DPROP installation).

The purpose of the subexit routine is usually not DB2-to-IMS propagation. Instead, its purpose is usually to:

- Propagate changed DB2 rows to other tables, or
- Perform other generalized functions, such as auditing changed DB2 rows.

DPROP calls your subexit routine when the HUP is invoked by the DB2 Data Capture function. DPROP calls the subexit routine even if you have not defined a PR and even if propagation has been emergency stopped.

However, your subexit will not be invoked when the HUP issues a rollback of the unit of work or an abend. This is not a problem since, in this case, the SQL update can be considered nonexistent.

When your subexit routine is invoked, the HUP provides it with both the data and the description of the changed row.

Although DPROP calls the DB2 Data Capture subexit routine, it is not part of its propagation procedure. Its call occurs regardless of whether:

- Propagation requests exist.
- Propagation is suspended.
- Propagation is deactivated.
- Propagation is emergency stopped.

Therefore, your DB2 Data Capture subexit routine cannot benefit from DPROP support functions.

Your exit routine can be written in Assembler, COBOL, PL/I, or C. DPROP support for exit routines written in HLL requires LE/370 Version 1 Release 2.

The DB2 Data Capture subexit routine is called when the HUP receives captured DB2 data. This applies to IMS batch and online dependent regions accessing DB2. Your DB2 Data Capture subexit routine runs within the same unit of work (UOW) as the updating application program and propagation request. Avoid using functions affecting PR processing, including:

- Execution of SQL COMMIT and ROLLBACK
- IMS CHKP, SETS, ROLS, ROLL, and ROLB calls
- IMS INIT STATUS GROUPA and GROUPB calls
- Execution of IFI calls requesting captured data
- ABENDs of your exit
How To Write a DB2 Data Capture Subexit Routine

Because the DB2 Data Capture subexit routine is not considered to be part of propagation, DPROP does not have special requirements for it.

DB2 Data Capture Subexit Routine Interface

When DPROP receives the changed data, it performs normal propagation. After processing a PR for the table for which data was captured, it calls your DB2 Data Capture subexit routine.

The HUP calls your subexit routine with the following parameters:

- A 64-byte anchor area.
- The HEC. The HEC is a DPROP control block that contains pointers to areas passed by the DB2 Data Capture exit.

Upon entry to your subexit routine, Register 1 contains the address of the list. This list is two fullwords long and contains the addresses of the parameters in the order listed above.

64-Byte Anchor Area

DPROP gives you 64 bytes as a general-purpose storage area. You can use it for whatever you want. Initially, the area is set to all binary zeros, and DPROP never changes it again.

The anchor area exists in virtual storage, and remains yours for the duration of the exit.

- For IMS Batch and BMP regions, the anchor area lasts for the duration of the application program.
- For MPP regions, the anchor area lasts for the duration of the IMS Program Controller Subtask. This spans multiple MPP executions.

HEC Interface

The HEC is the second parameter passed to your DB2 Data Capture subexit routine when the HUP calls it. It is used to provide the pointers to the areas received from the DB2 Data Capture (DB2CDC) and passed to your exit. These areas describe and contain the captured changed data, and are listed below:

- **QWHC**: Is the DB2 Instrumentation Facility standard header mapped by DSNDQWHC
- **QWHS**: Is the DB2 Instrumentation Facility correlation data mapped by DSNDQWHS
- **CDCDD**: Contains the Data Capture table description and is mapped by the QW0185 DSECT within DSNDQW02
- **CDCDA**: Contains the Data Capture data row and is also mapped by the QW0185 DSECT within DSNDQW02

For inserts and deletes there is one data row with the data of the inserted or deleted row. For updates there is one data row containing the after-image and one data row with the before-image of the updated row.
Your exit routine must not modify the HEC or the data pointed to by this control block.

Figure 70 provides an overview of the interface defined through the HEC.

As shown in the numbered sections of the figure, the interface consists of:

1. One HEC control block that provides various pointers.
2. A pointer to the DB2 Instrumentation Facility standard header data that contains specific DB2 information based on the active trace.
3. A pointer to the DB2 Instrumentation Facility correlation data header containing information about correlation and authorization.
4. A pointer to the Data Capture table description of the changed table and its columns.
5. A pointer to the Data Capture Data (data row) record containing the after image of the captured row. For SQL INSERT and DELETE, this is the only data row passed to your exit routine.
6. A pointer to the Data Capture Data (data row) record containing the **before** image of the captured row. This data row is only present for update operations.

7. A field containing the reason code returned by DB2 for the generated IFI call to retrieve the captured data. See *DB2 Messages and Codes* for a description of IFI reason codes.

**HEC Control Block DSECT**

You can generate the following DSECT in your assembler exit routine by coding the EKYHCHEC macro statement. For HLL exit routines, you can include or copy one of the following members to map the HUP Exit Communication Block:

- **EKYHCHCC** Exit routines written in COBOL
- **EKYHCHCP** Exit routines written in PL/I
- **EKYHCHCK** Exit routines written in C
Figure 71 (Part 1 of 2). HUP Exit Communication Block
The QWHS and QWHC Control Blocks

The IFI standard header data and IFI correlation data are passed as received from the DB2 Instrumentation Facility.

**DSNDQWHS**  Is the DB2 provided macro which maps the standard header data

**DSNDQWHC**  Is the DB2 provided macro which maps the correlation data

Refer to *DB2 Administration Guide* for information about these control blocks.

The Table Description and Data Row Control Blocks

The Data Capture table description contains a description of the captured data. It is always present when the HUP calls your DB2 Data Capture subexit routine.

The Data Capture Data (data row) contains a row's data. When the HUP calls your DB2 Data Capture subexit routine, it passes one or two data row areas, depending on the type of SQL operation that caused the data to be captured:

- For **INSERT** and **DELETE**, there is only one data row that contains either the inserted or deleted row.
- For **UPDATE**, there are two data rows, one containing the image of the row before the update, and one containing the image after the update operation.

Both data rows have the same format and are described by the same Data Capture table description that is passed to your exit routine.

---

*Figure 71 (Part 2 of 2). HUP Exit Communication Block*
The table description and data row are composed of a header common to both, and a data part, which is different for each control block type:

- The header part describes the table, using its qualified table name and the time stamp of the table description. For the data row, it also contains the RBAs of log records, the operation code, and the operation code qualifier.

- The data part of the table description contains a description of the columns of the table. The description is similar to the SQLDA.

- The data part of the data row contains the row data, as described in the table description data part.

You can generate the following DSECT (provided by DB2) in your assembler exit routine by coding the DSNDQW02 macro statement. This macro contains the QW0185 DSECT that represents the mapping of the table description and data row control blocks that the DB2 Data Capture uses.

For HLL exit routines, you can include or copy one of the following members to map the table description and data row control blocks:

- **EKYHCQ2C**: Exit routines written in COBOL
- **EKYHCQ2P**: Exit routines written in PL/I
- **EKYHCQ2K**: Exit routines written in C
Figure 72 (Part 1 of 2). Table Description and Data Row Control Blocks
The Table Description and Data Row Header

The following describes the fields of the table description and data row header part in more detail:

**QW0185LN**  
Length of the total table description or data row (header and data).

**QW0185TP**  
Contains the CDC control block type:

- **S**  
  For the DB2CDC table description

- **D**  
  For the DB2CDC data row

**QW0185RC**  
Reason code describing errors for this table and used only for the data row. If a severe error was detected for this table, the HUP calls your DB2 Data Capture subexit routine only if there is no PR defined for the captured data. In the other case, to keep propagated data consistent, the HUP enforces the rollback of the changes. The reason codes that your DB2 Data Capture subexit routine must handle are:

- **X'00E60A01'**  
The following message is returned in the data portion:
VIOLATION OF INSTALLATION DEFINED EDIT PROCEDURE proc_name, REASON CODE: reason_code

'X'00E60A08' The following message is returned in the data portion:

COLUMN column_name ON TABLE table_name IN VIOLATION OF INSTALLATION DEFINED FIELD PROCEDURE RT: return_code, RS: reason_code, MSG: message_token

'X'00E60A09' The following message is returned in the data portion:

INCORRECT DATA RETURNED FORM FIELD PROCEDURE fieldproc_name FOR TABLE table_name AND COLUMN column_name, MSG: message_token

'X'00E60A0A' The following message is returned in the data portion:

AN INSTALLATION FIELD PROCEDURE HAS RETURNED A RETURN CODE IN REGISTER 15 OTHER THAN AN EXPECTED 0 OR 4

'X'00E60A0B' This code indicates that although the date or time install option was specified as LOCAL, a date or time column value of the row has been returned in ISO format. The DB2 Data Capture never calls date and time exits.

QW0185QT The qualified table name, which is composed of the table creator (QW0185CR) and table name (QW0185TB).

QW0185CR The creator name (authorization ID), which is 8 bytes long and padded with blanks.

QW0185TB The table name, which is 18 bytes long and padded on the right with blanks.

QW0185TS The time stamp (internal format) of the table description from the catalog.

QW0185TL The time stamp (internal format) of the log record within the log buffer CI. This field is present only in the data row (QW0185TP=D).

QW0185UR RBA of the first log record for this unit of work. This field is present only in the data row (QW0185TP=D).

QW0185LR RBA of log record of this data row. This field is present only in the data row (QW0185TP=D).

QW0185PC Operation code describing the type of row image and the SQL operation that performed the data change. This field is present only in the data row (QW0185TP=D). The possible values of QW0185PC are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>Insert</td>
</tr>
<tr>
<td>UB</td>
<td>Update before-image</td>
</tr>
<tr>
<td>UA</td>
<td>Update after-image</td>
</tr>
<tr>
<td>DE</td>
<td>Delete</td>
</tr>
</tbody>
</table>

QW0185RI Operation code qualifier present only in the data row (QW0185TP=D). This field is either blanks, or RI if the operation is a result of a referential constraint enforcement of a DELETE SET NULL or CASCADE operation.
The Table Description Data
The table description data portion contains a similar form of an SQLDA that describes the table. It is like the standard SQLDA external format, except for the field where you usually specify the address of the data area for a particular column. In the CDC table description, this field is already set and contains the offset to the column within the data row data section, which is optionally prefixed by a null indicator variable.

The data portion of the table description consists of four variables followed by an arbitrary number of occurrences of a sequence of five variables, collectively called QW0185VR.

QW0185ID An eye catcher for storage dumps containing CDCDD
QW0185BC Length of the table description data portion, which is \((QW0185NO \times 44) + 16\)
QW0185NO Total number of occurrences of QW0185VR
QW0185LD The number of columns described by occurrences of QW0185VR

The following five variables are collectively called QW0185VR and occur QW0185NO times in the table description. Each occurrence of QW0185VR describes a column in the captured table.

QW0185ST Tells the data type of the column and whether it has an associated indicator variable. For a description of the type codes, see Figure 73 on page 270.
QW0185LE Defines the external length of a value of the column, as follows:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Length attribute in bytes</td>
</tr>
<tr>
<td>Graphic</td>
<td>Length attribute in bytes</td>
</tr>
<tr>
<td>Decimal</td>
<td>byte 1 = precision</td>
</tr>
<tr>
<td></td>
<td>byte 2 = scale</td>
</tr>
<tr>
<td>Float</td>
<td>4 (bytes) for single precision</td>
</tr>
<tr>
<td></td>
<td>8 (bytes) for double precision</td>
</tr>
<tr>
<td>Smallint</td>
<td>2 (bytes)</td>
</tr>
<tr>
<td>Integer</td>
<td>4 (bytes)</td>
</tr>
<tr>
<td>Date</td>
<td>10 (bytes) or LOCAL value</td>
</tr>
<tr>
<td>Time</td>
<td>8 (bytes) or LOCAL value</td>
</tr>
<tr>
<td>Time stamp</td>
<td>26 (bytes)</td>
</tr>
</tbody>
</table>

QW0185SD Contains the CCSID (Coded Character Set Identifier) in bytes 3 and 4. It is a two-byte (unsigned) binary number that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages.

QW0185SI Contains a flag byte and the offset of this column into the data row. The flag byte indicates if the column can be nullable or not. If the column can be NULL, then the column data in the data row is prefixed by an indicator variable (2 bytes). The offset points to the null indicator variable instead of the data for the column; the data immediately follows the indicator and starts at offset + 2. The indicator variable is a two-byte field in the data row containing X’FFFFFF’ (value -1) if the field is null, or X’0000’ if the field contains data.
The format of the QW0185SI field is:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flag byte. If the highest bit (bit 0) is on, then the column is prefixed with a null indicator variable, and the real data starts at offset + 2. The rest of the bits are reserved.</td>
</tr>
<tr>
<td>2-4</td>
<td>Offset into the data, or indicator variable for this column. This offset must be added to the data row data portion address (QW0185DR) to compute the virtual storage address of the column data or indicator variable.</td>
</tr>
</tbody>
</table>

| QW0185SN | Length of name (QW0185NL) and name of the column (QW0185CN). |
| QW0185NL  | Contains the length of the column name. |
| QW0185CN  | Contains the name of the column. |

Figure 73 lists values of the QW0185ST field of the table description and their meanings. There are two values for each data type. The first value means that the column does not have a null indicator and does not allow nulls; the second means that the column has a null indicator and allows nulls. For more information about data types refer to the DB2 SQL Reference.

<table>
<thead>
<tr>
<th>Values</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>384/385</td>
<td>Date</td>
</tr>
<tr>
<td>388/389</td>
<td>Time</td>
</tr>
<tr>
<td>392/393</td>
<td>Time stamp</td>
</tr>
<tr>
<td>448/449</td>
<td>Variable-length character string</td>
</tr>
<tr>
<td>452/453</td>
<td>Fixed-length character string</td>
</tr>
<tr>
<td>456/457</td>
<td>Long character string</td>
</tr>
<tr>
<td>460/461</td>
<td>Variable-length, optionally null terminated character string (C)</td>
</tr>
<tr>
<td>464/465</td>
<td>Variable-length graphic string</td>
</tr>
<tr>
<td>468/469</td>
<td>Fixed-length graphic string</td>
</tr>
<tr>
<td>472/473</td>
<td>Long graphic string</td>
</tr>
<tr>
<td>480/481</td>
<td>Floating point</td>
</tr>
<tr>
<td>484/485</td>
<td>Decimal</td>
</tr>
<tr>
<td>496/497</td>
<td>Large Integer</td>
</tr>
<tr>
<td>500/501</td>
<td>Small Integer</td>
</tr>
</tbody>
</table>
The Data Row Data
The data row data portion starts at label QW0185DR. It contains actual data mapped according to the table description, with DB2 calculated offsets into the data for each column.

SQL inserts (IN) and SQL deletes (DE) are passed as one row pointed to by HECCDCDA, a single image that contains all the columns in the table.

SQL updates are passed as two rows, an after-image (UA) pointed to by HECCDCDA, and a before-image (UB) pointed to by HECCDCDB. Both images contain all the columns of the table.

As applicable, the rules of the external form of a table description dictate how the following data items are handled:

- A string of fields, ordered as they were specified in the external form of a table description of the table, and in standard SQL external format.
- EDITPROCs and FIELDPROCs are called as in standard SQL. The returned data is as decoded by an EDITPROC or any FIELDPROCs that apply, the same as in standard SQL.
- DBCS data is supported as in standard SQL.
- VARCHARs are padded to maximum length, but they contain the actual length in the first two bytes of the data.
- Nulls are represented by an indicator variable (two bytes), which precedes the field, but this field is not included in the length.

Exit Routine Processing
Using the information in the control blocks described above (HEC, table description, and data row), you can do your processing in any way you choose. This section describes some of the things you must consider when developing your DB2 Data Capture subexit routine.

Calling Your Exit Routine
DPROP loads your DB2 Data Capture subexit routine before its first call, and keeps it in virtual storage until the OS/VS task terminates. In MPP regions, this spans multiple MPP executions. Before calling your exit routine, the HUP determines if there is a PR for the captured data, and performs propagation, using generalized or user mapping cases, if applicable. If standard propagation must be aborted, then the HUP does not call your DB2 Data Capture subexit routine. This is because the whole unit of work is rolled back, and changes that the application program performs are made nonexistent.

DPROP uses standard OS/VS conventions when calling your exit routine.

Register 1 Points to the parameter list described above.
Register 13 Contains the address of a register save area.
Register 14 Contains the return address.
Register 15 Contains the entry point address of the exit routine

Upon entering the exit routine, the register contents must be saved into the caller's save area. If your exit routine calls other routines that use standard MVS linkage conventions, it must also provide a save area of its own. The exit routine must return to its caller using normal OS/VS conventions after restoring the registers.
DPROP does not analyze the return code that your DB2 Data Capture subexit routine returns in register 15. Also, like the other DPROP exit routines, your DB2 Data Capture subexit routine gains control in AMODE 31, and must return control in AMODE 31.

The DB2 Data Capture subexit routine can be called multiple times during the processing of an SQL statement, if the statement updates or deletes more than one row. The number of calls, and the order in which they are made, depends on the DB2 process sequence of the rows, and is unpredictable for DPROP and the DB2 Data Capture subexit routine.

**Exit Routine Logic**

Your exit routine can do any processing with the supplied captured data. For performance reasons, it is recommended that your exit routine generate static SQL calls. Avoid using functions that have a detrimental effect on the performance of the application program (such as performing an OPEN and CLOSE on an MVS file each time the exit routine is called). It is also recommended that the DBRMs of your DB2 Data Capture subexit routine be package bound. The DB2 plans created for the propagating application programs must then list the packages.

Because the exit routine executes in the same environment as the propagating application program, it can generate the same type of IMS calls and SQL statements as the application program can.

The DBRM of your DB2 Data Capture subexit routine must be included in the DB2 plans of those application programs that synchronously propagate the changed data. If your exit generates IMS calls, use the AIB interface described in *IMS/ESA Application Programming: DL/I Calls*, which allows your exit routine to generate calls without the address of the IMS PCBs.

Any changes you make to propagated data from within your DB2 Data Capture subexit routine are not captured and cannot be propagated.

A DB2 Data Capture subexit routine must not perform functions that are not supported by the environment in which it is running. For example, an exit routine running in an MPP region must not write to OS files, and the exit routine must not generate STIMER macros in an IMS environment.

It is recommended that you code and link-edit your program as reusable.

**Return Codes**

This section discusses how to return from your exit routine to DPROP. Remember that you must return control to the caller in AMODE 31, using the normal MVS conventions described in the previous section.

DPROP does not accept return codes from your DB2 Data Capture subexit routine, because this exit is not intended for propagation. Therefore, the DB2 Data Capture subexit routine cannot use the DPROP error handling techniques.
Saving Information Across Calls
You can save information across calls to the exit routine. Save the information in the 64-byte anchor area passed at entry to your DB2 Data Capture subexit routine. If this area is not large enough, generate a GETMAIN and save the address of the storage in the 64-byte anchor area.

Updating Your DB2 Data Capture Subexit Routine
DPROP does not provide any online change logic to replace an existing load module copy of your exit routine with a new version of the load module. If you need to change your exit routine, stop the affected IMS regions before performing the change. A change of the exit routine without stopping the IMS regions causes unpredictable results. For example, some MPP regions use the new version of the exit routine, while other regions use the old version. After the change, you can restart the IMS regions.

Telling DPROP About Your Subexit Routine
This section discusses how you can inform DPROP that you want to use a DB2 Data Capture subexit routine. To do this, during DPROP generation, specify which DB2 Data Capture subexit routine must be called when changes are captured for DB2 tables. The exit name you define applies to a whole DPROP system. The DB2 Data Capture subexit routine is called for all captured data, whether or not propagation for it exists.

Sample DB2 Data Capture Subexit Routine
The sample DB2 Data Capture subexit routine in Figure 74 on page 274 is an example of DB2-to-DB2 propagation. In this case, the subexit routine intercepts updates to the table TABLE02 and propagates the same changes to a mirror table TABLE0M.

The source code in Figure 74 on page 274 is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYEDB2A. Following the source code are definitions related to the sample DB2 Data Capture subexit routine.
**Figure 74 (Part 1 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)**

```assembly
1       MACRO
2       SQLSECT &TYPE
3       GBLC &SQLSECT
4       AIF ('&TYPE' EQ 'RESTORE').REST
5       &SQLSECT SETC '&SYSECT'
6       MEXIT
7       .REST ANOP
8       &SQLSECT CSECT
9       MEND
11      PRINT NOGEN
12 *** START OF SPECIFICATIONS ***
13 *
14   MODULE NAME = EKYEDB2A
15 *
16   DESCRIPTIVE NAME = SAMPLE 'DB2 CDC SUBEXIT ROUTINE'
17 *
18   STATUS: V1 R2 Mmzqr!z!t
19 *
20   FUNCTION = EKYEDB2A IS A SAMPLE DPROP 'DB2 CDC SUBEXIT ROUTINE'
21   WHICH ILLUSTRATES HOW TO USE THE INFORMATION PASSED
22   BY HUP TO SUCH A USER EXIT ROUTINE.
23   BECAUSE PROPAGATION TO IMS DATABASES IS PERFORMED BY
24   DPROP, THE SCOPE OF A DB2 SUBEXIT ROUTINE SHOULD NOT
25   BE SUCH A PROPAGATION.
26   - FOR PROPAGATION OF CHANGES OF THE DB2 DATA YOU
27   SHOULD USE DPROP'S:
28   -- GENERALIZED MAPPING CASES FOR PROPAGATION
29   TO IMS DATABASES WHERE THE PRESCRIBED RULES
30   CAN BE APPLIED
31   -- USER MAPPING CASE FOR PROPAGATION TO IMS
32   DATABASES WHERE THE RULES OF THE GENERALIZED
33   MAPPING CASES ARE NOT FLEXIBLE ENOUGH
34   -- AND USE A DB2 SUBEXIT FOR ANY OTHER PURPOSE,
35   EXCEPT PROPAGATION TO IMS DATABASES, SUCH AS:
36   -- MONITORING CHANGES OF THE DB2 DATA
37   -- SECURITY CHECKING
38   -- PROPAGATION TO OTHER ENVIROMENTS AS IMS DB
39   BECAUSE THE SCOPE OF SUCH A DB2 CDC SUBEXIT ROUTINE
40   SHOULD NOT BE IMS PROPAGATION, DPROP WILL INVoke IT
41   REGARDLESS:
42   - IF THERE EXISTS A DPROP PROPAGATION REQUEST OR NOT
43   - IF DPROP PROPAGATION HAS BEEN SUSPENDED
44   - IF DPROP PROPAGATION HAS BEEN DEACTIVATED
45   - IF DPROP PROPAGATION HAS BEEN EMERGENCY STOPPED
46   HOWEVER, IT IS NOT INVOKED:
47   - IF THERE IS A PROPAGATION REQUEST FOR THE CHANGED
48   DATA WHICH CANNOT BE SUCCESSFULLY APPLIED AND
49   ---
50   - IF THE DROP ERROR LOGIC REQUESTS A ROLLBACK OF
51   THE CHANGES MADE TO THE DB2 DATA
```

---

**Customization Guide**

274
THE DB2 CDC SUBEXIT ROUTINE IS INVOKED ONCE FOR EACH
RETRIEVED UPDATE EVENT IN THE IFI DATA STREAM. IF
THE ORIGINATING APPLICATION SQL STATEMENT AFFECTED
MULTIPLE ROWS, THEN THE DB2 CDC SUBEXIT ROUTINE WILL
BE INVOKED BY DPROP MULTIPLE TIMES, UNTIL ALL UPDATE
EVENTS HAVE BEEN PROCESSED BY IT. FOR EACH SINGLE
INVOCATION, THE CAPTURED DATA IS PASSED AS FOLLOWS:

- A CHANGED DATA CAPTURE DATA DEFINITION (CDCDD)
  IS ALWAYS PASSED TO YOUR EXIT. THIS AREA CONTAINS
  A DEFINITION OF THE ROW DATA IN A SIMILAR FORM
  AS IN THE SQLDA.

- A CHANGED DATA CAPTURE DATA ROW (CDCDA) WHICH
  CONTAINS THE COLUMN VALUES OF THE AFFECTED ROW. THIS AREA IS ALWAYS PASSED TO YOUR EXIT AND
  REPRESENTS EITHER THE ONLY DATA ROW FOR INSERT
  AND UPDATE OPERATIONS, OR CONTAINS THE AFTER
  IMAGE OF THE ROW IN CASE OF UPDATE OPERATIONS.

- FOR UPDATE OPERATIONS, YOUR DB2 CDC SUBEXIT
  ROUTINE, WILL RECEIVE AN ADDITIONAL CHANGED
  DATA CAPTURE DATA ROW (CDCDA). THIS AREA
  CONTAINS THE BEFORE IMAGE OF THE AFFECTED ROW.

DISCLAIMERS:

- THIS SAMPLE EXIT IS BY PURPOSE VERY SIMPLE,
  IN ORDER TO AVOID TO OBSCURE THE MOST ESSENTIAL
  ASPECTS OF THE LOGIC OF A DB2 CHANGED DATA
  CAPTURE SUBEXIT ROUTINE.

- THE SCOPE OF THIS SAMPLE EXIT IS THE DB2 TO
  DB2 PROPAGATION. ANY DATA UPDATE (MADE UNDER
  IMS ATTACH) TO THE TABLE 'TABLE2' IS PROPAGATED
  'TABLEM'. BOTH TABLES ARE IDENTICAL AND HAVE
  THE FOLLOWING COLUMNS:

  -- KEYFLD1 CHAR(2) NOT NULL
  -- KEYFLD2 CHAR(6) NOT NULL
  -- FAMILY VARCHAR(30)
  -- FIRST VARCHAR(20)
  -- CITY VARCHAR(35)

  EACH TABLE CONTAINS AN UNIQUE INDEX WITH THE
  COLUMNS KEYFLD1 AND KEYFLD2.

NOTES:

DEPENDENCIES = NONE

RESTRICTIONS = THE DB2 CHANGED DATA CAPTURE SUBEXIT RUNS
WITHIN THE SAME UNIT-OF-WORK (UOW) AS THE
UPDATING APPLICATION PROGRAM AND PROBABLE
DPROP PROPAGATION REQUEST. THEREFORE YOU
MUST AVOID THE USAGE OF FUNCTIONS AFFECTING
THE PROCESS OF THESE, SUCH AS:

- THE EXECUTION OF SQL COMMIT AND ROLLBACK
- IMS CHKP, SETS, ROLS, ROLL AND ROLB CALLS
- IMS INIT STATUS GROUPA AND GROUPB CALLS
- THE EXECUTION OF IFI CALLS REQUESTING
  CAPTURED DATA
- ABENDS OF YOUR EXIT
Figure 74 (Part 3 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Chapter 5. DB2 Data Capture Subexit Routine

Figure 74 (Part 4 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
265 * - ANALYZE OPERATION CODE AND BRANCH ACCORDINGLY *
266 * *
267 * - IF THERE IS AN INVALID OPERATION CODE *
268 * - ISSUE WTO TO INFORM OPERATOR *
269 * - RETURN TO CALLING PROGRAM *
270 * *
271 * - IF OPERATION WAS 'INSERT' *
272 * - INSERT ROW IN MIRROR TABLE USING NEW VALUES *
273 * *
274 * - IF OPERATION WAS 'UPDATE' *
275 * - UPDATE THE ROW IN MIRROR TABLE USING OLD KEYFIELD VALUES IN THE WHERE CLAUSE *
276 * *
277 * - IF OPERATION WAS 'DELETE' *
278 * - DELETE THE ROW USING NEW KEYFIELD VALUES IN THE WHERE CLAUSE *
279 * *
280 * (3) CHECK RESULT OF MIRROR TABLE UPDATE *
281 *
282 * - CHECK THE RESULTING SQL CODE *
283 *
284 * - IF UPDATE OF MIRROR TABLE WAS SUCCESSFUL *
285 * - CONTINUE WITH RETURN TO CALLING PROGRAM *
286 *
287 * - IF MIRROR TABLE UPDATE FAILED *
288 * - EXECUTE THE SQL ERROR LOGIC *
289 *
290 * (4) IF SQL ERROR OCCURED *
291 *
292 * - PREPARE PARAMETER LIST FOR DSNTIAR *
293 * - CALL DSNTIAR GO GET FORMATTED SQL ERROR MESSAGE *
294 * - WTO ANY NON-BLANK MESSAGE LINE RETURNED BY DSNTIAR *
295 * - CONTINUE WITH RETURN PROCESSING *
296 *
297 * (5) RETURN PROCESSING *
298 *
299 * - RELOAD REGISTER AND RETURN TO DPROP *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 ******************* END-OF-LOGIC *****************************
308 *******************************************************
309 *******************************************************
310 *
311 * ASSEMBLER DCLGEN FOR TABLE TABLEM *
312 *
313 *******************************************************
314 ************** TABLE DECLARATION FOR TABLE TABLEM **************
315 EXEC SQL *
316 DECLARE TABLEM TABLE *
317 ( *
318  KEYFLD1 CHAR(2) NOT NULL *,
319  KEYFLD2 CHAR(6) NOT NULL *,
320  FAMILY VARCHAR(30) *,
321  FIRST VARCHAR(20) *,
322  CITY VARCHAR(35) *
323 )
324 *******************************************************

Figure 74 (Part 5 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
**Figure 74 (Part 6 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)**

```assembly
326 *-----------------------------------------------------------------------*
327 *          MODULE OWN SAVEAREA - MUST PREFIX THE WORKAREA         *
328 *-----------------------------------------------------------------------*

000000 330 WRK DSECT ,  ENTER DSECT DECLARATION
000000 0000000000000000  MODULE OWN WORKAREA

333 *-----------------------------------------------------------------------*
334 *          DEFINITION OF SQL HOST VARIABLES                      *
335 *-----------------------------------------------------------------------*

337 ****** -- FIELD DEFINITIONS FOR TABLE TABLEM

000048 338 NEW_KEYFLD1 DS CL2  CHAR(2) (NOT NULL)
00004A 00002 339 LEN_KEYFLD1 EQU *NEW_KEYFLD1
000050 00006 340 NEW_KEYFLD2 DS CL6  CHAR(6) (NOT NULL)
000050 00020 341 LEN_KEYFLD2 EQU *NEW_KEYFLD2

000070 00016 343 NEW_FAMILY EQH DC H,CL30  VARCHAR(30)
000086 00025 344 NEW_FIRST DS H,CL20  VARCHAR(20)
000086 00007 345 OLD_FIRST EQH DC H,CL35  VARCHAR(35)

354 ****** -- OLD KEY FIELD DEFINITIONS FOR TABLE TABLEM

000082 355 OLD_KEYFLD1 DS CL2  CHAR(2) (NOT NULL)
0000B4 356 OLD_KEYFLD2 DS CL6  CHAR(6) (NOT NULL)

358 *-----------------------------------------------------------------------*
359 *          AREA USED TO ISSUE ERROR MESSAGES                      *
360 *-----------------------------------------------------------------------*

0000BA 0000000000000000 362 WRKWTOM DC X(WTODSNTL)'O'  AREA FOR WTO PARMLIST COPY
0000C7 363 WRKWTOTM EQU WRKWTO+4+9,110  DEFINITION OF INSERTED TEXT

365 *-----------------------------------------------------------------------*
366 *          AREA USED TO INVOKE DSNTIAR MESSAGE FORMATTER          *
367 *-----------------------------------------------------------------------*

000138 369 WRKDSNT DS OF  DSNTIAR PARMLIST
000138 00000000 370 WRKDSNT1 DC A(+-+)  - ADDRESS OF SQLCA
00013C 00000000 371 WRKDSNT2 DC A(+-+)  - ADDRESS OF WRKMSG
000140 80000000 372 WRKDSNT3 DC A(+-+X'80000000')  - ADDRESS OF LINE LENGTH
000144 373 WRKMSG DC OF'0'  DSNTIAR MESSAGE AREA
000144 044C 374 WRKMSG1 DC AL2(10+L'WRKMSG1')  LENGTH OF MESSAGE Area
000146 4040404040404040 375 WRKMSG1 DC CL110'  ' MESSAGE LINE 1
0001B4 4040404040404040 376 WRKMSG2 DC CL110'  ' MESSAGE LINE 2
000222 4040404040404040 377 WRKMSG3 DC CL110'  ' MESSAGE LINE 3
000290 4040404040404040 378 WRKMSG4 DC CL110'  ' MESSAGE LINE 4
0002F4 4040404040404040 379 WRKMSG5 DC CL110'  ' MESSAGE LINE 5
00036C 4040404040404040 380 WRKMSG6 DC CL110'  ' MESSAGE LINE 6
0003DA 4040404040404040 381 WRKMSG7 DC CL110'  ' MESSAGE LINE 7
000448 4040404040404040 382 WRKMSG8 DC CL110'  ' MESSAGE LINE 8
0004B6 4040404040404040 383 WRKMSG9 DC CL110'  ' MESSAGE LINE 9
000524 4040404040404040 384 WRKMSG10 DC CL110'  ' MESSAGE LINE 10

386 *-----------------------------------------------------------------------*
387 *          SQL COMMUNICATION AREA                                  *
388 *-----------------------------------------------------------------------*

390 ***$$$
EXEC SQL INCLUDE SQLCA

SQL WORKAREA DEFINITION

---

Figure 74 (Part 7 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
EXECUTE CSECT AND AMODE/RMODE DECLARATIONS

---------------------------------------------------------------------

EXECUTE CSECT AND AMODE/RMODE DECLARATIONS

---------------------------------------------------------------------

EKYEDB2A CSECT , ENTER CSECT OF SUBEXIT ROUTINE
EKYEDB2A AMODE 31 EXIT IS CALLED IN AMODE 31
EKYEDB2A RMODE ANY EXIT CAN BE LOADED ANYWHERE

---------------------------------------------------------------------

GENERATE SAVE-ID WITH EXITNAME AND COMPILATION DATE AND TIME

---------------------------------------------------------------------

LCLC &SAVEID DEFINE LOCAL CHAR VARIABLE
&SAVEID SETC 'EKYEDB2A DPR12'.'-'.'&SYSDATE'.'-'.'&SYSTIME'

---------------------------------------------------------------------

SAVE REGISTERS AND ESTABLISH MODULE ADDRESSABILITY

---------------------------------------------------------------------

SAVE (14,12),,&SAVEID DEFINE ID-BLOCK AND SAVE REGS

---------------------------------------------------------------------

GET ENTRY POINT IN BASE REG

---------------------------------------------------------------------

LET'S POINT TO PASSED PARAMETERS

---------------------------------------------------------------------

LM R9,R10,(R1) GET POINTER TO PARAMETERS

---------------------------------------------------------------------

SAVEAREA AND ESTABLISH ADDRESSABILITY OF WORKAREA

---------------------------------------------------------------------
CHAINING DS 0H
FORWARD CHAIN OUR SAVEAREA
BACKWARD CHAIN THE PASSED ONE
SETUP CORRECT SAVEAREA POINTER
DECLARE WORKAREA ADDRESIBILITY

EXPLANATIONS ABOUT PASSED DATA

AT THIS POINT REGISTER 10 POINTS TO THE HUP
EXTERNAL CONTROL BLOCK (HEC). THE HEC CONTAINS
ITSELF POINTERS TO THE CHANGED DATA CAPTURE DATA
WHICH WAS RETRIEVED BY DPROP USING IFI CALL I85:

- HECQWHS POINTS TO THE IFI STANDARD HEADER AREA.
  THIS AREA IS MAPPED BY THE DSNDQWHS
  MACRO. IF DATA FROM THIS CONTROL BLOCK
  IS NEEDED, THE FOLLOWING INSTRUCTIONS
  CAN BE USED TO ESTABLISH ADDRESIBILITY
  OF IT:

  L RX,HECQWHS
  USING QWHS,RX
  ...

- HECQWHC POINTS TO THE IFI CORRELATION DATA AREA.
  THIS AREA IS MAPPED BY THE DSNDQWHC
  MACRO. IF DATA FROM THIS CONTROL BLOCK
  IS NEEDED, THE FOLLOWING INSTRUCTIONS
  CAN BE USED TO ESTABLISH ADDRESIBILITY
  OF IT:

  L RX,HECQWHC
  USING QWHC,RX
  ...

- HECCDCDD POINTS TO THE DB2 CHANGED DATA CAPTURE
  DATA DEFINITION (CDCDD). DPROP WILL
  ALWAYS PASS A DATA DEFINITION OF THE
  MODIFIED TABLE TO YOUR CHANGED DATA
  CAPTURE SUBEXIT ROUTINE. THIS AREA
  IS MAPPED BY THE QW018S DSECT IN THE
  DSNDQW02 MACRO. AFTER A PREFIX COMMON
  TO CDCDD AND CDCDA (SEE BELOW) THIS
  AREA CONTAINS A DESCRIPTION OF THE
  COLUMNS IN THE TABLE, WHICH IS IN A
  SIMILAR MANNER AS IN THE STANDARD
  EXTERNAL SQLDA. NOTE, THAT QQ195S
  CONTAINS THE OFFSET OF THE COLUMN
  WITHIN THE DATA ROW (CCDDA) AND THAT
  THE LENGTH OF GRAPHIC AND VARGRAPHIC
  FIELDS IS SPECIFIED IN NUMBER OF BYTES
  (IN OPPOSITION TO THE SQLDA WHICH
  CONTAINS THE NUMBER OF DOUBLE BYTES).

- HECCDCDA POINTS TO THE FIRST OR ONLY DATA ROW
  OF THE CHANGED DATA CAPTURE DATA ROW
  (CDCDA). THIS AREA IS ALWAYS PASSED
  TO YOUR EXIT ROUTINE AND IT WILL
  CONTAIN EITHER THE ONLY IMAGE OF THE
  ROW (FOR INSERT OR DELETE OPERATIONS)
  OR THE AFTER IMAGE (FOR UPDATES).
  THIS AREA IS ALSO MAPPED BY THE QW018S
  DSECT OF THE DSNDQW02 MACRO. AFTER A
  PREFIX COMMON TO CCDDA AND CDCDD (SEE
  ABOVE), THIS AREA CONTAINS THE COLUMN
  VALUES OF THE AFFECTED ROW.
- HECCDCDB POINTS TO THE CHANGED DATA CAPTURE
- DATA ROW (CCDAA). THIS AREA CONTAINS
- THE BEFORE IMAGE OF THE AFFECTED ROW.
- AND IS THEREFORE ONLY PRESENT IF THE
- ORIGINATING SQL CALL WAS AN UPDATE.
- THIS AREA IS ALSO MAPPED BY THE QW018S
- DSECT OF THE DSNDQW Macro. AFTER A
- PREFIX COMMON TO CCDAA AND CCDDB (SEE
- ABOVE), THIS AREA CONTAINS THE COLUMN
- VALUES OF THE AFFECTED ROW.

**EXECUTE FETCH OF USED HOST VARIABLES**

- ADDRESS CCDAA AND ANALYZE IF THIS IS THE TABLE
- WE ARE LOOKING FOR (TABLEO2)
- SETUP OLD KEY FIELD VALUES FOR UPDATE OPERATIONS
- SETUP NEW FIELD VALUES FOR ANY OPERATION
- ANALYZE OPERATION CODE AND BRANCH ACCORDINGLY
- IF THERE IS AN INVALID OPERATION CODE
- ISSUE WTO TO INFORM OPERATOR
- RETURN TO CALLING PROGRAM
- IF OPERATION WAS 'INSERT'
- INSERT ROW IN MIRROR TABLE USING THE NEW VALUES
- IF OPERATION WAS 'UPDATE'
- UPDATE THE ROW IN MIRROR TABLE USING OLD KEYFIELD
  VALUES IN THE WHERE CLAUSE
- IF OPERATION WAS 'DELETE'
- DELETE THE ROW USING NEW KEYFIELD VALUES IN
  THE WHERE CLAUSE

**ADDRESS CCDAA AND ANALYZE IF REALLY TABLEO2 IN PROCESS**

**SETUP OLD KEY FIELD VALUES FOR UPDATE OPERATIONS**

**SETUP NEW FIELD VALUES (FOR ALL OPERATIONS)**

---

Figure 74 (Part 10 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 11 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 12 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 13 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 14 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
889 ******************************************************
890 *
891 * (4)  AN SQL ERROR OCCURED *
892 *
893 *  - PREPARE PARAMETER LIST FOR DSNTIAR *
894 *  - CALL DSNTIAR GO GET FORMATTED SQL ERROR MESSAGE *
895 *  - WTO ANY NON-BLANK MESSAGE LINE RETURNED BY DSNTIAR *
896 *  - CONTINUE WITH RETURN PROCESSING *
897 *
898 ******************************************************

900 *
901 *  PREPARE PARAMETER LIST FOR DSNTIAR *
902 *

00036A
904 SQLERR  DS  0H
00036A 4110 D594 00594 905 LA  R1,SQLCA  GET POINTER TO SQLCA
00036E 5010 D138 00138 906 ST  R1,WRKDSNT1 AND STORE AS PARAMETER 1
000372 4110 D144 00144 907 LA  R1,WRKMSG  GET POINTER TO MESSAGE AREA
000376 5010 D13C 0013C 908 ST  R1,WRKDSNT2 AND STORE AS PARAMETER 2
00037A 4110 C6E0 006E0 909 LA  R1,=AL('WRKMSG1)  GET POINTER TO LINE LENGTH
00037E 5010 D140 00140 910 ST  R1,WRKDSNT3 AND STORE AS PARAMETER 3
000382 9680 D140 00140 911 OI  WRKDSNT3,X'80'  INDICATE LAST IN LIST
000386 D201 D144 C70E 00144 0070E 912 MVC  WRKMSGL,=AL(10+L'WRKMSG1)  SETUP LENGTH OF MSG AREA

914 ******************************************************
915 *  CALL DSNTIAR TO GET FORMATTED SQL ERROR MESSAGE *
916 *

00038C 4110 D138 00138 918 LA  R1,WRKDSNT  GET POINTER TO PARMLIST
919 LINK  EP=DSNTIAR THEN LINK TO DSNTIAR MODULE
0003A6 49F0 C710 00710 926 CH  R15,=H'4'  WAS MESSAGE FORMATTED?
0003AA 4720 C30A 0030A 927 BH  RETURN NO --> THEN IGNORE ERROR

929 ******************************************************
930 *  WTOA ANY NON-BLANK MESSAGE LINE RETURNED BY DSNTIAR *
931 *

0003AE 4130 D146 00146 933 LA  R3,WRKMSG1  POINT FIRST MESSAGE LINE
0003B2 4120 000A 0000A 934 LA  R2,10  INITIALIZE THE LOOP REGISTER
0003B6 935 DSNTLOOP  DS  0H
0003B6 D560 3000 C712 00000 00712 936 CLC  O(L'WRKMSG1,R3),=CL(L'WRKMSG1)'  IS THE LINE BLANK?
0003BC 4780 C30A 0030A 937 BE  RETURN YES --> THEN LEAVE THE LOOP
0003CD 5270 D0BA C5A0 0008A 005A0 938 MVC  WRKWT,WTDSSNTM ELSE MOVE WTK SKEI TO WORKAREA
0003C6 D260 DCC7 3000 00CC7 00000 939 MVC  WRKWTOM,O(R3) SETUP TEXT RETURNED BY DSNTIAR
0003D2 4130 306E 0006E 943 LA  R3,1'WRKMSG1(0,R3) POINT NEXT MESSAGE LINE
0003D6 4620 C386 00386 944 BCT  R2,DSNTLOOP AND REPEAT THE WTO LOOP

946 ******************************************************
947 *
948 *  (5)  RETURN PROCESSING *
949 *
950 *  - RELOAD REGISTER AND RETURN TO DPROP *
951 *
952 ******************************************************

954 ******************************************************
955 *  RELOAD REGISTERS AND RETURN TO DPROP *
956 ******************************************************

0003DA 958 RETURN  DS  0H
0003DA 5800 D004 00004 959 L  R13,4(0,R13) POINT CALLERS SAVEAREA
960 RETURN (14,12),RC=0 AND RETURN TO DPROP

Figure 74 (Part 15 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 16 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 17 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
**Definitions**

- **Read-only constants**
- **Literal pool**
- **Equates**

---

### Host Variable Mapping Table for Before Image Columns

<table>
<thead>
<tr>
<th>COLBTAB</th>
<th>DS</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00494</td>
<td>COLTAB</td>
<td>OF</td>
</tr>
<tr>
<td>1115</td>
<td>1116</td>
<td>ENTRY FOR OLD_KEYFLD1 HOST VARIABLE</td>
</tr>
<tr>
<td>1117</td>
<td>DC</td>
<td>AL2(7) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>00496</td>
<td>D2C5E8C6D3C4F140</td>
<td>1118 DC CL18'KEYFLD1' COLUMN NAME</td>
</tr>
<tr>
<td>0048A</td>
<td>01C4</td>
<td>1119 DC AL2(CHAR) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>0049A</td>
<td>0000</td>
<td>1120 DC AL2(0) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>004AC</td>
<td>0082</td>
<td>1121 DC AL2(OLD_KEYFLD1-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>004AE</td>
<td>0002</td>
<td>1122 DC AL2(LEN_KEYFLD1) LENGTH OF HOST VARIABLE</td>
</tr>
<tr>
<td>1123</td>
<td>ENTRY FOR OLD_KEYFLD2 HOST VARIABLE</td>
<td></td>
</tr>
<tr>
<td>1124</td>
<td>DC</td>
<td>AL2(7) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>004B0</td>
<td>D2C5E8C6D3C4F240</td>
<td>1125 DC CL18'KEYFLD2' COLUMN NAME</td>
</tr>
<tr>
<td>004C4</td>
<td>01C4</td>
<td>1126 DC AL2(CHAR) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>004CE</td>
<td>0000</td>
<td>1127 DC AL2(0) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>004CB</td>
<td>0084</td>
<td>1128 DC AL2(OLD_KEYFLD2-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>004CA</td>
<td>0006</td>
<td>1129 DC AL2(LEN_KEYFLD2) LENGTH OF HOST VARIABLE</td>
</tr>
<tr>
<td>1130</td>
<td>END OF TABLE MARKER</td>
<td></td>
</tr>
<tr>
<td>1131</td>
<td>DC</td>
<td>X'FFFF' END OF TABLE MARKER</td>
</tr>
<tr>
<td>004CC</td>
<td>FFFF</td>
<td></td>
</tr>
</tbody>
</table>

---

### Host Variable Mapping Table for After Image Columns

<table>
<thead>
<tr>
<th>COLATAB</th>
<th>DS</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>004D0</td>
<td>COLTAB</td>
<td>OF</td>
</tr>
<tr>
<td>1137</td>
<td>1138</td>
<td>ENTRY FOR NEW_KEYFLD1 HOST VARIABLE</td>
</tr>
<tr>
<td>1139</td>
<td>DC</td>
<td>AL2(7) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>004D2</td>
<td>D2C5E8C6D3C4F140</td>
<td>1140 DC CL18'KEYFLD1' COLUMN NAME</td>
</tr>
<tr>
<td>004E4</td>
<td>01C4</td>
<td>1141 DC AL2(CHAR) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>004E6</td>
<td>0000</td>
<td>1142 DC AL2(0) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>004E8</td>
<td>0048</td>
<td>1143 DC AL2(NEW_KEYFLD1-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>004EA</td>
<td>0002</td>
<td>1144 DC AL2(LEN_KEYFLD1) LENGTH OF HOST VARIABLE</td>
</tr>
<tr>
<td>1145</td>
<td>ENTRY FOR NEW_KEYFLD2 HOST VARIABLE</td>
<td></td>
</tr>
<tr>
<td>1146</td>
<td>DC</td>
<td>AL2(7) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>004EE</td>
<td>D2C5E8C6D3C4F240</td>
<td>1147 DC CL18'KEYFLD2' COLUMN NAME</td>
</tr>
<tr>
<td>00500</td>
<td>01C4</td>
<td>1148 DC AL2(CHAR) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>00502</td>
<td>0000</td>
<td>1149 DC AL2(0) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>00504</td>
<td>004A</td>
<td>1150 DC AL2(NEW_KEYFLD2-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>00506</td>
<td>0006</td>
<td>1151 DC AL2(LEN_KEYFLD2) LENGTH OF HOST VARIABLE</td>
</tr>
<tr>
<td>1152</td>
<td>ENTRY FOR NEW_FAMILY HOST VARIABLE</td>
<td></td>
</tr>
<tr>
<td>1153</td>
<td>DC</td>
<td>AL2(6) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>0050A</td>
<td>C6C1D4C9D3E8A040</td>
<td>1154 DC CL18'FAMILY' COLUMN NAME</td>
</tr>
<tr>
<td>0051C</td>
<td>01C1</td>
<td>1155 DC AL2(VARCHAR+NULL) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>0051E</td>
<td>00AC</td>
<td>1156 DC AL2(IND_FAMILY-WRK) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>00520</td>
<td>0050</td>
<td>1157 DC AL2(NEW_FAMILY-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>00522</td>
<td>0020</td>
<td>1158 DC AL2(LEN_FAMILY) LENGTH OF HOST VARIABLE</td>
</tr>
<tr>
<td>1159</td>
<td>ENTRY FOR NEW_FIRST HOST VARIABLE</td>
<td></td>
</tr>
<tr>
<td>1160</td>
<td>DC</td>
<td>AL2(5) LENGTH OF COLUMN NAME</td>
</tr>
<tr>
<td>00524</td>
<td>C6C9D9E2E4A040</td>
<td>1161 DC CL18'FIRST' COLUMN NAME</td>
</tr>
<tr>
<td>00526</td>
<td>0050</td>
<td>1162 DC AL2(VARCHAR+NULL) COLUMN DATA TYPE</td>
</tr>
<tr>
<td>0053A</td>
<td>00AE</td>
<td>1163 DC AL2(IND_FIRST-WRK) OFFSET OF NULL INDICATOR</td>
</tr>
<tr>
<td>0053C</td>
<td>0070</td>
<td>1164 DC AL2(NEW_FIRST-WRK) OFFSET OF HOST VARIABLE</td>
</tr>
<tr>
<td>0053E</td>
<td>0016</td>
<td>1165 DC AL2(LEN_FIRST) LENGTH OF HOST VARIABLE</td>
</tr>
</tbody>
</table>

---

Figure 74 (Part 18 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
1166 ^----- ENTRY FOR NEW_CITY HOST VARIABLE
000540 0004 000542 C39E3E840404040 1167 DC AL2(4) LENGTH OF COLUMN NAME
000554 01C1 1168 DC CL1B'CITY' COLUMN NAME
000556 0080 1169 DC AL2(VARCHAR+NULL) COLUMN DATA TYPE
000558 0086 1170 DC AL2(IND_CITY-WRK) OFFSET OF NULL INDICATOR
00055A 0025 1171 DC AL2(NEW_CITY-WRK) OFFSET OF HOST VARIABLE
1172 DC AL2(LEN_CITY) LENGTH OF HOST VARIABLE
1173 ^----- END OF TABLE MARKER
00055C FFFF 1174 DC X'FFFF' END OF TABLE MARKER

---------------------------------------------------------------------

WTO MACRO LIST FORMATS

--------- WTO ERRORT WTO 'EKYEDB1E INVALID OPERATION CODE IN CDC DATA DEFINITION', ROUTCDE=11, MF=L
--------- WTOERRDT WTO 'EKYEDB3E UNEXPECTED COLUMN DATA TYPE ENCOUNTERED', ROUTCDE=11, MF=L
--------- WTOCOLEM WTO 'EKYEDB5I COLUMN IN ERROR: -++++----1----+-', ROUTCDE=11, MF=L

LITERAL POOL

--------- LTORG , EXPAND LITERAL POOL
0006CB 0000000B 1222 =A(WRKLEN)
0006BB 00000004 1223 =F'64'
0006BD 00000000 1224 =XLA'00000000'
0006BE 00000000 1225 =V(DSNHLI)
0006BB 0000005B 1226 =F'88'
0006BD 0000001C 1227 =F'28'
0006E0 0000000E 1228 =A(L'WRKMSG1)
0006E4 00000000 1229 =F'00000000'
0006E8 E3C1C035F0F24 1230 =CL1B'TABLE02'
0006FA C905 1231 =C'IN'
0006FC E4C1 1232 =C'UA'
0006FE C4C5 1233 =C'DE'
000700 01C4 1234 =X'01C4'
000702 0002 1235 =H'2'
000704 0006 1236 =H'6'
000706 01C1 1237 =X'01C1'
000708 001E 1238 =H'30'
00070A 0014 1239 =H'20'
00070C 0023 1240 =H'35'
00070E 044C 1241 =AL2(10+L'WRKMSG1)
000710 0004 1242 =AL2(10+L'WRKMSG1)
000712 0000 1243 =X'FFFF'

Figure 74 (Part 19 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Figure 74 (Part 20 of 21). Sample DB2 Data Capture Subexit Routine (Assembler)
Definitions for Sample DB2 Data Capture Subexit Routine

The following statements illustrate how to specify the use of the DB2 Data Capture subexit routine and illustrate the environment that was set up for the exit routine shown in Figure 74 on page 274.

DPROPGEN Definitions

Figure 75 on page 295 shows a DBDGEN definition for the Segment exit routine in Figure 74 on page 274.
CREATE TABLE Statement for Source Table

Figure 76 shows a CREATE TABLE statement for the source table for the Segment exit routine in Figure 74 on page 274.

```
CREATE TABLE TABLEmzqr!z!t2
  (KEYFLD1 CHAR(2) NOT NULL,
   KEYFLD2 CHAR(6) NOT NULL,
   FAMILY VARCHAR(35),
   FIRST VARCHAR(20),
   CITY VARCHAR(35),
   PRIMARY KEY (KEYFLD1, KEYFLD2))
DATA CAPTURE CHANGES IN DUmzqr!z!t96277.DPROPTS2 ;
```

Figure 76. CREATE TABLE Statement for Source Table

Note: The DATA CAPTURE CHANGES clause specifies that the changed DB2 rows are captured and that the DB2CDCEX routine (the HUP) is called when a row of this table is changed.

CREATE TABLE Statement for Mirror Table

Figure 77 on page 296 shows a CREATE TABLE statement for the mirror table for the Segment exit routine in Figure 74 on page 274.
CREATE TABLE TABLE0M
  (KEYFLD1 CHAR(2) NOT NULL,
   KEYFLD2 CHAR(6) NOT NULL,
   FAMILY VARCHAR(30) ,
   FIRST VARCHAR(20) ,
   CITY VARCHAR(35) ,
   PRIMARY KEY (KEYFLD1, KEYFLD2))
IN DU096277.DPROPTSM ;

CREATE UNIQUE INDEX DPROPXM
  ON TABLE0M (KEYFLD1, KEYFLD2)
  USING VCAT KOE ;

Figure 77. CREATE TABLE Statement for Mirror Table

Note: The mirror table cannot have the DATA CAPTURE CHANGES clause
because table updates done within the DB2 Data Capture exit cannot be captured
themselves.
Chapter 6. EKYRESLB Dynamic Allocation Exit Routine

DPROP needs to load some DPROP modules from an APF-authorized library allocated to the EKYRESLB DD name. The EKYRESLB DD name is either:

- Allocated through a JCL DD statement that you provide, or
- Dynamically allocated by DPROP to a data set name that your System Administrator provided during DPROP installation.

If neither of these methods suits your needs, then you can provide an EKYRESLB Dynamic Allocation exit routine. For example, the EKYRESLB Dynamic Allocation exit routine can be useful if your installation uses an online change philosophy based on two load module libraries (for example, DPROP.RESLB1 and DPROP.RESLB2), and switches dynamically between these two libraries. Your EKYRESLB Dynamic Allocation exit routine can be used to decide dynamically which one of the two libraries must be dynamically allocated; for example, making the decision based on a specification located in a SYS1.PARMLIB member, or in a linklist load module.

Providing an EKYRESLB Dynamic Allocation exit routine is optional. Its load module name must be EKYDAEX0 and DPROP loads it from the usual //STEPLIB, //JOBLIB, linklist, LPA concatenation.

Your exit routine can be written in Assembler, but not in COBOL, PL/I, or C.

DPROP calls this exit routine with two different call functions: an AL (ALLOCATE) call function and a DE (DEALLOCATE) call function.

AL During an ALLOCATE call, your exit routine must dynamically allocate the EKYRESLB DD statement (if not already allocated).

DPROP tells your exit routine whether the EKYRESLB DD statement is already allocated or not. If upon return from your exit routine, the EKYRESLB DD statement is not allocated, DPROP dynamically allocates the data set name that your System Administrator identified during DPROP installation.

DE During the DEALLOCATE call, your exit routine can dynamically deallocate the EKYRESLB DD statement, or return without doing any processing.

Your exit routine is called with one ALLOCATE and one DEALLOCATE call function within each OS/VS task executing DPROP functions. This can occur multiple times within the same job step, for example, in MPP regions after pseudo-ABENDs that do not result in a job step ABEND. This can also occur when the RUP is called to perform asynchronous data propagation.

Make sure that each allocation is performed in the same way. To avoid inconsistent allocations, your exit routine must deallocate the EKYRESLB DD Statement during DEALLOCATE calls only if your exit routine performed the original allocation.
Interface Control Block

Code the EKYDAE macro statement to create the following DSECT in your Assembler exit routine.

The interface control block is followed by a detailed description of its fields.
Figure 78. Interface Control Block for EKYRESLB Dynamic Allocation Exit Routine
DAENAME  Contains the constant EKYDAE, which is used to identify the control block in a storage dump.

DAECALL  The call function that describes whether the exit routine is called to allocate or deallocate the EKYRESLB DD name.

DAEALLOC  When called for an ALLOCATE call function, your exit routine must test the content of this field. When called for a DEALLOCATE call function, this field has no meaning. The content can be:

- **Y** The EKYRESLB is already allocated; for example, it was allocated either through a JCL DD statement, or because your exit routine called it, but it was never deallocated.
- **N** The EKYRESLB DD statement is not yet allocated. Your exit routine must allocate the EKYRESLB DD statement dynamically. If your exit routine does not allocate the EKYRESLB DD statement, DPROP dynamically allocates it to the data set name that your System Administrator provided during DPROP installation.

DAEUSER  Your exit routine can use this field to exchange information between the ALLOCATE and DEALLOCATE call.

At entry to an ALLOCATE call, DPROP sets this field to binary zeros. DPROP does not change the content of DAEUSER after this first call to your exit routine.

---

**Exit Routine Processing**

Your EKYDAEX0 routine must be written in Assembler and must conform to the following linkage conventions:

1. Your exit routine is called and must return in AMODE 31. The call parameter that DPROP provides to your exit routine is usually located above the 16-MB line.

2. On entry, your exit routine must save the registers into the save area that the caller provides, and must provide a save area of its own.

   The exit routine must return to its caller using normal OS/VS conventions after restoring the registers.

3. On entry:

   - **Register 1** Points to a parameter list pointing to one single parameter, an interface control block.
   - **Register 13** Points to a register save area.
   - **Register 14** Contains the return address.
   - **Register 15** Contains the entry point address of your exit routine.

4. It is recommended that your exit routine be written and linked as reentrant.

When dynamically allocating the EKYRESLB DD statement, your exit routine must not specify deallocation at CLOSE. This is because DPROP opens and closes the EKYRESLB DD name more than once.
To avoid possible conflicts with STIMER and STIMERM macros generated when application programs perform synchronous data propagation, DPROP calls the exit in an MVS subtask created specifically for the call of your exit routine. DPROP attaches and detaches this subtask for every call of your exit routine.

**Return Codes**

When returning, your exit routine must provide a return code in register 15.

A nonzero return code results in an ABEND.

**Telling DPROP about The EKYRESLB Dynamic Allocation Exit**

To activate your EKYRESLB dynamic allocation exit, compile and link edit your exit routine with the load module name EKYDAEX0 into the //JOBLIB, //STEPLIB, and linklist LPA concatenation.

During DPROP Installation, your DPROP System Administrator can create a dummy, IEFBR14-type, EKYDAEX0 load module in your DPROP RESLIB. In this case, to use your real EKYDAEX0, one of the following must be done:

- The dummy, IEFBR14-type, EKYDAEX0 load module must be deleted from the DPROP RESLIB.
- The load module library containing your real EKYDAEX0 module must be concatenated ahead of the DPROP RESLIB in the //JOBLIB, STEPLIB, and LINKLIB LPA concatenation.

**Sample EKYRESLB Dynamic Allocation Exit**

The sample EKYRESLB dynamic allocation exit below is provided in the DPROP Sample Source Library (EKYSAMP) under the member name EKYEDA1A. To activate the dynamic allocation exit, you must link edit the load module as EKYDAEX0 in the //STEPLIB, //JOBLIB, linklist, or LPA concatenation.
**Figure 79 (Part 1 of 12). Sample EKYRESLB Dynamic Allocation Exit**

```plaintext
2 PRINT NOGEN
3 ********** START OF SPECIFICATIONS ***********************
4 * MODULE NAME = EKYEDA1A *
5 * DESCRIPTIVE NAME = SAMPLE 'EKYRESLB DYNAMIC ALLOCATION EXIT' *
6 * ROUTINE' *
7 * STATUS: V1 R2 M0 *
8 * FUNCTION = EKYEDA1A IS A SAMPLE DPROP USER EXIT ROUTINE *
9 * USED TO ALLOCATE DYNAMICALLY THE EKYRESLB *
10 * DD STATEMENT. *
11 * EKYEDA1A IS CALLED WITH ONE SINGLE PARAMETER: *
12 * THE EKYOED PARAMETER BLOCK. *
13 * IN THIS PARAMETER BLOCK THE FIELD DAECALL *
14 * CONTAINS THE CALL FUNCTION. THE CALL FUNCTION *
15 * IS EITHER: *
16 * - 'AL' (= 'ALLOCATE') *
17 * - 'DE' (= 'DE-ALLOCATE') *
18 * THE FUNCTIONS OF THIS SAMPLE EXIT ROUTINE CAN BE *
19 * SKETCHED AS FOLLOWS: *
20 * FOR AN 'AL' CALL FUNCTION ('ALLOCATE'): *
21 * ---------------------------------------- *
22 * WHEN BEING CALLED WITH AN 'AL' CALL-FUNCTION THIS *
23 * SAMPLE EXIT ROUTINE CHECKS IN INFORMATION PROVIDED *
24 * BY THE CALLER IN EKYEDA WHETHER //EKYRESLB *
25 * IS ALREADY ALLOCATED. *
26 * - IF //EKYRESLB IS ALREADY ALLOCATED, THE SAMPLE *
27 * EXIT ROUTINE RETURNS WITHOUT FURTHER PROCESSING. *
28 * - IF //EKYRESLB IS NOT ALREADY ALLOCATED, THE *
29 * SAMPLE EXIT ROUTINE USES MVS DYNALLOC SERVICES *
30 * TO ALLOCATE DYNAMICALLY THE //EKYRESLB *
31 * DD STATEMENT WITH A DISPOSITION OF 'SHR'. *
32 * THE DATASET-NAME ALLOCATED TO //EKYRESLB IS *
33 * A HARD-CODED/FIXED DATA-SET NAME. IN REAL-LIFE, *
34 * YOUR INSTALLATION WILL PROBABLY PROVIDE SOME *
35 * ADDITIONAL LOGIC ALLOWING TO ALLOCATE *
36 * DIFFERENT/VARIABLE DATA-SET NAMES TO EKYRESLB. *
37 * FOR EXAMPLE, THIS CAN BE ACHIEVED BY READING A *
38 * SYS1.PARMLIB MEMBER CONTAINING THE DATASET-NAME *
39 * TO BE ALLOCATED. *
40 * 'DE' CALL FUNCTION ('DE-ALLOCATE') *
41 * ------------------------------------- *
42 * WHEN BEING CALLED WITH A 'DE' CALL-FUNCTION, THIS *
43 * SAMPLE EXIT ROUTINE CHECKS WHETHER THE ALLOCATION *
44 * OF //EKYRESLB WAS PERFORMED BY THE SAMPLE EXIT *
45 * ROUTINE. *
46 * - IF THIS IS NOT THE CASE, THE SAMPLE *
47 * EXIT ROUTINE RETURNS WITHOUT FURTHER PROCESSING. *
48 * - ELSE, THE SAMPLE EXIT ROUTINE *
49 * USES MVS DYNALLOC SERVICES TO *
50 * DE-ALLOCATE DYNAMICALLY THE //EKYRESLB *
51 * DD STATEMENT. *
52 * *
53 *
```
NOTES:

1) IF WRITING YOUR OWN EXIT ROUTINE, THERE IS NO REAL NEED TO DE-ALLOCATE THE //EKYRESLB DD-STATEMENT. FOR A 'DE' CALL FUNCTION, YOUR ROUTINE CAN RETURN WITHOUT ANY PROCESSING. IN THIS CASE, //EKYRESLB WILL REMAIN ALLOCATED UNTIL THE END OF THE JOBSTEP.

IN FACT, WE RECOMMEND THAT YOU DO NOT DE-ALLOCATE THE //EKYRESLB DD-STATEMENT, SINCE THIS REDUCES THE AMOUNT OF CODING THAT YOU MUST PROVIDE FOR YOUR OWN EXIT ROUTINE.

2) BE CAREFUL, IFIGNORING ABOVE RECOMMENDATION. YOUR EXIT SHOULD DE-ALLOCATE DURING A 'DE' CALL THE //EKYRESLB DD-STATEMENT ONLY IF THE ALLOCATION HAS ALSO BEEN DONE PREVIOUSLY BY YOUR EXIT. IGNORING THIS WARNING CAN RESULT IN SOME ENVIRONMENTS (FOR EXAMPLE MPP REGIONS) IN DIFFERENT/INCONSISTENT CONSECUTIVE ALLOCATIONS OF //EKYRESLB.

ACTIVATION OF THIS EXIT ROUTINE=

THIS EXIT ROUTINE GETS ACTIVATED BY COMPILING AND LINKING THIS EXIT-Routine INTO THE USUAL JOBLIB/STEPLIB/LINKLIB-CONCATENATION USED FOR THE EXECUTION OF YOUR DPROP JOBSTEPS.

NOTE THAT THE LOAD MODULE NAME OF THIS EXIT ROUTINE MUST BE EKYDAEXO (NOT EKYDAE1A) IN ORDER TO GET INVOKED BY DPROP.

RESTRICTIONS:

THIS EXIT ROUTINE SHOULD NOT PERFORM ANY FUNCTION WHICH IS NOT SUPPORTED IN THE ENVIRONMENT IT EXECUTES (FOR EXAMPLE, IF DPROP IS USED TO PERFORM SYNCHRONOUS PROPAGATION IN MPP REGIONS, THEN THE EXIT ROUTINE SHOULD NOT PERFORM ANY FUNCTION WHICH IS NOT SUPPORTED BY IMS IN A MPP ENVIRONMENT).

REGISTER CONVENTIONS=

R13= ADDRESS OF SAVE AREA
R12= MODULE BASE REGISTER
R11= BAS REGISTER TO CALL SUBROUTINE
R9 = EKYDAE INTERFACE PARAMETER BLOCK
PATCH LABEL = - (NONE)

MODULE TYPE = PROCEDURE
PROCESSOR = ASSEMBLER
MODULE SIZE = LESS THAN 1000 BYTES.
ATTRIBUTES = REENTRANT
RMODE = ANY
AMODE = 31
ENTRY POINT = EKYDAE1A
PURPOSE = SEE FUNCTION
LINKAGE = STANDARD OS/VS ASSEMBLER LINKAGE CONVENTIONS.
INPUT : R1 = POINTING TO A STANDARD PARAMETER ADDRESS LIST.
1ST AND ONLY PARAMETER: EKYDAE CONTROL-BLOCK
OUTPUT : FOR A 'AL' CALL, IF THE //EKYRESLB DD-STATEMENT HAS BEEN ALLOCATED:
- DEALLOC IS SET TO 'Y'

Figure 79 (Part 2 of 12). Sample EKYRESLB Dynamic Allocation Exit
EXIT-NORMAL=

STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.

RETURN CODE = 0

EXIT-ERROR=

STANDARD OS/VS ASSEMBLER RETURN CONVENTIONS.

RETURN CODE = 4

ABEND-CODE OF EKYEDA1A = 1106

ABEND-REASON CODES = X'99999999'

ERROR MESSAGES ISSUED BY EKYEDA1A

EKYEDA1E : FAILURE DURING DYNAMIC ALLOCATION.

EKYEDA2E : INVALID-CALL FUNCTION.

EKYEDA3E : FAILURE DURING DYNAMIC DEALLOCATION.

ADDITIONAL ERROR-MESSAGES MIGHT BE ISSUED BY SVC 99/DYNALLOC.

EXTERNAL REFERENCES

ROUTINES= NONE

CONTROL BLOCKS = DAE INTERFACE CB FOR DYNALLOC EXIT ROUTINE

S99RB SVC 99 REQUEST BLOCK

S99R8X SVC 99 REQUEST BLOCK EXTENSION

S99UNIT SVC 99 TEXT UNITS

MACROS USED FROM MACRO LIBRARY=

SAVE - SAVE REGISTERS

GETMAIN - OS/VS GETMAIN

DYNALLOC - OS/VS SVC 99 CALL

EKYDAE - INTERFACE CONTROL-BLOCK FOR DYNALLOC

EXIT ROUTINE.

IEFZB4D2 - OS/VS SVC 99 DYNALLOC KEYS

CHANGE ACTIVITY= NONE

************ END OF SPECIFICATIONS ************************************

************ LOGIC OF EKYEDA1A *************

MAIN LINE LOGIC:

I) MODULE ENTRY LOGIC:

- PROVIDE REGISTER EQUATES

- GENERATE A MODULE SAVEID

- SAVE REGISTERS AND ESTABLISH MODULE-BASE REGISTER

- LOAD ADDRESSES OF CALL PARAMETER

- GETMAIN AN AREA CONTAINING

- A MODULE SAVE AREA AND MODULE WORKSPACE.

- CLEAR THE GETMAIN AREA.

- CHAIN MODULE SAVE AREA AND SAVE AREA OF CALLER.
2) FOR 'AL' (="ALLOCATE") CALLS

IF THE EKYRESLB DD-STATEMENT IS ALREADY ALLOCATED:
RETURN WITHOUT FURTHER PROCESSING.

IF THE EKYRESLB DD-STATEMENT IS NOT ALREADY ALLOCATED:

PREPARE INFORMATION REQUIRED TO CALL THE MVS DYNALLOC MACRO FOR DYNAMIC ALLOCATION OF THE EKYRESLB DD STATEMENT WITH DISP=SHR.

THE PREPARED DYNALLOC CALL-PARAMETERS REQUEST AMONG OTHER THAT MVS GENERATES AND WRITES ERROR MESSAGES ABOUT DYNALLOC FAILURE.

ISSUE DYNALLOC MACRO.

IF RETURN-CODE FROM DYNALLOC IS NON-ZERO:
-- ISSUE ERROR-MESSAGE
-- BRANCH TO THE RETURN-CODE 4 LOGIC (THIS WILL RESULT IN A ABEND ISSUED BY DPROP).

IF RETURN-CODE FROM DYNALLOC IS ZERO:
-- RECORD THAT THE EKYRESLB DD STATEMENT HAS BEEN ALLOCATED BY EKYEDA1A.
-- BRANCH TO THE RETURN-CODE 0 LOGIC.

3) FOR 'DE' (="DE-ALLOCATE") CALLS

IF IT IS NOT THIS SAMPLE EXIT ROUTINE WHICH ALLOCATED PREVIOUSLY //EKYRESLB:
-- THE EXIT RETURNS WITHOUT FURTHER PROCESSING.

IF IT IS THIS SAMPLE EXIT ROUTINE WHICH ALLOCATED PREVIOUSLY //EKYRESLB:

PREPARE INFORMATION REQUIRED TO CALL THE MVS DYNALLOC MACRO FOR DYNAMIC ALLOCATION OF THE EKYRESLB DD STATEMENT WITH DISP=SHR.

THE PREPARED DYNALLOC CALL-PARAMETERS REQUEST AMONG OTHER THAT MVS GENERATES AND WRITES ERROR MESSAGES ABOUT DYNALLOC FAILURE.

ISSUE DYNALLOC MACRO.

-- BRANCH TO RETURN-CODE 0 LOGIC.

4) RETURN LOGIC

-- FREEMAIN AREA CONTAINING SAVE-AREA AND WORKSPACE.
-- RESTORE REGISTERS OF THE CALLER
-- RETURN TO THE CALLER.

******** END-OF-LOGIC ********

******** MODULE ENTRY LOGIC ********

Figure 79 (Part 4 of 12). Sample EKYRESLB Dynamic Allocation Exit
Figure 79 (Part 5 of 12). Sample EKYRESLB Dynamic Allocation Exit
Chapter 6. EKYRESLB Dynamic Allocation Exit Routine

Figure 79 (Part 6 of 12). Sample EKYRESLB Dynamic Allocation Exit

```
00004A 1801   345  LR  R0,R1  SET UP
00004C 4110 00DF  000F  346  LA  R1,GETML  ...FOR A
000050 18FF  347  SR  R15,R15  ...ZEROING
000052 0E0E  348  MVCL R0,R14  ...MVCL

350 +-------------------------------------------------------------+
351 +                  CHAIN TOGETHER OUR SAVEAREA AND THE HIGHER-LEVEL SAVEAREA +
352 +                  AND LOAD INTO R13 THE ADDRESS OF OUR SAVEAREA +
353 +-------------------------------------------------------------+

000054 5080 0008  00008  355  ST  R11,8(R13)  CHAIN OUR SAVEAREA INTO HIGHER
000058 5088 0004  00004  356  ST  R13,4(R11)  CHAIN HIGHER SAVEAREA INTO OUR
00005C 1800  357  LR  R13,R11  R13=A(OUR SAVEAREA)
000000  358  USING GETM,R13  ESTABLISH BASE REGISTER FOR WORKAREA

360 +-------------------------------------------------------------+
361 +                  BRANCH DEPENDING ON CALL-FUNCTION. +
362 +-------------------------------------------------------------+

00005E D501 9008 C308 00008 0030B  364  CLC DAECALL,=CL2'AL' CALLED TO ALLOCATE EKYRESLB?
000064 4780 0076  00076  365  BE  ALLOC  ...YES>>B
000068 D501 9008 C30A 0000B 0030A  366  CLC DAECALL,=CL2'DE' CALLED TO DE-ALLOCATE EKYRESLB?
000066 4780 0184  00184  367  BE  DEALLOC  ...YES>>B

000072 47F0 02C2  00242  368  B  INVFUNC  CALL-FUNCTION IS INVALID

370 +--------------------------------------------------------------------+
371 +                  THE EXIT IS INVOKED TO PERFORM A DYNAMIC +
372 +                  ALLOCATION OF THE EKYRESLB DD STATEMENT +
373 +--------------------------------------------------------------------+
374 +                  'AL' (ALLOCATE) CALL +
375 +                  THE EXIT IS INVOKED TO PERFORM A DYNAMIC +
376 +                  ALLOCATION OF THE EKYRESLB DD STATEMENT +
377 +--------------------------------------------------------------------+
378 +--------------------------------------------------------------------+
379 +--------------------------------------------------------------------+
380 +--------------------------------------------------------------------+

000076  382  ALLOC  DS  OH
000076 95E8 900A  0000A  383  CLI DAELALLOCC,DAEALLOY  EKYRESLB ALREADY ALLOCATED?
00007A 4770 0088  00088  384  BNE  ALLOC20  ...NO>>LETS DO THE ALLOCATION
00007E D700 9020 9020 00020 00020  385  XC  OURALLO,OURALLO  CLEAR OURALLO FLAG
000084 47F0 0294  00294  386  B  RETURN0  ...AND RETURN

388 +--------------------------------------------------------------------+
389 +                   PERFORM THE DYNAMIC ALLOCATION BY PREPARING: +
390 +                   - DYNALLOC REQUEST BLOCK +
391 +                   - DYNALLOC REQUEST BLOCK EXTENSION +
392 +                   - DYNALLOC TEXT UNITS +
393 +                   AND BY CALLING THE DYNALLOC MACRO. +
394 +                   +
395 +                   PLEASE REFER TO MVS/ESA DOCUMENTATION, IF YOU NEED +
396 +                   EXPLANATIONS ON THIS SUBJECT (SEE 'MVS/ESA APPLICATION +
397 +                   DEVELOPMENT GUIDE: AUTHORIZED ASSEMBLER LANGUAGE +
398 +                   PROGRAMS (GC28-1645)' CHAPTER 'REQUESTING SVC 99 +
399 +                   FUNCTIONS'). +
400 +--------------------------------------------------------------------+

000088  402  ALLOC20  DS  OH

404 --------  PREPARE SVC 99 REQUEST BLOCK

000088 D792 D04C D04C 0004C 0004C  406  XC  Z99RB(Z99END-Z99RB),Z99RB CLEAR SVC 99 AREA
00008E 4110 D04C  0004C  408  LA  R1,Z99RB  GET POINTER TO RB AREA
000092 5010 D048  00048  409  ST  R1,Z99RBPTR  STORE ADDRESS INTO RBPTR
000096 9680 D048  00048  410  OI  Z99RBPTR,'X'80'  AND TURN HIGH BIT ON
00009A 9214 D04C  0004C  411  MVI Z99RLN,Z99RBEND-Z99RB  SETUP CB LENGTH
```

Chapter 6. EKYRESLB Dynamic Allocation Exit Routine  307
Figure 79 (Part 7 of 12). Sample EKYRESLB Dynamic Allocation Exit
**478** *DYNAMIC ALLOCATION FAILED.*

**479** +

**480** *

**481** + *ISSUE ERROR-MESSAGE.*

**482** + *BRANCH TO RETURN WITH A RETURN-CODE 4.*

**483** *

---

**485** ALERROR DS OH

**486** WTO 'EKYEDA1A: DYNAMIC ALLOCATION OF //EKYRESLB FAILED', C ROUTCDE=11

---

**511** DEALLOC DS OH

**512** TM DURALLO,DURALLOY DID WE ALLOCATE?

**513** BZ RETURN.....NO>>>RETURN

---

**526** ------- PREPARE SVC 99 REQUEST BLOCK

**528** XC Z99RB(Z99END-Z99RB),Z99RB CLEAR SVC 99 RB/PARMS

---

**00013E**

---

**000184**

---

**00018B 47F0 C29C 0029C 497** B RETURN4

---

**00018C D792 D04C D04C 0004C 0004C 528** LA R1,Z99RB GET POINTER TO RB AREA

---

**000192 4110 D04C 0004C 530** ST R1,Z99RBPTR STORE ADDRESS INTO RBPTR

---

**000196 5010 D04B 00048 531**

**00019A 9680 D048 00048 532**

**00019C 9214 D04C 0004C 533**

**0001A2 9202 D04D 0004D 534**

**0001A6 9200 D04E 0004E 535**

**0001AA 4110 D084 00084 536**

**0001AE 5010 D054 00054 537**

**0001B2 4110 D060 00060 538**

**0001B6 5010 D058 00058 539**

---

**541** ------- PREPARE REQUEST BLOCK EXTENSION

---

**548** ------- PREPARE POINTER LIST TO UNIT TEXT-UNITS

---

**0001CC 4110 D090 00090 550**

**0001DD 5010 D084 00084 551**

**0001D4 9680 D084 00084 552**

---

**Figure 79 (Part 8 of 12). Sample EKYRESLB Dynamic Allocation Exit**
554 ------- PREPARE DDNAME TEXT UNIT

000108 D201 D090 C312 00090 00312 556 MVC Z99T1KEY,=AL2(DALDDNAM) SETUP UNIT KEY
00010E D201 D092 C314 00092 00314 557 MVC Z99T1NUM,=AL2(1) SETUP NUMBER OF ENTRIES
00010A D201 D094 C316 00094 00316 558 MVC Z99T1LNG,=AL2(8) SETUP PARM LENGTH
000110 D207 D096 C200 00096 00200 559 MVC Z99T1DDN(8),=CL8'EKYRESLB' SETUP DDNAME IN PARM

561 ------- LETS CALL SVC 99 FOR DYNAMIC DE-ALLOCATION

0001F0 4110 D048 00048 563 LA R1,Z99RBPTR R1=A(RB-POINTER)
0001F4 48 D2mzqr!z!t 564 DYNALLOC , AND CALL SVC 99

0001F6 12FF 568 LTR R15,R15 DE-ALLOCATION OK?
0001F8 478mzqr!z!t C294 569 BZ RETURNmzqr!z!t ...YES>>>RETURN

571 *--------------------------------------------------------------------*
572 * DYNAMIC DE-ALLOCATION FAILED *
573 * *
574 * - ISSUE ERROR-MESSAGE. *
575 * - BRANCH TO RETURN WITH A RETURN-CODE 0 *
576 * (SINCE DE-ALLOCATION FAILURES DO NOT PREVENT *
577 * SUCCESSFUL DPROP OPERATIONS, IT IS BY PURPOSE THAT WE *
578 * RETURN WITH RC=0 -- AS OPPOSED TO RC=4). *
579 *--------------------------------------------------------------------*

581 WTO 'EKYEDA3E: DYNAMIC DE-ALLOCATION OF //EKYRESLB FAILED', C
582 ROUTCDE=11

00023E 47F0 C294 00294 592 B RETURN0 RETURN WITH ZERO RC
594 *--------------------------------------------------------------------*
595 *--------------------------------------------------------------------*
596 *--------------------------------------------------------------------*
597 **** ***
598 *** INVALID CALL FUNCTION IN DAECALL. ***
599 **** ***
600 *--------------------------------------------------------------------*
601 *--------------------------------------------------------------------*
602 *--------------------------------------------------------------------*

000242 604 INVFUNC DS 0H
605 WTO 'EKYEDA3E: INVALID CALL-FUNCTION FOR EKYEDA1A', C
606 ROUTCDE=11

00027E 5860 C304 00304 616 L R6,=X'99999999' R6= ABEND REASON CODE
616 ABEND 1106,REASON=(R6),DUMP
627 *--------------------------------------------------------------------*
628 *--------------------------------------------------------------------*
629 *--------------------------------------------------------------------*
630 **** ***
631 **** RETURN TO CALLER OF EXIT ***
632 **** - RETURN TO CALLER OF EXIT ***
633 **** ***
634 *--------------------------------------------------------------------*
635 *--------------------------------------------------------------------*
636 *--------------------------------------------------------------------*
637 *--------------------------------------------------------------------*

000294 639 RETURN0 DS 0H
000294 41F0 0000 00000 640 LA R15,0 LOAD 0 AS RETURN-CODE
000298 47F0 C2A0 002A0 641 B RETURN99

00029C 643 RETURN0 DS 0H
00029C 41F0 0004 00004 644 LA R15,4 LOAD 4 AS RETURN-CODE

Figure 79 (Part 9 of 12). Sample EKYRESLB Dynamic Allocation Exit
**Figure 79 (Part 10 of 12). Sample EKYRESLB Dynamic Allocation Exit**
Figure 79 (Part 11 of 12). Sample EKYRESLB Dynamic Allocation Exit
Figure 79 (Part 12 of 12). Sample EKYRESLB Dynamic Allocation Exit
Chapter 7. TSMF Callable Interface

This Chapter describes the timestamp marker facility (TSMF) callable interface. This interface is used with LOG-ASYNC.

The timestamp marker facility (TSMF) callable interface allows a user application program to create a stop timestamp marker (TSM) for one or more propagation groups. Refer to *IMS DPROP Reference* for details on the use of TSMs.

The user application program can pass the stop timestamp in ISO/DB2 format (local time) or in MVS TOD time format (GMT time).

The user application program must include the object module EKYT099X in its link-edit. EKYT099X is an assembler module provided with IMS DPROP to dynamically link the user application program with the TSMF. This means that if changes are made to IMS DPROP, the user application program does not need to be relinked.

The TSMF callable interface provides an alternative to the SCU for creating group stop timestamps. The JCL to run the user application program should fulfill the requirements for the JCL used to run the SCU with the CREATETSM STOP control statement. Refer to *IMS DPROP Reference* for details on using the SCU.

TSMF Callable Interface Parameters

The user application program invokes one of two entry points within EKYT099X, depending on the format of timestamp being passed:

- EKYB097X (RC,TODTIME,ID,GRPLIST_COUNT,GRPLIST_ARRAY)
- EKYB098X (RC,DB2TIME,ID,GRPLIST_COUNT,GRPLIST_ARRAY)

The parameters to be used when you call EKYT099X are detailed in Figure 80.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>No. of Bytes</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>BIN(31)</td>
<td>4</td>
<td>Passes the return code back to the caller.</td>
</tr>
<tr>
<td>DB2TIME</td>
<td>BIT(64)</td>
<td>8</td>
<td>Contains ISO/DB2 format timestamp.</td>
</tr>
<tr>
<td>TODTIME</td>
<td>CHAR(26)</td>
<td>26</td>
<td>Contains MVS TOD format timestamp.</td>
</tr>
<tr>
<td>ID</td>
<td>CHAR(8)</td>
<td>8</td>
<td>Contains the timestamp ID, left aligned, padded with blanks. All blanks means that an ID is not supplied.</td>
</tr>
<tr>
<td>GRPLIST_COUNT</td>
<td>BIN(31)</td>
<td>4</td>
<td>Contains the number of group IDs.</td>
</tr>
<tr>
<td>GRPLIST_ARRAY</td>
<td>CHAR(8)</td>
<td>8</td>
<td>Array of group IDs. Each group ID is 8 bytes, left-aligned, padded with blanks.</td>
</tr>
</tbody>
</table>
To call the callable interface from PL/I you must declare the assembler module (EKYT97X or EKYT98X) as an external module and then declare the variables and array used in the call to the TSMF callable interface. Refer to Figure 81 for an example of these declarations.

```plaintext
/* DECLARE THE ASSEMBLER MODULE (EKYT98X) AS AN EXTERNAL MODULE. */
DCL EKYT98X ENTRY
  (FIXED BIN(31), /* return code */
   CHAR(26), /* USERTIME in ISO/DB2 format */
   CHAR(8), /* TSMID */
   FIXED BIN(31), /* Count of group IDs */
   (3) CHAR(8)) /* Array of group IDs, in this case 3 */
EXTERNAL OPTIONS(ASSEMBLER INTER);

/* DECLARE LOCAL VARIABLES */
DCL CURRENT_TSTAMP CHAR(26) INIT('1993-01-01-00.00.00.000000');

/* DECLARE THE VARIABLES USED IN THE CALL TO THE TSMF CALLABLE INTERFACE. */
DCL 1 TSMF, 2 RC FIXED BIN(31) INIT(0000),
2 USERTIME CHAR(26) INIT('0001-01-01-01.01.01.000001'),
2 TSMID CHAR(8) INIT('DEFAULT '),
2 GRPLIST_COUNT FIXED BIN(31) INIT(0003);

/* DECLARE THE ARRAY USED IN THE CALL TO THE TSMF. */
/* IN THIS EXAMPLE AN ARRAY WITH THREE GROUP NAMES IS USED. YOU CAN */
/* DEFINE AS MANY GROUP NAMES AS YOU WISH BY MAKING THE ARRAY BIGGER */
/* AND PASSING THE NUMBER OF GROUP NAMES IN THE GRPLIST_COUNT VARIABLE. */
DCL GRPLIST (3) CHAR(8) INIT('GROUP01 ','GROUP02 ','GROUP3 ');
```

Figure 81. TSMF Callable Interface, Declarations for PL/I
Figure 82 is an example of how to call the TSMF callable interface from a PL/I program by using EKYT98X. The PL/I program uses a timestamp which is in DB2 format.

```pli
/* SET THE VARIABLE VALUES AS APPROPRIATE TO YOUR SITUATION */
TSMF.USERTIME = CURRENT_TIMESTAMP;

/* CALL THE TSMF CALLABLE INTERFACE PASSING A TIMESTAMP IN DB2 FORMAT */
CALL EKYT98X(TSMF.RC,
              TSMF.USERTIME,
              TSMF.TSMID,
              TSMF.GRPLIST_COUNT,
              GRPLIST);

/* CHECK THE RETURN CODE FROM THE TSMF CALLABLE INTERFACE AND HANDLE ANY ERRORS WHICH OCCUR. */
IF TSMF.RC != mzqr!z!t THEN
  DO;
    /* handle error */
  END;
END;
```

Figure 82. TSMF Callable Interface, Call from a PL/I Program
Calling the TSMF Callable Interface from COBOL

To call the callable interface from COBOL you must declare the local variables and then declare the variables used in the call to the TSMF callable interface. Refer to Figure 83 for an example of this.

```cobol
* DECLARE LOCAL VARIABLES
WORKING-STORAGE SECTION.
01 CURRENT-TSTAMP PIC(26) VALUE '1993-01-01-00.00.00.000000'.
01 .........
01 .........

* DECLARE THE VARIABLES USED IN THE CALL TO THE TSMF CALLABLE INTERFACE.
* IN THIS EXAMPLE THREE GROUP NAMES ARE SPECIFIED IN THE VARIABLE TS-GROUPS. YOU CAN DEFINE AS MANY GROUP NAMES AS YOU WISH BY MAKING THE VARIABLE TS-GROUPS BIGGER AND PASSING THE NUMBER OF GROUP NAMES IN THE TS-GROUP-COUNT VARIABLE.

01 TSMF-PARAMETERS.
  03 TS-RETURN-CODE PIC 9(8) COMP VALUE ZERO.
  03 TS-USERTIME PIC X(26) VALUE '1994-02-17-13.00.00.000000'.
  03 TS-TSMID PIC X(8) VALUE 'TSM000003'.
  03 TS-GROUP-COUNT PIC 9(8) COMP VALUE 3.
  03 TS-GROUPS PIC X(24) VALUE 'GROUP01 GROUP02 GROUP03'.

Figure 83. TSMF Callable Interface, Declarations for COBOL

Figure 84 is an example of how to call the TSMF callable interface from a COBOL program by using EKYT98X. The PL/I program uses a timestamp which is in DB2 format.

* SET THE VARIABLE VALUES AS APPROPRIATE TO YOUR SITUATION
TS-USERTIME = CURRENT-TSTAMP.

* CALL THE TSMF CALLABLE INTERFACE PASSING A TIMESTAMP IN DB2 FORMAT
CALL 'EKYT98X' USING BY REFERENCE
  TS-RETURN-CODE,
  TS-USERTIME,
  TS-TSMID,
  TS-GROUP-COUNT,
  TS-GROUPS.

IF TS-RETURN-CODE NOT EQUAL ZERO THEN
  ** handle error **
```

Figure 84. TSMF Callable Interface, Call from a COBOL Program
## Return Codes from the TSMF Callable Interface

The TSMF Callable Interface provides the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful creation of stop timestamp for group(s)</td>
</tr>
<tr>
<td>4</td>
<td>Warning message has been issued. One or more of the groups may already have a group stop timestamp equal to the timestamp passed by the user application.</td>
</tr>
<tr>
<td>8</td>
<td>Error message has been issued. The group stop timestamp is not created. This result occurs when there is insufficient information supplied to the callable interface. For example: <em>Unable to Open SCF</em> means that the user did not supply the //EKYSCF DD statement.</td>
</tr>
<tr>
<td>12</td>
<td>Error message has been issued. Invalid parameter passed by the user application.</td>
</tr>
<tr>
<td>16</td>
<td>Error message has been issued. The group stop timestamp was not created, probably because of an internal DPROP error. It is unlikely that this error would occur as a result of invalid data supplied by the user application.</td>
</tr>
<tr>
<td>20</td>
<td>Error message has been issued. The group stop timestamp was not created. This error can probably be traced to environmental considerations that are not specific to the request. For example: <em>Out of Storage</em> means that the request would complete normally if there was only one user or if sufficient resources were supplied to the system.</td>
</tr>
</tbody>
</table>
Chapter 8. EMF Callable Interface

The event marker facility (EMF) callable interface allows a user application program to create an event marker (EM) for one or more Propagation Data Streams. Refer to IMS DPROP Reference for details on the use of EMs.

The EMF callable interface provides an alternative to the Capture System Utility (CUT) for creating Event Markers. The JCL to run the user application program should fulfill the requirements for the JCL used to run the CUT with the EM control statement. Refer to IMS DPROP Reference for details on using the CUT.

Note: when the EMF callable interface is called by IMS Batch Application Programs or by a non-IMS Batch Application Programs that issue their own MQSeries calls, then these Application Programs:

- must issue their MQSeries calls through the use of the CSQBRSTB batch stub of MQSeries (this is the RRS batch stub, that provides a RRS-based two phase commit coordination between multiple Resource Managers).
- should not issue following types of MQSeries calls: MQCMIT and MQBACK.

EMF Callable Interface Parameters

The user application program invokes EKYI950X as follows:

EKYI950X (RC,RESERVD,ID,PRSTREAM_COUNT,PRSTREAM_ARRAY)

The parameters to be used when you call EKYI950X are detailed in Figure 85.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>No. of Bytes</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>BIN(31)</td>
<td>4</td>
<td>Passes the return code back to the caller.</td>
</tr>
<tr>
<td>RESERVD</td>
<td>CHAR(26)</td>
<td>26</td>
<td>A reserved Field</td>
</tr>
<tr>
<td>ID</td>
<td>CHAR(8)</td>
<td>8</td>
<td>Contains the Event Marker ID, left aligned, padded with blanks.</td>
</tr>
<tr>
<td>PRSTREAM_COUNT</td>
<td>BIN(31)</td>
<td>4</td>
<td>Contains the number of PRSTREAM Names.</td>
</tr>
<tr>
<td>PRSTREAM_ARRAY</td>
<td>CHAR(8)</td>
<td>8</td>
<td>Array of PRSTREAM Names. Each PRSTREAM Name is 8 bytes, left-aligned, padded with blanks.</td>
</tr>
</tbody>
</table>
Calling the EMF Callable Interface from COBOL

To call the callable interface from COBOL you must declare the local variables and then declare the variables used in the call to the EMF callable interface. Refer to Figure 86 for an example of this.

```cobol
* DECLARE LOCAL VARIABLES
* WORKING-STORAGE SECTION.
  01 .................
  01 .................

* DECLARE THE VARIABLES USED IN THE CALL TO THE EMF CALLABLE INTERFACE.
* IN THIS EXAMPLE THREE PRSTREAM NAMES ARE SPECIFIED IN THE TS-PRSTREAMS. YOU CAN DEFINE AS MANY PRSTREAM's AS YOU WISH BY MAKING THE VARIABLE TS-PRSTREAMS BIGGER AND PASSING THE NUMBER OF PRSTREAM NAMES IN THE TS-PRSTREAM-COUNT VARIABLE.

  01 TS-RETURN-CODE PIC 9(8) COMP VALUE ZERO.
  03 TS-PRSTREAM-COUNT PIC 9(8) COMP VALUE 3.
  03 TS-EMID PIC X(8) VALUE 'EMmzqr!z!t3'.
  03 TS-PRSTREAMS PIC X(24) VALUE 'PRSTRmzqr!z!t1 PRSTRmzqr!z!t2 PRSTRmzqr!z!t3'.

  03 TS-RESERVD PIC X(26) VALUE ' '.

Figure 86. EMF Callable Interface, Declarations for COBOL

Figure 87 is an example of how to call the EMF callable interface from a COBOL program by using EKYI950X.

```cobol
* CALL THE EMF CALLABLE INTERFACE

  CALL 'EKYI950X' USING BY REFERENCE
  TS-RETURN-CODE,
  TS-RESERVD,
  TS-EMID,
  TS-PRSTREAM-COUNT,
  TS-PRSTREAMS.

  IF TS-RETURN-CODE NOT EQUAL ZERO THEN
    * handle error *

Figure 87. EMF Callable Interface, Call from a COBOL Program

Return Codes from the EMF Callable Interface

The EMF Callable Interface provides the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful creation of Event Marker.</td>
</tr>
</tbody>
</table>
Warning: the requested Event Marker has not been created, for example, because the IMS DPROP Capture System is in emergency stopped status or because the Jobstep executes with a 'PROP OFF' Control Statement in the //EKYIN File.

Error: the requested Event Marker has not been created, for example, because a specified PRSTREAM name is not defined in the //EKYTRANS File.
Chapter 9. User-Implemented Asynchronous Data Propagation (USER-ASYNC)

IMS DPROP Version 3 supports two methods of asynchronous propagation:
MQ-ASYNC and LOG-ASYNC. These methods of asynchronous propagation are fully described in the appropriate Administrators Guide for your propagation mode.

This chapter describes a third method of asynchronous propagation:
USER-ASYNC. USER-ASYNC propagation is implemented by combining IMS DPROP components with user-provided programs. USER-ASYNC propagation was previously documented in the IMS DataPropagator for OS/390 and z/OS library when IMS DataPropagator for OS/390 and z/OS did not support either MQ-ASYNC nor LOG-ASYNC. With the advent of MQ-ASYNC and of LOG-ASYNC, you will no longer be required to develop programs to implement your own USER-ASYNC solutions. Instead, you can use MQ-ASYNC or LOG-ASYNC methods.

However, if you still want to implement your own solution, this chapter outlines what is required to develop a USER-ASYNC solution.

User asynchronous propagation can be based on either of the following:

- The IMS Asynchronous Data Capture function to harden the data on the log
- A user-written IMS Data Capture exit routine to capture the data and harden it

Refer to the following for information on the IMS Asynchronous Data Capture function and user-written IMS Data Capture exit routines:

- IMS/ESA Administration Guide: Database Manager
- IMS/ESA Utilities Reference: Database Manager
- IMS/ESA Customization Guide

For a detailed description of the log records written by the IMS Asynchronous Data Capture function, see:

- IMS/ESA Customization Guide

Implementation Based on IMS Asynchronous Data Capture Function

IMS application programs update the IMS databases. The IMS Asynchronous Data Capture function writes the changed data to the IMS log.

Later, a program that you write gathers the changed data from the IMS log data sets. This program is often referred to as the selector. It selects and gathers changed data to be propagated from all those IMS logs that contain changed data. It makes the changed data available (in sequential files, for example) for processing by another program that you write, the receiver.

When you want to apply the updates, the receiver accesses or receives the changed data, and calls the RUP to update the DB2 table.

The selector and receiver are discussed in more detail in “Writing A Selector Program” on page 326 and “Writing A Receiver Program” on page 327.

Figure 88 on page 323 provides an overview of this implementation.
Figure 88. HR Asynchronous Propagation With the IMS Asynchronous Data Capture Function

An implementation based on the IMS Asynchronous Changed Data Capture function supports propagation of updates performed by IMS application programs executing in the following environments:

- IMS Fast Path Regions
- IMS MPP Regions
- IMS BMP Regions
- IMS Batch Regions
- CICS (only when executing with DBCTL)

Implementation Based on User-Written IMS Data Capture Exit

IMS application programs update the IMS databases. The IMS data capture function provides the changed data to your IMS Data Capture exit routine, which is referred to as the sender program. Your sender program must either store the IMS updates until you want to apply the updates to the DB2 table, or send them directly to the receiver. When you want to apply the updates, the receiver accesses or receives the changed data, and calls the RUP to update the DB2 table. The sender and receiver are discussed in more detail in “Writing A Selector Program” on page 326 and “Writing A Receiver Program” on page 327.

Figure 89 on page 324 provides an overview of this implementation.
An implementation based on a user-written IMS Data Capture exit routine supports propagation of updates performed by IMS application programs executing in the following environments:

- IMS Fast Path Regions
- IMS MPP Regions
- IMS BMP Regions
- IMS Batch Regions

This implementation does not support propagation of updates performed by IMS application programs executing in a CICS environment.

**Developing Your Asynchronous System**

This section explains how you can develop your asynchronous system.

**Setting Up Your Asynchronous System**

You must determine the exact processes that the selector, sender, and receiver use to call the RUP asynchronously.

Because the IMS Data Capture function does not call the RUP directly, your programs must provide several processing features that are described in this section. Keep these features in mind while developing your asynchronous system.
Calling the RUP
The fact that data propagation is asynchronous must be invisible to the RUP. That is, your programs must call the RUP in exactly the same way as the IMS Data Capture function during synchronous propagation. Therefore, if you develop asynchronous propagation based on an IMS Data Capture exit routine, your sender must record all the information passed to it by the IMS Data Capture Function. If you develop asynchronous propagation based on the IMS Asynchronous Changed Data Capture Function, your selector must record all the information available in the IMS log records containing changed data. The receiver must call the RUP using this information exactly as the IMS Data Capture function uses it. This is discussed further in “Writing A Receiver Program” on page 327.

Programming languages supported
The RUP can be called from a program written in Assembler, COBOL, PL/I, or C languages. However, this too must be transparent to the RUP. Support of COBOL and PL/I programs assumes that the RUP can function as though it were called by a program written in Assembler.

Handling the Changed Data
While you want your programs to be efficient and provide a reasonable throughput, your programs must ensure that the propagated data changes are presented to the RUP in the correct sequence. Your programs must also avoid losing propagated data, or propagating changes multiple times. These situations cause inconsistencies between the IMS data and DB2 data.

When called for asynchronous propagation, the RUP always propagates IMS inserts, including those made with an IMS processing option load. Your programs must filter out the inserts that you do not want propagated.

Your programs must also provide some operational support; for example, avoid losing changed data in both normal and abnormal situations.

Propagation Failures
With asynchronous data propagation, failures do not automatically trigger a coordinated backout of the IMS update and the DB2 updates. If you encounter a propagation failure, the RUP signals the failure to your calling receiver program. The RUP does not perform a rollback.

Your receiver program must provide the logic to handle any propagation failures that can occur. The receiver must not call the RUP after a propagation failure until the problem is fixed. This can cause many more data inconsistencies and propagation failures.

More information on error handling for the receiver is discussed in “Writing A Receiver Program.” Also, because the RUP can abend, your programs must provide restart logic.

You can also provide trace and audit capabilities for those parts of your system that the DPROP tracing functions do not trace.
Sync Point Processing
Your asynchronous propagation system must perform its own sync point processing. You can begin processing after completing each original unit of work (UOW).

Splitting the IMS Data
The sender and selector store IMS data. To increase efficiency, split this data into parts. Then call multiple copies of the receiver in parallel. Each copy of the receiver is called in a distinct address space to process its portion of the IMS data.

You can split the IMS data many different ways. Examples include splitting the data by DBD name, segment type, key range values of the root segment, and so on.

Writing A Selector Program
The selector gathers the log records containing changed data from the IMS log datasets, and makes the data available for receiver processing.

If multiple IMS subsystems are updating the same databases, the selector needs to merge the IMS log records containing changed data in a sequence consistent with that in which IMS generated them. If your selector and receiver programs maintain the correct sequence, this ensures that your data remains consistent between IMS and DB2.

The selector creates output data sets containing the changed data to be propagated. If the receiver executes on a different remote MVS system, the output data sets can be transmitted with file transfer programs.

Processing log records containing changed data requires a detailed understanding of the format of these log records. Refer to IMS/ESA Customization Guide for a detailed description of these log records.

Writing A Sender Program
The sender program is defined to IMS in the DBDGEN as an IMS Data Capture exit routine. The sender stores the propagated data changes and IMS Data Capture interface information, or sends this information directly to the receiver. If the IMS updates are made in an IMS online environment, the sender can continuously send the updates to the receiver or a remote destination by inserting the data into IMS output messages; or, if the sender and receiver are on different MVS systems, the messages can be sent across MSC or ISC links. If you plan to temporarily store the updates before sending them to the receiver, you can store them in the:

- IMS log
- IMS full-function database
- DEDB sequential dependent segments
- MVS flat file

If you store the changed data on the IMS log, use the Remote Recovery Data Facility (RRDF) when you send the data to a remote destination or to the receiver. For more information on RRDF, refer to RRDF Program Description and Operations.
Remember to present the updates to the RUP in a sequence consistent with that
which IMS created. If your sender and receiver programs maintain the correct
sequence, your data remains consistent between IMS and DB2.

See IMS/ESA Customization Guide for details on the IMS Data Capture interface
that IMS uses to call the sender. Also, see the next section for more details on
duplicating the interface to call the RUP.

Writing A Receiver Program

The receiver program receives the changed data information from the sender or
selector, and calls the RUP to update the propagated DB2 table. You can write the
receiver program in Assembler, COBOL, C, or PL/I. Whatever language you
choose, the RUP must run as though it were called from an Assembler program.

Your receiver must provide the necessary JCL to call the RUP. Also, because the
RUP accesses the DPROP directory, you must provide a usable DB2 plan. If you
are using DPROP for both synchronous and asynchronous data propagation, you
must generate two DPROP systems.

The fact that data propagation is asynchronous must be invisible to the RUP.
Therefore, your receiver must duplicate the IMS Data Capture interface. This and
other requirements for the receiver are discussed in more detail below.

For each job step in which your receiver program calls the RUP, the receiver must
generate:

1. One initialization call to the housekeeping module EKYZ800X. This module
initializes the DPROP environment.
2. One call to the RUP for each changed data segment. Again, the receiver must
provide all the IMS Data Capture interface information.
3. Termination calls to the housekeeping module to complete DPROP activities.

When calling the RUP, it is very important to present the updates to the RUP in a
sequence that ensures the consistency of your data between IMS and DB2. To do
this, present the updates to the RUP in the same sequence as that in which IMS
created them. The receiver must maintain the concept of the original unit of work
(UOW) while presenting updates to the RUP.

Interface Used to Call the RUP

Your receiver program must duplicate the IMS Data Capture Interface to call the
RUP. The IMS Data Capture interface consists of eight parts; one part is a
parameter that is passed to the RUP, and this part contains pointers to the other
parts. The parameter and pointers include:

1. A DL/I XPCB control block containing pointers to the next parts
2. One or more DL/I XSDB control blocks
3. The output of a DL/I INQY call, which IMS created when the DL/I data changed
4. The fully concatenated key of the changed DL/I segment
5. The changed DL/I segment (for replace calls, both the before- and
   after-images)
6. The hierarchical parent and ancestors of the changed segment, if the IMS data capture function provided them

7. The DBD version ID

8. A 256-byte area reserved for the RUP

The RUP is called with only one parameter: the XPCB. It contains both a description of the data change and pointers to the other information listed above. When your receiver program calls the RUP, the receiver program must provide the XPCB parameter. The XPCB must have the same format and content as when used in the IMS Data Capture interface.

Your receiver must also provide the RUP with access to all the other information in the list above. For example, you must provide the same XSDB control blocks that IMS provided to the Data Capture exit routine (the sender) when the segment was changed; all of the XSDB fields must be filled in before calling the RUP. You must provide the same INQY call output.

Observe the following conventions for the 256-byte area pointed to by the XPCB:

- The receiver must initialize this area with binary zeroes before the first call to the RUP, and must not change its content afterward.
- If your receiver program uses more than one XPCB copy, then each XPCB copy must point to the same copy of the 256-byte area.

This interface information is described in detail in *IMS/ESA Customization Guide*. The return and reason codes that the RUP returns to your program are discussed in “Error Handling” on page 337.

To make maintenance and migration activities easier, avoid link-editing your receiver program with the RUP. Instead, the receiver must call the RUP dynamically. If the receiver is written in Assembler:

1. Generate an MVS LOAD macro to load the RUP (EKYRUP00) and save its entry point address.
2. Provide all interface information.
3. Branch to the RUP entry point (using a BASR).

If the receiver is written in COBOL, you can call the RUP dynamically. Use a Call Identifier statement.

If the receiver is written in PL/I, refer to PL/I documentation for the interface used to call an Assembler program.

**XPCB and XSDB Interfaces:** The XPCB is the only parameter passed by your receiver to the RUP. It is used to provide information about the changed data and to point to XSDBs. An XSDB points to, and describes, either a changed segment occurrence or a physical ancestor of a changed segment.

IMS defines the XPCB and the XSDB control blocks.
As shown by the numbered sections of the figure, the interface consists of:

1. One XPCB control block, which provides a description of the changed data and contains various pointers.

2. A pointer to the fully concatenated key of the changed segment.

3. A pointer to the XSDB control block describing the changed segment. This XSDB points to the data of the changed segment.

4. For Replace operations, a pointer to an XSDB describing the segment before it was replaced. The XSDB also points to the data of the before image of the segment.
5. A pointer to the first XSDB in a chain of XSDBs for the hierarchical ancestors of the changed segment. The chain is in descending hierarchical order, with each XSDB pointing to the segment data of the segment itself and to the next XSDB in descending order.

6. A pointer to the DBD version ID.

7. A pointer to an area containing the output of an implied IMS INQY ENVIRON call.

8. A pointer to a 256-byte area reserved for RUP-usage.

**The XPCB Control Blocks:** Figure 91 on page 331 shows the DSECT for the XPCB. In the figure, each field is marked with a note number, which refers to a note (located after the figure) describing how the receiver should set the field.

You can generate the XPCB control block (together with the XSDB and the output area of an IMS INQY call) by coding the EKYRCDL1 macro statement.
Notes:

1. Before calling the RUP, the receiver should set:
   a. Blanks in the XPCB fields
   b. Binary zeroes in the XPCB fields

2. Before calling the RUP, the receiver should initialize the following fields of the XPCB with constants as follows:
   - **XPCBEYE**: Should be initialized with the value XPCB
   - **XPCBVER**: Should be initialized with the value V1
   - **XPCBREL**: Should be initialized with the value R1

3. Before calling the RUP, the receiver should set the XPCB fields identified with 3, to the value provided by IMS, either:
   - In the XPCB, when calling your user-written IMS Data Capture exit routine
In the CAPD block of the changed data capture IMS log records, if using the IMS Asynchronous Data Capture function

4. **XPCBVERA** (pointer to the DBD Version ID):

Before calling the RUP, the receiver should provide in this field a pointer to a variable-length character string that contains the DBD version. Unless the character string is set from the DBD VERSION= keyword, it will be the time stamp of the DBDGEN. The first two bytes are a halfword containing the length of the string, and are followed by the string itself.

The DBD Version ID must have the same value provided by IMS either:

- Via the XPCBVERA pointer, when calling your user-written IMS Data Capture exit routine
- In the changed data capture IMS log record, if using the IMS Asynchronous Data Capture function

5. **XPCBINQA** (pointer to INQY ENVIRON output area):

Before calling the RUP, the receiver should provide in this field a pointer to an area that has the same layout as the output area of a INQY ENVIRON DL/I call. See “The INQY ENVIRON output area” on page 335 for more information on the output area.

6. **XPCBCKEYA** (pointer to the fully concatenated key)

Before calling the RUP, the receiver should provide in this field a pointer to the fully concatenated key of the changed IMS segment.

The fully concatenated key must have the same value provided by IMS either:

- Via the XPCBCKEYA pointer, when calling your user-written IMS Data Capture exit routine
- In the changed data capture IMS log record, if using the IMS Asynchronous Data Capture function

7. **XPCBXSDBD** (pointer to the XSDB describing changed segment):

Before calling the RUP, the receiver should provide in this field a pointer to the XSDB control block describing the changed IMS segment.

Your receiver should set this field to zero before calling RUP if IMS does not provide a description of the changed IMS segment:

- In an XSDB control block, when calling your user-written IMS Data Capture exit routine, or
- In a CAPD_DATA block in the changed data capture IMS log records

8. **XPCBXSDBB** (pointer to the XSDB describing the “before-image” of the changed segment):

Before calling the RUP, the receiver should provide in this field a pointer to the XSDB control block describing the before-image of the changed IMS segment.

Your receiver should set this field to zero before calling RUP if IMS does not provide a description of the before-image of the changed IMS segment either:

- In an XSDB control block, when calling your user-written IMS Data Capture exit routine, or
- In a CAPD_DATA block within the changed data capture IMS log records
9. **XPCBXSDBP** (pointer to the first XSDB describing the segments in the hierarchic path, in descending order, above the changed segment):

Before calling the RUP, the receiver should provide in this field a pointer to the XSDB control block that describes the first segment in the hierarchic path above the changed segment.

Your receiver should set this field to zero before calling RUP if IMS does not provide a description of the segments in the hierarchic path above the changed segment either:

- In XSDB Control blocks when calling your user-written IMS Data Capture exit routine, or
- In CAPD_DATA blocks within the changed data capture IMS log records.

10. **XPCBEXIWP** (pointer to a 256-byte area reserved for RUP):

Before calling the RUP, the receiver should provide in this field a pointer to a 256-byte area reserved for RUP usage.

Your receiver should observe the following conventions for the 256-byte that area the XPCB points to:

- The receiver must initialize this area with binary zeroes before the first call to the RUP.
- The receiver must not change its content afterward.

If your receiver uses more than one XPCB copy, then each XPCB copy should point to the same copy of the 256-byte area.

11. **XPCBRC** and **XPCBRSNC** (return code and reason code)

The RUP returns on call completion a return code and a reason code in these fields. See “Error Handling” on page 337 for a description of the return codes and reason codes.

**The XSDB Control blocks:** Figure 92 on page 334 shows the DSECT for the XSDB. In the figure, each field is marked with a note number, which refers to a note (located after the figure) describing how the receiver should set the field.

You can generate the XSDB control block (together with the XPCB and the output area of an IMS INQY call) by coding the EKYRCDL1 macro statement.
Figure 92. Extended Segment Data block (XSDB)

Notes:

1. Before calling the RUP, the receiver should set:
   a. Blanks in the XSDB fields
   b. Binary zeroes in the XSDB fields

2. Before calling the RUP, the receiver should initialize the following fields of the XSDB with constants as follows:

   XSDBEYE Should be initialized with the value XSDB
   XSDBVER Should be initialized with the value V1
   XSDBREL Should be initialized with the value R1

3. Before calling the RUP, the receiver should set the XSDB fields identified with 3 to the value provided by IMS, either:

   a. In the XSDB, when calling your user-written IMS Data Capture exit routine
   b. In the CAPD_DATA block of the changed data capture IMS log records, if using the IMS Asynchronous Data Capture function

4. XSDBNXSDB (pointer to the next XSDB describing the segments in the hierarchic path, in descending order, above the changed segment).

Before calling the RUP, the receiver should provide in this field a pointer to that XSDB control block that describes the next segment in the hierarchic path above the changed segment.

Your receiver should set this field to zero before calling the RUP if IMS did not provide a description of the next segment in the hierarchic path above the changed segment either:

a. In an XSDB Control block, when calling your user-written IMS Data Capture exit routine, or
b. In a CAPD_DATA block within the changed data capture IMS log records.
5. **XSDBPHPY** (physical path accessibility):
   Before calling the RUP, the receiver should set this field to N if either:
   - IMS set this field to N when calling your user-written IMS Data Capture exit routine, or
   - The DEL_ON_PHY_PATH flag is set to On in the CAPD_DATA block of the changed data capture IMS log records, if using the IMS Asynchronous Data Capture function).
   In other cases, XSDBPHPY should be set to Y.

6. **XSDBKEYA** (address of physical key)
   Before calling the RUP, the receiver should provide in this field a pointer to the keyfield of the segment described by this XSDB.
   Your receiver should set this field to zero before calling RUP if IMS did not provide a pointer to the keyfield either:
   - In the XSDB Control block when calling your user-written IMS Data Capture exit routine, or
   - In a CAPD_DATA block within the changed data capture IMS log records.

7. **XSDBSEGA** (address of segment data)
   Before calling the RUP, the receiver should provide in this field a pointer to the segment described by this XSDB.
   Your receiver should set this field to zero before calling the RUP if IMS did not provide either one of the following:
   - In the XSDB control block a pointer to the segment data when calling your user-written IMS Data Capture exit routine, or
   - The segment data in the changed data capture IMS log records.

8. **XSDBDBD** (physical database name)
   Before calling the RUP, the receiver should provide in this field the same value as in XPCBDBD.

   **The INQY ENVIRON output area:** Figure 93 on page 336 shows the DSECT for the output area of the INQY output area. In the figure, each field is marked with a note number, which refers to a note (located after the figure) describing how the receiver should set the field.

   You can generate the DSECT for the INQY ENVIRON output area, together with the XPCB and the XSDB, by coding the EKYRCDL1 macro statement.
Figure 93. INQY ENVIRON Output Area

Notes:

1. Before calling the RUP, the receiver should set binary zeroes in the INQEAPA field.

2. Before calling the RUP, the receiver should set the fields identified with 2 to the value provided by IMS, either:
   - In the INQY_ENVIRON output area, when calling your user-written IMS Data Capture exit routine.
   - In the LOG_INQY_PSBNAME, LOG_INQY_TRANNAME, and LOG_INQY_USERID fields of the LOG_DCAP_DATA portion of the changed data capture IMS log records, if using the IMS Asynchronous Data Capture function. These fields can be set to blank, if the Receiver does not find the required information in the IMS log records.

3. Before calling the RUP, the receiver should set the fields identified with 3:
   - To the value provided by IMS in the INQY_ENVIRON output area, when calling your user-written IMS Data Capture exit routine.
   - To blanks or binary zeroes, if using the IMS Asynchronous Data Capture function.

4. INQERTA

Before calling the RUP, the receiver should provide in this field a pointer to a variable-length field. The two first bytes of the variable-length field are a halfword containing the length of the field, and are followed by:
   - In the case where the Data Staging Area of DataPropagator Relational is being fed by the RUP:
In the case where DataPropagator Relational is not being used, a recovery token that must have the same value as that provided by IMS, either:

- Via the INQERTA pointer, when calling your user-written IMS Data Capture exit routine
- In the RECOVTKN field of the LOG_DCAP_DATA portion of the changed data capture IMS log record, if using the IMS Asynchronous Data Capture function

**Error Handling**

When it is called for asynchronous data propagation, the RUP recognizes three types of propagation errors. They are failures caused:

- By deadlocks
- By unavailable resources
- For other reasons (typically mapping errors)

Your receiver must handle propagation failures. If not handled correctly, propagation failures can result in data inconsistencies.

The RUP does not perform rollbacks when it encounters an error. Therefore, design your receiver to generate rollbacks that preserve the concept of the original UOW. This can help you to maintain the correct sequence of updates presented to the RUP.

When the RUP returns a return code of 8 (indicating an error), it writes error messages to the //EKYPRINT data set, the DPROP audit trail, and the optional trace data set //EKYTRACE to help with diagnosis.

**Return codes and reason codes:** The RUP places the following return (RC) and reason (RSNC) codes in the XPCB when a propagation error occurs:

- **RC=0, RSNC=0**
  - Propagation completed successfully. For PRs defined with ERROPT=IGNORE, the RUP can return RC=0, RSNC=0 even if propagation failed.

- **RC=0, RSNC=4**
  - This is a warning. The RUP completed propagation without errors. However, the number of successfully processed PRs is zero. This can occur, for example, if the DPROP directory did not contain a PR defined for the segment type of the changed data.

- **RC=8, RSNC=4**
  - Propagation failed because of a DB2 deadlock. DB2 performed a rollback for the entire UOW. The receiver can restart processing of the failed UOW.

- **RC=8, RSNC=8**
  - Propagation failed because of a DB2 deadlock. However, rollback processing for the failing UOW was not performed. The receiver can generate an SQL rollback and restart processing of the failed UOW. This combination of codes is never returned if you are running under IMS.
**RC=8, RSNC=12**

Propagation failed due to an unavailable resource problem. Rollback processing for the failing UOW was *not* performed. The receiver must generate a rollback and terminate, or Abend. To maintain data consistency, you must solve the unavailable resource problem before restarting processing of the failed UOW.

**RC=8, RSNC=16**

Propagation failed because of another type of error. Rollback processing for the failing UOW was not performed.

For PRs defined with ERROPT=BACKOUT, the receiver must generate a rollback and terminate, or Abend. To maintain data consistency, you must solve the error before restarting processing of the failed UOW.

For PRs defined with ERROPT=IGNORE, the RUP rarely returns with RC=8, RSNC=16. Instead, it often provides diagnosis information and returns to the receiver without error indications. The RUP still generates error messages to the /EKYPRT data set and the optional trace data set. The RUP can also write error messages to the audit trail and snaps to the trace data set. The amount of these messages to the audit trail and snaps to the trace data set can be controlled with the MAXERROR value during PR generation.

After writing error messages, the RUP processes any remaining PRs for the same segment type and returns to the receiver with zero return and reason codes.

**Calling the Housekeeping Module EKYZ800X**

The first call to DPROP within an address space must be an initialization call to the DPROP housekeeping module (EKYZ800X). This call tells DPROP what environment it is in and that it is being called asynchronously.

The last call to DPROP within an address space must be a termination call to the housekeeping module. This tells DPROP to perform its cleanup processing (for example, to close files).

Avoid generating initialization and termination calls frequently because of their significant performance impact.

The housekeeping module must be called according to the standard OS/VS conventions for calling Assembler modules.

**Parameters for the initialization Call:** For DPROP initialization, call the module with the following two parameters:

- **Call Function** A four-byte character field that contains the string *INIT*
- **Environment** A four-byte character field that describes the environment. DPROP uses this value to determine which language interface to use for SQL calls. The value of the string must be one of the following:
  - **IMS** The receiver is running in an IMS environment. DPROP generates its SQL statements through the language interface of the IMS Attach facility.
  - **TSO** The receiver is running in a TSO environment. DPROP generates its SQL statements through the language interface of the TSO Attach facility.
The receiver is running in a Call Attach facility (CAF) environment. DPROP generates its SQL statements through the language interface of the CAF Attach.

If running in a CAF environment, the receiver must establish a CAF connection to DB2 before calling the housekeeping module, and close the connection after terminating the housekeeping module. The receiver can establish the connection with CAF CONNECT and OPEN requests; it can close the connection with CAF CLOSE and DISCONNECT requests.

The RUP and the housekeeping module must be called from the task that establishes and closes the CAF connection.

Remember that you must link-edit any DPROP exit routine that generates SQL statements with the SQL language interface of the proper DB2 Attach.

**Parameter for the termination Call:** For DPROP termination, only the first parameter is required:

- **Call Function** A four-byte character field containing the string TERM.
- **Environment** This parameter is optional. The housekeeping module does not use it, but it can be useful for consistency with the initialization call.

**Calling the Module:** As with the RUP, the receiver must not be link-edited with the housekeeping module. Instead, you must call the module dynamically using the same methods described above.

**Return codes:** This section describes the return codes from the housekeeping module. After a termination call, the module (EKYZ800X) always returns with a zero in Register 15 (R15).

After an initialization call, the following codes can be returned in R15:

- **0** DPROP initialization was successful.
- **4** DPROP initialization failed because a DB2 deadlock condition was encountered. Rollback processing for the failing UOW was performed. The receiver can restart any processing done during the failing UOW, and regenerate the INIT call.
- **8** DPROP initialization failed because a DB2 deadlock condition was encountered. However, rollback processing was not performed. The receiver can generate an SQL rollback, and restart any processing done during the failing UOW.
- **12** DPROP initialization failed because of an unavailable resource. Rollback processing for the failing UOW was not performed. The receiver must either generate an SQL rollback and return (if running under TSO/CAF), or Abend.
- **16** DPROP initialization failed for another type of error. The receiver must either generate an SQL rollback and return (if running under TSO/CAF), or Abend.
Supported Environments and Restrictions

This section describes the environment DPROP supports when your receiver calls DPROP:

- The RUP runs as a DB2 application; your receiver can call it in the following environments:
  - In an IMS/ESA batch or BMP region
  - under TSO foreground or TSO batch
  - In a DB2 CAF environment

IMS MPP or IFP regions, and CICS environments are not supported. DPROP does not test if it is being called in one of these environments.

- If your receiver is running in a TSO or CAF environment, any DPROP exit routines must not generate DL/I calls, because IMS does not support TSO or CAF.

- DPROP does not support access to remote or distributed DB2 systems. This applies to both synchronous and asynchronous data propagation systems.

- The RUP and the housekeeping module (EKYZ800X) must be link-edited in AMODE 31, and must be called in AMODE 31. Remember that, for maintenance and migration considerations, you must not link-edit your receiver with these modules.

- DPROP must be called by a program running in the ordinary problem-program mode, with PSW protection key 8. DPROP cannot be called in:
  - SVC or SRB mode
  - Cross memory mode
  - Access register mode
  - Authorized mode
  - Any protection key other than 8

- If DPROP is running in a subtask, the attaching mother task must not share any OS/VS virtual storage subpool with the subtask, if it intends to reattach DPROP after termination. Also, in a multiple task environment, higher level OS/VS tasks must not preload DPROP modules.

JCL Requirements

To run the receiver, in a TSO or IMS environment, you must provide:

- The JCL that DB2 and DB2 Attach require (the IMS, TSO, or CAF Attach), and
- The JCL that DPROP requires.

This section describes DPROP's JCL requirements. In addition to the DPROP JCL for //STEPLIB and //EKYRESLB, this includes:

1. An //EKYPRINT DD statement, which the RUP uses to write error messages. It is typically coded as:
   ```
   //EKYPRINT DD SYSOUT=A
   ```

2. An //EKYLOG DD statement or an //EKYTRACE DD statement, which contain information that the RUP usually writes to the IMS log (for example, traces, snaps, or error messages). They are typically coded as:
   ```
   //EKYLOG DD DSN=xxxx,DISP=(,CATLG),
   //UNIT=xxx,SPACE=(CYL,(nn,nn))
   //EKYTRACE DD SYSOUT=A
   ```
3. An optional //EKYIN DD statement, which is used to provide a TRACE control statement used to activate the DPROP Trace module. Refer to *IMS DPROP Reference* for the syntax of the TRACE statement.

**Binding a DB2 Plan for the Receiver**

When your receiver calls DPROP, DPROP generates SQL statements to access DPROP directory tables and to update the propagated tables. Therefore, you need to bind a DB2 plan for running the receiver.

For details on how to perform the bind, see the appropriate *Administrators Guide* for your propagation mode.

---

**Installation Considerations: Asynchronous Data Propagation**

If you are using DPROP for both synchronous and asynchronous propagation, you must define two different DPROP systems, each with its own DPROP directory.

When installing each DPROP system, you must define whether the system is used for synchronous or asynchronous data propagation. Specify the type of system in the System Type field of the EKYGP42E installation panel:

- **SYNC** For synchronous propagation
- **ASYNC** For asynchronous propagation

Specifying the type of system is part of the DPROP generation and customization process. Refer to *IMS DPROP Installation Guide* for more details.

**The Status Change Utility (SCU)**

The Status Change utility supports only the following control statements for an asynchronous DPROP system:

- **INIT DPROP=**
- **DISPLAY STATUS**

Because you cannot activate or deactivate PRs in an asynchronous DPROP system, the RUP considers all PRs in such a system as active.

**Multiple MVS Images**

As explained in the appropriate *Administrators Guide* for your propagation mode, additional restrictions apply if IMS and DB2 reside on different MVS images.

DPROP must execute on the target MVS system, the same MVS system that owns the DB2 tables. The DPROP directory must also reside on this system.

MVG and MVGU must run on the target MVS system. They must have access to the DBDLIB of the source MVS system, the system on which IMS runs. If the MVS images share DASD, the DBDLIB can reside on the shared storage. Otherwise, an up-to-date copy of the DBDLIB must be provided on the target MVS system.

As described in the appropriate *Administrators Guide* for your propagation mode and *IMS DPROP Reference*, special considerations are required for:

- Use of the CCU
Database Maintenance

With asynchronous propagation, you can reorganize or repair the DB2 side while you are collecting or storing updates. Then, when you are applying those updates to DB2, you can stop updating the IMS database and perform database maintenance activities on the IMS side. Conversely, when you are collecting updates on the IMS side, you can perform database table space maintenance on the DB2 side.

With asynchronous propagation, out-of-sync conditions are normal, and data can be truly synchronized for only brief periods. Asynchronous propagation gives you some allowance for unavailable resources that is not possible in the synchronous environment. You can generally operate on the IMS and DB2 components independently without having one affect the other.

With asynchronous propagation in IMS online or BMP environments, your stored updates must be backed out with a failing IMS UOW if you use a database (full-function or DEDB) or a message queue for warehousing the propagating updates. The IMS synchronization point manager performs this backout. If you use the IMS Batch Backout utility or dynamic backout in IMS batch jobs, your propagating updates must also be backed out from databases used to store changes intended for the receiving program.

However, if you use the IMS log or an MVS flat file for storing propagating updates, then no backout can occur if failure occurs. You must restore data integrity by eliminating the failed updates from the log or flat file.

Recovering the DPROP Directory

For information on recovering the DPROP directory, see the appropriate Administrators Guide for your propagation mode.
Appendix A. Calling the Trace Module

This appendix describes the interface for the optional DPROP trace module from a Propagation exit routine. Some reasons for calling the trace module are also discussed. For more general information about DPROP trace support, refer to IMS DPROP Diagnosis.

To complement the RUP's and HUP's trace support, your Propagation exit routine can also call the trace module (module EKYR410X) directly. By activating the DPROP Trace with the appropriate debug level, you can request that the RUP and HUP trace the parameters, control blocks, and other areas involved in calling your exit routine. (For information about debug levels see the IMS DPROP Diagnosis.) The RUP and HUP can also trace this information when your exit routine signals a propagation failure by returning a nonzero return code. Therefore, your exit routine does not need to provide the code for tracing this interface information.

Typically, your exit routine can call the trace module for two purposes:

1. To trace updating SQL calls (HR propagation) and IMS calls (RH propagation) that your exit routine creates upon request. If the PICDBLV2 bit is on in the Propagation Interface Control Block (PICB), you are requesting the tracing of SQL calls and IMS Calls.

2. To trace information needed for problem determination. If you have a propagation failure, even if you have not requested tracing, you can snap or trace whatever information you think is needed to solve the problem. When your exit routine returns with an error return code, the RUP snaps or traces all relevant interface information.

The DPROP trace module can trace multiple items with each call. For example, when tracing an SQL call, each DB2 column involved in the call is traced as a separate item. the trace module writes its results to the //EKYTRACE data set, to the //EKYLOG data set, or to the IMS log. To find out how to format and print these records and interpret the trace output, refer to IMS DPROP Diagnosis.

Trace Module Interface

Your exit routine must use standard OS/VS linkage conventions when calling the trace module.

Register 1 Points to the parameter list described below
Register 13 Contains the address of the standard save area
Register 14 Contains the return address
Register 15 Contains the entry address of the DPROP trace module

Parameter list

The first parameter in the parameter list pointed to by Register 1 must be the address of the Trace Request Block (TRB). Following this address, the parameter list must include the address of one Trace Element Descriptor (TED) for each item included in the trace. The TRB and TED are described below.
The trace module must be called in AMODE 31, and returns control to your exit routine in AMODE 31.

The sample Propagation exit routine (see Figure 52 on page 190) contains a macro called SETTED. This macro simplifies calling the trace module. You can create a similar macro to use in your system. See Figure 52 on page 190, where the SETTED macro is used in the sample Propagation exit routine.

---

**Trace Request Block (TRB)**

Figure 94 on page 345 contains the DSECT for the TRB. Following the DSECT, the fields are described in detail.

The EKYTRB DSECT is provided in the DPROP macro library. Code the EKYTRB macro statement to create the DSECT in your exit routine.
1 EKYTRB
2************************** START OF CONTROL BLOCK SPECIFICATION **********
3++
4++ CONTROL BLOCK NAME: *
5++ EKYTRB (TRB) *
6++
7++ DESCRIPTIVE NAME: *
8++ DPROP TRACE REQUEST BLOCK (TRB) *
9++ = = = *
10++
11****************************** END OF CONTROL BLOCK SPECIFICATION **********
12++
13++ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM". *
14++
15++ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. *
16++ ALL RIGHTS RESERVED. *
17++
18++ U.S. GOVERNMENT USERS RESTRICTED RIGHTS - *
19++ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY *
20++ GSA ADP SCHEDULE CONTRACT WITH IBM CORP. *
21++
22++ LICENSED MATERIALS - PROPERTY OF IBM. *
23++
24****************************** END OF CONTROL BLOCK SPECIFICATION **********
25++
26++ STATUS: V1 R2 M *
27++
28++ FUNCTION: *
29++ A TRB IS USED FOR THE COMMUNICATION BETWEEN A *
30++ 'PROPAGATION USER EXIT ROUTINE' AND THE DPROP TRACE *
31++ FUNCTION. *
32++
33++ WHEN INVOKING THE DPROP-TRACE FUNCTION, THE CALLING *
34++ USER EXIT MUST PROVIDE THE TRB AS FIRST CALL-PARAMETER. *
35++
36++ THE TRB PROVIDES INFORMATION ABOUT THE TRACE REQUEST. *
37++
38++ MODULE TYPE= MACRO *
39++ PROCESSOR= ASSEMBLER H *
40++
41++ ACQUIRED BY MODULE INVOKING THE TRACE *
42++
43++ INNER CONTROL BLOCKS: NONE *
44++
45++ MACROS USED FROM MACRO LIBRARY: NONE *
46++
47++ CHANGE ACTIVITY: *
48++ KMP0057 12/13/90 *
49++
50****************************** END OF CONTROL BLOCK SPECIFICATION **********

000000 54+TRB DSECT
000000 E309C240 55+TRBEYE DC C'TRB ' EYE-CATCHER
000004 00000000 56+TRBPTD DC A(0) ADDRESS OF THE DPROP-PTD CONTROL BLOCK

58********
59******** NAME OF OBJECTS ASSOCIATED WITH THE TRACE
60********

000008 4040404040404040 62+TRBTABQ DC CL8' ' TABLE-NAME QUALIFIER ASSOC. W. TRACE
000010 4040404040404040 63+TRBTABN DC CL18' ' TABLE-NAME ASSOCIATED WITH THE TRACE
000022 4040 64+ DC CL2' ' NAME ASSOCIATED WITH THE TRACE
000024 4040404040404040 65+TRBDBN DC CL8' ' DBD-NAME ASSOCIATED WITH THE TRACE
00002C 4040404040404040 66+TRBSEGN DC CL8' ' SEG-NAME ASSOCIATED WITH THE TRACE

Figure 94 (Part 1 of 2). Trace Request Block
TRB Field Descriptions

TRBEYE
Your exit routine must set this field to TRB. The trace module validates its content.

TRBPTD
Your exit routine must provide the address of the PTD control block in this field. This PTD address can be found in the PICPTD field of the PICB.

When performing HR propagation, your exit routine must also set the next two fields, which are used in the trace records to identify data objects associated with the trace. DPROP includes the data you provide below, in both the trace record (to allow selective trace formatting), and the formatted trace output.

TRBTABQ  The table name qualifier of the table involved in the trace
TRBTABN  The unqualified table name of the table involved in the trace

When performing RH propagation, your exit routine must set the next two fields, which are used in the trace records to identify data objects associated with the trace. DPROP includes the data you provide, in both the trace record (to allow selective trace formatting), and the formatted trace output.

TRBDBN  The name of the physical IMS database involved in the trace
TRBSEGN  The name of the physical IMS segment involved in the trace

Your exit must also set the next field:

TRBSOLI  The Propagation exit routine must set this field to determine if the trace was requested by the user. If the user requested it, the exit routine must set this field to Y. If the user did not request it, (for example, if errors occurred), the exit routine must set this field to N.

Trace Element Descriptor (TED)

This section describes the Trace Element Descriptor (TED). You specify one TED in the keyword list for each item you want to trace.

DPROP distinguishes between the following three different types of items that can be traced:

- Header items
- Subheader items
- Data items
The DPROP Trace module formats each of the three types of items differently. In each TED in the parameter list, your exit routine must identify the type of item the TED describes.

DPROP requires that the first TED in the keyword list describe a header item. TEDs describing subheader items are optional; they can be provided to make reading of the formatted trace easier by helping to structure the information presented in the trace output. An exit routine provides one or more TEDs that describe data items to be traced.

Figure 95 is an example of a formatted trace. The figure and the explanations that follow show how the DPROP trace module formats the different item types.

---

Figure 95. Example of Formatted Trace

This is an example of formatted trace. The DPROP Trace module creates it, using the following TEDs:

1. A TED for a header item, which provides a text string that is printed exactly as entered (the text string “PROPAGATING SQL-UPDATE FOR TABLE=PROD.PARTS” at the top of the figure).

   DPROP Trace formatting prefixes the text string of a header item with asterisks, the time, and the date.

   For header items, the DPROP formatting routine prints additional lines with identifying information (DPR ID, IMS ID, and so forth.)

2. A TED for a data item. The second TED consists of the text string “SQLCODE:,” followed on the next print line by the snapped SQL error code.

   Note the difference between TEDs for header items and TEDs for data items:
   - TEDs for header items (and subheader items) provide only a text string.
TEDs for data items provide both:

a. A descriptive text string (in the example: “SQLCODE:")) printed on the first print line. DPROP formatting prefixes the text string with a period and some blanks to help identify it.

b. A virtual storage area to be snapped both in hexadecimal and character/EBCDIC format printed on the following lines. DPROP Trace formatting prefixes each print line with the virtual storage address of the first byte represented in the print line.

3. A TED for a subheader item, consisting of the text string “COLUMNS IN WHERE CLAUSE”

In this example, the DPROP trace module's caller provides a subheader item to add additional structure to the formatted trace. It identifies the columns used in the WHERE clause of the SQL statement, and columns that the SQL statement propagates.

DPROP Trace formatting prefixes the text string in a subheader item with three dots and some blanks for easier identification.

4. A TED for a data item (the BRANCH OFFICE).
5. A TED for a data item (the PART NBR).
6. A TED for a subheader item (PROPAGATED COLUMNS).
7. A TED for a data item (the MANUFACTURER).
8. A TED for a data item (the ZIP CODE).
9. A TED for a data item (the CITY).
10. A TED for a data item (the PRICE).

Figure 96 on page 349 shows the DSECT for the Trace Element Descriptors. Field descriptions follow the figure.

The EKYTED DSECT is provided in the DPROP macro library. Code the EKYTED macro statement to create the DSECT in your exit routine.
1 EKYTED
2+******************************** START OF CONTROL BLOCK SPECIFICATION ***************
3+  
4+ CONTROL BLOCK NAME:  
5+ EKYTED (TED)  
6+  
7+ DESCRIPTIVE NAME:  
8+ DPROP TRACE ELEMENT DESCRIPTOR (TED)  
9+  
10+  
11+************************************************************
12+  
13+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".  
14+  
16+ ALL RIGHTS RESERVED.  
17+  
18+ U.S. GOVERNMENT USERS RESTRICTED RIGHTS -  
19+ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY  
20+ GSA ADP SCHEDULE CONTRACT WITH IBM CORP.  
21+  
22+ LICENSED MATERIALS - PROPERTY OF IBM.  
23+  
24+************************************************************
25+  
26+ STATUS: V1 R2 M  
27+  
28+ FUNCTION:  
29+ WHEN INVOKING THE DPROP TRACE FUNCTION, THE CALLING  
30+ MODULE MUST PROVIDE ONE TED FOR EACH:  
31+ - TRACE-HEADER  
32+ - TRACE-SUBHEADER  
33+ - DATA-AREA  
34+ WHICH SHOULD BE TRAced/SNAPPED.  
35+  
36+ MODULE TYPE= MACRO  
37+ PROCESSOR= ASSEMBLER H  
38+  
39+ ACQUIRED BY MODULE INVOKING THE TRACE  
40+  
41+ INNER CONTROL BLOCKS: NONE  
42+  
43+ MACROS USED FROM MACRO LIBRARY: NONE  
44+  
45+ CHANGE ACTIVITY:  
46+ KMP0057 12/13/90  
47+  
48+************************************************************ END OF CONTROL BLOCK SPECIFICATION  

000000 52+TED DSECT
000000 E3C5C440 53+TED EYE DC C'TED ' EYE-CATCHER
000004 40 54+TEDTYPE DC C' ' TYPE OF TRACE ITEM
0000C8 55+TEDYPH EQU C'H' ... HEADER
0000E2 56+TEDYPS EQU C'S' ... SUB-HEADER
0000C4 57+TEDYPD EQU C'D' ... DATA
000050 58+TEDALIGN DC C' ' ALIGNMENT FOR SNAP-FORMATTING
000003 59+TEDALIGL EQU C'L' ...L = LEFT ALIGNMENT
000040 60+TEDALIGB EQU C' ' ...BLANK= NO LEFT ALIGNMENT
000006 0000 61+ DC XL2'00' RESERVED
000008 00000000 62+TEDTXA DC A(0) PTR TO TEXT-STRING
00000C 00000000 63+TEDTXL DC F'0' LENGTH OF TEXT-STRING
000010 00000000 64+TEDMA DC A(0) VIRTUAL STORAGE ADDR OF AREA TO BE SNAPPED
000014 00000000 65+TEDALEN DC F'0' LENGTH OF AREA TO BE SNAPPED
000018 00000000 66+TEDALEF DC F'0' ALET OF DATA (MUST BE ZERO IN THIS RELEASE)
00001C 0000000000000000 67+ DC 2F'0' RESERVED/MUST BE ZERO

Figure 96 (Part 1 of 2). Trace Element Descriptor
Figure 96 (Part 2 of 2). Trace Element Descriptor

TED Field Descriptions

**TEDEYE**

Your exit routine must set this field to TED. The trace module validates its content.

**TEDTYPE**

The type of the item to be traced. DPROP recognizes three types of items:

- **Header**
  
  For a header, your exit routine must set this field to **H**. You must also provide a text string to be used as the header, and store its address in TEDTXTA and its length in TEDTXTL. For a header item, the fields TEDMA, TEDALEN, and TEDALIGN do not apply. Therefore, you do not need to provide values for these fields.

- **Subheader**
  
  For a subheader, your exit routine must set this field to **S**. You must also provide a text string to be used as the subheader, and store its address in TEDTXTA and its length in TEDTXTL. For a subheader item, the fields TEDMA, TEDALEN, and TEDALIGN do not apply. Therefore, you do not need to provide values for these fields.

- **Data**
  
  For data, your exit routine must set this field to **D**. It must store the address of the data item to be traced in TEDMA, and the length in TEDALEN. Depending on the length of the data item, the trace is formatted on one or more print lines.

  Your exit routine must also provide a descriptive text string explaining what information is being traced. This text string is printed in the formatted trace output. The address of the text string must be placed in TEDTXTA, and the length in TEDTXTL. When tracing a DB2 column, it is recommended that the text string be the DB2 column name.

  Also for a data item, your exit routine must set TEDALIGN to indicate whether the traced area must be left-aligned on the formatted print line.

  For more information on headers, subheaders, and data items, and on the format of the trace output, see *IMS DPROP Diagnosis*.

**TEDALIGN**

This field determines if the first byte of the formatted trace output is aligned to the left of the page. If you want the output to be left-aligned, set this field to **L**. If you do not want the output left-aligned, the field must be blank.
Left-alignment can make the trace output much easier to read, especially when the output length is small and you do not need to locate the area using virtual storage address. DPROP uses left-alignment when tracing SQL calls, and it is recommended that your exit routine use the same convention.

If, however, the traced area is large, or you want to locate traced information using a virtual storage address, do not align the trace output on the left. The output then resembles a storage dump. This can be useful when tracing entire control blocks or work areas. It simplifies location of information when you search using virtual addresses.

To see an example of formatting with left-alignment, refer to Figure 95 on page 347. To see an example of formatting without left-alignment, see *IMS DPROP Diagnosis*.

**TEDTXTA**  
The address of the text string that is printed in the formatted trace output.

**TEDTXTL**  
The length of the text string that is printed in the formatted trace output.

**TEDMA**  
For a data item, the address of the area in storage that is traced.

**TEDALEN**  
For a data item, the length of the area in storage that is traced.
Appendix B. Sample Segment Exit Control Blocks

This appendix contains sample Segment exit control blocks which map the existing DPROP interface control blocks. This appendix provides the exit control blocks in three languages:

- COBOL
- PL/I
- C

The Assembler version of the Segment exit control block is shown in Figure 7 on page 28.
Sample Segment Exit Control Block for COBOL

Figure 97 shows an example of the EKYRCDAX control block in COBOL. This control block, called EKYRCDXC, resides in the DPROP Sample Source library (EKYSAMP).

000100************ START OF CONTROL BLOCK SPECIFICATION ************ 00010000
000200+ * 00020000
000300+ CONTROL BLOCK NAME: + 00030000
000400+ EKYRCDXC (DAX) + 00040000
000500+ * 00050000
000600+ DESCRIPTIVE NAME: + 00060000
000700+ DPROP COBOL SEGMENT EXIT INTERFACE BLOCK + 00070000
000800+ * 00080000
000900+ COBOL VERSION OF EKYRCDAX + 00090000
001000+ * 00100000
001100+******************************************************************** 00110000
001200+ * 00120000
001300+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM". + 00130000
001400+ * 00140000
001500+ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. + 00150000
001600+ ALL RIGHTS RESERVED. + 00160000
001700+ * 00170000
001800+ U.S. GOVERNMENT USERS RESTRICTED RIGHTS - + 00180000
001900+ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY + 00190000
002000+ GSA ADP SCHEDULE CONTRACT WITH IBM CORP. + 00200000
002100+ * 00210000
002200+ LICENSED MATERIALS - PROPERTY OF IBM. + 00220000
002300+ * 00230000
002400+************************************************************************* 00240000
002500+ * 00250000
002600+ STATUS: V1 R2 M0 + 00260000
002700+ * 00270000
002800+ FUNCTION: + 00280000
002900+ THIS IS THE COBOL CONTROL BLOCK USED TO INTERFACE BETWEEN + 00290000
003000+ - DPROP OR DXT + 00300000
003100+ AND + 00310000
003200+ - A USER'S SEGMENT EXIT ROUTINE (THESE USER + 00320000
003300+ EXIT ROUTINES ARE CALLED BY DXT 'USER DATA + 00330000
003400+ EXIT ROUTINES') + 00340000
003500+ * 00350000
003600+ THERE IS ONE DAX CONTROL BLOCK FOR EACH SEGMENT + 00360000
003700+ EXIT ROUTINE, LASTING FOR THE DURATION OF THE EXIT + 00370000
003800+ IN VIRTUAL STORAGE. + 00380000
003900+ FOR SYNCH PROPAGATION IN MPP REGIONS: + 00390000
004000+ - THIS IS THE DURATION OF THE IMS PROGRAM CONTROLLER + 00400000
004100+ 'SUBTASK'. + 00410000
004200+ FOR SYNCH PROPAGATION IN BATCH/BMP REGIONS, FOR + 00420000
004300+ CCJ AND DLU PROCESSING, AND FOR ASYNCH PROPAGATION + 00430000
004400+ (DEPENDING ON HOW ASYNCH PROPAGATION IS IMPLEMENTED): + 00440000
004500+ - THIS IS THE DURATION OF THE JOBSTEP. + 00450000
004600+ * 00460000
004700+************************************************************************* 00470000
004800+ IMPORTANT NOTES: + 00480000
004900+ ******************************************************* 00490000
005000+ - SINCE THE SAME USER EXIT ROUTINE CAN BE INVOKED BOTH + 00500000
005100+ BY DPROP AND BY DXT: CHANGES TO THIS CONTROL BLOCK MUST + 00510000
005200+ BE COORDINATED BETWEEN DPROP DEVELOPMENT AND DXT + 00520000
005300+ DEVELOPMENT. + 00530000
005400+ * 00540000
005500+ - FIELDS MARKED IN THE COMMENT WITH '**DXT ONLY**' + 00550000
005600+ HAVE NO MEANING, WHEN THE SEGMENT USER EXIT + 00560000

Figure 97 (Part 1 of 5). COBOL Interface Control Block for a Segment Exit Routine
Figure 97 (Part 2 of 5). COBOL Interface Control Block for a Segment Exit Routine
Figure 97 (Part 3 of 5). COBOL Interface Control Block for a Segment Exit Routine
Figure 97 (Part 4 of 5). COBOL Interface Control Block for a Segment Exit Routine
Figure 97 (Part 5 of 5). COBOL Interface Control Block for a Segment Exit Routine
Sample Segment Exit Control Block for PL/I

Figure 98 shows an example of the EKYRCDAX control block in PL/I. This control block, called EKYRCDXP, resides in the (EKYSAMP) library.

```
1/******************* START OF CONTROL BLOCK SPECIFICATION *******************
  *
  * Control Block name:     *
  * EKYRCDXP (DAX)         *
  *
  * Descriptive name:      *
  * DPROP PL/I segment exit interface block.       *
  *
  * PL/I version of EKYRCDAX

******************************************************************************

  * THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".


  * ALL RIGHTS RESERVED.

  * U.S. GOVERNMENT USERS RESTRICTED RIGHTS -
  * USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
  * GSA ADP SCHEDULE CONTRACT WITH IBM CORP.

  * LICENSED MATERIALS - PROPERTY OF IBM.

******************************************************************************

  * STATUS: V1 R2 M0

  *
  * Function:
  * This is the PL/I control block used to interface between
  * - DPROP OR DXT
  * and
  * - a user segment exit routine (these user exit routines are
    called by DXT "user data exit routines")

  * There is one DAX control block for each segment exit routine,
  * lasting for the duration of the exit in virtual storage.

  * For synchronous propagation in MPP regions:
    * this is the duration of the IMS program controller subtask.
  * For synchronous propagation in batch/BMP regions, for CCU and
    DLU processing, and for asynchronous propagation (depending
    on how asynchronous propagation is implemented):
    * this is the duration of the jobstep.

******************************************************************************

  * Important note:
  * - Fields marked in the comment with '***** DXT only *****' have
    no meaning, when the segment user exit routine is invoked by
    DPROP.

******************************************************************************

  * Change activity:    *
  * None

*************** END OF CONTROL BLOCK SPECIFICATION ***************
```

```
1DECLARE 1 DAX BASED(DAX_POINTER),

/* This section of the control block may not be modified by exit */
2 DAXPFX,   /* Prefix of control block (32 bytes) */
3 DAXNAME CHAR(8),  /* eye catcher ("DVRXCDAX") */
3 DAXRSVD CHAR(24),   /* reserved for DXT internal use */
```

Figure 98 (Part 1 of 5). PL/I Interface Control Block for a Segment Exit Routine
Figure 98 (Part 2 of 5). PL/I Interface Control Block for a Segment Exit Routine
Figure 98 (Part 3 of 5). PL/I Interface Control Block for a Segment Exit Routine
Figure 98 (Part 4 of 5). PL/I Interface Control Block for a Segment Exit Routine
3 DAXSMESG CHAR(64), /* Text of message passed from exit routine to DPROP/DXT. All blanks means no message.

If caller is DPROP:
Message will be written to various destinations according to usual DPROP/RUP error handling logic in message EKYR9801 or EKYR981E.

If caller is DXT: text of message will be written to SYSPRINT dataset in message DVRA950, (underscore is replaced by one of several digits) has effect for all calls. */

3 DAXDPRPM CHAR(24), /* Storage reserved for data exit. */
3 DAXRSVD2 CHAR(32), /* Reserved for DXT use. */
3 DAXCRT1 CHAR(128); /* Work space (scratchpad), may be used by the exit as desired. */

Figure 98 (Part 5 of 5). PL/I Interface Control Block for a Segment Exit Routine
Sample Segment Exit Control Block for C

Figure 99 shows an example of the EKYRCDAX control block in C. This control block, called EKYRCDXXK, resides in the (EKYSAMP) library.

```c
/*------------------- START OF CONTROL BLOCK SPECIFICATION -------------------*/

/*
   Control Block name:
   EKYRCDXX (DAX)
*/

/*
   Descriptive name:
   DPROP C language segment exit interface block.
   C language version of EKYRCDAX
*/

/*
   THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM"
*/

/*
   ALL RIGHTS RESERVED.
*/

/*
   U.S. GOVERNMENT USERS RESTRICTED RIGHTS -
   USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
   GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
*/

/*
   LICENSED MATERIALS - PROPERTY OF IBM.
*/

/*
   STATUS: V1 R2 M1
*/

/*
   Function:
   This is the C language control block used to interface between
   - DPROP OR DXT
   and
   - a user's segment exit routine (these user exit routines
     are called by dxt 'user data exit routines')
*/

/*
   There is one DAX control block for each segment exit routine,
   lasting for the duration of the exit in virtual storage.
   For synchronous propagation in MPP regions:
   - this is the duration of the IMS program controller subtask.
   For synchronous propagation in batch/BMP regions, for CCU and
   DLU processing, and for asynchronous propagation (depending
   on how asynchronous propagation is implemented):
   - this is the duration of the jobstep.
*/

/*
   Important notes:
   Since the same user exit routine can be invoked both by DPROP
   and by DXT: changes to this control block must be coordinated
   between DPROP development and DXT development.
*/

/*
   Fields marked in the comment with '***** DXT only *****' have
   no meaning, when the segment user exit routine is invoked by
   DPROP.
*/

/*
   Change activity:
   None
*/

/*------------------- END OF CONTROL BLOCK SPECIFICATION -------------------*/

#pragma page(1)

typedef
struct
{
   /* DAX */
   /*
   * This section of the control block may not be modified by exit
   */
}*/

Figure 99 (Part 1 of 5). C Interface Control Block for a Segment Exit Routine
Figure 99 (Part 2 of 5). C Interface Control Block for a Segment Exit Routine
/* of DL/I ioarea format buffer. */

/***************************************************************************/
/* DAXOUTLN */
/* Length of output segment-record to be built by segment exit routine. */
long daxflen; /* Alternate name, refers to length of DPROP format buffer. */

/***************************************************************************/
char *daxsyspr; /* **** DXT only **** pointer to sysprint DCB */

/***************************************************************************/
/* DAXENV */
/* Environment subfields (12 bytes) */
unsigned char daxopsys[4]; /* Operating system: */
"ESA " if MVS/ESA
**DXT only** "XA " if MVS/XA
**DXT only** "MVS " if MVS */

unsigned char daxtrans[4]; /* DB/DC environment: */
"BAT " if IMS BATCH/BMP
"MPP " if IMS MPP
"IFP " if Fast Path
"CICS" if CICS
" " if none of the above */

unsigned char daxprogm[4]; /* Calling program: */
"DXT " if DXT
"DPRS" if DPROP SYNCH PROP
"DPRA" if DPROP ASYNCH PROP
"DPRC" if DPROP CCU PROP
"DPRL" if DPROP DLU */

/***************************************************************************/
#pragma page(1)

unsigned char daxexit[8]; /* Name of this exit routine */

unsigned char daxdbnm[8]; /* Name of IMS data base. */
If caller is DPROP:
name of physical DBD.
If caller is DXT:
name of used DBD (can be name of a physical or logical DBD) */

unsigned char daxdprpn[24]; /* Reserved */

long daxasgno; /* **** DXT only **** number of DAXASEGS array elements */

char daxasegs[15][12]; /* **** DXT only **** array of ancestor segments */

unsigned char daxdprct[4]; /* If caller is DPROP, exit is called to process: */
"ISRT" - a DL/I or DB2 insert
"DLET" - a DL/I or DB2 delete
"REPL" - a DL/I or DB2 replace (after-image) */

unsigned char daxrepl; /* If caller is DPROP and if DAXDPRCT is "REPL": */
"A" - after replace
"B" - before replace */

Figure 99 (Part 3 of 5). C Interface Control Block for a Segment Exit Routine
unsigned char daxseg; /* If caller is DPROP, 
type of segment processed: 
"U" - updated IMS segment 
"A" - ancestor of updated seg 
"I" - internal segment */

unsigned char daxpsup; /* If caller is DPROP: description 
whether propagation-suppression is 
allowed: 
"N" - suppression not allowed 
"Y" - suppression is allowed */

unsigned char fill01; /* Reserved */

unsigned char daxisegm[8]; /* If caller is DPROP and for RH 
propagation: name of segment to 
process. Same as physical IMS 
segment name in DAXSEGM if not 
mapping case 3 entity (internal) 
segment in process. */

#pragma page(1)

char *daxiddsb; /* If caller is DPROP and for RH 
propagation: pointer to DL/I DB 
segment buffer. 
This buffer contains contains the 
before image of the IMS segment if: 
- DAXDRCT equals REPL, or DLET 
- or 
- DAXSEGT equals DAXSEGTI 
(internal seg) 
else contains all binary zeroes 
in other cases. 
Buffer is read only for 
the exit routine. */

char *daxidds; /* If caller is DPROP and for RH 
propagation: length of the 
'before-change' IMS DB segment 
pointed-to by DAXIDDSB. */

unsigned char fill22[22]; /* Filler */

/*********************************************************************************/
/*The next group of fields may be modified by the exit routine. */
/*********************************************************************************/

unsigned char daxentrd; /* Set by exit to "x" indicating 
that the exit has been entered. */

unsigned char daxinctl; /* Set by exit to "x" indicating 
that exit is in control. */

long daxretc; /* Return code. 
0 = normal, output data returned. 
4 = <<DXT ONLY>>> 
8 = If caller is DPROP: 
Propagation of the DL/I 
changed data will be 
suppressed. 
If caller is DXT: 
DXT should not consider data 
to be eligible for extract. 
12 = ERROR 
If caller is DPROP: 
Propagation failure. 
DPROP/RUP will go through 
its usual error handling 
logic. */

Figure 99 (Part 4 of 5). C Interface Control Block for a Segment Exit Routine
If caller is DXT:
DXT should terminate.

16 = ERROR
If caller is DPROP:
RUP will abend.
If caller is DXT:
DXT should terminate DEM execution. */

#pragma page(1)

unsigned char daxsmesg[64]; /* Text of message passed from exit routine to DPROP/DXT. All blanks means no message. */

If caller is DPROP:
Message will be written to various destinations according to usual DPROP/RUP error handling logic in message EKYR980I or EKYR981E.

If caller is DXT: text of message will be written to SYSPRINT dataset in message DVRA0_50, (underscore is replaced by one of several digits) has effect for all calls. */

unsigned char daxdprpm[24]; /* Storage reserved for data exit. */
unsigned char daxrsvd2[32]; /* Reserved for DXT use. */
unsigned char daxscrt1[128]; /* Work space (scratchpad), may be used by the exit as desired. */

} EKYRCDA;

#pragma page(1)

---

**Figure 99 (Part 5 of 5). C Interface Control Block for a Segment Exit Routine**
Appendix C. Sample Field Exit Control Blocks

This appendix contains sample Field exit control blocks which map the existing DPROP interface control blocks. This appendix provides the exit control blocks in three languages:

- COBOL
- PL/I
- C

Figure 28 on page 115 shows the Assembler version of the Field exit control block.
Sample Field Exit Control Block for COBOL

Figure 100 shows an example of the EKYRCUDT control block in COBOL. This control block, called EKYRCUDC, resides in the DPROP Sample Source library (EKYSAMP).

000100************ START OF CONTROL BLOCK SPECIFICATION ************ 00010000
000200+ * 00020000
000300+ CONTROL BLOCK NAME: * EKYRCUDC (UDT)
000400+ + 00040000
000500+ + 00050000
000600+ DESCRIPTIVE NAME: + 00060000
000700+ DPROP COBOL FIELD EXIT INTERFACE + 00070000
000800+ + 00080000
000900+ COBOL VERSION OF EKYRCUDT + 00090000
001000+ + 00100000
001100+*********************************************************** 00110000
001200+ + 00120000
001300+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM". + 00130000
001400+ + 00140000
001500+ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. + 00150000
001600+ + 00160000
001700+ + 00170000
001800+ U.S. GOVERNMENT USERS RESTRICTED RIGHTS - + 00180000
001900+ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY + 00190000
002000+ GSA ADP SCHEDULE CONTRACT WITH IBM CORP. + 00200000
002100+ + 00210000
002200+ LICENSED MATERIALS - PROPERTY OF IBM. + 00220000
002300+ + 00230000
002400+*********************************************************** 00240000
002500+ + 00250000
002600+ STATUS: V1 R2 M0 + 00260000
002700+ + 00270000
002800+ FUNCTION: + 00280000
002900+ THIS IS THE COBOL CONTROL BLOCK USED TO INTERFACE BETWEEN + 00290000
003000+ - DPROP OR DXT + 00300000
003100+ AND + 00310000
003200+ - A USER'S FIELD EXIT ROUTINE (THESE USER + 00320000
003300+ EXIT ROUTINES ARE CALLED BY DXT 'USER DATA TYPE + 00330000
003400+ EXIT ROUTINES') + 00340000
003500+ + 00350000
003600+ THERE IS ONE CONTROL BLOCK FOR EACH FIELD + 00360000
003700+ EXIT ROUTINE, LASTING FOR THE DURATION OF THE EXIT + 00370000
003800+ IN VIRTUAL STORAGE. + 00380000
003900+ FOR SYNCH PROPAGATION IN MPP REGIONS: + 00390000
004000+ - THIS IS THE DURATION OF THE IMS PROGRAM CONTROLLER + 00400000
004100+ SUBTASK. + 00410000
004200+ FOR SYNCH PROPAGATION IN BATCH/BMP REGIONS, FOR + 00420000
004300+ ASYNCH PROPAGATION, AND FOR CCU PROCESSING: + 00430000
004400+ - THIS IS THE DURATION OF THE JOBSTEP. + 00440000
004500+ + 00450000
004600+*********************************************************** 00460000
004700+ IMPORTANT NOTES: + 00470000
004800+ + 00480000
004900+ + 00490000
005000+ - SINCE THE SAME USER EXIT ROUTINE CAN BE INVOKED BOTH + 00500000
005100+ BY DPROP AND BY DXT: CHANGES TO THIS CONTROL BLOCK MUST + 00510000

Figure 100 (Part 1 of 4). COBOL Interface Control Block for a Field Exit Routine
005200+ BE COORDINATED BETWEEN DPROP DEVELOPMENT AND DXT * 00520000
005300+ DEVELOPMENT. * 00530000
005400+ * 00540000
005500---------------------------* 00550000
005600+ * 00560000
005700+ CHANGE ACTIVITY: * 00570000
005800+ * 00580000
005900************** END OF CONTROL BLOCK SPECIFICATION ************ 00590000
006000+ 06060000
006100 01 EKYRCUDC. 00610000
006200+ 06620000
006300-----------------------------* 00630000
006400+ THIS SECTION OF THE CB MAY NOT BE MODIFIED BY THE EXIT. * 00640000
006500-----------------------------* 00650000
006600+ 00660000
006700 02 UDTPFX. 00670000
006800 03 UDTTNAME PIC X(8). 00680000
006900+ NAME OF CONTROL BLOCK. MAPS TO DVRXCUDT 00690000
007000 03 FILLER PIC X(24). 00700000
007100+ RESERVED FOR DXT USE 00710000
007200+ 00720000
007300 02 UDTPFXE. 00730000
007400+ PREFIX EXTENSION 00740000
007500+ 00750000
007600 03 UDTPNMOD. 00760000
007700 04 UDTCALL PIC X(2). 00770000
007800+ TYPE OF CALL TO EXIT...
007900 88 UDTCRTG VALUE "ST". 00790000
008000+ "SRC --> TRG" CALL ISSUED BY DXT AND BY DPROP 00800000
008100+ DURING HR MAPPING. EXIT SHOULD CONVERT THE DATA 00810000
008200+ FROM THE USER FORMAT TO THE DPROP FORMAT. 00820000
008300 88 UDTCGSR VALUE "TS". 00830000
008400+ "TRG --> SRC" CALL ISSUED BY DPROP DURING RH 00840000
008500+ MAPPING. EXIT SHOULD CONVERT DATA FROM THE 00850000
008600+ DPROP FORMAT TO THE USER FORMAT. 00860000
008700 88 UDTCDEFN VALUE "DF". 00870000
008800+ **NOT** ISSUED BY DPROP. 00880000
008900+ DEFINITION CALL ISSUED BY DXT-UIM FOR EACH DATATYPE. 00890000
009000+ EXIT CAN VALIDATE REQUEST AND RETURN REQUIRED VALUES 00900000
009100+ 00910000
009200 04 FILLER PIC X(2). 00920000
009300+ RESERVED FOR DXT USE 00930000
009400 04 UDTPNVRN. 00940000
009500+ ENVIRONMENTAL INFORMATION 00950000
009600 05 UDTOPSYS PIC X(4). 00960000
009700+ OPERATING SYSTEM CALLING PROGRAM IS EXECUTING IN: 00970000
009800 88 UDTOSMVS VALUE "MVS ". 00980000
009900+ INDICATES DXT IS RUNNING IN MVS/370 ENVIRONMENT. 00990000
010000 88 UDTOSKA VALUE "XA ". 01000000
010100+ INDICATES DXT IS RUNNING IN MVS/XA ENVIRONMENT. 01010000
010200 88 UDTOSESA VALUE "ESA ". 01020000

Figure 100 (Part 2 of 4). COBOL Interface Control Block for a Field Exit Routine
Figure 100 (Part 3 of 4). COBOL Interface Control Block for a Field Exit Routine
015400*        LENGTH INDICATOR FOR DPROP FORMAT (DXT ONLY) 01540000
015500 88 UDTPBYIN VALUE "N". 01550000
015600*        LENGTH OF DPROP FORMAT RESIDES WITH THE DEFINITION. 01560000
015700 88 UDTPBYIV VALUE "V". 01570000
015800*        LENGTH OF DPROP FORMAT VARIES AND MUST BE RETURNED 01580000
              AT "DEFINITION" TIME. 01590000
015900*        FILLER PIC X(1). 01600000
016100*        RESERVED FOR DXT USE 01610000
016200 04 UDTPBYTV PIC S9(4) COMP. 01620000
016300*        LENGTH OF FIELD IN DPROP FORMAT 01630000
016400 04 UDTSCL1 PIC X(1). 01640000
016500*        SCALE INDICATOR FOR DPROP FORMAT (DXT ONLY) 01650000
016600 88 UDTSCLN VALUE "N". 01660000
016700*        SCALE OF DPROP FORMAT RESIDES WITH THE DEFINITION. 01670000
016800 88 UDTSCLV VALUE "V". 01680000
016900*        SCALE OF DPROP FORMAT VARIES AND MUST BE RETURNED 01690000
              AT "DEFINITION" TIME. 01700000
017100 04 FILLER PIC X(3). 01710000
017200*        RESERVED FOR DXT USE 01720000
017300 04 UDTSCLV PIC S9(4) COMP. 01730000
017400*        VALUE OF SCALE IN DPROP FORMAT 01740000
017500*------------------------------------------------------------------ 01750000
017600*        THIS SECTION IS THE COMMUNICATIONS AREA BETWEEN THE EXIT 01760000
017700*        AND DPROP/DXT. 01770000
017800*------------------------------------------------------------------ 01780000
017900 03 UDTPXICOM. 01790000
018000*        DEFINE A COMMUNICATIONS AREA 01800000
018100 04 UDTPRPR2 PIC X(24). 01810000
018200*        RESERVED 01820000
018300 04 UDTPCRT1 PIC X(128). 01830000
018400 04 UDTPXTWS REDEFINES UDTPCRT1 PIC X(128). 01840000
018500*        USER EXIT WORK AREA 01850000
018600 04 UDTPTRWD PIC X(1). 01860000
018700*        'ENTERED' FLAG - SET TO X BY EXIT TO INDICATE 01870000
018800*        THAT DATA TYPE ROUTINE HAS BEEN ENTERED. 01880000
018900 04 UDTPXCT PIC X(1). 01890000
019000*        'IN-CONTROL' FLAG - SET TO X BY EXIT TO INDICATE 01900000
019100*        THAT DATA TYPE ROUTINE IS IN CONTROL. 01910000
019200 04 UDTPXCTD PIC X(1). 01920000
019300*        DATA RETURNED FROM EXIT IS NULL. 01930000
019400 88 UDTPXNLY VALUE "Y". 01940000
019500*        RETURN DATA IS NULL. 01950000
019600 88 UDTPXNNL VALUE "N". 01960000
019700*        RETURNED DATA IS NOT NULL. 01970000
019800 04 FILLER PIC X(1). 01980000
019900*        RESERVED 01990000
020000 04 UDTPXREC PIC S9(8) COMP. 02000000
020100*        USER EXIT RETURN CODE 02010000
020200*        0 - SUCCESSFUL COMPLETION 02020000
020300*        OTHER - ERROR ENDED 02030000
020400*        IF CALLER IS DPROP: 02040000
020500*        4  ----> RUP WILL USE IST USUAL 02050000
020600*        ERROR HANDLING LOGIC. 02060000
020700*        ¾4  ----> RUP ABENDS 02070000
020800 04 UDTPXMSG PIC X(64). 02080000
020900*        USER EXIT MESSAGE TEXT INSERTED INTO DPROP/DXT 02090000
021000*        MESSAGE. IF CALLER IS DPROP, TEXT WILL BE 02100000
021100*        INSERTED INTO MSG EKRY9701/EKRY971E. 02110000

Figure 100 (Part 4 of 4). COBOL Interface Control Block for a Field Exit Routine
Sample Field Exit Control Block for PL/I

Figure 101 shows an example of the EKYRCUDT control block in PL/I. This control block, called EKYRCUDP, resides in the (EKYSAMP) library.

```pli
1/******************************************************************************
  * START OF CONTROL BLOCK SPECIFICATION
  *
  * Control Block name:
  *   EKYRCUDP
  *
  * Descriptive name:
  *   DPROP PL/I field exit interface.
  *
  * PL/I version of EKYRCUDT.
  *
*******************************************************************************/

  * THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
  *
  *
  * U.S. GOVERNMENT USERS RESTRICTED RIGHTS -
  * USE, duplication, or disclosure restricted by
  * GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
  *
  * LICENSED MATERIALS - PROPERTY OF IBM.
  *
*******************************************************************************/

  * Status: V1 R2 M0
  *
  * Function:
  *   This is the PL/I control block used to interface between
  *   - DPROP or DXT
  *   and
  *   - a user's field exit routine (these user exit routines
  *     are called by DXT 'user data type exit routines')
  *
  * There is one control block for each field exit routine,
  * lasting for the duration of the exit in virtual storage.
  *
  * For synchronous propagation in MPP regions:
  *   - this is the duration of the IMS program controller subtask.
  * For synchronous propagation in Batch/BMP regions, for
  * asynchronous propagation, and for CCU processing:
  *   - this is the duration of the jobstep.

*******************************************************************************/

  * Important Notes:
  *
  * Since the same user exit routine can be invoked both by
  * DPROP and by DXT: changes to this control block must be
  * coordinated between DPROP development and DXT development.

*******************************************************************************/

  * Change activity:
  *   None.

*******************************************************************************/

IDECLARE UDT_PTR POINTER;
DECLARE 1 EKYRCUDP BASED (UDT_PTR),
*******************************************************************************/

  * This section of the control block may not be modified by the exit.

*******************************************************************************/

  2 UDTPFX, /* DXT prefix (32 bytes) */
  3 UDTTNAME CHAR(8), /* Name of block, "DVRXCUDT" */
  3 UDTXADDR POINTER, /* Address of loaded routine */

*******************************************************************************/

Figure 101 (Part 1 of 4). PL/I Interface Control Block for a Field Exit Routine
Figure 101 (Part 2 of 4). PL/I Interface Control Block for a Field Exit Routine
Figure 101 (Part 3 of 4). PL/I Interface Control Block for a Field Exit Routine
4 UDENTRD CHAR(1),
  /********************************************************************
  * 'entered' flag - set to "X" by exit to indicate that *
  * data type routine has been entered. *
  /********************************************************************/

4 UDTINCTL CHAR(1),
  /********************************************************************
  * 'in-control' flag - set to "X" by exit to indicate that *
  * data type routine is in control. *
  /********************************************************************/

4 UDTNULLT CHAR(1),
  /********************************************************************
  * "Y" - indicates data returned from exit is NULL. *
  * "N" - indicates data returned from exit is NOT NULL *
  /********************************************************************/

4 FILLDIC CHAR(1), /* Reserved */

4 UDTXRETC FIXED BIN(31),
  /********************************************************************
  * User exit return code: *
  * 0 - successful completion else error encountered. *
  * if caller is DPROP: *
  * 4 ----> RUP will use its usual error handling logic. *
  * >4 ----> RUP ABENDS. *
  /********************************************************************/

4 UDTXMESG CHAR(64);
  /********************************************************************
  * user exit message text inserted into DPROP/DXT *
  * message if caller is DPROP, text will be inserted *
  * into message EKYR97D/EYKR971E. *
  /********************************************************************/
Sample Field Exit Control Block for C

Figure 102 shows an example of the EKYRCUDT control block in C. This control block, called EKYRCUDK, resides in the (EKYSAMP) library.

```c
typedef struct EKYRCUDT {
  /* This section of the control block may not be modified by the exit. */
  /* Start of control block specification */
  char udttname[8]; /* Name of block, "DVRXCUDT" */
  char *udtaddr;  /* Address of loaded routine */
  unsigned char fill2mzqr[2mzqr]; /* Reserved for DXT use */
  char udtpfx[32]; /* DXT prefix (32 bytes) */
  /* End of control block specification */
} EKYRCUDT;

Figure 102 (Part 1 of 4). C Interface Control Block for a Field Exit Routine
/***/  /* Prefix extension (300 bytes) udtfpfx */  /* (76 bytes) utpmod */  
unsigned char udtcall[2];  
/*****  Type of call to exit:  
* "DF" - This is NOT issued by DPROP. But is a  
* definition call issued by DXT-UIM for each datatype.  
* Exit can validate request and return required values.  
* "ST" - Source->target call issued by DXT and by DPROP  
* during HRF mapping. Exit should convert data from the  
* DL/I IO area format to DPROP supported target datatype.  
* "TS" - Target->source call issued by DPROP during RH  
* mapping. Exit should convert data from DPROP supported  
* datatype to DL/I IO area format not issued by DXT.  
* 
*****

unsigned char fill02[2];  /* Reserved for DXT use */  /* Environmental information */  (12 bytes) udtenvrn */

unsigned char udtropsys[4];  
/*****  Operating system calling program is executing in:  
* When UDTOPSYS = "MVS ".  
* This indicates DXT is running in MVS/370 environment.  
* When UDTOPSYS = "XA ".  
* This indicates DXT is running in MVS/XA environment.  
* When UDTOPSYS = "ESA ".  
* This indicates DXT is running in MVS/ESA environment.  
* 
*****

unsigned char udtrans[4];  
/*****  DB/DC environment: 'BAT ' if IMS Batch/BMP  
* 'MPP ' if IMS MPP  
* 'IFP ' if Fast Path  
* 'CICS' if CICS  
* '' if none of the above.  
* 
*****

#pragma page(1)

unsigned char udtprog[4];  
/*****  Calling program: 'DXT ' if DXT  
* 'DPRS' if DPROP synchronous PROP  
* 'DPRU' if DPROP asynchronous PROP  
* 'DPRC' if DPROP CCU processing  
* 'DPRL' if DPROP DLU  
* 
*****

unsigned char udtexit[8];  /* Name of the user exit */  
char *udtpcbcb;  /* *** DXT only ***  
Address list of all PCB addresses if DLI environment */

unsigned char udtprop[24];  /* Additional work space */  
/*****  This section contains data pertinent to the source field  
* 
*****

unsigned char udstype[2];  /* source data type value */  
unsigned char udstbyt;  

---

*Figure 102 (Part 2 of 4). C Interface Control Block for a Field Exit Routine*
/*************************************************************************/
/* Source bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t3b[3]; /* Reserved for DXT use */
short udttbyti; /* Number of target bytes */
unsigned char udtttype[2]; /* Target data type value */
unsigned char udttbytv; /* Target bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t1a; /* Reserved for DXT use */
short udtsbytv; /* Number of source bytes */
unsigned char udtsclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t1b; /* Reserved for DXT use */
short fillmzqr!z!t3a[3]; /* Reserved for DXT use */
unsigned char udttclsi; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t01b; /* Reserved for DXT use */
short udtssbytv; /* Source bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t01a; /* Reserved for DXT use */
short udtssclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t03a[3]; /* Reserved for DXT use */
short udttsclv; /* Value of target scale */
unsigned char udtsscli; /* Value of source scale */
unsigned char udtttype[2]; /* Target data type value */
unsigned char udttbytv; /* Target bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char udttscli; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t03b[3]; /* Reserved for DXT use */
short udttsbytv; /* Number of target bytes */
unsigned char udttclsi; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char udttscli; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t012[2]; /* Reserved for DXT use */
short udttclsi; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char udttscli; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t011; /* Reserved for DXT use */
short udttsbytv; /* Number of target bytes */
unsigned char udtttype[2]; /* Target data type value */
unsigned char udttbytv; /* Target bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t010; /* Reserved for DXT use */
short fillmzqr!z!t03a[3]; /* Reserved for DXT use */
unsigned char udttsclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t013; /* Reserved for DXT use */
short udtssbytv; /* Number of source bytes */
unsigned char udtssclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t014; /* Reserved for DXT use */
short udttsclv; /* Value of target scale */
unsigned char udttscli; /* Target scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t015; /* Reserved for DXT use */
short udtsscli; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t016; /* Reserved for DXT use */
short udttsbytv; /* Number of target bytes */
unsigned char udtttype[2]; /* Target data type value */
unsigned char udttbytv; /* Target bytes indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t017; /* Reserved for DXT use */
short fillmzqr!z!t03b[3]; /* Reserved for DXT use */
unsigned char udttsclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/
unsigned char fillmzqr!z!t018; /* Reserved for DXT use */
short udtssbytv; /* Number of source bytes */
unsigned char udtssclv; /* Source scale indicator (DXT only). */
/* "N" - indicates value resides with the definition. */
/* "V" - indicates value is returned at "DEF" call to UIM. */
***************************************************************************/

Figure 102 (Part 3 of 4). C Interface Control Block for a Field Exit Routine

Appendix C. Sample Field Exit Control Blocks 379
unsigned char udtnullt;
/*
   * "Y" - indicates data returned from exit is NULL.
   * "N" - indicates data returned from exit is NOT NULL
   */
unsigned char fill0ic; /* Reserved */
long udtxretc;
/* User exit return code:
   * 0 - successful completion else error encountered.
   * if caller is DPROP:
   * 4 ---> RUP will use its usual error handling logic.
   * >4 ---> RUP ABENDS.
   */
unsigned char udtxmesg[64];
/*
   * user exit message text inserted into DPROP/DXT
   * message if caller is DPROP, text will be inserted
   * into message EKYR9701/EYKR971E.
   */
} EKYRCUDT;

#pragma page(1)

Figure 102 (Part 4 of 4). C Interface Control Block for a Field Exit Routine
Appendix D. Sample Propagation Exit Control Blocks

This appendix contains sample Propagation exit control blocks which map the existing DPROP interface control blocks. This appendix provides the exit control blocks in three languages:

- COBOL
- PL/I
- C

Chapter 4, “Propagation Exit Routines” on page 153 shows the Assembler version of the Propagation exit control blocks.
Sample Propagation Exit Control Blocks for COBOL

Figure 103 shows an example of the Propagation Exit control blocks in COBOL. These control blocks, called EKYRPCCC, EKYRCDLC, EKYHCHCC, and EKYHCQ2C reside in the DPROP Sample Source library (EKYSAMP).

COBOL Propagation Exit Interface (PIC)

Figure 103 shows a COBOL Propagation Exit Interface.

```
000100 000200+ START OF CONTROL BLOCK SPECIFICATION 00010000
000300+ CONTROL BLOCK NAME: EKYRPCCC (PIC) 00030000
000400+ DESCRIPTIVE NAME: DPROP COBOL PROPAGATION EXIT INTERFACE 00040000
000500+ COBOL VERSION OF EKYRPCIC 00050000
000600+ ALL RIGHTS RESERVED. 00060000
000600+ LICENSED MATERIALS - PROPERTY OF IBM. 00060000
000700+ CONTROL BLOCK NAME: 00070000
000800+ COBOL VERSION OF EKYRPCIC 00080000
000900+ U.S. GOVERNMENT USERS RESTRICTED RIGHTS - ALL RIGHTS RESERVED. 00090000
001000+ LICENSED MATERIALS - PROPERTY OF IBM. 00100000
001100+ LICENSED MATERIALS - PROPERTY OF IBM. 00110000
001200+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM". 00120000
001300+ COPYRIGHT IBM CORP. 1989, 1992. 00130000
001400+ ALL RIGHTS RESERVED. 00140000
001500+ COPYRIGHT IBM CORP. 1989, 1992. 00150000
001600+ LICENSED MATERIALS - PROPERTY OF IBM. 00160000
001700+ LICENSED MATERIALS - PROPERTY OF IBM. 00170000
001800+ COPYRIGHT IBM CORP. 1989, 1992. 00180000
001900+ LICENSED MATERIALS - PROPERTY OF IBM. 00190000
002000+ COPYRIGHT IBM CORP. 1989, 1992. 00200000
002100+ LICENSED MATERIALS - PROPERTY OF IBM. 00210000
002200+ LICENSED MATERIALS - PROPERTY OF IBM. 00220000
002300+ LICENSED MATERIALS - PROPERTY OF IBM. 00230000
002400+ LICENSED MATERIALS - PROPERTY OF IBM. 00240000
002500+ LICENSED MATERIALS - PROPERTY OF IBM. 00250000
002600+ LICENSED MATERIALS - PROPERTY OF IBM. 00260000
002700+ LICENSED MATERIALS - PROPERTY OF IBM. 00270000
002800+ LICENSED MATERIALS - PROPERTY OF IBM. 00280000
002900+ LICENSED MATERIALS - PROPERTY OF IBM. 00290000
003000+ LICENSED MATERIALS - PROPERTY OF IBM. 00300000
003100+ LICENSED MATERIALS - PROPERTY OF IBM. 00310000
003200+ LICENSED MATERIALS - PROPERTY OF IBM. 00320000
003300+ LICENSED MATERIALS - PROPERTY OF IBM. 00330000
003400+ LICENSED MATERIALS - PROPERTY OF IBM. 00340000
003500+ LICENSED MATERIALS - PROPERTY OF IBM. 00350000
003600+ LICENSED MATERIALS - PROPERTY OF IBM. 00360000
003700+ LICENSED MATERIALS - PROPERTY OF IBM. 00370000
003800+ LICENSED MATERIALS - PROPERTY OF IBM. 00380000
003900+ LICENSED MATERIALS - PROPERTY OF IBM. 00390000
004000+ LICENSED MATERIALS - PROPERTY OF IBM. 00400000
004100+ LICENSED MATERIALS - PROPERTY OF IBM. 00410000
004200+ LICENSED MATERIALS - PROPERTY OF IBM. 00420000
004300+ LICENSED MATERIALS - PROPERTY OF IBM. 00430000
004400+ LICENSED MATERIALS - PROPERTY OF IBM. 00440000
004500+ LICENSED MATERIALS - PROPERTY OF IBM. 00450000
004600+ LICENSED MATERIALS - PROPERTY OF IBM. 00460000
004700+ LICENSED MATERIALS - PROPERTY OF IBM. 00470000
004800+ LICENSED MATERIALS - PROPERTY OF IBM. 00480000
004900+ LICENSED MATERIALS - PROPERTY OF IBM. 00490000
005000+ LICENSED MATERIALS - PROPERTY OF IBM. 00500000
005100+ LICENSED MATERIALS - PROPERTY OF IBM. 00510000
```

Figure 103 (Part 1 of 5). COBOL Propagation Exit Interface
Figure 103 (Part 2 of 5). COBOL Propagation Exit Interface
010300*-----------------------------------------------* 01030000
010400* 01040000
010500 02 PICENTRD PIC X. 01050000
010600* SET BY EXIT ROUTINE TO 'X', INDICATES 01060000
010700* THAT EXIT HAS BEEN ENTERED 01070000
010800 02 PICINCL PIC X. 01080000
010900* SET BY EXIT ROUTINE TO 'X', INDICATES 01090000
011000* THAT EXIT IS IN CONTROL 01100000
011100* 01110000
011200*** RETURN CODE AND ERROR MESSAGE 01120000
011300* 01130000
011400 02 PICXRETC PIC S9(4) COMP. 01140000
011500* RETURN CODE 01150000
011600* ...4: SQL ERROR. SQL ERROR CODE IS IN THE FIELD 01160000
011700* SQLCODE OF THE SQLCA 01170000
011800* ...8: DLI ERROR. AIBRETRN, AIBREASN AND DL/I 01180000
011900* STATUS CODE IN PCB POINTED BY AIBRSA1 01190000
012000* ..12: ERROR OTHER THAN SQL ERROR: 01200000
012100* SOME RESOURCES NOT AVAILABLE 01210000
012200* ..16: ERROR OTHER THAN SQL ERROR: 01220000
012300* NOT A RESOURCE AVAILABILITY PROBLEM. 01230000
012400* ..20: SHOULD NOT OCCUR/SHOULD ABEND 01240000
012500 02 PICXMESG. 01250000
012600* USER EXIT ERROR/WARNING MESSAGE 01260000
012700* DPROP WILL WRITE THE MESSAGE TO VARIOUS 01270000
012800* DESTINATIONS ACCORDING TO USUAL DPROP/RUP 01280000
012900* ERROR HANDLING LOGIC. 01290000
013000 03 PICXML1. 01300000
013100* 1ST MESSAGE LINE 01310000
013200 04 PICXMLS1 PIC X(8). 01320000
013300* ...8 BYTES MESSAGE ID 01330000
013400 04 PICXMLGB PIC X. 01340000
013500* ...ONE BLANK 01350000
013600 04 PICXMLTX PIC X(61). 01360000
013700* ...61 TEXT BYTES IN 1ST MESSAGE LINE 01370000
013800 03 PICXML2 PIC X(70). 01380000
013900* 2ND MESSAGE LINE 01390000
014000 03 PICXML3 PIC X(70). 01400000
014100* 3RD MESSAGE LINE 01410000
014200 03 PICXML4 PIC X(70). 01420000
014300* 4TH MESSAGE LINE 01430000
014400 02 FILLER PIC X(12). 01440000
014500* RESERVED FOR DPROP 01450000
014600* 01460000
014700*** NAME OF OBJECTS ASSOCIATED WITH ERROR 01470000
014800* 01480000
014900 02 PICDBN PIC X(8). 01490000
015000* DBNAME ASSOCIATED WITH THE ERROR 01500000
015100 02 PICSEGN PIC X(8). 01510000
015200* SEG NAME ASSOCIATED WITH THE ERROR 01520000
015300 02 PICTABQ PIC X(8). 01530000

Figure 103 (Part 3 of 5). COBOL Propagation Exit Interface
Figure 103 (Part 4 of 5). COBOL Propagation Exit Interface
Figure 103 (Part 5 of 5). COBOL Propagation Exit Interface
Figure 104 shows a COBOL DL/I capture interface.
005200 02  XPCBREL   PIC XX.              00520000
005300*   XPCB RELEASE INDICATOR          00530000
005400 02  XPCBEXIT   PIC X(8).           00540000
005500*   SEGMENT USER EXIT NAME           00550000
005600 02  XPCBCRC   PIC S9(4) COMP.      00560000
005700*   RETURN-CODE                      00570000
005800 02  XPCBCRCSC  PIC S9(4) COMP.     00580000
005900*   REASON-CODE                      00590000
006000 02  XPCBDBD   PIC X(8).            00600000
006100*   PHYSICAL DATA BASE NAME          00610000
006200 02  XPCBVERA   POINTER.            00620000
006300*   ADDRESS OF DB VERSION ID         00630000
006400 02  XPCBSEG   PIC X(8).            00640000
006500*   PHYSICAL SEGMENT NAME             00650000
006600 02  XPCBCALL   PIC X(4).           00660000
006700*   "CALL FUNCTION" DEFINED BY IMS/ESA 00670000
006800*   ISRT: INSERT                      00680000
006900*   REPL: REPLACE                    00690000
007000*   DLET: DELETE                     00700000
007100*   CASC: CASCADING DELETE           00710000
007200*   DLJP: NOW ALSO DELETED FROM LOGICAL PATH 00720000
007300 02  XPCBCPCALL  PIC X(4).          00730000
007400*   "PHYSICAL UPDATE TYPE" DEFINED BY IMS 00740000
007500*   ISRT: INSERT                      00750000
007600*   REIN: RE-INSERT VIA LOGICAL PATH  00760000
007700*   REPL: REPLACE                    00770000
007800*   DLET: DELETE                     00780000
007900*   DLPP: DELETED ONLY FROM PHYSICAL PATH 00790000
008000 02  FILLED     PIC X(4).           00800000
008100*   RESERVED                           00810000
008200 02  XPCBPBCA  POINTER.             00820000
008300*   ADDRESS OF DB PCB                 00830000
008400 02  XPCBPBCN  PIC X(8).            00840000
008500*   NAME OF DB PCB                    00850000
008600 02  XPCBINQA  POINTER.             00860000
008700*   ADDRESS OF "INDY" OUTPUT          00870000
008800 02  XPCBIOPA  POINTER.             00880000
008900*   ADDRESS OF I/O PCB                00890000
009000 02  FILLER    PIC S9(4) COMP.      00900000
009100*   RESERVED                           00910000
009200 02  XPCBCKEYL  PIC S9(4) COMP.     00920000
009300*   LENGTH OF CONCATENATED KEY         00930000
009400 02  XPCBCKEYA  POINTER.             00940000
009500*   ADDRESS OF CONCATENATED KEY        00950000
009600 02  XPCBSDBD   POINTER.             00960000
009700*   ADDRESS OF XSDB FOR DATA           00970000
009800 02  XPCBSDDB  POINTER.              00980000
009900*   ADDRESS OF XSDB FOR REPL DATA      00990000
010000 02  XPCBSDBP  POINTER.              01000000
010100*   ADDRESS OF XSDB FOR PATH DATA      01010000
010200 02  FILLER     PIC X(12).           01020000

Figure 104 (Part 2 of 4). COBOL DL/I Capture Interface
Figure 104 (Part 3 of 4). COBOL DL/I Capture Interface

Appendix D. Sample Propagation Exit Control Blocks
Figure 104 (Part 4 of 4). COBOL DL/I Capture Interface

015400*---------------------------------------------------------------* 0154000
015500* 0155000
015600 01  DBPCB.
015700* 0157000
015800 02  DBPCR0BD  PIC X(8).  01580000
015900*  DBD NAME  01590000
016000 02  DBPCBLEY  PIC XX.  01600000
016100*  LEVEL FEEDBACK  01610000
016200 02  DBPCBSTC  PIC XX.  01620000
016300*  DBPCBPXY  PIC X(8).  01630000
016400 02  DBPCBPXY  PIC X(8).  01640000
016500*  DBPCBPXY  PIC S9(8) COMP.  01650000
016600 02  DBPCBPXY  PIC S9(8) COMP.  01660000
016700*  DBPCBMKL  PIC S9(8) COMP.  01670000
016800 02  DBPCBMKL  PIC S9(8) COMP.  01680000
016900*  DBPCBMKL  PIC S9(8) COMP.  01690000
017000 02  DBPCBMKL  PIC S9(8) COMP.  01700000
017100*  DBPCBMKL  PIC S9(8) COMP.  01710000
017200 02  DBPCBMKL  PIC S9(8) COMP.  01720000
017300*  DBPCBMKL  PIC S9(8) COMP.  01730000
017400*  DBPCBMKL  PIC S9(8) COMP.  01740000
017500 02  DBPCBMKL  PIC S9(8) COMP.  01750000
017600*  DBPCBMKL  PIC S9(8) COMP.  01760000
017700*  DBPCBMKL  PIC S9(8) COMP.  01770000
017800*---------------------------------------------------------------* 01780000

390  Customization Guide
COBOL HUP Exit Communication Block (HEC)

Figure 105 shows a COBOL HUP exit communication block.

```cobol
000100 ******************** START OF CONTROL BLOCK SPECIFICATION ********************
000200  * 00020000
000300  CONTROL BLOCK NAME:  * EKYHCHEC (HEC)
000400  * 00040000
000500  * 00050000
000600  DESCRIPTIVE NAME:  * 00060000
000700  DPROP COBOL HUP EXIT COMMUNICATION BLOCK  * 00070000
000800  * 00080000
000900  COBOL VERSION OF EKYHCHEC  * 00090000
001000  * 00100000
001100  ************************************************************************************
001200  * 00120000
001300  THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".  * 00130000
001400  * 00140000
001500  5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992.  * 00150000
001600  ALL RIGHTS RESERVED.  * 00160000
001700  * 00170000
001800  U.S. GOVERNMENT USERS RESTRICTED RIGHTS -  * 00180000
001900  USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY  * 00190000
002000  GSA ADP SCHEDULE CONTRACT WITH IBM CORP.  * 00200000
002100  * 00210000
002200  LICENSED MATERIALS - PROPERTY OF IBM.  * 00220000
002300  * 00230000
002400  ************************************************************************************
002500  * 00250000
002600  STATUS: V1 R2 M  * 00260000
002700  * 00270000
002800  FUNCTION:  * 00280000
002900  THIS IS THE COBOL CONTROL BLOCK USED TO PASS INFORMATION  * 00290000
003000  GOT BY DPROP FROM THE DB2 CHANGED DATA CAPTURE EXIT  * 00300000
003100  (USING IFI CALLS) TO THE PROPAGATION EXIT ROUTINE.  * 00310000
003200  * 00320000
003300  THE HEC IS NEWLY BUILD FOR EACH EXIT CALL AND DOES  * 00330000
003400  CONTAIN DATA TO BE RETAINED BETWEEN EXIT CALLS.  * 00340000
003500  * 00350000
003600  CHANGE ACTIVITY:  * 00360000
003700  * 00370000
003800  ******************** END OF CONTROL BLOCK SPECIFICATION ********************
003900  00390000
004000  01 HEC.  00400000
004100  02 HECQWHS  00410000
004200  02 HECQW  PIC X(8).  00420000
004300  EYE CATCHER  00430000
004400  02 FILLER  PIC X(8).  00440000
004500  RESERVED  00450000
004600  02 HECQW  00460000
004700  ----- POINTERS TO IFI HEADER AREAS  00470000
004800  02 HECQWHS  00480000
004900  02 HECQW  00490000
005000  ADDRESS OF THE DB2 IFI STANDARD HEADER AREA  00500000
005100  02 HECQWHC  00510000
```

Figure 105 (Part 1 of 2). COBOL HUP Exit Communication Block
Figure 105 (Part 2 of 2). COBOL HUP Exit Communication Block
Figure 106 shows a COBOL IFC copyarea for IFCIDS 0185.

```
000100*************** START OF CONTROL BLOCK SPECIFICATION *************** 00010000
000200+   * 00020000
000300+ CONTROL BLOCK NAME:  EKYHCQ2C  000400+   * 00040000
000500+   * 00050000
000600+ DESCRIPTIVE NAME:   * 00060000
000700+ DPROP COBOL IFC COPYAREA FOR IFCIDS 0185  000800+   * 00080000
000900+ COBOL VERSION OF A PORTION OF ASM MACRO DSNQW02  000900+   * 00100000
001000+   * 00110000
001100+--------------------------------------------------------------------------------------
001200+   * 00120000
001300+ THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".  001400+   * 00140000
001500+ 5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992.  001600+   * 00150000
001700+ ALL RIGHTS RESERVED.  001800+   * 00160000
001900+ U.S. GOVERNMENT USERS RESTRICTED RIGHTS -  001900+   * 00170000
001900+ USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY  001900+   * 00180000
002000+ GSA ADP SCHEDULE CONTRACT WITH IBM CORP.  002100+   * 00190000
002100+   * 00210000
002200+ LICENSED MATERIALS - PROPERTY OF IBM.  002300+   * 00220000
002300+   * 00230000
002400+--------------------------------------------------------------------------------------
002500+   * 00240000
002600+ STATUS: V1 R2 M0  002700+   * 00250000
002800+   * 00260000
002800+   * 00270000
002900+ COPYAREA FOR ICF EVENTS.  002900+   * 00280000
003000+   * 00290000
003000+   * 00300000
003100+ QW0185 IS WRITTEN FOR READS REQUESTS FOR IFCID 185.  003100+   * 00310000
003200+ IT CONTAINS A HEADER SECTION WHICH IS FOLLOWED BY  003200+   * 00320000
003300+ A DATA SECTION.  003300+   * 00330000
003400+   * 00340000
003500+ THE DATA PORTION OF QW0185 BEGINS WITH FIELD:  003500+   * 00350000
003600+ - QW0185DD, IF QW0185TP=S  003600+   * 00360000
003700+ OR  003700+   * 00370000
003800+ - QW0185DR, IF QW0185TP=D  003800+   * 00380000
003900+   * 00390000
004000+   * 00400000
004100+ IF QW0185TP = S, THE DATA PORTION CONSISTS OF  004100+   * 00410000
004200+ FIELDS FOLLOWED BY AN ARBITRARY NUMBER OF OCCURRENCES  004200+   * 00420000
004300+ OF THE QW0185VR STRUCTURE.  004300+   * 00430000
004400+   * 00440000
004500+ IF QW0185TP = D, THE DATA PORTION CONSISTS OF:  004500+   * 00450000
004600+ ----> THE DATA ROW, IF QW0185RC = 0  004600+   * 00460000
004700+ OR  004700+   * 00470000
004800+ ----> AN ERROR MESSAGE, OTHERWISE.  004800+   * 00480000
004900+   * 00490000
005000+--------------------------------------------------------------------------------------
005100+   * 00500000
005100+--------------------------------------------------------------------------------------
```

Figure 106 (Part 1 of 4). COBOL IFC Copyarea for IFCIDS 0185
005200 01  QW0185  PIC X.  00520000
005300*  00530000
005400*---------------------------------------------------------------*  00540000
005500*  QW0185A IS THE STRUCTURE CONTAINING THE TABLE DESCRIPTION  +  00550000
005600*  IN ITS DATA PORTION (QW0185P=S).  +  00560000
005700*  THE DATA PORTION (QW0185DD) CONSISTS OF 4 FIELDS FOLLOWED  +  00570000
005800*  AN ARBITRARY NUMBER OF OCCURRENCES OF THE QW0185VR STRUCTURE  +  00580000
006000*  AN ARBITRARY NUMBER OF OCCURRENCES OF THE QW0185VR STRUCTURE  +  00600000
006100*  +  00610000
006200*---------------------------------------------------------------*  00620000
006300*  00630000
006400 01  QW0185A REDEFINES QW0185.  00640000
006500*  00650000
006600 02  QW0185LN  PIC 9(8) COMP.  00660000
006700*  LENGTH OF TOTAL DB2CDC DATA  00670000
006800 02  QW0185TP  PIC X.  00680000
006900*  TYPE OF CONTROL BLOCK  00690000
007000  88  QW0185DS  VALUE "S".  00700000
007100*  DB2CDC TABLE DESCRIPTION  00710000
007200  88  QW0185DD  VALUE "D".  00720000
007300*  DB2CDC DATA ROW  00730000
007400  02  FILLER  PIC XXX.  00740000
007500*  RESERVED  00750000
007600  02  QW0185RC  PIC X(4).  00760000
007700*  REASON CODE DESCRIBING ERROR FOR THIS DATA PORTION  00770000
007800  02  QW0185OT.  00780000
007900*  QUALIFIED TABLE NAME  00790000
008000  03  QW0185CR  PIC X(8).  00800000
008100*  CREATOR OF TABLE (AUTH ID)  00810000
008200  03  QW0185TB  PIC X(18).  00820000
008300*  TABLE NAME  00830000
008400  02  QW0185TS  PIC X(10).  00840000
008500*  TIMESTAMP OF TABLE DESCRIPTION FROM CATALOG  00850000
008600  02  QW0185TL  PIC X(10).  00860000
008700*  TIMESTAMP OF LOG BUFFER CI WHEN IT IS EXTERNALIZED  00870000
008800*  OR WHEN THE BUFFER IS INITIALIZED  00880000
008900  02  QW0185UR  PIC X(8).  00890000
009000*  RBA OF THE FIRST LOG RECORD FOR THIS UNIT OF WORK.  00900000
009100  02  QW0185LR  PIC X(8).  00910000
009200*  RBA OF LOG RECORD FROM WHICH THIS DB2CDC DATA ROW  00920000
009300  02  FILLER  PIC XX.  00930000
009400*  OPERATION CODE, NOT USED WHEN QW0185P=S.  00940000
009500  02  QW0185RI  PIC X(8).  00950000
009600*  OPERATION CODE QUALIFIER.  00960000
009700  88  QW0185RE  VALUE "RI".  00970000
009800*  RESULT OF A REFERENTIAL CONSTRAINT ENFORCEMENT OF  00980000
009900*  A DELETE SET NULL OR CASCADE, IF QW0185P = "D".  00990000
010000  02  FILLER  PIC X(8).  01000000
010100*  RESERVED  01010000
010200*  01020000

Figure 106 (Part 2 of 4). COBOL IFC Copyarea for IFCIDS 0185
Figure 106 (Part 3 of 4). COBOL IFC Copyarea for IFCIDS 0185

01300♦--------------------------------------------------------------- DATA PORTION --------------------------------- 0130000
01400♦ 02 QW01B5D0. 0140000
01500♦ 02 QW01B5ID PIC X(8). 0150000
01600♦ "EYE CATCHER = "CDCDD " 0160000
01700♦ 03 QW01B51D PIC X(8). 0170000
01800♦ 03 QW01B5RC PIC S9(8) COMP. 0180000
01900♦ 03 QW01B5NS PIC S9(4) COMP. 0190000
02000♦ TOTAL NUMBER OF OCCURRENCES OF QW01B5VR 0200000
02100♦ NUMBER OF COLS DESCRIBED BY OCCURRENCES OF QW01B5VR 0210000
02200♦ OCCURS 1. 0220000
02300♦ DESCRIBES A COLUMN IN A CAPTURED TABLE 0230000
02400♦ 04 QW01B5ST PIC S9(4) COMP. 0240000
02500♦ OFFSET OF THIS COLUMN INTO THE DATA ROW 0250000
02600♦ 04 FILLER REDEFINES QW01B5SI. 0260000
02700♦ 05 FILLER PIC XX. 0270000
02800♦ 05 QW01B5S5 PIC S9(4) COMP. 0280000
02900♦ OFFSET IS IN A HALF WORD 0290000
03000♦ 04 QW01B5SN. 0300000
03100♦ LENGTH OF NAME AND NAME OF THE COLUMN 0310000
03200♦ 05 QW01B5SL PIC S9(4) COMP. 0320000
03300♦ LENGTH OF COLUMN NAME 0330000
03400♦ 05 QW01B5CN PIC X(30). 0340000
03500♦ NAME OF COLUMN 0350000
03600♦--------------------------------------------------------------- 0360000
03700♦--------------------------------------------------------------- 0370000
03800♦ 03 QW01B5B IS THE STRUCTURE CONTAINING THE DATA ROW OR ERROR 0380000
03900♦ MESSAGE IN ITS DATA PORTION (QW01B5TP=D). 0390000
04000♦ 03 QW01B5RC = 0 0400000
04100♦ 04 THE DATA PORTION (QW01B5DR) CONSISTS OF: 0410000
04200♦ - THE DATA ROW, IF QW01B5RC = 0 0420000
04300♦ 04 QW01B5VR - AN ERROR MESSAGE, OTHERWISE. 0430000
04400♦ 04 QW01B5SR 0440000
04500♦ 04 QW01B5V 0450000
04600♦ 04 QW01B5S 0460000
04700♦--------------------------------------------------------------- 0470000
04800♦ 01 QW01B5B REDEFINES QW01B5. 0480000
04900♦ 02 FILLER PIC X(74). 0490000
05000♦ 02 QW01B5FC PIC XX. 0500000
05100♦--------------------------------------------------------------- 0510000
05200♦--------------------------------------------------------------- 0520000
05300♦--------------------------------------------------------------- 0530000

Appendix D. Sample Propagation Exit Control Blocks  395
**Figure 106 (Part 4 of 4).** COBOL IFC Copyarea for IFCIDS 0185

```
01400*  OPERATION CODE, USED WHEN QW0185TP=0.  01540000
015500*  IT HAS ONE OF THE FOLLOWING VALUES:  01550000
015600  88  QW0185IN  VALUE "IN".  01560000
015700*  INSERT  01570000
015800  88  QW0185UB  VALUE "UB".  01580000
015900*  UPDATE BEFORE IMAGE  01590000
016000  88  QW0185UA  VALUE "UA".  01600000
016100*  UPDATE AFTER IMAGE  01610000
016200  88  QW0185DE  VALUE "DE".  01620000
016300*  DELETE  01630000
016400  02  FILLER  PIC X(8).  01640000
016500*  REDEFINITION OF 8 BYTES MORE.  01650000
016600*  01660000
016700*---------------------------- DATA PORTION -----------------------------* 01670000
016800*  01680000
016900  02  QW0185DR  PIC X OCCURS 1.  01690000
017000*  DATA ROW OR ERROR MESSAGE  01700000
017100*  01710000
017200*---------------------------------------------* 01720000
```
Sample Propagation Exit Control Blocks for PL/I

Figure 107 on page 398 shows an example of the Propagation Exit control block in PL/I. These control blocks, called EKYRCPCP, EKYRCDLP, EKYHCQ2P, and EKYHCQ2P, reside in the (EKYSAMP) library.

PL/I Propagation Exit Interface (PIC)

Figure 107 on page 398 shows a PL/I Propagation exit interface.
Figure 107 (Part 1 of 3). PL/I Propagation Exit Interface
2 PICDBLEV CHAR(1), /* Debug level in effect. Hex'02':
   external trace of propagating SQL statements and DL/I calls. */
2 FILL01 CHAR(1), /* Reserved */
2 PICPTD POINTER, /* Address of DPROP PTD */
2 PICPRID CHAR(8), /* PR-ID */
2 PICPRSET CHAR(8), /* PRSET-ID */
2 PICPRST CHAR(26), /* PR timestamp */
2 FILL02 CHAR(2), /* Reserved */
2 PICPCBLA CHAR(8), /* PCB label as specified on PR */
2 FILL56 CHAR(56), /* Reserved */
2 PICOPSYS CHAR(4), /* Operating system. 'ESA': MVS/ESA */
2 PICTRANS CHAR(4), /* IMS region type:
   'MPP' = MPP region
   'IFP' = IMS fast path region
   'BMP' = IMS BMP region
   'BAT' = IMS batch region
   *' = none of above */
2 PICPROGM CHAR(4), /* calling program
   'DPRS' - DPROP synch propagation
   'DPRA' - DPROP asynch propagation */
2 FILL12A CHAR(12), /* Reserved for DPROP */
1 /* This section is used by exit to provide information to DPROP */
2 PICENTRD CHAR(1), /* Set by exit routine to 'X', to
   indicate that exit has been entered */
2 PICINCTL CHAR(1), /* Set by exit routine to 'X', to
   indicate that exit is in control */
/* Return code and error message */
2 PICXRETC FIXED BIN(15), /* Return code:
   4 = SQL error. SQL error code is in
   the field SQLCODE of the SQLCA.
   8 = DLI error. AIBRETRN, AIBREASN
   and DL/I status code in PCB
   pointed by AIBRSA1.
   12 = error other than SQL error,
   some resources not available.
   16 = error other than SQL error, not
   a resource availability problem.
   20 = should not occur/should abend. */
2 PICXMESSD, /* User exit error/warning message DPROP
   will write the message to various
   destinations according to the usual
   DPROP/RUP error handling logic.
   (280 byte area) */
3 PICXML1, /* 1st message line (70 text bytes) */
4 PICXMLSIG CHAR(8), /* 8 byte message ID */
4 PICXMLSIGB CHAR(1), /* one blank */
4 PICXMTEXT CHAR(61), /* 61 text bytes */
3 PICXML2 CHAR(70), /* 2nd message line (70 text bytes) */

Figure 107 (Part 2 of 3). PL/I Propagation Exit Interface

Appendix D. Sample Propagation Exit Control Blocks
3 PICML3 CHAR(70), /* 3rd message line (70 text bytes) */
3 PICML4 CHAR(70), /* 4th message line (70 text bytes) */
2 FILL2B CHAR(12), /* Reserved for DPROP */

*******************************************************************************/
* Names of objects associated with error
*******************************************************************************/
2 PICDBN CHAR(8), /* DBD name */
2 PICSEGN CHAR(8), /* Segment name */
2 PICTABQ CHAR(8), /* Table name qualifier */
2 PICTABN CHAR(18), /* Table name */
2 FILL14 CHAR(14), /* Reserved for DPROP */

*******************************************************************************/
* Exit Work Area
*******************************************************************************/
2 FILLFLO FLOAT BIN(8), /* for double word alignment */
(in ASM: DS mzqr!z!tD)
2 PICSWORK CHAR(256), /* Work area for the exit */
2 FILL16A CHAR(16), /* Reserved for DPROP */

*******************************************************************************/
* SQL communication area (SQLCA).
*******************************************************************************/
2 PICSQLCA CHAR(136),
2 FILL16B CHAR(16),

*******************************************************************************/
* DLI application interface block (AIB).
*******************************************************************************/
2 PICAIB, /* AIB initialized by DPROP */
3 AIBID CHAR(8), /* Eyecatcher */
3 AIBLEN FIXED BIN(31), /* DFSAIB ALLOCATED LENGTH */
3 AIBSFUNC CHAR(8), /* Subfunction code */
3 AIBRSM1 CHAR(8), /* Resource name 1 */
3 AIBRSM2 CHAR(8), /* Resource name 2 */
3 FB1(2) FIXED BIN(31), /* Reserved */
3 AIBOALEN FIXED BIN(31), /* Output area length (max) */
3 AIBOAUASE FIXED BIN(31), /* Output area length (used) */
3 FB2(2) FIXED BIN(31), /* Reserved */
3 FIXB(2) FIXED BIN(15), /* Reserved */
3 AIBRETNRN FIXED BIN(31), /* Return code */
3 AIBREASN FIXED BIN(31), /* Reason code */
3 FB3 FIXED BIN(31), /* Reserved */
3 AIBRSA1 POINTER, /* Resource address 1 */
3 AIBRSA2 POINTER, /* Resource address 2 */
3 AIBRSA3 POINTER, /* Resource address 3 */
3 FB4(10) FIXED BIN(31), /* Reserved */
2 FB5(4) FIXED BIN(31); /* Reserved */
PL/I (RUP) DL/I Capture Interface

Figure 108 shows a PL/I DL/I capture interface.

```
/*---------------------- START OF CONTROL BLOCK SPECIFICATION ------------*/
  * Control Block name:
  * EKYRCDLP
  *
  * Descriptive name:
  * DPROP RUP: PL/I DL/I capture interface.
  * PL/I VERSION OF EKYRCDLP.
  *
*******************************************************************************/

  * THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
  *
  * ALL RIGHTS RESERVED.
  *
  * U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
  * GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
  *
  * LICENSED MATERIALS - PROPERTY OF IBM.
  *
*******************************************************************************/

  * STATUS: V1 R2 Mmzqr!z!t
  *
  * FUNCTION:
  *
  * EKYRCDLP is an include library member providing descriptions
  * for the interface-areas used to communicate between
  * - DL/I changed data capture
  * - the EKYRUPmzqr!z!tmzqr!z!t DL/I changed data exit routine
  *
  * EKYRCDLP contains descriptions of following areas:
  * - the DL/I XPCB
  * - the DL/I XSDB
  * - the DL/I DBPCB
  *
  * CHANGE ACTIVITY:
  * None
  *
*******************************************************************************/

END OF CONTROL BLOCK SPECIFICATION

```
CASC: Cascading delete
DLLP: now also deleted from logical path /*
2 XPCBPCALL CHAR(4), /* "Physical Update Type" defined by IMS
   ISRT: Insert
   REIN: Re-insert via logical path
   REPL: Replace
   DLET: Delete
   DLPP: Deleted only from physical path /*
2 FILL CHAR(4), /* Reserved */
2 XPCBPCBA POINTER, /* Address of DB PCB */
2 XPCBCBN CHAR(6), /* Name of DB PCB */
2 XPCBINOA POINTER, /* Address of "INQY" output */
2 XPCBIDPA POINTER, /* Address of I/O PCB */
2 FILLER FIXED BIN(15), /* Reserved */
2 XPCBCKEYL FIXED BIN(15), /* Length of concatenated key */
2 XPCBCKEYLA POINTER, /* Address of concatenated key */
2 XPCBXSDBD POINTER, /* Address of XSDB for data */
2 XPCBXSDBB POINTER, /* Address of XSDB for REPL data */
2 XPCBXSDBP POINTER, /* Address of XSDB for path data */
2 XPCBFL1 FIXED BIN(31), /* Reserved */
2 XPCBFL12 FIXED BIN(31), /* Reserved */
2 XPCBFL13 FIXED BIN(31), /* Reserved */
2 XPCBEXIWP POINTER, /* Address of 256-byte area reserved for exit */
2 XPCBFL4 FIXED BIN(31), /* Reserved */
2 XPCBFL5 FIXED BIN(31), /* Reserved */
2 XPCBTIMST CHAR(8), /* Timestamp of call */
2 XPCBFL6 FIXED BIN(31), /* Reserved */
1/*-------------------------------------------------------------------------------------*/
/* Extended Segment Data -- XSDB */
***************************************************************************************/
DECLARE XSDB_POINTER POINTER;
DECLARE 1 XSDB BASED(XSDB_POINTER),
   2 XSDBEYE CHAR(4), /* XSDB" eyecatcher */
   2 XSDBVER CHAR(2), /* XSDB version indicator */
   2 XSDBREL CHAR(2), /* XSDB release indicator */
   2 XSDBXSDBD POINTER, /* Next XSDB pointer */
   2 XSDBXSDB CHAR(8), /* Physical data base name */
   2 XSDBSEG CHAR(8), /* Physical segment name */
   2 XSDBPHP CHAR(1), /* Physical path accessibility */
      If value is "y" then segment is accessible via physical path.
      If value is "N" then segment is not accessible via physical path. */
   2 FILLER CHAR(3), /* Reserved */
   2 XSDBSEGELV FIXED BIN(15), /* Segment data base level */
   2 XSDBKEYL FIXED BIN(15), /* Length of physical key */
   2 XSDBKEYA POINTER, /* Address of physical key */
   2 XSDBFL1 FIXED BIN(15), /* Reserved */
   2 XSDBSEGEL FIXED BIN(15), /* Length of segment data */
   2 XSDBSEGPOA POINTER, /* Address of segment data */
   2 XSDBFL12 FIXED BIN(31), /* Reserved */
   2 XSDBFL13 FIXED BIN(31), /* Reserved */
   2 XSDBFL14 FIXED BIN(31); /* Reserved */

Figure 108 (Part 2 of 3). PL/I (RUP) DL/I Capture Interface
DECLARE 01 DBPCB,
  2 DBPCBDBD CHAR(8), /* DBD name */
  2 DBPCBLEV CHAR(2), /* Level feedback */
  2 DBPCBSTC CHAR(2), /* Status codes (returned to user) */
  2 DBPCBPPO CHAR(4), /* Processing options */
  2 DBPCBPFX FIXED BIN(31), /* Prefix address */
  2 DBPCBSF0 CHAR(8), /* Segment name feedback */
  2 DBPCBMKL FIXED BIN(31), /* Current length of KFBA or GSAM feedback area */
  2 DBPCBSNS FIXED BIN(31), /* Number of sensitive segments in the PCB */
  2 DBPCBKFD CHAR(0); /* Key feedback area */

Figure 108 (Part 3 of 3). PL/I (RUP) DL/I Capture Interface
PL/I HUP Exit Communication Block

Figure 109 shows a PL/I HUP exit communication block.

```plain
Figure 109 (Part 1 of 2). PL/I HUP Exit Communication Block

DECLARE HEC_POINTER POINTER;
DECLARE 1 HEC BASED(HEC_POINTER),
2 HEC/EYE CHAR(8), /* Eye catcher ("EKY HEC") */
2 HEC/RESV1 CHAR(8), /* Reserved */
2 HEC/QWHS POINTER, /* Addr DB2 IFI standard header area */
2 HEC/QWHC POINTER, /* Addr DB2 IFI correlation data area */
2 HEC/DCD POINTER, /* Address of CDC data description */
2 HEC/DCDA POINTER, /* Address of CDC data row: */
2 HEC/DCDB POINTER, /* Address of CDC data row: */
```

Figure 109 (Part 1 of 2). PL/I HUP Exit Communication Block
/ * Return code from IFI call */
2 HECRARC2 FIXED BIN(31), / * IFCRC2 reason code */

/ * DBDname/SEGname/PCBlabel area (mapped by HECDSLDS below) */
2 HECDBSLA POINTER, / * Address of DBD/SEG/PCBlabel area HECDSLDS */
2 HECDBSLN FIXED BIN(31), / * Number of entries in HECDSLDS */

/ * Reserved space */
2 HECRESV2 CHAR(16);

/*******************************************************************************/
* For propagation exit routines only, the HECDBSLA field points to *
* an area (for DB2 subexit routines this field is zero). This area *
* contains 24 byte entries (in top to bottom hierarchy) which were *
* defined to DPROP for the PR in process. The number of entries in *
* this list is contained in the HECDBSLN field. *
*******************************************************************************/
DECLARE HECDSLDS_PTR POINTER;
DECLARE 1 HECDSLDS(1) BASED(HECDSLDS_PTR),
2 HECDBDNM CHAR(8), / * DBD name */
2 HECSEGNM CHAR(8), / * SEGMENT name */
2 HECPCBNM CHAR(8); / * PCB label name */

*******************************************************************************/

Figure 109 (Part 2 of 2). PL/I HUP Exit Communication Block
PL/I IFC Copyarea for IFCIDS 0185

Figure 110 shows a PL/I IFC copyarea for IFCIDS 0185.

DECLARE HECCDCDD_PTR POINTER;
DECLARE 1 QWO185A BASED(HECCDCDD_PTR),

2 QWO185LN FIXED BIN(31), /* Length of total db2cdc data */
2 QWO185TP CHAR(1), /* type of control block */
   * S - DB2CDC table description *
   * D - DB2CDC data row */
2 FILLER1 CHAR(3), /* Reserved */
2 QWO185RC CHAR(4), /* Reason code describing error */
   * for this data portion */
2 QWO185OT, /* qualified table name */
3 QWO185CR CHAR(8), /* Creator of table (Auth id) */
3 QWO185TB CHAR(18), /* table name */
2 QWO185TS CHAR(1), /* Timestamp of table */
   * description from catalog */
2 QWO185TL CHAR(10), /* Timestamp of log buffer ci */
   * when it is externalized or */
   * when the buffer is initialized */
2 QWO185UR CHAR(8), /* RBA of the first log record */
   * for this unit of work. */
2 QWO185LR CHAR(8), /* RBA of log record from */
   * which this DB2CDC data row */
2 FILLER2 CHAR(2), /* Operation code, */
   * not used when QWO185TP=S. */
2 QWO185RI CHAR(2), /* operation code qualifier. */
   * R1 - result of a referential */
   * constraint enforcement of */
   * a delete set null or */
   * cascade, if QWO185TP = "D", */
   */
2 FILLER3 CHAR(6), /* Reserved */
1/**************************** DATA PORTION *****************************/

2 QWO185DD,
3 QWO185ID CHAR(8), /* Eye catcher = "CDCDD */
3 QWO185BC FIXED BIN(31), /* Length of the QWO185DD section */
3 QWO185NO FIXED BIN(15), /* Total number of occurrences */
   * of QWO185VR */
3 QWO185LD FIXED BIN(15), /* Number of cols described by */
   * occurrences of QWO185VR */
3 QWO185VR(1), /* Describes a column in */
   * a captured table */
4 QWO185ST FIXED BIN(15), /* Tells the data type of */
   * the column and whether */
   * it has an associated */
   * indicator variable */
4 QWO185LE FIXED BIN(15), /* Defines the external lg */
   * of a value from the column */
4 QWO185SD FIXED BIN(31), /* contains the CCSID */
4 QWO185SI, /* offset of this column */
   * into the data row */
5 FILLER CHAR(2),
5 QWO185SX FIXED BIN(15), /* Offset is in a half word */
4 QWO185SN, /* Length of name and */
   * name of the column */
5 QWO185NL FIXED BIN(15), /* Length of column name */
5 QWO185CN CHAR(30); /* Name of column */
DECLARE HECCDA_PTR POINTER;
DECLARE 1 QW0185B BASED(HECCDA_PTR),
  2 FILLER5 CHAR(74), /* Redefinition of 74 bytes of the header portion */
  2 QW0185PC CHAR(2), /* Operation code, used when QW0185TP=d. */
  * IN - Insert
  * UB - Update before image
  * UA - Update after image
  * DE - Delete
  2 FILLER6 CHAR(8),

Figure 110 (Part 3 of 3). PL/I IFC Copyarea for IFCIDS 0185
Sample Propagation Exit Control Blocks for C

Figure 111 on page 410 shows an example of the Propagation Exit control blocks in C. These control blocks, called EKYRCCK, EKYRCDLK, EKYHCHCK, and EKYHCQ2K, reside in the (EKYSAMP) library.

C Propagation Exit Interface (PIC)

Figure 111 on page 410 shows a C Propagation exit interface.
typedef struct /* PICXMESG */
{
    /* User exit error/warning message DPROP
       will write the message to various
       destinations according to the usual
       DPROP/RUP error handling logic.
       (280 byte area) */
    unsigned char picxmsgi[8]; /* 8 byte message id */
    unsigned char picxmsgb; /* one blank */
    unsigned char picxmtxt[61]; /* 61 text bytes */
} PICXML1;

Figure 111 (Part 1 of 4). C Propagation Exit Interface
typedef struct {
    PICXML1  picxml1;
    unsigned char picxml2[70]; /* 2nd message line (70 text bytes) */
    unsigned char picxml3[70]; /* 3rd message line (70 text bytes) */
    unsigned char picxml4[70]; /* 4th message line (70 text bytes) */
} PICMESG;

typedef struct {
    /* PICAIB */
    unsigned char aibid[8]; /* Eyecatcher */
    long    aiblen;     /* DFSAIB allocated length */
    unsigned char aibsfunc[8]; /* Subfunction code */
    unsigned char aibrsm1[8]; /* Resource name 1 */
    unsigned char aibrsm2[8]; /* Resource name 2 */
    long    fbi[2];     /* Reserved */
    long    aiboalen;   /* Output area length (max) */
    long    aiboausage; /* Output area length (used) */
    long    fb2[2];     /* Reserved */
    short   fixb[2];    /* Reserved */
    long    aibetnm;    /* Return code */
    long    aibrean;    /* Reason code */
    long    fb3;        /* Reserved */
    char    aibrtsa1;   /* Resource address 1 */
    char    aibrtsa2;   /* Resource address 2 */
    char    aibrtsa3;   /* Resource address 3 */
    long    fb4[10];    /* Reserved */
} PICAIB;

typedef struct {
    /* EKYRCPIC */
    unsigned char piceye[8]; /* Eye catcher ("EKYRCPIC") */
    unsigned char picexit[8]; /* Name of the exit routine */
    unsigned char piccall[2]; /* Type of call to exit:
                                 HR = hierarchical to relational
                                 RH = relational to hierarchical */
    unsigned char picdblev; /* Debug level in effect. hex'02':
                              external trace of propagating SQL
                              statements and DL/I calls. */
    unsigned char fillb01;  /* Reserved */
    char    *picptd;       /* Address of DPROP PTD */
    unsigned char picprid[8]; /* PR-ID */
    unsigned char picprset[8]; /* PRset-ID */
    unsigned char picprtsst[26]; /* PR timestamp */
}

Figure 111 (Part 2 of 4). C Propagation Exit Interface
unsigned char fillmzqr2[2]; /* Reserved */
unsigned char picpcbla[8]; /* PCB label as specified on PR */
unsigned char fill56[56]; /* Reserved */
unsigned char picopsys[4]; /* Operating system. ESA : MVS/ESA */
unsigned char pictrans[4]; /* IMS region type:  
'MPP' = MPP region  
'IFP' = IMS fast path region  
'BMP' = IMS BMP region  
'BAT' = IMS batch region  
'' = none of above */
unsigned char picprogm[4]; /* Calling program:  
'DPRS' - DPROP synch propagation  
'DPRA' - DPROP asynch propagation */
unsigned char fill12a[12]; /* Reserved for DPROP */

#pragma page(1)

/* This section is used by exit to provide information to DPROP */
unsigned char picentrdr; /* Set by exit routine to 'X', to indicate that exit has been entered */
unsigned char picinctl; /* Set by exit routine to 'X', to indicate that exit is in control */

/* Return code and error message */
short picxretc; /* Return code:  
4 = SQL error. SQL error code is in the field SQLCODE of the SQLCA.  
8 = DLI error. AIBRETRN, AIBREASN and DL/I status code in PCB pointed by AIBRSA1.  
12 = error other than SQL error, some resources not available.  
16 = error other than SQL error, not a resource availability problem.  
20 = should not occur/should abend. */

PICXMESG picxmesg;
unsigned char fill12b[12]; /* Reserved for DPROP */

/* Names of objects associated with error */
unsigned char picdbn[8]; /* DBD name */
unsigned char picsegn[8]; /* Segment name */
unsigned char pictabq[8]; /* Table name qualifier */
unsigned char pictabn[18]; /* Table name */
unsigned char fill14[14]; /* Reserved for DPROP */

#pragma page(1)

---

Figure 111 (Part 3 of 4). C Propagation Exit Interface
The exit work area can be used to save information across calls to the exit (e.g. to save the addresses of getmaind areas across calls to the exit).

```c
unsigned char floater[4];
unsigned char picswork[256]; /* Work area for the exit */
unsigned char fill16a[16];    /* Reserved for DPROP */

/* SQL communication area (SQLCA). */
struct sqlca picsqsqlca;
unsigned char fill16b[16];
PICAIB picaib;

long fb5[4];       /* Reserved */
} EKYRCPIC;

#pragma page(1)
```

Figure 111 (Part 4 of 4). C Propagation Exit Interface

**C (RUP) DL/I Capture Interface**

Figure 112 on page 414 shows a C DL/I capture interface.
/*************** START OF CONTROL BLOCK SPECIFICATION *************/
/*
 * Control Block name: EKYRCDLK
 * Descriptive name: DPROP RUP: C DL/I capture interface.
 * C VERSION OF EKYRCDL1.
*/

/*
 * THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".
 * ALL RIGHTS RESERVED.
 * U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
 * LICENSED MATERIALS - PROPERTY OF IBM.
*/

/*
 * STATUS: V1 R2 M0
 * FUNCTION:
 * EKYRCDLI is a copyarea providing descriptions for the interface-areas used to communicate between
 * - DL/I changed data capture
 * - the EKYRUP00 DL/I changed data exit routine
 * EKYRCDLI generates descriptions of following areas:
 * - the DL/I XPCB
 * - the DL/I XSDB
 * - the DL/I DBPCB
 * CHANGE ACTIVITY:
 * None

*************** END OF CONTROL BLOCK SPECIFICATION *************/

#pragma page(1)

typedef struct
{
    unsigned char segikey[6];
    unsigned char segidat1[7];
    unsigned char segidat2[4];
    unsigned char segidat3[8];
} SEGI;

/**************************** Extended Segment Data -- XSDB ****************************/
typedef struct /* XSDB */
{
    unsigned char xsdbeye[4]; /* "XSDB" eyecatcher */
    unsigned char xsdbver[2]; /* XSDB version indicator */

Figure 112 (Part 1 of 3). C (RUP) DL/I Capture Interface
typedef struct XPCB {
    unsigned char xpcbeye[4];  /* XPCB" eyecatcher */
    unsigned char xpcbver[2];  /* XPCB version indicator */
    unsigned char xpcbrel[2];  /* XPCB release indicator */
    unsigned char xpcbexit[8]; /* Segment user exit name */
    short xpcbrbc;             /* Return-code */
    short xpcbrscn;            /* Reason-code */
    unsigned char xpcbdbd[8];  /* Physical data base name */
    char *xpcbvera;            /* Address of DBD version ID */
    unsigned char xpcbseg[8];  /* Physical segment name */
    unsigned char xpcbcall[4]; /* "Call function" defined by IMS/ESA ISRT: Insert 
                                REPL: Replace 
                                DLET: Delete 
                                CASC: Cascading delete 
                                DLLP: now also deleted from logical path */
    unsigned char xpcbpcall[4]; /* "Physical update type" defined by IMS ISRT: Insert 
                                  REIN: Re-insert via logical path 
                                  REPL: Replace 
                                  DLET: Delete 
                                  DLPP: Deleted only from physical path */
    unsigned char fill[4];     /* Reserved */
    char *xpcbpcba;            /* Address of DB PCB */
    unsigned char xpcbpcbn[8]; /* Name of DB PCB */
    char *xpcbinqa;            /* Address of "INQY" output */
    char *xpcbciopa;           /* Address of I/O PCB */
    short filler;             /* Reserved */
    short xpcbckeyl;          /* Length of concatenated key */
} XPCB;

#pragma page(1)

/* X E t x t e d  D a t a B a s e  P C B  --  X P C B */

Figure 112 (Part 2 of 3). C (RUP) DL/I Capture Interface
char *xcpcbckeya; /* Address of concatenated key */
char *xcpcbxsdbd; /* Address of XSDB for data */
char *xcpcbxsdbb; /* Address of XSDB for REPL data */
char *xcpcbxsdbp; /* Address of XSDB for path data */
long xpcbfil1; /* Reserved */
long xpcbfil2; /* Reserved */
long xpcbfil3; /* Reserved */
char *xcpcbexiwp; /* Address of 256-byte area reserved for exit */
long xpcbfil4; /* Reserved */
long xpcbfil5; /* Reserved */
unsigned char xpcbtimst[8]; /* Timestamp of call */
long xpcbfil6; /* Reserved */
}
XPCB;

#pragma page(1)

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* D B P C B */
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
typedef struct
{
    unsigned char DBPCBD[8]; /* DBD name */
    unsigned char DBPCBLEV[2]; /* Level feedback */
    unsigned char DBPCBSTC[2]; /* Status codes (returned to user) */
    unsigned char DBPCBPRO[4]; /* Processing options */
    long DBPCBFX; /* Prefix address */
    unsigned char DBPCBSFD[8]; /* Segment name feedback */
    long DBPCBMKL; /* Current length of KFBA or GSAM feedback area */
    long DBPCBNSS; /* Number of sensitive segments in the PCB */
    unsigned char DBPCBKFD; /* Key feedback area */
} DBPCB;

#pragma page(1)

Figure 112 (Part 3 of 3). C (RUP) DL/I Capture Interface

C HUP Exit Communication Block

Figure 113 on page 417 shows a C HUP exit communication block.
/********************** START OF CONTROL BLOCK SPECIFICATION **********************
* 
* Control Block name:
*   EKYHCHCK (HEC)
* 
* Descriptive name:
*   DPROP C HUP exit communication block.
* 
* C version of EKYHCHEC.
* 
******************************************************************************
* 
* THIS PRODUCT CONTAINS “RESTRICTED MATERIALS OF IBM”.
* 
* 
* U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY
* GSA ADP SCHEDULE CONTRACT WITH IBM CORP.
* 
* LICENSED MATERIALS  - PROPERTY OF IBM.
* 
******************************************************************************
* 
* Status: V1 R2 Mmzqr!z!t
* 
* Function:
*   This is the PL/I control block used to pass information received by DPROP from the DB2 changed data capture exit
*   (using IFI calls) to the propagation exit routine.
*   The HEC is newly built for each exit call and does contain data to be retained between exit calls.
* 
******************************************************************************
* 
* Change activity:
*   None

#pragma page(1)

******************************************************************************
* 
* For propagation exit routines only, the HECDBSLA field points to an area (for DB2 subexit routines this field is zero).
* this list is contained in the HECDBSLN field.
* 
******************************************************************************

typedef struct { unsigned char hecdbdnm[8]; /* DBD name */ unsigned char hecsegnm[8]; /* Segment name */ unsigned char hecpcbnm[8]; /* PCB label name */ } HECDSLDS1;

typedef struct /* HECDSLDS array */
{
  HECDSLDS1 hecdsls[30];
} HECDSLDS;

Figure 113 (Part 1 of 2). C HUP Exit Communication Block
typedef struct
{
    / * HEC */
    unsigned char heceye[8]; / * Eye catcher ("EKY HEC ") */
    unsigned char hecresv1[8]; / * Reserved */

    / * Pointers to IFI header areas */
    char *hecqwhs; / * Addr DB2 ifi standard header area */
    char *hecqwhc; / * Addr DB2 ifi correlation data area */

    / * Pointers to CDC data areas */
    void *heccdcdd; / * Address of CDC data description (always passed to exit) */
    void *heccdcda; / * Address of CDC data row: (always passed to exit)
                     - only data for INSERT/DELETE
                     - OR contains the after image for UPDATE operations */
    char *heccdcdb; / * Address of CDC data row: - Zero for INSERT and DELETE
                     - Otherwise, BEFORE image of row for UPDATE operations. */

    / * Return code from IFI call */
    long hecrarc2; / * IFCRC2 reason code */

    / * DBDname/SEGname/PCBlabel area (mapped by HECDSLDS below) */
    HECDSLDS *hecdbsla; / * Address of DBD/SEG/PCBlabel area HECDSLDS */
    long hecdbsln; / * Number of entries in hecdslds */

    / * Reserved space */
    unsigned char hecresv2[16]; / * reserved */
} HEC;

#pragma page(1)

Figure 114 (Part 2 of 2). C HUP Exit Communication Block

C IFC Copyarea for IFCIDS 0185

Figure 114 on page 419 shows a C IFC copyarea for IFCIDS 0185.
/** START OF CONTROL BLOCK SPECIFICATION **/

* Control Block name: EKYHCQ2K
* Descriptive name:
  * DPROP C IFC copyarea for IFCIDS 0185
  * C VERSION OF A PORTION OF ASM MACRO DSNDQW02

******************************************************************************

THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM".

5685-124 (C) COPYRIGHT IBM CORP. 1989, 1992. ALL RIGHTS RESERVED.

U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION, OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM CORP.

LICENSED MATERIALS - PROPERTY OF IBM.

******************************************************************************

STATUS: V1 R2 M

FUNCTION:

Copyarea for IFC events.

QW0185 is written for reads requests for IFCID 185.

It contains a header section which is followed by a data section.

The data portion of QW0185 begins with field:

- QW0185DD, if QW0185TP=S
  * or
- QW0185DR, if QW0185TP=D

If QW0185TP = S, the data portion consists of four fields followed by an arbitrary number of occurrences of the QW0185VR structure.

If QW0185TP = D, the data portion consists of:

--- the data row, if QW0185RC = 0
  * or
--- an error message, otherwise.

******************************************************************************

END OF CONTROL BLOCK SPECIFICATION

******************************************************************************

#pragma page(1)

typedef struct /* Describes a column in a captured table */
{
  short qw0185st; /* Tells the data type of the column and whether it has an associated indicator variable */
}

Figure 114 (Part 1 of 3). C IFC Copyarea for IFCIDS 0185
short qw0185le; /* Defines the external lg of a value from the column */
long qw0185sd; /* Contains the ccsid */
char filler[2];
short qw0185sx; /* Offset of this column into the data row */
short qw0185nl; /* Length of column name */
char qw0185cn[30]; /* Name of column */
}

QW0185VR;

#endif

typedef struct
{
  long qw0185ln; /* Length of total db2cdc data */
  char qw0185tp; /* Type of control block */
  * S - DB2CDC table description
  * D - DB2CDC data row */
  char filler1[3]; /* Reserved */
  char qw0185rc[4]; /* Reason code describing error */
  * for this data portion */
  char qw0185cr[8]; /* Creator of table (auth id) */
  char qw0185tb[18]; /* Table name */
  char qw0185ts[10]; /* Timestamp of table */
  * description from catalog */
  char qw0185tl[10]; /* Timestamp of log buffer CI */
  * when it is externalized or */
  * when the buffer is initialized */
  char qw0185ur[8]; /* RBA of the first log record */
  * for this unit of work. */
  char qw0185ir[8]; /* RBA of log record from */
  * which this DB2CDC data row */
  char filler2[2]; /* Operation code, */
  * not used when QW0185TP=S. */
  char qw0185ri[2]; /* operation code qualifier */
  * RI - result of a referential */
  * constraint enforcement of */
  * a delete set null or */
  * cascade, if QW0185TP = "D". */
  */
  char filler3[6]; /* reserved */
  */
} DATA PORTION

*******************************************************************************/

Figure 114 (Part 2 of 3). C IFC Copyarea for IFCIDS 0185
typedef struct {
    char filler5[74]; /* Redefinition of 74 bytes */
    char qw0185pc[2]; /* Operation code, */
                        /* used when QW0185TP=D. */
                        /* IN - Insert */
                        /* UB - Update before image */
                        /* UA - Update after image */
                        /* DE - Delete */
    char filler6[8];
} QW0185B;

Figure 114 (Part 3 of 3). C IFC Copyarea for IFCIDS 0185
This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, Program, or service may be used. Any functionally equivalent product, Program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the
Programming Interface Information

This publication is intended to help you administer IMS DataPropagator, hereafter called IMS DPROP.

This publication also documents general-use programming interface and associated guidance information provided by IMS DPROP.

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

**Notice**

This chapter documents general-use programming interface and associated guidance information.

**Trademarks**

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- AD/Cycle
- AT
- CICS
- CICS/ESA
- CICS/MVS
- COBOL/370
- Database 2
- DataPropagator
- DataRefresher
- DB2
- DFSMS
- DXT
- IBM
- IMS
- IMS Client Server/2
- IMS/ESA
- Information Warehouse
- Language Environment
- MVS
- MVS/ESA
- OS/390
- QMF
- RACF
- SAA
- z/OS
Bibliography

The IMS DataPropagator for z/OS
Version 3 Release 1 Library

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Book Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC27-1216</td>
<td>Administrators Guide for Log Asynchronous Propagation</td>
</tr>
<tr>
<td>GC27-1217</td>
<td>Administrators Guide for MQSeries Asynchronous Propagation</td>
</tr>
<tr>
<td>GC27-1215</td>
<td>Administrators Guide for Synchronous Propagation</td>
</tr>
<tr>
<td>SC27-1544</td>
<td>Concepts</td>
</tr>
<tr>
<td>GC27-1214</td>
<td>Customization Guide</td>
</tr>
<tr>
<td>GC27-1209</td>
<td>Diagnosis</td>
</tr>
<tr>
<td>GC27-1211</td>
<td>An Introduction</td>
</tr>
<tr>
<td>GC27-1212</td>
<td>Installation Guide</td>
</tr>
<tr>
<td>GC27-1213</td>
<td>Messages and Codes</td>
</tr>
<tr>
<td>GC27-1210</td>
<td>Reference</td>
</tr>
<tr>
<td>GC27-1208</td>
<td>Licensed Program Specification</td>
</tr>
</tbody>
</table>

Other Books Referenced in This Book

The following books are referred to in this book or might be helpful in understanding customization tasks:

- DB2 Administration Guide, SC26-4888
- DB2 Messages and Codes, SC26-4892
- DB2 SQL Reference, SC26-4890
- DXT Writing Exit Routines, SC26-4636
- IMS/ESA General Information, GC26-3068
- IMS/ESA Customization Guide, SC26-3064-00
- IMS/ESA Administration Guide: Database Manager, SC26-3065-00
- IMS/ESA Application Programming: DL/I Calls, SC26-3062-00
- IMS/ESA Utilities Reference: Database Manager, SC26-4627-00
- IMS/ESA Utilities Reference: System, SC26-4629-00
- OS/390 Language Environment Programming Reference, SC26-4841
- IBM SAA AD/Cycle PL/I MVS & VM Language Reference, SC26-3114
- Remote Recovery Data Facility Program Description and Operation, LY37-3710-03
Glossary of Terms and Abbreviations

A

abort record.  An IMS DataPropagator propagation log record (38nn or 5938), indicating that the associated unit of work will not be committed by IMS and should not be propagated to DB2.  Compare with commit record.

ACB.  Application control block. Located in IMS.

ACDC.  Asynchronous changed data capture.

Apply Program.  A component of IMS MQ-DPROP that reads the MQSeries messages containing the changed data and passes it to the RUP. RUP transforms the changed data into relational format and updates the DB2 target tables.

Archive utility. A utility that filters out propagation log records from the records written to the IMS logs and writes them to Changed Data Capture data sets (CDCDSs).

asynchronous changed data capture. An IMS function that captures the changes needed for IMS DPROP asynchronous propagation and saves them on the IMS logs. The function is mandatory for IMS DPROP asynchronous propagation and is either implemented by an SPE (IMS 3.1) or built into the program (subsequent releases of IMS).

asynchronous propagation. The propagation of data at a later time, not within the same unit of work as the update call.

Audit Extract utility. An IMS DPROP utility that inserts the IMS DPROP audit records written to SMF into the IMS DPROP audit table.

AUDU.  Audit Extract utility.

CDCDS Registration utility.  An IMS DPROP asynchronous propagation utility that registers new CDCDS to DBRC.

CDCDS Unregistration utility.  An IMS DPROP asynchronous propagation utility that deletes CDCDS entries from DBRC.

CDU.  CDCDS Unregistration utility.

CEC.  central electronics complex.

Changed Data Capture data set (CDCDS).  The data sets that the archive utility uses to store the IMS DPROP asynchronous propagation log records filtered during the archive process. CDCDSs contain only the propagation log records. These log records are used by the Selector in place of the corresponding SLDSs, that contain all IMS changes.

Changed Data Capture exit routine.  See DB2 Changed Data Capture exit routine.

Changed Data Capture function.  See DB2 Changed Data Capture function.

commit record.  An IMS DPROP asynchronous propagation log record (9928, 37nn, 41nn, or 5937) indicating that the associated unit of work has been committed by IMS and should be propagated to DB2. Compare with abort record.

collapsed key.  See “IMS concatenated key” and “conceptual concatenated key.”

conceptual concatenated key.  The conceptual concatenated key of a segment consists of the concatenated keys of the segment’s immediate physical parent and physical ancestors. Unlike the Conceptual fully Concatenated key, the conceptual concatenated key does not include the concatenated key of the segment itself.

conceptual fully concatenated key.  The conceptual fully concatenated key is an IMS DPROP concept useful for the propagation of entity segments that do not have a unique IMS fully concatenated key; but that are nevertheless uniquely identifiable.

The conceptual fully concatenated key of a segment consists of these parts:

- the concatenated key of the segment
- the concatenated keys of the segment’s physical parent and physical ancestors

The conceptual fully concatenated key is therefore the combination of these parts:

B

Batch Log data set.  A data set that an IMS batch job uses to store propagation log records needed for IMS DPROP asynchronous propagation.

C

CAF.  Call attach facility.

CCU.  Consistency Check utility.

CDCDS.  Changed Data Capture data sets.
• the IMS fully concatenated key
• the ID fields (if any) of the segment that contribute to the concatenated key of the segment
• the ID fields (if any) of the physical parent or ancestors that contribute to the concatenated keys of the physical parent or ancestor

So, the conceptual fully concatenated key is equal to that hypothetical IMS fully concatenated key, that you would see if including the ID fields into the IMS key-field at each hierarchical level.

The concept of conceptual fully concatenated key allows the support of segments with a unique conceptual fully concatenated key, much in the same way as segments with a unique IMS fully concatenated key.

concatenated key. The concatenated key is an IMS DPROP concept useful for the propagation of entity segments that are neither unique under their parent nor have a unique IMS key, but that are nevertheless uniquely identifiable through ID fields.

The concatenated key is a combination of these fields that identify the segment uniquely under its parent:
• the non-unique IMS key field (if any)
• ID fields

For segments having a unique IMS key field, the conceptual key and the IMS key field are identical.

Consistency Check utility (CCU). An IMS DPROP utility that checks whether the data that has been propagated between IMS and DB2 databases is consistent. If not, it reports the inconsistencies and generates statements the DBA can use to fix the inconsistencies. The CCU is applicable when generalized mapping cases are being used.

containing IMS segment. An IMS segment that contains internal segments (embedded structures) propagated by mapping case 3 Propagation Requests. It is referred to interchangeably as a “containing IMS segment” or “containing segment.”

containing segment. See containing IMS segment.

CRU. CDCDS Registration utility.

Data Extract Manager (DEM). A DataRefresher component that extracts the IMS data to which changes will subsequently be propagated. DEM also creates control statements for the DB2 Load utility to load the extracted IMS data into DB2 tables.

data propagation. The application of changes to one set of data to the copy of that data in another database system. See also synchronous propagation and IMS DPROP asynchronous propagation.

DataRefresher DEM. DataRefresher data extract manager.

DataRefresher Map Capture exit routine (MCE). See Map Capture exit routine.

DataRefresher UIM. See User Input Manager.

DBRM. Database Request Module.

DB2 commit count. The number of IMS commit records that the IMS DPROP asynchronous propagation receiver is to apply to DB2 before it issues a DB2 commit.

DB2 Changed Data Capture exit routine. The routine to which the DB2 Changed Data Capture function passes the DB2 changes it has captured for propagation. This routine can be the IMS DPROP HUP routine, that propagates data, or your own exit routine.

DB2 Changed Data Capture function. A DB2 function that captures the DB2 changes needed for data propagation.

DB2 Changed Data Capture subexit routine. An optional IMS DPROP exit routine invoked whenever the HUP is called by DB2 changed data capture. The DB2 Changed Data Capture subexit routine can typically be used to perform generalized functions such as auditing all of the captured DB2 changes.

DB2-to-IMS propagation. Propagation of changed DB2 tables to IMS segments. It can be either:
• One-way DB2-to-IMS propagation
• DB2-to-IMS propagation, as part of two-way propagation

DBD. Database definition. The collection of macroparameter statements that describes an IMS database. These statements describe the hierarchical structure, IMS organization, device type, segment length, sequence fields, and alternate search fields. The statements are assembled to produce database description blocks.

DBDLIB. Database definition library.

DBPCB. Database program communication block.
DEDB. Data entry database.

DEM. Data Extract Manager.

directory. See IMS DPROP directory.

DLU. DL/1 Load Utilities. IMS DPROP utilities that are used to create (or re-create) the IMS databases from the content of the propagated DB2 tables. You can use DLU if you have implemented DB2 to IMS or two-way propagation.

DPROP-NR. The abbreviation for IBM IMS DataPropagator MVS/ESA through Version 2.2. At Version 3.1 the product name changed to IMS DataPropagator, abbreviated as IMS DPROP.

E

EKYMQCAP. The Capture component of MQ-DPROP. EKYMQCAP is an IMS data Capture exit routine. It runs as an extension to the updating IMS application programs, but it is transparent to them. EKYMQCAP obtains the changed data from the IMS Data Capture function and sends this data via MQSeries messages to the Apply Program.

EKYRESLB Dynamic Allocation exit routine. An IMS DPROP exit routine that can be used to allocate dynamically the IMS DPROP load module library to the EKYRESLB DD-name.

entity segment. The data being mapped from IMS to DB2 comes from a single hierarchic path down to an entity segment and from any segments immediately subordinate to the entity segment. The segments subordinate to the entity segment can have zero or one occurrence beneath a single occurrence of the entity segment. This type of subordinate segment is called an extension segment (as it extends the data in the entity segment). See also mapping case 2.

extract request (ER). A DataRefresher request to extract IMS data. Extract requests become IMS DPROP propagation requests once they are validated by the IMS DPROP MCE.

F

Field exit routine. An IMS DPROP exit routine you can write to complement the logic of IMS DPROP's generalized mapping cases. Field exit routines are typically used to convert an individual IMS data field between a customer format IMS DPROP does not support and a format you have defined in your propagation request.

fully concatenated key. See IMS fully concatenated key and conceptual fully concatenated key.

G

generalized mapping cases. The mapping cases provided by IMS DPROP. See mapping case 1, mapping case 2 and mapping case 3.

group definition file. The file that the Group Unload utility (GUU) uses to store the IMS sources that it extracts from the IMS DPROP directory tables. See also, SCF Compare job and SCF Apply job.

Group Unload utility (GUU). The IMS DPROP asynchronous propagation utility that extracts details of all IMS sources for the specified propagation group from the IMS DPROP directory tables at the receiver site and writes them to the Group Definitions File. See also, SCF Compare job and SCF Apply job.
GUU. Group Unload utility.

Hierarchical update program (HUP). The IMS DPROP component that does the actual DB2-to-IMS propagation. HUP is the IMS DPROP-provided DB2 Changed Data Capture exit routine. The DB2 Changed Data Capture function calls HUP and provides to HUP the changed IMS rows.

Hierarchical to Relational propagation. This is one-way hierarchical to relational propagation: the one-way propagation of changed IMS segments to DB2 tables. The terms hierarchical to relational propagation and one-way IMS-to-DB2 propagation are interchangeable.

HUP. Hierarchical Update program.

HSSR. High speed sequential retrieval.

ID fields. Identification (ID) fields are non-key fields that:

- uniquely identify a segment under its parent
- do not change their value

Typical examples of IMS segments with ID fields, are segments where the data base administrator has not defined the ID fields as part of the IMS Key field. For example because the IMS applications need to retrieve the segment in another sequence than the ascending sequence of the ID fields.

Identification fields. See ID fields.

IMS concatenated key. For an IMS segment, the concatenated key consists of:

- The key of the segment's immediate parent, and
- The keys of the segment's ancestors

Unlike the IMS fully concatenated key of the segment, the concatenated key does not include the key of the segment itself.

A logical child segment has two concatenated keys: a physical concatenated key and a logical concatenated key. The physical concatenated key consists of the key of the segment's physical parent and the keys of the physical ancestors of the physical parent. The logical concatenated key consists of the key of the segment's logical parent and the keys of the physical ancestors of the logical parent.

IMS Data Capture exit routine. The routine to which the IMS Data Capture function passes the IMS changes it has captured for propagation. For synchronous propagation, this routine can be the IMS DPROP RUP routine, that propagates data, or your own exit routine. For IMS DPROP asynchronous propagation, the data capture exit routine is a program you write that gets the changed data from IMS. Other programs that you write will later invoke IMS DPROP with the changed IMS data.

IMS data capture function. An IMS function that captures the changes needed for data propagation.

IMS DPROP. The abbreviated name for the IBM IMS DataPropagator product. Previously, this product was called IMS DataPropagator, abbreviated as DPROP-NR.

IMS DPROP directory. A set of DB2 tables containing the mapping and control information necessary to perform propagation.

IMS fully concatenated key. For an IMS segment, the fully concatenated key consists of:

- The key of the segment,
- The key of the segment's immediate parent, and
- The keys of the segment's ancestors.

Unlike the IMS concatenated key of the segment, the fully concatenated key includes the key of the segment itself.

IMS INQY data. The first 9904 (update) record in each IMS unit of work (UOW) contains IMS INQY data (transaction name, PSB name, and user ID). This information is written to the PRDS for the propagation group as the first record of the UOW.

IMS log files. The files that IMS uses to store details of all changes to IMS data. See also, batch log data sets, online data sets (OLDSs), system log data sets (SLDSs), and Changed Data Capture data sets (CDCDSs).

IMS logical concatenated key. One of the two IMS concatenated keys of a logical child segment (the other is an IMS physical concatenated key). The logical concatenated key consists of:

- The key of the segment's logical parent, and
- The keys of the physical ancestors of the logical parent.

IMS physical concatenated key. One of the two IMS concatenated keys of a logical child segment (the other is an IMS logical concatenated key). The physical concatenated key consists of:

- The key of the segment's physical parent, and
- The keys of the physical ancestors of the physical parent.

IMS-to-DB2 propagation. This is the propagation of changed IMS segments to DB2 tables. Distinguish between:
• One-way IMS-to-DB2 propagation
• IMS-to-DB2 propagation, as part of two-way propagation

**internal segments.** Internal Segments is the IMS DPROP and DataRefresher term for structures embedded in IMS Segments, that are propagated through mapping case-3 propagation requests. Each embedded structure (i.e. each internal segment), is propagated to a different table; each occurrence of the embedded structure to one row of the table.

**invalid unit of work.** An IMS UOW that is missing a first record (containing the INQY data). If the IMS DPROP asynchronous propagation Selector detects an invalid unit, it responds according to what you specified on the INVUOW keyword of the SELECT control statements. If you specified:

**IGNORE** The Selector continues processing

**STOP** The Selector issues an error message and terminates

**ISC.** Inter-system communications.

**ISPF.** Interactive system production facility or Interactive structured programming facility.

**IXF.** Integrated exchange format.

**L**

**LOG-ASYNC.** The IMS log-based, asynchronous propagation functions of IMS DPROP.

Once the IMS log records are archived (IMS Online Logs) or de-allocated (IMS Batch Logs) by IMS and then stored in time-stamp sequence, LOG-DPROP reads the IMS logs to find the changed data and then stores the changed data in PRDS datasets. The Receiver component of IMS DPROP reads the PRDSs, transforms the data into the relational format, and applies the changes to the target DB2 tables.

See asynchronous propagation.

**logical concatenated key.** See IMS logical concatenated key

**M**

**Map Capture exit (MCE) routine.** The map capture exit routine provided by DPROP. MCE is used when you provide mapping information through DataRefresher. MCE is called by DataRefresher during mapping and data extract to perform various validation and checking operations. The IMS DPROP MCE should be distinguished from the DataRefresher Map Capture exit, the DataRefresher routine that calls MCE.

**mapping case.** A definition of how IMS segments are to be mapped to DB2 tables. IMS DPROP distinguishes between mapping case 1, mapping case 2, and user mapping cases.

**mapping case 1.** One of the generalized mapping cases provided by IMS DPROP. Mapping case 1 maps one single segment type, with the keys of all parents up to the root, to a row in a single DB2 table.

**mapping case 2.** One of the generalized mapping cases provided by IMS DPROP. Mapping case 2 maps one single segment type, with the keys of all parents up to the root, plus data from one or more immediately subordinate segment types (with a maximum of one occurrence of each segment type per parent), to a row in a single DB2 table.

**mapping case 3.** One of the generalized mapping cases provided by IMS DPROP. Mapping case 3 supports the propagation of segments containing embedded structures. A typical example of an embedded structure is a repeating group of fields. Each embedded structure can be propagated to/from a different table. Mapping case 3 propagates each occurrence of an embedded structure, with the key of the IMS segment, and the keys of the physical parent and ancestor, to/from a row of one DB2 table.

• the remaining data of the IMS segment (that is the fields that are not located in a embedded structure) can be propagated to/from another table.

**Mapping Verification and Generation (MVG).** An IMS DPROP component that validates the mapping information for each propagation request and stores it in the IMS DPROP directory. For a propagation request belonging to a generalized mapping case, MVG generates an SQL update module. MVG is invoked internally by MCE and MVGU.

**Mapping Verification and Generation utility (MVGU).** An IMS DPROP utility invoked by the DBA. MVGU creates propagation requests when DataRefresher is not used to provide mapping information (i.e., when you put the mapping information directly into the MVG input tables). MVGU also deletes or rebuilds propagation requests in the IMS DPROP directory.

**master table.** The IMS DPROP directory master table, that is created when IMS DPROP is initialized. It consists of one row, containing system and error information.

**MCE.** Map Capture exit routine.

**MIT.** Master Index Table.

430 Customization Guide
MQ-ASYNC. The MQSeries-based, asynchronous propagation functions of IMS DPROP.

An IMS Data Capture Exit routine provided by IMS DPROP obtains the IMS Database changes in real time from IMS and sends the changes via MQSeries messages to an IMS DPROP Apply program. The Apply program reads the MQSeries messages, transforms the data into relational format, and then applies the new data to the target DB2 tables.

MQ-ASYNC supports both near-real time propagation and automated point-in-time propagation.

MQSeries. A family of IBM licensed programs that provide message queuing services.

MQSeries for OS/390. The members of the MQSeries that run on OS/390 systems.

MSDB. Main storage database.

MSC. Multisystem communication.

MVG. Mapping Verification and Generation.

MVG input tables. A group of DB2 tables into which the DBA stores propagation request definitions when DataRefresher is not used to provide mapping information. Once the propagation requests are stored, the DBA invokes MVGU. MVGU invokes MVG, that validates the propagation request and copies the mapping definitions from the MVG input tables to the IMS DPROP directory.

MVGU. Mapping Verification and Generation utility.

P

PCB. Program communication block.

persistent MQSeries message. An MQSeries message that survives a restart of the MQSeries Queue Manager.

physical concatenated key. See IMS physical concatenated key.

Point In Time Propagation. An Asynchronous propagation is said to operate in ‘Point In Time’ mode, when the data content of the target databases matches the content of the source databases at a previous, clearly identified Point In Time. For example, a Point In Time Propagation can be used to reflect in the content of the target databases the logical end of a business day, or the logical end of business month, or the end of specific Batch jobstream that updated the source databases.

PR. Propagation request.

PR ID. Propagation request identifier.

PRCT. Propagation Request Control Table

PRDS. Propagation Request Data Set

PRDS register file. A data set created by the IMS DPROP asynchronous propagation Selector that contains details of the associated PRDS.

PRDS register table. An IMS DPROP directory table that is created at the Receiver site when IMS DPROP is installed. The table is initially empty and you must populate it, using the PRU REGISTER control statements.

PRDS Registration utility (PRU). An IMS DPROP asynchronous propagation utility that registers PRDSs in the PRDS Register Table.

propagation. See data propagation.

Propagation Data Stream. A stream of changed IMS data that flows in MQSeries messages from the Capture Component of IMS DPROP to the Apply Component of IMS DPROP. Propagation data streams are defined with PRSTREAM control statements in the //EKYTRANS file of EKYMQCAP.

propagation delay. The time elapsed between the update of the IMS source database by the application programs and the update of the target DB2 table by IMS DPROP.

Propagation exit routine. An IMS DPROP exit routine you can write to propagate data when the generalized
mapping cases don’t meet your needs. A Propagation exit routine must provide all the logic for data mapping, field conversion, and propagation.

propagation group. A subset of the propagation requests in the IMS DPROP directory propagation request table (IMS DPROP asynchronous only).

You can define as many propagation groups as you like, but any propagation request can be associated with one and only one propagation group.

propagation log records. IMS log records that the IMS DPROP asynchronous propagation Selector writes to PRDSs:

- 9904 (update) records
- Commit or abort records
- SETS/ROLS records

propagation request control table (PRCT). An IMS DPROP directory table that is created at the Receiver site when IMS DPROP is installed. It contains details of all propagation requests defined to IMS DPROP and, in combination with the RCT, enables the Receiver to ascertain:

- Which propagation requests are assigned to which Receivers
- The activity status of all defined Receivers
- The activity status of all propagation requests that are assigned to defined Receivers

Propagation Request data set (PRDS). A sequential file into which the IMS DPROP asynchronous propagation Selector writes all propagation log records for a propagation group.

propagation request (PR). A request to propagate data between IMS and DB2. You define propagation requests for each segment type that is to be propagated.

PR set. A group of logically related propagation requests, identified by having the same PRSET ID. PR sets are typically used when you propagate the same IMS data to multiple sets of DB2 tables.

PRU. PRDS Registration utility.

PSB. Program specification block.

R

RCT. Receiver control table.

Receiver. An IMS DPROP asynchronous propagation component that retrieves the propagation log records from a PRDS and passes them to the RUP, that uses them to update the DB2 target tables.

Applies to LOG-DPROP.

RECEIVER control statement. A control statement that is input directly into the IMS DPROP asynchronous propagation Receiver JCL to specify:

- The name of the Receiver that is to process a PRDS
- The names of the DB2 subsystem to be accessed and the DB2 plan
- The number of committed UOWs to process before a DB2 commit is issued

Applies to LOG-DPROP.

Receiver control table (RCT). An IMS DPROP directory table, that is created at the Receiver site when IMS DPROP is installed. The table is initially empty and you must populate it, using the SCU CREATEREC control statement. It contains details of all Receivers and, in combination with the PRCT, enables the Receiver to ascertain:

- Which propagation requests are assigned to which Receivers
- The activity status of all defined Receivers
- The activity status of all propagation requests that are assigned to defined Receivers

Applies to LOG-DPROP.

Relational to Hierarchical propagation. This is one-way relational to hierarchical propagation: the one-way propagation of changed DB2 tables to IMS segments. The terms relational to hierarchical propagation and one-way DB2-to-IMS propagation are interchangeable.

relational update program (RUP). The IMS DPROP component that does the actual IMS to DB2 propagation. RUP is the IMS DPROP-provided IMS Data Capture exit routine.

For synchronous propagation, the IMS Data Capture function calls RUP with the changed IMS segments.

For user asynchronous propagation, your routine gets the changes from IMS and later calls RUP.

For IMS DPROP asynchronous propagation, the Receiver gets the changes from the Selector-Receiver Interface and later calls RUP. In either case, RUP propagates the changes to DB2.

RIR. RIR is an IMS DPROP abbreviation for DB2 Referential Integrity Relationship. Database administrators can define RIRs between tables in order to request that DB2 catches and prevents update anomalies in the relational databases.

Implementation of RIRs between propagated tables is:

- Optional for one-way IMS to DB2 propagation
• Strongly recommended for DB2 to IMS and two-way propagation

**RTT.** Resource translation table.

**RUP.** Relational Update program.

**RUP control block table.** A single IMS DPROP directory table that contains one RUP propagation control block (PRCB) for each propagated segment type. Each RUP PRCB contains details of the relevant database and segment.

**SCF.** Selector Control File.

**SCF Apply job.** Uses the SCF control statements to create new propagation groups and to list and modify existing propagation groups in the SCF.

**SCF Compare job.** Used to compare the contents of the Group Definitions File with the propagation groups in the SCF and to generate SCF control statements to bring the SCF into line with the Group Definitions File.

**SCF control statements.** Can be generated automatically by the IMS DPROP asynchronous propagation GUU or input directly into the IMS DPROP asynchronous propagation SCF Apply utility JCL. The control statements modify the contents of the SCF records.

**SCU.** Status Change utility.

**segment exit routine.** An IMS DPROP exit routine you can write to complement the logic of the generalized mapping cases. Segment exit routines are typically used to convert a changed data segment from the form it has in your IMS database to a form you have defined in your propagation request.

**SELECT control statements.** Control statements that are input directly into the IMS DPROP asynchronous propagation Selector JCL to define the execution options for the Selector.

**Selector.** An IMS DPROP asynchronous propagation component that collects propagation log records from the IMS log files and writes them to PRDSs for later processing by the IMS DPROP asynchronous propagation Receiver component.

**Selector control file.** Created at Selector installation or generation time and contains the following control information that is essential to the operation of the Selector:

**SCS.** System Log Data Set.

**SNAP.** System network analysis program

**Source System.** An OS/390 system where IMS source databases of the IMS DPROP propagation reside.

**SQL update module.** A module generated by MVG for each propagation request belonging to a generalized mapping case. An SQL update module contains all the SQL statements required to propagate to DB2 the changed IMS data for that propagation request.

**SSM.** Subsystem member. An IMS JCL parameter that identifies the PDS member that describes the connection between IMS and the DB2 subsystems.

**Status Change utility (SCU).** An IMS DPROP utility that:

1. Changes the status of propagation requests in the synchronous environment. Propagation requests can be active, inactive, or suspended. The SCU also performs a variety of other service functions.

2. Maintains the Timestamp Marker Facility and populates the RCT and the PRCT in IMS DPROP asynchronous propagation.

**synchronous propagation.** The propagation of data within the same unit-of-work as the update call.

**Target System.** An OS/390 system where DB2 target tables of the IMS DPROP propagation reside.

**Timestamp Marker Facility.** Supports the statements that create, assign, and delete timestamp markers in the SCF. It is run as part of the SCU.

**TSMF.** Timestamp Marker Facility.

**TSMF Callable Interface.** A facility that allows a user application to create a stop timestamp for one or more propagation groups.

**Two-way propagation.** The combination of IMS-to-DB2 propagation and DB2-to-IMS propagation for the same data.

**TW propagation.** See two-way propagation.
**U**

**UIM.**  User Input Manager.

**ULR.**  Uncommitted Log Record.

**uncommitted log records (ULR).**  When the IMS DPROP asynchronous propagation Selector terminates, it writes all uncommitted log records (propagation log records that have not yet been either committed or aborted by IMS) to the uncommitted log record data set. On a subsequent Selector execution, these records will be either written to the appropriate PRDS (if they have been committed by IMS) or deleted from the uncommitted log record data set (if they have been aborted by IMS).

**UOW.**  Unit of work.

**USER-ASYNC.**  The User asynchronous propagation functions of IMS DPROP.

**user exit.**  See exit routines.

**User Input Manager (UIM).**  A DataRefresher component to which you describe your IMS databases and the mapping between IMS databases and DB2 tables. The mapping is defined by submitting extract requests. You can specify on an extract requests that the UIM is to invoke the DataRefresher Map Capture exit routine provided by IMS DPROP and pass it the DataRefresher mapping definitions of the extract request.

**user mapping case.**  A mapping case you can develop if the generalized mapping cases don't meet your needs.

**V**

**Virtual Lookaside Facility (VLF).**  An MVS/ESA component that is a specific implementation of data spaces. IMS DPROP exploits VLF for a high-performance retrieval of mapping information and other control information.

**VLF.**  Virtual Lookaside Facility.
Index

Numerics
64-byte anchor area
  Field exit routine 114, 124
  saving information across calls 41
  Segment exit routine 26, 41

A
asynchronous propagation
  database maintenance 342
  developing 322
  installing 341
  TSMF callable interface 314
  writing 326

B
Before-Change IMS DB segment buffer 26
binding a DB2 plan for the receiver 341

C
  exit routines in 15
  sample control block 377
  sample Propagation exit control block 409
  sample Segment exit control block 363
  call
    normal 17
    reverse 17
    saving information across 185
  callable interface
    See EMF callable interface
    See TSMF callable interface
  CCU Propagation exit routine 154
  COBOL
    exit routines in 15
    sample control block 369
    sample Propagation exit control block 382
    sample Segment exit control block 353
  conversion routine, DPROP 1

D
Data Capture exit routine
  See also DB2 Data Capture exit routine
  See also IMS Data Capture exit routine
  subexit routine 259
  data conversions 122
  Data Extract
    See DataRefresher
  data propagation
    selective suppression 44
    user asynchronous 322
  data type exit routine 3, 4
  DataRefresher
    and exit routines 9
    propagation exit routine 186
    Segment exit routine 18
  DataRefresher (Data Extract)
    CREATE DATATYPE statement 126
    DEFINITION call 126
    Field exit routine
      calling 111
      PR 125
    Segment exit routine
      informing DPROP 43
      interface control block duration 41
      selective suppression of data propagation 44
      variable definitions 126
  DB2
    Data Capture exit routine, IBM-supplied 7
    Data Capture subexit routine
      64-byte anchor area 260
      calling 260
      data row 264, 267
      data row data 271
      description 10, 259
      environment 259, 272
      HEC 260, 262
      informing DPROP about 273
      performance 272
      processing 271
      QWHS control block 264
      requirements 271
      return codes 272
      sample exit routine 273, 294
      table description 264, 267
      table description data 269
      updating your exit routine 273
      writing 260
    plan, binding for the receiver 341
    propagating data to multiple tables 188
    definitions for Segment exit routine
      first sample 68
      second sample 75
  DLU (DL/I Load utilities)
    Propagation exit routine 154
  DPROP
    buffer
      format 113
      segment 38

© Copyright IBM Corp. 1991,2001
DPROP (continued)
conversion routine 1
Directory, recovering 342
segment buffer 25
trace module 343

DXT
See DataRefresher
Dynamic Allocation exit routine (EKYRESLB) 11

E
EKYRESLB Dynamic Allocation exit routine
description 297
Interface Control Block
details 298
source code 298
JCL requirements 300
processing 300
return codes 301
sample 301
source code 301
telling DPROP about 301
using 11
EKYTED DSECT 346
EKYTRB DSECT 344
EMF callable interface
description 12, 319
parameters 319
return codes 320
sample COBOL program 320
environments
for receiver programs 340
error
dealing
asynchronous data propagation 337
Field exit routine 124
Propagation exit routine 185
Segment exit routine 40
messages
Field exit routine 124
Propagation exit routine 161, 185
Segment exit routine 40
processing, in exit routines 9
exit control blocks
field samples 368
propagation samples 381
segment samples 352
exit routine
third sample 97
exit routines 1
DataRefresher, relationship to 9
DB2
Data Capture exit routine 7
Data Capture subexit routine 10
description 12
EKYRESLB Dynamic Allocation exit routine 11
exit routines (continued)
error processing 9
Field exit first sample 138
Field exit routine 3, 110
Field exit second sample 143
general considerations 11
general description 1
guidelines in developing 122
high-level languages
C 15
COBOL 15
general information 13
PL/I 15
IMS Data Capture exit routine 5
processing 5—9, 300
Propagation exit first sample 188
Propagation exit routine 4, 153
Propagation second sample 234
relationship to DataRefresher 1
restrictions 122
Segment exit routine 3
tracing 41, 185
Extended Program Communication Block
See XPCB
Extended Segment Data Block
See XSDB

F
Field exit routine
64-byte anchor area 114, 124
data conversions 122
DataRefresher, relationship to 9
description 1, 3
DPROP format buffer 113
evironment 111, 122
error
handling techniques 124
messages 124
HUP, returning to 123
interface control block
details 118
source code 114
structure 112
performance 123
processing
requirements 122
sequence 1
requirements 111, 122
return codes 123
RUP, returning to 123
sample control blocks
C 377
COBOL 369
PL/I 373
sample exit routine 127, 143
Field exit routine (continued)
specifying
  DataRefresh 125
  MVG input tables 126
tracing 125
typical uses 110
updating your exit routine 125
user format buffer 113
validating results 123
writing 111

G
generalized mapping
  HUP and exit routine processing 7
  RUP and exit routine processing 5

H
HEC (HUP exit communication block)
  control block 173
  DB2 data capture subexit routine 260
  propagation exit routine 171
high-level languages (HLLs)
  exit routines in
    C 15
    COBOL 15
    general information 13
    PL/I 15
  requirements 14
housekeeping module
  calling during asynchronous propagation 338
  return codes 339
HUP (Hierarchical Update program)
  and exit routine processing 7
calling
  Field exit routine 111
  Propagation exit routine 153, 155
  Segment exit routine 17, 23
DB2 data capture subexit routine
calling 260
duration 271
programmng conventions 271
validation 271
Field exit routine
calling 17
validating results 123
propagation exit routine
duration 182
programmng conventions 182
validation 182
HUP Exit Communication Block
  See HEC

I
IMS
  Data Capture exit routine
    IBM-supplied, description 5
    user-supplied comparison 4
  DB segment buffer 24, 38
  multiple segments, propagating data to 188
  installation, asynchronous propagation 341
interface control block
  DSECT
    field exits 114
    generating 27
duration 41
  EKYRESLB dynamic allocation exit routine
details 298
  source code 298
  field descriptions 33
  Field exit routine
details 118
  source code 114
  structure 112
  generating DSECT 298
  Propagation exit routine 156, 160
  Segment exit routine
description 23
  source code 27

J
JCL requirements
  for a receiver program 340
  for EKYDAEX0 dynamic allocation exit routine 300

K
keys, mapping 154

L
logic
  See processing

M
mapping
  logic
    DPROP to IMS 19, 21
    IMS segments with internal segments 20
    IMS segments with no internal segments 19
    IMS to DPROP 19, 21
    provided by segment exit 19
    user format 3
  MCE, SUBMIT command 141
MQ-ASYNC propagation
  EMF callable interface 319
multiple MVS images 341
MVG (Multiple Verification and Generation)
input tables
  Field exit routine 126
  Propagation exit routine 187
  Segment exit routine 43
verification
  Field exit routine 123, 127
  Segment exit routine 43

N
normal call 17

P
performance
  DB2 data capture subexit routine 272
  Field exit routine 123
  propagation exit routine 183
  Segment exit routine 39
PL/I
  exit routines in 15
  sample control block 373
  sample Propagation exit control block 397
  sample Segment exit control block 358
processing
  DB2 data capture subexit routine 271
  errors in exit routines 9
  Field exit routine 122
  HUP and exit routines 7
  propagation exit routine 182
  RUP and exit routines 5
  Segment exit routine 38
programming conventions
  DB2 data capture subexit routine 271
  propagation exit routines 182
programs
receiver
  description 12
  writing 327
selector
  description 12
  writing 326
sender
  description 12
  writing 326
propagation
exit routines
  CCU 154
  DLU 154
  supported DPROP functions 154
requests
  entered through DataRefresher UIM 43
  entering in MVG input tables 43
  telling DPROP about your propagation exit routine 186

propagation (continued)
  setting up field exit routine 125
  to multiple DB2 tables 188
  to multiple IMS segments 188
  user asynchronous 322
Propagation exit control blocks
  sample source code
    C 409
    COBOL 382
    PL/I 397
Propagation exit routine
  binding the PR 188
  calling the trace module 343
CCU 154
  compared to IMS Data Capture exit routine 4
  data row 175, 178
  description 4, 163
  DL/I calls 162
  DPROP, returning to 184
  environment 153, 183
  error
    handling techniques 185
    messages 161, 185
  general description 1
  HEC 171
  HR-propagation only 188
  HUP 155
  informing DPROP 186
  interface control block
    example 156
    field descriptions 160
  keys, mapping 154
  multiple
    calls 182
    DB2 tables 188
    IMS segments 188
  performance 183
  processing 182
  referential integrity relationships 154
  relationship to DataRefresher 10
  requirements 153, 182
  return codes 161, 184
  returning to DPROP 184
  RUP 155
sample
  EKYEPR1A 189, 230
  first sample 188, 230
  second sample 235, 254
specifying
  DataRefresher 186
  MVG input tables 187
  SQL statements 162, 187
  table description 175, 178
  unavailable resource problem 184
  updating your exit routine 185
  work area 162
Propagation exit routine (continued)
writing 154
XPCB
collection block 165
description 163
field descriptions 166
XSDB
chaining 169
collection block 165
field descriptions 169

Q
QWHC control block 175, 264
QWHS control block 175

R
receiver program
binding a DB2 plan for 341
description 12
JCL requirements 340
RUP, calling 327
writing 327
XPCB 328
XSDB 328
recovery, DPROP directory 342
referential integrity relationship (RIR), mapping 154
Relational Update program
See RUP (Relational Update program)
requirements
  DB2 data capture subexit routine 271
  Field exit routine 111, 122
  Propagation exit routine 153, 182
  Segment exit routine 18, 38
restrictions for receiver programs 340
return codes
  DB2 data capture subexit routine 272
  EKYRESLB dynamic allocation exit routine 301
  Field exit routine 123
  housekeeping module 339
  propagation exit routine 161, 184
  saving information across calls 273
  Segment exit routine 39
reverse call 17
RH-Propagation, interface for 171
RUP (Relational Update program)
  and exit routine processing 5
calling
    Field exit routine 111
    Propagation exit routine 153, 155
    Segment exit routine 17, 23
Field exit routine
calling 17
validating results 123
propagation exit routine
calling 182

RUP (Relational Update program) (continued)
propagation exit routine (continued)
duration 182
programming conventions 182
validation 182

S
sample exit routines
  DB2 data capture subexit routine
details 294
source code 273
  EKYRESLB dynamic allocation exit routine 301
  Field exit routine 127, 143
  first sample
details 230
source code 189
  Propagation exit routine 188, 234
  second sample
details 254
source code 235
  Segment exit routine
  first sample 45
  second sample 75
  third sample 97
sample Field exit
  first sample
details 138
source code 127
  second sample, source code 144
sample Propagation exit routine
  writing in HLL 234
sample Segment exit
  first sample
details 68
source code 45
  second sample
details 88
source code 76
  third sample
source code 97
sample Segment exit control block
  C 363
  COBOL 353
  PL/I 358
saving information across calls
  Field exit routine 124
  Propagation exit routine 185
  Segment exit routine 41
segment buffer, Before-Change IMS DB 26
Segment exit routine
  64-byte anchor area
description 26
  saving information across calls 41
  Before-Change IMS DB segment buffer 26
  call
    normal 17
Segment exit routine (continued)
call (continued)
    reverse 17
calling
    from DataRefresher 42
    from DPROP 42
    with DataRefresher 18
    with HUP 17
    with RUP 17
DataRefresher, relationship to 9
description 1, 3
DPROP
    returning to 39
    segment buffer 25, 38
environment 18, 39
error
    handling techniques 40
    messages 40
IMS DB segment buffer 24, 38
informing DPROP 43
interface control block
description 23
    field descriptions 33
    source code 27
mapping logic
    IMS segments with internal segments 20
    IMS segments with no internal segments 19
    provided by routine 19
performance 39
processing
    guidelines 38
    requirements 38
    sequence 1
requirements 18
return codes 39
returning to DPROP 39
sample
    first sample 45
    second sample 75
    third sample 97
selective suppression of data propagation 40, 44
specifying
    DataRefresher 43
    MVG input tables 43
SQL statements 39
tracing
    Field exit routine 125
    Segment exit routines 41
    your exit routine 185
user format
    buffer 113
    Field exit routine 3
user LOG-ASYNC propagation
    implementing
        general description 322
        receiver program 12
        selector program 12
        sender program 12
user mapping
    HUP and exit routine processing 8
    RUP and exit routine processing 5
USER-ASYNC propagation
    and database maintenance 342
    developing system
        receiver program, writing 327
status change utility (SCU) 341
subexit routine
    See DB2 Data Capture subexit routine
syntax diagrams - how to read xiv
trace element descriptor (TED)
description 346
details 350
    field descriptions 350
    formatted example 346
trace module
    calling 343
    interface 343
    TED
        See trace element descriptor 346
        TRB field descriptions 346
trace request block (TRB)
    DSECT 344
        field descriptions 346
tracing
    Field exit routine 125
    Segment exit routines 41
    your exit routine 185
trademarks 424
TSMF callable interface
description 11, 314
    parameters 314
    return codes 318
    sample COBOL program 317
    sample PL/I program 315
unavailable resources, RUP or HUP handling 184
updating your exit routine
    DB2 data capture subexit routine 273
    Field exit routine 125
    Propagation exit routine 185
    Segment exit routine 41
user format
    buffer 113
    Field exit routine 3
user LOG-ASYNC propagation
    implementing
        general description 322
        receiver program 12
        selector program 12
        sender program 12
user mapping
    HUP and exit routine processing 8
    RUP and exit routine processing 5
USER-ASYNC propagation
    and database maintenance 342
    developing system
        receiver program, writing 327

440 Customization Guide
USER-ASYNC propagation (continued)
  developing system (continued)
  selector program, writing  326
  sender program, writing  326
  setting up  324
  error handling  337
  implementing
    IMS Asynchronous Data Capture Function  322
    user-written IMS Data Capture exit routine  323
  installation considerations
    general description  341
    multiple MVS images  341
    Status Change utility (SCU)  341

W
  work area, propagation exits  162

X
  XPCB (Extended Program Communication Block)
    DSECT  165
    Propagation exit routine  163
    receiver program  328
    XSDB  163, 328
  XSDB (Extended Segment Data Block)
    chained  169
    control block  169
    DSECT  169
    Propagation exit routine  163
    receiver program  328
How to send your comments

IMS DataPropagator for z/OS
Customization Guide
Version 3 Release 1
Publication No. SC27-1214-00

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other Data Management Tools documentation. You can use the following methods to send your comments.

- Send your comments by e-mail to comments@vnet.ibm.com and include the name of the product, the version number of the product, the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title, page number, or a help topic title).
- Complete the readers' comment form at the back of the book and return it by mail, by fax (800-426-7773 for the United States and Canada), or by giving it to an IBM representative.
How satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technically accurate</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Complete</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to find</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Well organized</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Grammatically correct and consistent</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Graphically well designed</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Overall satisfaction</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

Company or Organization

Phone No.