CICS VSAM Transparency for z/OS
Version 2 Release 1

Data Reengineering and Customization Guide

SC34-7250-00

CICS VSAM Transparency for z/OS
Version 2 Release 1

Data Reengineering and Customization Guide

SC34-7250-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 85.

This edition applies toVersion 2 Release 1 of the CICS VSAM Transparency for z/OS, program number 5655-Y01, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Contents

Preface

This book explains how to exploit the capabilities of CICS® VSAM Transparency reengineering to maximise the value of your data in DB2®.

To gain the maximum benefit from this manual, you must be familiar with all aspects of CICS VT, especially manual and automated mapping. You must understand the the built-in data reengineering capabilities provided by the product.

If you intend to write your own CICS VT exits, you should be aware that exits add additional processing overhead.

Who should use this manual?

This manual is aimed at z/OS® application programmers and DB2 database administrators.

Who this book is for

You must have practical experience of CICS VT manual and automated mapping to use this book effectively. The book also assumes a reasonable knowledge of application programming, ideally in COBOL, as well as DB2 programming experience.

Abstract

This book explains various ways that VSAM can be reengineered in DB2 using CICS VT. It is organized into the following chapters:

- Chapter 2, “Mapping a VSAM file to CICS VT,” on page 3 briefly explains the mapping process and highlights the main differences between the manual and automated mapping methods.
- Chapter 3, “Field level data reengineering by mapping,” on page 5 explains how you use the mapping process to exploit the built-in data reengineering capabilities of CICS VT.
- Chapter 4, “Overview of CICS VT exits,” on page 11 discusses how you identify the need for exits, their architecture, and the parameter lists that drive the user exit logic.
- Chapter 5, “Coding FBEs for field level reengineering,” on page 17 provides a number of typical reengineering challenges and looks at actual working exit code.
- Chapter 6, “FBE for managing a control record,” on page 31 details how to code an exit to deal with this common situation in VSAM.
- Chapter 7, “Record level reengineering,” on page 41 examines what you need to do to manage the migration of a single file to multiple DB2 tables
- Chapter 8, “Exit parameter lists,” on page 57 details the parameters that are passed to exits and how they are typically used.
- Chapter 9, “Generic assembler FBEs,” on page 67 describes a number of sample exits available to download.

- Chapter 10, "Built-in conversion routines," on page 73 describes the built-in data conversion capabilities in CICS VT.

Other product publications

The CICS VT documentation includes one other book:

CICS VT User's Guide.

To use CICS VT effectively requires knowledge and experience of DB2 and CICS, and the following IBM books contain information that the CICS VT user will find helpful.

DB2 SQL Reference
DB2 Utility Reference
CICS System Definition Guide
CICS DB2 Guide
CICS Performance Guide

It is assumed that whoever is responsible for writing and testing CICS VT user exits has the appropriate knowledge and experience in the chosen programming language. The samples in this manual are written in COBOL.

Notes on terminology

In this book, the term CICS, used without any qualification, refers to the CICS element of IBM® CICS Transaction Server for z/OS. The term DB2 refers to DB2 UDB for z/OS and OS/390®.

Conventions used in this book

The following conventions are used throughout this book:

Bold text is used for:
Emphasis of key parameters
Italic text is used for:
Variable names
`Monospace` text is used for:
JCL statements
SYSIN control cards
DB2 SQL
Code or syntax examples

Chapter 1. Introduction to CICS VT

Using CICS VT, you can migrate KSDS and RRDS data sets to DB2 without having to rewrite your application programs. You preserve your investment in existing VSAM-based applications, but open up the data to new uses using regular DB2 SQL calls.

A key element of CICS VT is the ability to reengineer data, enhancing its value to your business. Reengineering can be at a field or record level. An example of field level reengineering is converting a numeric date field in a VSAM file to a DATE column in DB2, automatically adding a century where appropriate. An example of record level reengineering is where your VSAM data set contains multiple record types, each with an entirely different record structure. With CICS VT, you can separate each record structure into a different DB2 table.

Field level reengineering can occur automatically as a result of the mapping process. More complex reengineering and data verification require user exit programs that are called by CICS VT at run time. User exits can be written in assembler or any LE/370 language.

The objective of this manual is to help you understand the reengineering facilities that are available in CICS VT, and the circumstances in which you use them. A number of typical reengineering scenarios are included, and sample exits are provided.

A thorough understanding of CICS VT is a prerequisite to reading this manual. You should read it with the *CICS VT User's Guide*.

Why reengineer your data?

Generally, data reengineering achieves a very simple objective: it enhances the value of the data in DB2 and therefore its value to your business. In some ways it is analogous to the data transformation that often occurs when you extract data from your operational databases into a data warehouse.

Prior to migrating a VSAM file, you should consider how you plan to use the data in DB2 in order to determine the required level of reengineering. For simple files, you may be able to achieve your required level of data reengineering using the facilities built into to CICS VT. These are activated as a result of the mapping process. For other files, you may have to write user exits.

You should also consider the effort that you are prepared to invest in the conversion. If you want to convert as quickly as possible to improve online availability, and you are not concerned about the initial DB2 design, the automated migration facility should meet your needs. This facility provides a degree of user-driven field level reengineering, but does not support record-level reengineering. When DB2 design is important, a combination of the manual mapping method and user exits will be required in many cases.

You can potentially migrate a data set more than once. Initially, you might decide to migrate with minimal or no reengineering, then at some stage in the future migrate it again with more complex reengineering. This can often happen with some of the files that you migrate early in your migration project. As you gain

more experience and knowledge of CICS VT, you realize that data sets you migrated previously could provide more business benefit if more reengineering occurs. CICS VT lets you migrate to a new DB2 design at any time.

Reengineering is not just for improving the value of your data. You can use it for data validation, and potentially eliminate data exceptions that you might be experiencing currently with your VSAM application files.

Chapter 2. Mapping a VSAM file to CICS VT

The process of mapping establishes the relationship between the record structure in a VSAM file and the definition of the table in DB2.

CICS VT provides an automated mapping facility and a manual mapping facility. Both use ISPF dialogs and are explained in detail in the *CICS VT User's Guide.*

Automated mapping facility

There are three steps in the automated mapping facility.

Step 1 Is a batch job or ISPF option that analyzes the VSAM cluster for the file to be migrated. Information such as the record length, key length and position, alternate indexes, and number of records are extracted into the CICS VT mapping tables.

Step 2 Uses ISPF dialogs to analyze the copybook for the file and relate it to the VSAM record on a field by field basis. Column names are generated from copybook field names. You can change column names and attributes in the ISPF dialogs.

Step 3 Generates the mapping data for the base cluster and each alternate index, and produces the DDL to create the appropriate DB2 tablespace, table and index objects. The runtime drivers (DIM and DDM) are generated in this step.

There are commands to enable you to suspend and resume mapping, and to save the DDL for processing at a later stage if required.

Manual mapping facility

The manual mapping method has four steps.

Step 1 Is a manual analysis of the copybook to determine the DB2 design to be implemented.

Step 2 Is the manual creation of the DB2 DDL to create the tablespace, table, and primary index.

Step 3 Requires that each copybook field position and attribute is associated with the appropriate DB2 column.

Step 4 Generates the runtime driver modules.

Each alternate index path must be mapped using the alternate index manual mapping utility.

Choosing the appropriate mapping method

You may have to use a combination of the automated and manual mapping methods. For example, assume that you have a VSAM file containing a single record type that you want to convert to a single DB2 table.

Typically, you will use the automated mapping method in this case. During the initial data migration phase, you discover that one or more fields in VSAM file

contain inconsistent data values that are causing data exception conditions. You decide to write an FBE to identify and correct the inconsistent values. You use the manual mapping method to update the mapping and specify the FBE. Any time you make a change to mapping information, you must generate the driver modules for the file to enable the changes to take effect. You use the manual mapping method to generate the driver modules.

The automated mapping facility does not provide the capability to view the mapping. Always use the manual mapping method to view or change mapping.

Use the manual mapping method in all cases where you are migrating a VSAM file to multiple DB2 tables.

Simplifying automated mapping by copybook editing

When you use the automated mapping facility, there are cases where you can reduce or avoid additional manual mapping by creating a version of the file copybook specifically for VT mapping.

To illustrate this, consider the following extract from a COBOL copybook:

```
09 OM-HEADDATE.
   11 OM-HEADDATE-CC PIC XX.
   11 OM-HEADDATE-YY PIC XX.
   11 OM-HEADDATE-MM PIC XX.
   11 OM-HEADDATE-DD PIC XX.
```

The automated mapping facility will generate four 2-byte columns for this field. If you want to map the field to `OM-HEADDATE` a single DATE column, you will have to delete the four mapping entries, insert a single 8-byte field, and map it to the appropriate DB2 column. You can eliminate part of this by updating the copybook to a single 8-byte field.

Chapter 3. Field level data reengineering by mapping

Most field level data reengineering is achieved by mapping. To illustrate this, look at the mapping for the sample VSAM data set provided with CICS VT, called VIDKSDS.

This is shown in Figure 1.

```
 ------------------- CICS VT: List of fields for VIDKSDS     Row 1 to 10 of 10
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CIRDL                                  +
Data set length: 00080    Table name: VID_ITEM                             +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                  Exit     Pic Par
- -------- ----- ----- ---- ------------------------------   -------- --- ---
_ VIDKEY   00006 00001   C  ITEM_NUMBER                    +
_ VIDF001  00012 00007   C  ITEM_NAME                      +
_ VIDF002  00006 00019   C  ITEM_COLOUR                    +
_ VIDF003  00004 00025   C  ITEM_WEIGHT                    +
_ VIDF004  00004 00029   P  ITEM_COST                      +
_ VIDF005  00003 00033   C  ITEM_REORDER_NO                +
_ VIDF006  00003 00036   C  ITEM_SUPP_CODE                 +
_ VIDF007  00005 00039   P  ITEM_DATE_FSHIP                +           Y
_ VIDF008  00002 00044   C  ITEM_SHELF_LIFE                +
_ VIDF009  00035 00046   C  ITEM_DESCRIPTION               +
_
****************************** Bottom of data ********************************
```

Figure 1. CICS VT sample VSAM file mapping

The data in the fields highlighted in Figure 1 is automatically reengineered due to the combination of the field attribute and the DB2 column data type.

- The field called VIDF005 is defined in the copybook as zoned decimal. It maps to column ITEM_REORDER_NO, which is a decimal column in DB2. CICS VT automatically transforms the data.
- The field called VIDF007 is defined in the copybook as packed decimal and contains a date. It maps to column ITEM_DATE_FSHIP, which is a DB2 DATE column. CICS VT automatically transforms the data based on the date field layout in the copybook, which is specified in the mapping.

The combination of copybook field type and the column type determines when data transformation occurs.

How to enable automatic reengineering

CICS VT uses built-in conversion routines to perform automatic reengineering. The specific routine used is based on the combination of the field type and the DB2 column data type.

The full set of built-in routines available in CICS VT is described in Chapter 10, "Built-in conversion routines," on page 73.

Specifying the VSAM field type

The names of the fields shown in Figure 1 (VIDKEY, VIDF001, VIDF002, and so on) are generated by the automated mapping facility. The manual mapping facility generates field names with the prefix VIDM. VT requires that field names are unique within each file and are limited to a maximum of 8 characters.

Each field corresponds to a DB2 column. In manual mapping, you specify the field type that corresponds to the field attribute. For example, a zoned decimal field has a field type of C. The field types supported by CICS T are character (C), signed packed decimal (P), unsigned packed decimal (U), hexadecimal (X), halfword (H), fullword (F), and two fullwords (B).

If a key or alternate index field is a group field that is converted to multiple columns, there are additional mapping considerations. These are described in the *CICS VT User's Guide* .

Mapping fields to columns

You specify the field name, its starting position, and the length and data type of each field that is mapped to a DB2 column.

The file VIDKSDS in Figure 1 on page 5 is mapped to the table shown in Figure 2. If you specify a combination of field type and DB2 column type that is not

```
CREATE TABLE creator.VID_ITEM
  (ITEM_NUMBER        CHARACTER(6)    NOT NULL,
   ITEM_NAME          CHARACTER(12)   NOT NULL,
   ITEM_COLOUR        CHARACTER(6)    NOT NULL,
   ITEM_WEIGHT        CHARACTER(4)    NOT NULL,
   ITEM_COST          DECIMAL(7,2)    NOT NULL,
   ITEM_REORDER_NO    SMALLINT        NOT NULL,
   ITEM_SUPP_CODE     CHARACTER(3)    NOT NULL,
   ITEM_DATE_FSHIP    DATE            NOT NULL,
   ITEM_SHELF_LIFE    CHARACTER(2)    NOT NULL,
   ITEM_DESCRIPTION   CHARACTER(35)   NOT NULL,
   PRIMARY KEY (ITEM_NUMBER))
   IN db.ts ;
```

Figure 2. SQL DDL for VIDKSDS

supported by CICS VT, you receive an error during DIM generation.

Note that each column in Figure 2 has the NOT NULL attribute. You can use nullable columns with CICS VT using either additional mapping parameters or by using a user exit. Sample exits in COBOL and assembler are provided to handle nullable columns.

No data reengineering

Reengineering does not occur if the field type that you specify in the mapping matches the DB2 column type. For example, no reengineering occurs if a field type of C is mapped to a CHAR column, or a field type of P is mapped to a DEC column.

In VIDKSDS shown in Figure 1 on page 5, there is no data reengineering for the fields that are not highlighted.

Adjacent fields in a VSAM file that map to adjacent columns in DB2 are built in a single operation in CICS VT, providing that no data reengineering is required.

Data reengineering

Data reengineering is occurring for the two fields that are mapped to the columns that are highlighted in the DDL in Figure 2 on page 6, as follows:

- The field called VIDF005 is 3 bytes starting at position 33 in the VSAM record. The field type is C, which means the data in the VSAM file is displayable alphanumeric. In the copybook, this field is defined as zoned decimal, and maps to the column ITEM_REORDER_NO. This column is defined in the VID_ITEM table as SMALLINT, which is a numeric data type in DB2. CICS VT automatically translates the zoned decimal field values into the appropriate SMALLINT data value on WRITE/REWRITE calls. For a GET/BROWSE call, CICS VT performs the data translation in reverse.
- The field called VIDF007 is 5 bytes starting at position 39 and has a field type of P. This means the data in this field in the VSAM file is packed decimal. It maps to the DB2 column ITEM_DATE_FSHIP that is defined as DATE in DB2. CICS VT automatically converts the packed-decimal field value to the appropriate date value according to the definition of the picture string. This is CCYYMMDD, as shown in Figure 1 on page 5, and is provided by the user.

In both these cases, data is automatically reengineered by CICS VT as a result of the combination of the VSAM field type and the DB2 column type in the mapping.

The DB2 column data type is not specified anywhere in the mapping process. CICS VT gets this information from the DB2 catalog tables during the driver generation process.

Date and time fields

The mapping in Figure 1 on page 5 shows that the VIDF007 field in VIDKSDS has the picture string CCYYMMDD. You must always specify a picture string when you map a field to a DB2 column with a data type of DATE, TIME, or TIMESTAMP, unless you are using an FBE to perform the reengineering.

DATE columns are the most common. A windowing facility is provided, enabling transformation to occur for VSAM fields that do not include a century value. For example, assume you have a 4-byte packed decimal field containing date information, and you map it to a DATE column. There are two different ways that you can specify the picture string in the mapping, which are as follows:

YYMMDD
: With this picture string, CICS VT adds a century of 19 to every date value in DB2.

Y*n*MMDD
: With this picture string, CICS VT uses date windowing, where *n* represents a decade. If $n = 4$, dates containing a year value less than 40 will have a century of 20 in DB2. If $n \geq 4$, CICS VT uses century = 19.

If your VSAM date field includes the century, such as VIDF0007 in the VIDKSDS sample file mapping in Figure 1 on page 5, date windowing is not required.

DATE picture strings

When you map a field to a DB2 DATE column, you must supply a picture clause.

A typical picture clause for a DATE column is a series of 2-byte constants, where CC represents century, YY represents year, MM represents month, and DD represents day. In the sample VIDKSDS file, the field VIDF007 is mapped to the DATE column ITEM_DATE_FSHIP. The mapping for VIDF007 is shown in Figure 3.

```
 --------------------------- CICS VT: Display field -------------------------
Command ===> ____________________________________________ Scroll ===> CSR

Data set name  . : VIDKSDS
Creator  . . . . : CIRDL                                                  +
Table  . . . . . : VID_ITEM                                               +
Data set length  : 00080

Field name . . . : VIDF007
Field length . . : 00005
Field type . . . : P
Column name  . . : ITEM_DATE_FSHIP                                        +
Starting position: 00039

Picture or FBE . : CCYYMMDD
Parameters . . . :

Special function :
Mapped from table: P
Build order  . . : 00008


Press:  PF3=Exit  PF1=Help
```

Figure 3. Mapping to a DATE column

Other characters can be specified in the picture string, and Table 1 shows a number of examples of the data conversion that is performed with various picture strings. In this case, the source data is in VSAM: ZZ in the picture string can be any characters apart from C, Y, M, or D.

Table 1. Picture strings for converting to DB2 DATE columns

Picture	Source value	Converted value
CCYY/MM/DD	2008/12/31	2008-12-31
CCYY:MM:DD	2008:12:31	2008-12-31
ZZCCMMDD	20081231	1908-12-31
CCZZMMDD	20081231	2000-12-31
CCYYZZDD	20081231	2008-01-31
CCYYMMZZ	20081231	2008-12-01

When data in Table 1 is retrieved from DB2 columns of these types, the effects of the same picture strings result in the field values in Table 2 being returned to your application program:

Table 2. Converting DB2 DATE columns back to VSAM format

Picture	Source value	Converted value
CCYY/MM/DD	2004-12-31	2004/12/31
CCYY:MM:DD	2004-12-31	2004:12:31
CCYYMM01	2004-12-31	20041201
CCZZMMDD	2000-12-31	20ZZ1231
CCYYZZDD	2004-01-31	2004ZZ31

Table 2. Converting DB2 DATE columns back to VSAM format (continued)

Picture	Source value	Converted value
CCYYMMZZ	2004-12-01	200412ZZ

CICS VT always uses the same picture string for converting VSAM to DB2 and DB2 to VSAM. A simple FBE is required to enable different picture strings for VSAM to DB2 and DB2 to VSAM conversions.

Other date formats

CICS VT does not provide direct support for Julian date formats, YYDDD or YYYYDDD. Sample COBOL and assembler exits are available to perform this transformation.

The COBOL version is explained in detail in "Converting between Julian and Gregorian date formats" on page 17.

TIME picture strings

Picture clauses in conversions to a TIME column consist of 2-byte constants shown here.

HH Hours

XX Minutes

SS Seconds

NN Microsecond

String constants in the picture clause, such as a forward slash (/) or colon (:), shown in Table 1 on page 8, can be used for TIME columns. Picture clauses in conversions to a TIMESTAMP column are a combination of the constants for DATE and TIME.

Built-in conversion routines

CICS VT uses a subset of the built-in conversion routines, based on the combination of the field type specified in the mapping and the data type of the DB2 columns.

For example, in the field VIDF005 in Figure 1 on page 5, the field type is "C" and the DB2 column type is SMALLINT. CICSVT uses the CHARSINT routine to convert the data from VSAM format, and SINTCHAR to convert from DB2 format toVSAM format. All the routines can be invoked by a user exit. Examples of calling the built-in routines in COBOL and assembler exits are covered later.

The following table defines the supported conversion combinations that can be achieved through mapping:

Field type	DB2 column type
C	CHAR, VARCHAR, SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, and TIMESTAMP
P and U	SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, and TIMESTAMP
H and F	DECIMAL

In the CICS VT mapping, field type “X” is treated the same as “C”. VSAM fields with type “B”, which is two fullwords, are assumed to contain numeric values that are only valid for the column type BIGINT.

Chapter 4. Overview of CICS VT exits

There are two types of data reengineering exits in CICS VT.

- Field build exits (FBEs) translate data from VSAM form into DB2 form and back again on retrieval. Data in this context can be a single field, a group of fields or an entire record. FBEs are invoked for all VSAM calls. A parameter list (VIDFBEP) is passed to the exit, and the exit logic is based on the parameter values.
- Insert, replace, delete exits (IRDs) operate at a record level and are invoked for a call that updates a file. A parameter list is passed to the exit (VIDIRDP), and the exit logic is based on the parameter values.

You specify exits during the mapping process. FBEs are specified at a field level and IRDs are specified at a file level. Exits are automatically invoked by CICS VT at run time. The same exits are used by CICS VT for batch and online programs.

FBEs and IRDs are invoked during the initial data migration process by the CICS VT conversion utility VIDLOAD.

FBEs and IRDs – differences and similarities

When you map a VSAM file, you define the association between the record in its VSAM and DB2 forms.

In Figure 1 on page 5, each field is mapped to one column in DB2. To process a GET-type VSAM call, CICS VT performs the following steps:

1. The DDM retrieves a row from DB2.
2. CICS VT takes each column value, transforming it when necessary, and moves it to the position in the VSAM record according to the mapping.
3. When every byte of the VSAM record is built, it is returned to the requesting application program.

To process a PUT/WRITE/REWRITE VSAM call, the process is as follows:

1. CICS VT takes each field value according to the mapping, transforming it where necessary, and moves it to the appropriate position in the DB2 record.
2. When every byte of the VSAM record is processed, the DDM issues the appropriate SQL UPDATE or INSERT call.

If you have mapped an FBE on a field, it is responsible for building the field values and performing any data transformation. It must do this for GET-type calls and PUT/WRITE/REWRITE calls. An example of how to map an FBE is shown in “Mapping an FBE” on page 14. You can map multiple FBEs in the same file, and you can use a single FBE to build several fields or columns. When an FBE ends, CICS VT builds the remaining fields in the record.

An IRD operates at record-level, and is only invoked for PUT/WRITE/REWRITE calls. You can only have one IRD for any DIM.

An IRD is always processed after the entire DB2 record is built. If you have FBEs and an IRD on the same file, the FBEs will always have executed before the IRD.

When you map an IRD, you specify the specific point in the VSAM call that the exit is invoked. The options are as follows:

- A *before* IRD is executed before the SQL INSERT/DELETE/UPDATE call is issued by the DDM.
- An *after* IRD is executed after the SQL INSERT/DELETE/ UPDATE call is issued by the DDM.
- A *before and after* IRD is executed before and after the SQL INSERT/DELETE/ UPDATE call issued by the DDM.

A parameter is passed to the IRD to indicate the point at which it is being invoked.

An example of mapping an IRD is shown in "Mapping an IRD" on page 15.

Data migration considerations

FBEs and IRDs are invoked by CICS VT during the initial data migration process. In general, an FBE does not need to consider the type of call being processed. It either builds DB2 data from VSAM data, or the other way around.

In virtually every case, an IRD must include functionality specifically for initial data migration as well as for PUT/WRITE/REWRITE calls. An IRD is sensitive to the type of call being processed.

Establishing when exits are required

CICS VT does not provide any facility to help you identify where exits are required. It is a manual process that needs detailed knowledge of the file, coupled with CICS VT experience. In some cases, application knowledge is essential, especially if data cleansing is required.

Because CICS VT experience is a factor of your own usage of the product, where possible you should start by migrating simple files. Initially, avoid files that need FBEs and IRDs to migrate to multiple tables. You should also try to avoid files that require SQL in an exit.

Copybook analysis

Before you convert a VSAM file, you must establish the target DB2 design. One way to do this is to import your copybook into a spreadsheet. Take each field in turn and specify the DB2 column name and data type.

There are a number of common things that you should look for during this process:

1. Highlight the fields that you want to migrate to DATE, TIME, or TIMESTAMP columns and note how the data is organized with the field. (CCYYMMDD, YYYYDDD for example).
2. Highlight fields that you know may potentially contain inconsistent field values.
3. Highlight columns that you want to be nullable.
4. Identify group fields with OCCURS clauses (arrays in PL/I).
5. Identify redefined fields.

In many circumstances, you may have a common reengineering requirement across a number of different files. For example, you may have a specific format for date fields that require an exit to convert to DB2 DATE format. You can consider

writing a generic exit and using it for multiple fields and files. Sample generic exits are discussed in Chapter 9, “Generic assembler FBEs,” on page 67 .

Data analysis

You should pay particular attention to copybook fields that migrate to DB2 column types that are value-sensitive, such as numeric and date columns. As part of your analysis before you start mapping a file, you might consider writing one-off programs to identify the relative consistency of your VSAM data.

One way to reduce the number of exits you potentially require is to ensure that your data is as consistent as possible. Uninitialized fields can cause data conversion problems, so you should consider coding a one-off program to correct inconsistent field values.

You should also bear in mind that data is sometimes provided within your application programs. For example, consider a file with a packed decimal key that is mapped to a decimal column in DB2. A CICS user optionally specifies a value that your program uses to issue a START BROWSE. If the user leaves the field blank, your program may use a key of low-values. This will result in a data conversion error in CICS VT because low-values is not a valid packed decimal value.

This situation may only emerge during application testing .

Data conversion errors during initial data migration

CICS VT assumes that all VSAM field values are valid.

For example, assume you have a packed decimal date field that you map to a DB2 date column. If the value in the VSAM file is not a valid date value on a WRITE/REWRITE call, DB2 disallows the SQL INSERT/UPDATE call. CICS VT interprets the SQL return code and sets an appropriate VSAM error code which is returned to your application program. The point at which the error occurs varies. It may be during the initial data migration or during initial DB2 load processing.

A common solution to data issues that arise during data migration is to code an FBE to perform data validation and correct invalid data values.

Runtime data conversion errors

If an invalid data value is provided by an application program for a key or alternate index field, no record is retrieved and an appropriate return code or CICS RESP code is set.

For example, assume that you have a file with a packed decimal key field or subfield which is mapped to a decimal column in DB2. If your application program issues a START BROWSE with a key of LOW-VALUES, an error is returned. DB2 returns SQL code -310 for an invalid decimal value and -181 for an invalid DATE or TIMESTAMP value. Invalid zoned decimal key data may result in an S0C7 data exception abend.

This is another scenario where an FBE may be desirable.

Mapping an FBE

You can map an FBE in both manual and automated mapping dialogs. Figure 4 shows how to add an FBE using the manual mapping CICS VT: Update field screen.

```
 --------------------------- CICS VT: Update field ---------------------------
Command ===> ____________________________________________________ Scroll ===> CSR

DIM name       : ITEMFL
Creator        : CICSVT                                                      +
Table          : HLL_ITEM                                                    +
Data set length: 00080

Field name . . . . . . : F07
Field length . . .  ===> 00003             (In bytes)
Field type . . . .  ===> C                 (C,P,U,F,H,B)
Column name  . . .  ===> ITEM_SUPP_CODE..... + (Look-up available)
Starting position   ===> 00036             ("1" = Beginning of data set)
Picture or FBE . .  ===> EXITL=SUPPFBE_____________  (example HH.XX.SS.NNNNNN)
                                                      (or MMDDYY)
                                                      (or EXITx=exit name)
Parameters . . . .  ===> ____________________________  Optional user parameters
Special function .  ===> ___               ("KEY", "PTH", "BKY", or blank)
Mapped from table   ===> P                 ("P"=Prim, "X"=Not mapped)
Build order  . . .  ===> 00007             ("1"=first, "2"=second and so on)


Press: Enter=Update  PF3=Exit  PF1=Help
```

Figure 4. Mapping an FBE

Figure 4 shows that an LE/370 FBE called SUPPFBE has been mapped for the field F07.

In some cases, virtually the entire record area may be built by a combination of an FBE and an IRD. This may be for a file containing multiple record types where the only field that is common to each record type is the key field. The mapping shown in Figure 5 represents a complex file that is managed by the LE/370 FBE MULTFBE and IRD MULTIRD.

```
 ------------------- CICS VT: List of fields for APPLCTL    -- Row 1 to 7 of 7
Command ===> ____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CIRSP                                  +
Data set length: 00263    Table name: TB_APPLCTL                           +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                 Exit     Pic Par
- -------- ----- ----- ---- ------------------------------  -------- --- ---
_ KY       00017 00001   C                                +
_ OBJECT   00006 00001   C  OBJECT_ID                      +
_ RECTYPE  00003 00007   C  REC_TYPE                       +
_ USERID   00008 00010   C  USER_ID                        +
_ FILLER   00233 00018   C                                + MULTFBE
_ ULSTCHG  00008 00251   C  USER_ID_LAST_CHG               +
_ CHGDTE   00005 00259   P  LAST_CHG_DATE                  + DATEFBE
****************************** Bottom of data ********************************
```

Figure 5. Mapping a complex file

The field called FILLER refers to an area in the VSAM record that is redefined with four different record types. The other mapped fields are common to each record.

MULTFBE and MULTIRD are written in COBOL and are explained in Chapter 7, “Record level reengineering,” on page 41.

The FBE DATEFBE is explained in “Reformatting a date field” on page 20.

Note that the field names shown in Figure 5 on page 14 are for illustration purposes. The actual field names used in manual mapping are generated.

Mapping an IRD

You map an IRD by updating the data set information using the manual mapping dialogs only.

This displays the screen in Figure 6.

```
---------------------- CICS VT: Update data set mapping -------------------------
 Command ===> ____________________________________________________________________

 DIM name                          ===> APPLCTL_

 Field build user exit name        ===> ________   If selected, you must build
                                                   all fields for the data set
                                                   with the specified exit

 I/R/D user exit name              ===> MULTIRD_

 I/R/D user exit processing order  ===> A         (B=Before,A=After,' '=Both)

 I/R/D user exit language          ===> L         (A=Assembler,L=LE enabled)

 DIM ready to be generated         ===> Y




 Enter=Update  PF3=Exit
```

Figure 6. Mapping an IRD

The IRD MULTIRD is written in an LE/370 language and is invoked after the DDM has processed the SQL INSERT/ DELETE/UPDATE call. (The field value for I/R/D user exit processing order is A.)

When you map an IRD, the exit processing order is the main mapping option to consider.

What do I code first?

FBEs and IRDs are used in the initial data migration process. There is no stand-alone exit testing facility with CICS VT so the first time your exits are used is during the initial data migration.

For files that you are migrating to a single DB2 table, code any FBEs and add them to the mapping, as shown in Figure 4 on page 14. Generate the CICS VT drivers and then perform the data migration process.

For files that you are migrating to multiple tables, the following approach is recommended:

1. Write the FBEs that operate at an individual field level.

2. Write the IRD to handle the initial load process only.
3. Migrate the data to DB2. Note that there is a special DD statement required for the VIDLOAD utility when a file is mapped to multiple DB2 tables.
4. Write the FBE that processes the additional tables.
5. Using the dual mode facility (DMF), test data retrieval.
6. Add code to the IRD exit to support update calls.
7. Test update calls.

You should use the VIDUNLOD utility to test data retrieval. Sample JCL is shown in Figure 7.

```
//VIDREAD  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//*
//CHECKIT EXEC PGM=VIDUNLOD,REGION=8M
//STEPLIB  DD DSN=appl.EXIT.LOAD,DISP=SHR
//         DD DSN=appl.DRIVERS.LOAD,DISP=SHR
//         DD DSN=VID.SVIDLODE,DISP=SHR
//FILEIN@  DD DISP=SHR,DSN=vsam.DUMMY.APPLCTL
//FILEIN   DD SUBSYS=(vids,db2x,APPLCTL)
//FILEOUT  DD DUMMY,LRECL=271,RECFM=FB
//VIDTRCE  DD SYSOUT=*
//VIDTRCEP DD *
TRACE P01,P03,P04,P05,P06,P08,DIM=APPLCTL
TRACSET CALLS=50
```

Figure 7. Sample JCL for VIDUNLOD CICS

Consider coding a simple program to test update calls for files that use FBE and IRD exits.

Chapter 5. Coding FBEs for field level reengineering

This section looks at a number of simple scenarios where the solution is an FBE. They all operate at a single field level and do not include SQL. They are written in COBOL.

- CJULGRG converts between Julian date and DB2 date formats using intrinsic functions.
- CNULLCL enables a predefined VSAM field value to be stored as a null value in DB2.
- DATEFBE formats a VSAM date field and converts it to DB2 DATE format by calling CICS VT built-in conversion routines.

This is not intended to be an exhaustive list of all possible situations where an FBE is required, but an illustration of actual customer situations and the type of FBE that was used. Some other potential scenarios requiring exits are discussed in "Other potential uses for FBEs" on page 28.

There are additional sample exits in the appendices.

You can also download sample CICS VT COBOL and assembler exits from: This link opens in a new window

Converting between Julian and Gregorian date formats

DB2 supports a combination of industry and IBM standard date formats. A local date format can be supported by a user-written exit routine. It is possible that the format you used when your VSAM files were designed may not match your chosen DB2 format.

A common requirement with CICS VT customers is to convert Julian date fields to DB2 DATE columns.In the context of this manual, Julian and Gregorian date formats are defined as follows:

Julian date
: This format of date is a combination of year plus a relative day number within the year. A typical example is 2008267 in the format YYYYDDD. This is equivalent to a calendar date of August 23rd 2008.

Gregorian date
: This format of date corresponds to any of the industry or IBM standard date formats supported by DB2. For example, the International Organization for Standardization (ISO) format is CCYY-MM-DD. 2008-08-23 is equivalent to the calendar date August 23rd 2008.

There is a sample assembler exit called JULGREG which is available for converting between Julian and Gregorian dates. It supports a VSAM field format of YYDDDHHMM.

COBOL code

```
 CBL LIB
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CJULGRG.
*
```

```
* THIS FBE CONVERTS A 7-BYTE UNSIGNED ZONED DECIMAL FIELD IN
* VSAM TO A DB2 DATE COLUMN. NO DATA VERIFICATION IS PERFORMED.
*
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION                .
 01 WS-DB2-FIELD              PIC 9(8) .
 01 WS-VSAM-FIELD             PIC 9(7) .
*
 01 WS-DB2-TEMP-FIELD                   .
    02 TEMP-YYYY              PIC 9(4) .
    02 TEMP-MM                PIC 9(2) .
    02 TEMP-DD                PIC 9(2) .
 01 WS-DB2-TEMP REDEFINES
    WS-DB2-TEMP-FIELD         PIC 9(8) .
*
 LINKAGE SECTION                        .
 01 VSAM-FIELD                PIC 9(7) .
 01 DB2-FIELD                           .
    02 DB2-YYYY               PIC 9(4) .
    02 FILLER                 PIC X    .
    02 DB2-MM                 PIC 9(2) .
    02 FILLER                 PIC X    .
    02 DB2-DD                 PIC 9(2) .
    COPY VIDFBEC                        .
*
 PROCEDURE DIVISION USING VSAM-FIELD, DB2-FIELD, EXITPARMS .
 MAIN-SECTION.
    SET ADDRESS OF VSAM-FIELD TO EXVSAFLD.
    SET ADDRESS OF DB2-FIELD TO EXDB2FLD.
    EVALUATE EXFUNCT
       WHEN 'D'      PERFORM BUILD-DB2-FIELD
       WHEN 'V'      PERFORM BUILD-VSAM-FIELD
    END-EVALUATE .
 MAIN-SECTION-END.
    GOBACK.
    EXIT.
 BUILD-VSAM-FIELD SECTION.
 10-BUILD-VSAM-FIELD.
     MOVE DB2-YYYY    TO TEMP-YYYY .
     MOVE DB2-MM      TO TEMP-MM .
     MOVE DB2-DD      TO TEMP-DD .
     COMPUTE WS-VSAM-FIELD  =
             FUNCTION INTEGER-OF-DATE(WS-DB2-TEMP) .
     COMPUTE VSAM-FIELD =
             FUNCTION DAY-OF-INTEGER(WS-VSAM-FIELD).
 10-BUILD-VSAM-FIELD-END.
    EXIT.
 BUILD-DB2-FIELD SECTION.
 10-BUILD-DB2-FIELD.
     COMPUTE WS-DB2-FIELD      =
             FUNCTION INTEGER-OF-DAY(VSAM-FIELD).
     COMPUTE WS-DB2-TEMP       =
             FUNCTION DATE-OF-INTEGER(WS-DB2-FIELD).
     MOVE TEMP-YYYY    TO DB2-YYYY .
     MOVE TEMP-MM      TO DB2-MM .
     MOVE TEMP-DD      TO DB2-DD .
 10-BUILD-DB2-FIELD-END.
     EXIT.
```

Notes for CJULGRG

This is a very simple exit that uses COBOL intrinsic functions to handle the data conversion between Julian and Gregorian date formats. No verification of VSAM field values is performed.

Using a nullable column in DB2

The classic definition of a null value is a value that is *not known at this time*. There is no standard concept of a null value in VSAM and from a CICS VT perspective every VSAM field has a value. Using a nullable column in DB2 is a common way to manage VSAM fields with default values that are inconsistent with the field attribute, such as SPACES in a packed decimal field.

Null values are controlled in DB2 using null indicator variables. CICS VT provides mapping support for nullable columns for fields with repeating predefined characters. You should note that DB2 columns that correspond to either the whole or part of the VSAM file key or an alternate index path cannot be nullable.

There may be situations when the mapping support for nullable columns is inadequate. In these cases, the solution is to write an FBE. An example of a COBOL FBE called CNULLCL follows:

COBOL code

```
 CBL LIB RMODE(ANY)
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CNULLCL.
*
* THIS FBE PROCESSES A 2-BYTES PACKED DECIMAL COLUMN THAT IS
* NULLABLE. IF THE VSAM FIELD VALUE IS SPACES, THE EXIT SETS
* THE COLUMN VALUE TO NULL. THE REVERSE IS PERFORMED.
*
* WHEN THE VSAM FIELD VALUE IS SPACES, THE NULL-INDICATOR VARIABLE
* IS SET TO ON AND A VALUE OF '@@' IS MOVED TO THE DB2 COLUMN.
* THE DB2 LOAD CONTROL CARD SPECIFIES NULLIF POS1:POS1 = '@'
* SO THAT THE INITIAL DATA LOAD SETS THE VALUE IN DB2 TO NULL.
*
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION               .
 01 ADDR-OF-NULL               POINTER.
 01 ADDR-OF-NULL2    REDEFINES ADDR-OF-NULL PIC S9(8) COMP-5.
 LINKAGE SECTION                      .
 01 VSAM-FIELD                PIC S9(3) COMP-3.
 01 VSAM-FIELD-NULL  REDEFINES VSAM-FIELD PIC X(2)          .
 01 DB2-FIELD                 PIC S9(3) COMP-3.
 01 DB2-FIELD-NULL   REDEFINES DB2-FIELD PIC X(2)           .
 COPY VIDFBEC                          .
 01 NULL-IND                  PIC S9(5) COMP-5.
*
 PROCEDURE DIVISION USING VSAM-FIELD, DB2-FIELD, EXITPARMS.
 MAIN-SECTION.
     SET ADDRESS OF VSAM-FIELD  TO EXVSAFLD.
     SET ADDRESS OF DB2-FIELD   TO EXDB2FLD.
     SET ADDR-OF-NULL           TO EXNULLS .
     COMPUTE ADDR-OF-NULL2  =
             ADDR-OF-NULL2 + EXNULOFF .
     SET ADDRESS OF NULL-IND    TO ADDR-OF-NULL.
         EVALUATE EXFUNCT
       WHEN 'D'     PERFORM BUILD-DB2-FIELD
       WHEN 'V'     PERFORM BUILD-VSAM-FIELD
    END-EVALUATE .
     MAIN-SECTION-END.
    GOBACK.
    EXIT.
 BUILD-VSAM-FIELD SECTION.
 10-BUILD-VSAM-FIELD.
     IF   NULL-IND NOT   = 0 MOVE SPACES TO VSAM-FIELD-NULL
     ELSE MOVE DB2-FIELD                TO VSAM-FIELD.
```

```
10-BUILD-VSAM-FIELD-END.
    EXIT.
BUILD-DB2-FIELD SECTION.
10-BUILD-DB2-FIELD.
    IF   VSAM-FIELD NOT NUMERIC THEN
         MOVE -1         TO NULL-IND
         MOVE '@@'       TO DB2-FIELD-NULL
    ELSE MOVE VSAM-FIELD TO DB2-FIELD.
10-BUILD-DB2-FIELD-END.
    EXIT.
```

Notes for CNULLCL

CICS VT maintains a pool of null indicators with an entry for every column in the table that is mapped to the VSAM file. The address of the null pool is in the parameter EXNULLS. For each field, the NULLOFF parameter is the offset in the null pool for a specific field. The exit calculates the address of the null pool variable for the DB2 column by adding the null pool offset to the null pool address.

CNULLCL depends on the column name being specified in the mapping of the field. Use the mapping example in Figure 4 on page 14 for field F07.

The statement MOVE '@@' TO DB2-FIELD-NULL in 10-BUILD-DB2-FIELD is specifically to handle initial data migration with the VIDLOAD utility. When the exit detects a non-numeric VSAM field value, it sets the null indicator on and moves @@' to the DB2 column. For a PUT/REWRITE/CALL, the null indicator is on so the column value is disregarded. For the initial data load, you add the following in the input statement for the DB2 LOAD utility:

```
NULLIF (start_pos:end-pos) = '@'
```

The test performed by the exit to establish if the VSAM field value is null is the statement IF VSAM-FIELD NOT NUMERIC. The values SPACES, low-values, and high-values will become a null value in DB2. On retrieval, if the DB2 column is null, the value built in the field is SPACES. Ensure you don't have any program logic that tests this field for a specific field value such as LOW-VALUES.

Reformatting a date field

This exit DATEFBE processes a date field defined as PIC S9(9) COMP-3. There are 6 significant digits in the date, which has a picture string of YYMMDD.

Without the exit, CICS VT would assume that the first byte of the field contains a valid value. For example, a field value of X'000081231C' would be converted incorrectly to a DB2 date value of 2000-08-12.

To build a DB2 value, this exit strips off the first byte, and then calls the CICS VT built-in conversion routine PACKDATE to convert to a DB2 date value. To build a VSAM field value, the exits calls the DATEPACK built-in conversion routine.

The parameter list that the exit passes to the built-in conversion routine includes the picture field Y5MMDD to exploit date windowing. This means that YY values greater than or equal to 50 become 20nn-MM-DD in DB2. Year values less than 50 become 19nn-MM-DD.

COBOL code

```
 CBL LIB
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DATEFBE.
*
* THIS EXIT PROCESSES A PIC S9(9) COMP-3 DATE FIELD THAT HAS
* 6 SIGNIFICANT DIGITS (YYMMDD). VT WILL PROCESS THIS FIELD
* FROM LEFT TO RIGHT, SO A FIELD VALUE OF X'000081231C' WOULD BE
* ERRONEOUSLY CONVERTED TO 2000-08-12 IN DB2.
*
* THE EXIT STRIPS THE FIRST BYTE AND THEN CALLS VIDCONV TO
* CONVERT THE PACKED DECIMAL VALUE TO A DB2 DATA VALUE. IT ALSO
* PERFORMS THE CONVERSION IN REVERSE.
*
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION                .
 01 VIDCONV                   PIC X(8) VALUE 'VIDCONV ' .
 01 DB2-TO-VSAM-PARMLIST                .
    02 DB2-ROUTINE-NO         PIC S9(8) COMP VALUE 49 .
    02 DB2-SOURCE-FIELD       PIC X(10)               .
    02 DB2-SOURCE-FIELD-LEN   PIC S9(8) COMP VALUE 10.
    02 DB2-SPICTURE-FIELD     PIC S9(8) COMP VALUE  0.
    02 DB2-DEST-FIELD         PIC S9(9) COMP-3       .
    02 DB2-DEST-FIELD-LEN     PIC S9(8) COMP VALUE  5.
    02 DB2-DPICTURE-FIELD-LEN PIC S9(8) COMP VALUE  6.
    02 DB2-DPICTURE-FIELD     PIC X(6) VALUE 'Y5MMDD'.
 01 VSAM-TO-DB2-PARMLIST .
    02 VS-ROUTINE-NO          PIC S9(8) COMP VALUE 19.
    02 VS-SOURCE-FIELD        PIC S9(7) COMP-3       .
    02 VS-SOURCE-FIELD-LEN    PIC S9(8) COMP VALUE  4.
    02 VS-SPICTURE-FIELD      PIC X(6) VALUE 'Y5MMDD'.
    02 VS-DEST-FIELD          PIC X(10)              .
    02 VS-DEST-FIELD-LEN      PIC S9(8) COMP VALUE 10.
    02 VS-DPICTURE-FIELD      PIC S9(8) COMP VALUE  0.
 LINKAGE SECTION                       .
 01 VSAM-FIELD                PIC S9(9) COMP-3.
 01 DB2-FIELD                 PIC X(10) .
     COPY VIDFBEC                      .
 PROCEDURE DIVISION USING VSAM-FIELD, DB2-FIELD, EXITPARMS .
 MAIN-SECTION.
    SET ADDRESS OF VSAM-FIELD   TO EXVSAFLD.
    SET ADDRESS OF DB2-FIELD    TO EXDB2FLD.
    EVALUATE EXFUNCT
       WHEN 'D'       PERFORM BUILD-DB2-FIELD
       WHEN 'V'       PERFORM BUILD-VSAM-FIELD
    END-EVALUATE .
 MAIN-SECTION-END.
    GOBACK.
    EXIT.
 BUILD-VSAM-FIELD SECTION.
 10-BUILD-VSAM-FIELD.
    MOVE DB2-FIELD     TO    DB2-SOURCE-FIELD.
    CALL VIDCONV       USING DB2-ROUTINE-NO
                             DB2-SOURCE-FIELD
                             DB2-SOURCE-FIELD-LEN
                             DB2-SPICTURE-FIELD
                             DB2-DEST-FIELD
                             DB2-DEST-FIELD-LEN
                             DB2-DPICTURE-FIELD.
    MOVE DB2-DEST-FIELD TO   VSAM-FIELD.
 10-BUILD-VSAM-FIELD-END.
    EXIT.
 BUILD-DB2-FIELD SECTION.
 10-BUILD-DB2-FIELD.
    MOVE VSAM-FIELD    TO    VS-SOURCE-FIELD.
    CALL VIDCONV       USING VS-ROUTINE-NO
```

```
                              VS-SOURCE-FIELD
                              VS-SOURCE-FIELD-LEN
                              VS-SPICTURE-FIELD
                              VS-DEST-FIELD
                              VS-DEST-FIELD-LEN
                              VS-DPICTURE-FIELD.
  MOVE VS-DEST-FIELD   TO   DB2-FIELD.
10-BUILD-DB2-FIELD-END.
  EXIT.
```

Notes for DATEFBE

The mapping for DATEFBE is shown in Figure 5 on page 14.

The exit illustrates how to call the VIDCONV utility, which is the CICS VT module that includes all of the built-in conversion routines shown in Chapter 10, "Built-in conversion routines," on page 73. This is the standard parameter list to call VIDCONV in COBOL.

Not all parameters are used each time you call VIDCONV. For example, the 4th parameter specifies the picture for the source field. There must be a valid value for this parameter when the source field is VSAM. A value is not required if the source field is DB2.

The first byte of the VSAM field is stripped off by moving the `PIC S9(9) COMP-3` field `VSAM-FIELD` in the linkage section to the working storage field `VS-SOURCE-FIELD`, which is defined as `PIC S9(7) COMP-3`. There are other techniques to do this in COBOL.

VIDCONV should always be invoked dynamically and must not be statically linked. You must include the CICS VT module VIDHIPLI in the linkage edit when an FBE calls VIDCONV.

General notes for FBEs

This section defines a number of factors that you should consider when coding FBEs. Unless stated otherwise, they apply to FBEs in any supported language.

Mixing exit languages

You can code multiple exits on the same DIM in different LE/370 languages.

Coding PL/I exits

To enable PL/I exits to be called by CICS VT, code the procedure statement as follows:

```
PLIFBE: PROC( vsam_field, db2-field, parm_list)
        OPTIONS( FETCHABLE BYVALUE ) ;
```

Exits that include SQL

There are a number of additional factors to consider when you code an exit that includes SQL statements.

Program preparation

As with any program that contains SQL statements, the DB2 precompiler must be used. In the COBOL sample exits included in this manual, the precompiler is invoked by the language compiler. You can use any of your own procedures to compile the exits.

The DB2 package should be bound into the same DB2 collection as your DDM driver module.

Linkage-edit

When your exit includes SQL calls, you must include the CICS VT module VIDHLIPI in your linkage edit step. Sample compile/link/bind JCL for a COBOL exit is shown in Appendix A, "JCL to compile COBOL exit," on page 77.

VIDDMPD DD statement

Your exit must include code to handle unexpected SQL return codes. Based on parameter values you specify, you can cause a call to abend or pass an unsuccessful return code back to your application program. These options are discussed in "Processing errors in an FBE."

Performance

FBEs that contain SQL may add a measurable processing overhead, depending on the number of SQL calls processed. It is essential that the SQL is as efficient as possible. For example, if your exit has to retrieve a single row from a table, it is more efficient to use a singleton select than a cursor declare/open/fetch/close. You must ensure that the SQL access path is efficient and always uses an index. Your SQL should not perform a tablespace scan or any sorting. Additional DB2 indexes are often required to support FBEs that issue SQL calls.

Calling other CICS VT routines

CICS VT establishes an LE/370 enclave for high-level language exits. Internal VT services support calls to routines outside the enclave, such as SQL or calls to the conversion routine VIDCONV.

If a failure occurs in an internal service, a return code of 16 is passed to the user exit.

Processing errors in an FBE

There are two parameters in the CICS VT FBE parameter list that you use to perform error processing.

There are four possible outcomes that you can choose:

1. The exit has completed successfully.
2. The exit has completed successfully and the call should be marked as complete.
3. The exit has not completed successfully and an invalid return code is passed to the application program.
4. The exit has not completed successfully and the call ends abnormally.

You set these conditions in your exit using the parameter EXRET. Each time an FBE is invoked, EXRET is set to SPACE. If you are reporting an SQL error, also set EXSQLCA to the address of the SQLCA in your exit.

Successful completion and continue processing the call

Assume that you have an exit called NUMVALF as shown in Figure 8. Its purpose is to validate that a numeric field value is valid. After your exit completes, CICS VT has to build the remaining fields before the call is complete.

```
 ------------------- CICS VT: List of fields for ITEMFL      Row 1 to 10 of 10
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CICSVT                                  +
Data set length: 00080    Table name: HLL_ITEM                              +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                  Exit     Pic Par
- -------- ----- ----- ---- ------------------------------   -------- --- ---
_ F01      00006 00001   C  ITEM_NUMBER                    +
_ F02      00012 00007   C  ITEM_NAME                      +
_ F03      00006 00019   C  ITEM_COLOUR                    +
_ F04      00004 00025   C  ITEM_WEIGHT                    +
_ F05      00004 00029   P  ITEM_COST                      + NUMVALF
_ F06      00003 00033   C  ITEM_REORDER_NO                +
_ F07      00003 00036   C  ITEM_SUPP_CODE                 +
_ F08      00005 00039   P  ITEM_DATE_FSHIP                +           Y
_ F09      00002 00044   C  ITEM_SHELF_LIFE                +
_ F10      00035 00046   C  ITEM_DESCRIPTION               +
****************************** Bottom of data ********************************
```

Figure 8. Error processing in an FBE

On invocation of an FBE, the parameter field EXRET always is set to SPACE and the field can ignored if the exit is successful. Any value in EXSQLCA is ignored if EXRET is SPACE.

Successful completion and the call is complete

In some cases, your exit needs to override CICS VT.

An illustration of this is the exit sample CTLRECF exit, which is described in Chapter 6, “FBE for managing a control record,” on page 31 . The mapping for this exit is shown in Figure 9 on page 25 .

```
 ------------------- CICS VT: List of fields for ITEMFL      Row 1 to 10 of 10
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CICSVT                                  +
Data set length: 00080    Table name: HLL_ITEM                              +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                 Exit     Pic Par
- -------- ----- ----- ---- ------------------------------  -------- --- ---
_ F01      00006 00001   C  ITEM_NUMBER                   + CTLRECF
_ F02      00012 00007   C  ITEM_NAME                     +
_ F03      00006 00019   C  ITEM_COLOUR                   +
_ F04      00004 00025   C  ITEM_WEIGHT                   +
_ F05      00004 00029   P  ITEM_COST                     + NUMVALF
_ F06      00003 00033   C  ITEM_REORDER_NO               +
_ F07      00003 00036   C  ITEM_SUPP_CODE                +
_ F08      00005 00039   P  ITEM_DATE_FSHIP               +           Y
_ F09      00002 00044   C  ITEM_SHELF_LIFE               +
_ F10      00035 00046   C  ITEM_DESCRIPTION              +
****************************** Bottom of data ********************************
```

Figure 9. Signalling a call is complete

The purpose of CTLRECF is to recognize that a call processes the control record, which exists in a separate DB2 table. The key of the control record is assumed to be the lowest key value in the file, in this case low values.

When CTLRECF processes a call for a regular record, it builds the appropriate field for the key, and then returns control to CICS VT to continue building the remainder of the record. In this case, EXRET is SPACE.

Now assume that there is a call to retrieve the control record. CTLRECF recognizes the key is low values, retrieves the row from the control table and builds the record area to be returned to the application. If CTLRECF ends and EXRET is SPACE, CICS VT assumes that the remaining fields in the file are to be built. This would overwrite the record area built by CTLRECF.

To prevent this, your exit must set EXRET = Y. This signals to CICS VT that the record area is fully built. CICS VT will not build the remaining fields.

When your exit sets EXRET = Y, any value in EXSQLCA is ignored.

Unsuccessful completion and return control to application

Your application may handle certain error conditions returned in VSAM. You can simulate this in an FBE by setting EXRET = P. This will result in return code = 08 being passed to your application program.

If your exit sets EXRET = P as a result of an unacceptable SQL code, it should also set the address of the SQLCA to EXSQLCA to ensure that details of the SQL error are reported. The following COBOL statement will cause CICS VT to produce a formatted SQL error message:

```
SET EXSQLCA TO ADDRESS OF SQLCA.
```

The formatted SQL error message is written to the DD statement VIDDMP in CICS and VIDDMPD in a batch job.

No data is returned to your application program when your exit sets EXRET = P. If CICS VT tracing is active, a return code of 0020 is shown in the trace point 8 record.

CICS VT builds the RPLFDBWD, which is translated by your application program into language specific return and reason codes. EXRET=P will result in return code 08 and reason code DB in your application program.

Unsuccessful completion and abnormally terminate

If your exit sets EXRET = E, a U3107 abend is issued in batch. In CICS, the transaction abend code is 3107.

In both cases, a formatted SQL error message is produced if EXSQLCA is set to the address of the SQLCA in your exit.

User generated error messages

Optionally, your exit can produce messages.

If you use DISPLAY in COBOL exits, the messages are written to SYSOUT in batch and to VIDPIPI in CICS. PUT statements in PL/I exits are written to SYSPRINT in CICS and batch.

Processing DATE columns in CICS VT exits

The CICS VT SQL driver modules are precompiled with DATE and TIME options set to ISO.

This means that DATE and TIME column data is retrieved by CICS VT in the following formats:

	Format	**Example**
DATE	YYYY-MM-DD	2008-12-31
TIME	HH:MM:SS	23:59:39

If you develop any CICS VT exits that process DATE or TIME column data retrieved by an SQL driver, the data will be in ISO format. This can be changed by updating the JCL used by the driver generation job. See member VIDPRE in *my*.SVIDCNFG.*custom*.

Working with variable length data

FBEs and IRDs do not help to manage a VSAM file that contains varying length records. The options for this type of VSAM file are discussed in the *CICS VT User's Guide.*

Terminating an LE/370 enclave

High-level language (HLL) exits run in an LE/370 enclave which CICS VT establishes with CEEPIPI. This places certain restrictions on the semantics of how your exit ends.

In the Language Environment® Programming Guide manual, terminating a program in any of the following ways will result in an abnormal termination:

C exit(), abort(), or signal handling function specifying a normal or abnormal termination

COBOL
STOP RUN statement

PL/I STOP or EXIT

A U313x in module VIDIPIPI abend is the likeliest outcome.

Passing data between exits

CICS VT provides a 16k work area for passing data between exits. The address of the work area is in EXWRKA. In an IRD, the address is in IRDWRKA. Use this as a scratch pad area.

The work area is shared by *all exits in the same DIM*. For example, if your file has 3 FBEs and one IRD, the same 16k area is shared by all 4 exits.

If you need to use the work area, you must ensure that you implement a technique to avoid one exit overwriting an area that is required by another exit. One technique to do this is to have a single copybook which describes the entire area and is used in every exit.

Performance implications of FBEs

In general, FBEs that manipulate bits and bytes in storage do not add any significant overhead.

The overhead in FBEs that include SQL calls is always likely to be more significant. In all cases, it is important to understand how CICS VT builds field and record areas in order to establish the optimum design for your FBEs. This is explained next.

IOAREA building in CICS VT

CICS VT optimizes IO area building by grouping fields together and processing them as a single field.

For example, assume you have a file whose fields map to DB2 columns with the equivalent DB2 column types (signed packed decimal to DEC, PIC 9 and PIC X to CHAR). Additionally, the field positions within your VSAM record correspond to the positions of the columns within your DB2 table. In this case, data transformation is not required and CICS VT builds the appropriate DB2 and VSAM fields in a single operation.

If a field is being transformed by CICS VT, such as a packed decimal field mapped to a DB2 DATE column, or an unsigned packed decimal field to a DEC column, multiple build operations are performed.

Handling multiple fields in an FBE

The sample FBEs process individual fields. An FBE can build multiple fields at the same time.

FBEs are invoked for each field that they map to. If you decide to build more than one field in an FBE, map the FBE on the first field only and leave the other fields it builds unmapped.

FBE for a key or AIX field

Each SQL call that is issued by a CICS VT DDM has a predicate. The predicate columns are the DB2 primary key columns for regular call or the alternate index columns if you are using an alternate index path. Your FBE may have to take this into consideration.

Your FBE uses EXFUNCT to decide if it is building a VSAM field value or a DB2 column value. If your FBE operates on a single field, the exit code is the same for a key and non-key field. An example is DATEFBE which is explained in "Reformatting a date field" on page 20.

More complex FBEs on a key or an AIX® field may have to use the fields EXVSABLD and EXDB2BLD in conjunction with EXFUNCT. An example of a more complex FBE is CTLRECF, which is explained in Chapter 6, "FBE for managing a control record," on page 31.

To illustrate this, consider the case where an application program issues a direct call, such as a START BROWSE, and the key field is managed by an FBE. When the call is processed in CICS VT, the exit is executed at multiple points in the call.

1. A key value is supplied by the application program. For the first invocation of the FBE, the key value must be transformed from its VSAM format into a DB2 format. In this case, EXFUNCT=D signifies that a DB2 field is being built and EXDB2BLD=N signifies that the entire DB2 record is not being built.
2. When the DDM retrieves a record from DB2, the exit is invoked for a second time to transform the DB2 column value to a VSAM format. This enables CICS VT to compare the specified key value with the retrieved key value. EXFUNCT=V signifies a VSAM field is being built and EXVSABLD=N signifies that the entire VSAM record area in not being built.
3. If the converted key matches the key value that is specified in the application call, the remainder of the VSAM record is built. EXFUNCT=V and EXVSABLD=Y in this case.

Your exit must taken account of the values of EXFUNCT, EXVSABLD, and EXDB2BLD to drive the appropriate logic.

Other potential uses for FBEs

FBEs for data transformation or validation are typically used where a VSAM field is mapped to either numeric or DATE/TIME columns in DB2. Common scenarios are discussed next. There are no generic exits available for these scenarios. However, they are all real customer situations that have been resolved by exits.

Scenario 1:

Assume you have a date field that is defined in your copybook as packed decimal and you map it to a DATE column in DB2. A user via a CICS screen inserts a new record, but the value of the date field is optional. Possible error scenarios are as follows:

- The user leaves the field blank and the program provides a default value of blanks.
- The user specifies an invalid date value, such as 31 April 2011.

In the first case, CICS VT gets an S0C7 abend trying to convert an invalid packed decimal value of blanks. In the second case, CICS VT gets a -181 SQL error trying to insert an invalid date into DB2.

In both cases, the error could be avoided by a simple FBE.

Scenario 2:

For this illustration, you have a zoned decimal field that is mapped to a DB2 SMALLINT column. To convert a zoned decimal value to a binary value, the CICS VT CHARSINT routine must initially convert the field value to a decimal number. If the zoned decimal field has a value of blanks, the initial conversion to packed decimal will result in a S0C7 abend. A simple FBE is needed to convert the blanks to a valid zoned decimal value such as 0. (x'F0')

In scenarios 1 and 2, data is reengineered. The conversion routines that are part of CICS VT can be called from within a user FBE. Sample code to do this is provided.

Scenario 3:

A data set in a banking application has a key field defined is as PIC S9(13) COMP-3 in COBOL or FIXED DEC(13) in PL/I. The field contains a 6-digit branch code and a 7-digit account number. To map the key field to a single DEC (13,0) column does not require an exit. To map it to two numeric columns requires an FBE.

Assembler exit coding rules

Exits are assembler and must be written as reentrant and reusable. A 16k work area is provided for and FBE and for IRD exits. Use the exit work area for your program save area. The work area is not changed by CICS VT and the same area is shared by FBE and IRD exits. This allows you to pass it between FBE and IRD exits.

Exits that contain SQL calls should use an EXEC SQL INCLUDE member containing a DSECT defining the SQL host variables. Alternatively, you can use a DSECT that is based on the 16K exit work area EXWRKA (and IRDWRKA).

For IRDs that include SQL, include the module VIDHLI in your linkage edit. This is to enable the exit to be shared between CICS and batch programs.

Register usage in assembler

If you are coding exits in assembler, you should not have any register issues with FBEs and IRDs that do not include SQL statements. A single base register is adequate in most circumstances. More that one base register may be required when exits include SQL.

Be aware that you have no influence over the assembler produced by the DB2 precompiler, and you may discover at assembly time that you have addressability issues. Complex exits may have to be designed as multiple routines and linked together.

You should avoid using register 12 if your exit includes SQL calls or calls to VIDCONV.

Chapter 6. FBE for managing a control record

VSAM files often have a control record that is used to control update programs. The control record structure is entirely different to the structure of the remaining records in the file.

This section defines how this can be solved within CICS VT.

Definition of requirement

A control record is a technique that is often used to control integrity of updates in batch programs. For example, assume that two programs PROGA and PROGB update FILEA. The first thing each program does is to read the control record to check the status of the UPDATE-IN-PROGRESS switch.

When PROGA starts, the UPDATE-IN-PROGRESS switch is off, and therefore the file is eligible for updating. PROGA sets the switch on, updates some records but abnormally terminates. PROGB starts, but the UPDATE-IN-PROGRESS switch is on, so PROGB ends because the file is ineligible for updating.

Because the control record structure is completely different to the structure of the other records in the file, the typical solution in CICS VT is to have the control record in its own table.

Definition of solution

A VSAM file that is migrated to more than one DB2 table normally requires a combination of an IRD and FBE exit. A file containing a control record can be migrated to two DB2 tables with just an FBE.

The FBE accesses the control table and the DDM accesses the main table.

The main record in the file is mapped to DB2, using either mapping method. Fields in these records can be reengineered using CICS VT or user FBEs.

To illustrate the solution, consider a file called ITEMFL, which contains a control record. The mapping is shown in Figure 10 on page 32.

```
 ------------------- CICS VT: List of fields for ITEMFL      Row 1 to 10 of 10
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CICSVT                                +
Data set length: 00080    Table name: HLL_ITEM                            +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                 Exit     Pic Par
- -------- ----- ----- ---- ------------------------------  -------- --- ---
_ F01      00006 00001   C  ITEM_NUMBER                   + CTLRECF
_ F02      00012 00007   C  ITEM_NAME                     +
_ F03      00006 00019   C  ITEM_COLOUR                   +
_ F04      00004 00025   C  ITEM_WEIGHT                   +
_ F05      00004 00029   P  ITEM_COST                     + NUMVALF
_ F06      00003 00033   C  ITEM_REORDER_NO               +
_ F07      00003 00036   C  ITEM_SUPP_CODE                +
_ F08      00005 00039   P  ITEM_DATE_FSHIP               +           Y
_ F09      00002 00044   C  ITEM_SHELF_LIFE               +
_ F10      00035 00046   C  ITEM_DESCRIPTION              +
****************************** Bottom of data ********************************
```

Figure 10. Mapping for a VSAM file with a control record

The FBE to manage the control record is called CTLRECF, as shown in Figure 10. The COBOL copybook for the file is:

```
   01 ITEM-FILE-RECORD                                    .
      02 ITEM-FILE-KEY                                    .
         03 ITEM-FILE-NUM              PIC 9(6)           .
      02 ITEM-FILE-DATA                                   .
         03 ITEM-FILE-NAME             PIC X(12)          .
         03 ITEM-FILE-CLR              PIC X(6)           .
         03 ITEM-FILE-WT               PIC X(4)           .
         03 ITEM-FILE-COST             PIC S9(5)V99 COMP-3.
         03 ITEM-FILE-REORDER-NUM      PIC 9(3)           .
         03 ITEM-FILE-SUPPLIER-CD      PIC X(3)           .
YYYYMMDD 03 ITEM-FILE-FSHIP-DT         PIC S9(9) COMP-3   .
         03 ITEM-FILE-SHELFLIFE        PIC X(2)           .
         03 ITEM-FILE-DESC             PIC X(35)          .
      02 ITEM-FILE-CONTROL-RECORD
         REDEFINES ITEM-FILE-DATA                        .
         03 ITEM-UPDATE-IN-PROGRESS    PIC X              .
            88 COMPLETE             VALUE '0'            .
            88 INCOMPLETE           VALUE '1'            .
        03 ITEM-FILE-UPDATE-PROG       PIC X(8)           .
        03 ITEM-FILE-UPDATE-JOB-NM     PIC X(8)           .
YYYYDDD 03 ITEM-FILE-LAST-UPDATE-DT    PIC S9(7) COMP-3   .
HHMMSS  03 ITEM-FILE-LAST-UPDATE-TM    PIC S9(7) COMP-3   .
        03 ITEM-FILE-RECORDS-DELETED   PIC S9(7) COMP-3   .
        03 ITEM-FILE-RECORDS-INSERTD   PIC S9(7) COMP-3   .
        03 ITEM-FILE-RECORDS-UPDATED   PIC S9(7) COMP-3   .
        03 ITEM-FILE-CONTROL-REMARKS   PIC X(37)          .
```

The DDL for the two tables is:

```
CREATE TABLE CICSVT.HLL_ITEM
  (ITEM_NUMBER        CHARACTER(6)   NOT NULL,
   ITEM_NAME          CHARACTER(12)  NOT NULL,
   ITEM_COLOUR        CHARACTER(6)   NOT NULL,
   ITEM_WEIGHT        CHARACTER(4)   NOT NULL,
   ITEM_COST          DECIMAL(7,2)   NOT NULL,
   ITEM_REORDER_NO    SMALLINT       NOT NULL,
   ITEM_SUPP_CODE     CHARACTER(3)   NOT NULL,
   ITEM_DATE_FSHIP    DATE           NOT NULL,
   ITEM_SHELF_LIFE    CHARACTER(2)   NOT NULL,
   ITEM_DESCRIPTION   CHARACTER(35)  NOT NULL,
   PRIMARY KEY (ITEM_NUMBER))
   IN CVTDB.ITEMFIL1   ;
```

```
CREATE TABLE CICSVT.HLL_ITEM_CONTROL
  (ITEMUP_NUMBER      CHARACTER(6)   NOT NULL,
   ITEMUP_COMPLETE    CHARACTER(3)   NOT NULL,
   ITEMUP_PROGRAM     CHARACTER(8)   NOT NULL,
   ITEMUP_JOBNAME     CHARACTER(8)   NOT NULL,
   ITEMUP_LAST_DATE   DATE           NOT NULL,
   ITEMUP_LAST_TIME   TIME           NOT NULL,
   ITEMUP_REC_DELETES INTEGER        NOT NULL,
   ITEMUP_REC_INSERTS INTEGER        NOT NULL,
   ITEMUP_REC_UPDATES INTEGER        NOT NULL,
   ITEMUP_REMARKS     CHARACTER(37)  NOT NULL,
   PRIMARY KEY (ITEMUP_NUMBER))
   IN VS2DB.ITEMFIL2   ;
```

Field level reengineering

The field `ITEM-UPDATE-IN-PROGRESS` in the control record is a 1-byte character field with a value of `0` or `1`. It maps to a CHAR(3) DB2 column which will have a value of `YES` or `NO`. The transformation is performed by CTLRECF.

The field `ITEM-FILE-LAST-UPDATE-DT` in the control record is a packed decimal date field in Julian format – YYYYDDD. The exit CTLRECF converts it to DB2 DATE format using COBOL intrinsic functions.

The field `ITEM-FILE-LAST-UPDATE-TM` in the control record is a packed decimal time field in the format HHMMSS. CTLRECF converts it to a DB2 TIME column using the appropriate built-in CICS VT conversion routines (VIDCONV).

Limitations of this solution

A number of minor operational limitations are imposed by this solution.

- The control record must be the very first record in the file. The exit assumes the key is LOW-VALUES.
- The control record is never deleted. It is only read or updated.
- The control record must be manually migrated to the control record table. Ideally, DB2 RI should be used to ensure that neither the dummy control record in HLL_ITEM nor the actual control record in HLL_ITEM_CONTROL is deleted.
- A record with a key of LOW-VALUES must exist in the main table (HLL_ITEM). The record is generated by the exit during the initial data migration.

Notes for CTLRECF

The exit must build the key field every time it is called, regardless of whether the call is to process the control record or a record in the HLL_ITEM table. It also has to consider that it may be invoked several times in the same call.

This is described in “FBE for a key or AIX field” on page 27.

Several other points should be noted:

- No data verification is performed in the exit. It assumes that all field values are correct.
- The three fields in the control record table that are reengineered are typical examples of reengineering fields.
- A different technique is used for each reengineered field to demonstrate that there are different ways to reengineer a field.

- Note the use of EXRET. When the exit is processing a request for the control record, EXRET is set to Y to prevent CICS VT from processing the fields that are mapped to the HLL_ITEM table.

There are more efficient ways to code this program but a key objective is to make it easy to understand.

Error processing

The only error handling is associated with the two SQL statements. Any non-zero SQL code will result in the formatted SQLCA being written to VIDDMPD in batch and VIDDMP in CICS. The error messages from the DISPLAY statements will go to the DD SYSOUT in batch and VIDPIPI in CICS. EXRET is set to E if an invalid SQL code is encountered and this causes a U3107 abend.

COBOL code

```
 CBL SQL('HOST(COB2),APOSTSQL,SOURCE,XREF'),LIB,TEST(SYM)
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CTLRECF.
*****************************************************************
* *
* THIS IS A CICS/VT FBE EXIT TO MANAGE VSAM FILE THAT CONTAINS *
* A CONTROL RECORD WITH A 6-BYTE KEY OF LOW-VALUES, WHICH IS *
* STORED IN ITS OWN TABLE. THE FILE IS MAPPED IN CICS VT TO THE*
* TABLE THAT CONTAINS ALL POF THE OTHER FILE RECORDS. *
* *
* IN THE APPLICATION, THE CONTROL RECORD IS ALWAYS ACCESSED BY *
* A DIRECT READ SPECIFYING THE LOW-VALUES KEY. *
* *
* THE EXIT IS MAPPED ON THE KEY FIELD. IF A REGULAR RECORD IS *
* ACCESSED, THE EXIT BUILDS THE KEY VALUE AND ENDS. IF THE *
* CONTROL RECORD IS BEING ACCESSED, THE EXIT RETRIEVES IT FROM *
* THE CONTROL RECORD TABLE. *
* *
* IN THE APPLICATION, THE CONTROL RECORD IS NEVER DELETED. IT *
* IS ONLY RETRIEVED OR UPDATED. *
* *
* THE KEY IS 6-BYTES AND IS CHARACTER DATA IN BOTH VSAM AND DB2*
* *
* NOTE 1: *
* THE EXIT REQUIRES THAT A DUMMY CONTROL RECORD MUST EXIST IN *
* THE MAIN RECORD TABLE. THE EXIT BUILDS A DUMMY RECORD WHEN *
* THE FILE IS INTIALLY MIGRATED. *
* *
* NOTE 2: *
* THE CONTROL RECORD MUST BE MANUALLY INSERTED INTO THE CONTROL*
* RECORD TABLE AT INITIAL DATA MIGRATION. *
* *
*****************************************************************
 ENVIRONMENT DIVISION.
*
 DATA DIVISION.
*
WORKING-STORAGE SECTION .
        01 WS-UPDATE-IN-PROGRESS       PIC X(3)  .
        01 WS-DB2-FIELD                PIC 9(8)  .
        01 WS-VSAM-FIELD               PIC 9(7)  .
        01 WS-DB2-TEMP-FIELD .
           02 TEMP-YYYY                PIC 9(4)  .
          02 TEMP-MM                   PIC 9(2)  .
          02 TEMP-DD                   PIC 9(2)  .
        01 WS-DB2-TEMP REDEFINES
                    WS-DB2-TEMP-FIELD  PIC 9(8)   .
        01 WS-FILE-LAST-UPDATE-DT-TEMP           .
```

```
           02 DB2-YYYY               PIC X(4)   .
           02 DASH1                  PIC X .
           02 DB2-MM                 PIC X(2)   .
           02 DASH2                  PIC X .
           02 DB2-DD                 PIC X(2)   .
        01 WS-FILE-LAST-UPDATE-DT           REDEFINES
              WS-FILE-LAST-UPDATE-DT-TEMP   PIC X(10).
      *****************************************************************
      * VARIABLES USED IN ERROR MESSAGES
      *****************************************************************
        01 WS-DATE.
           02 WS-CENTURY            PIC 99.
           02 WS-YEAR               PIC 99.
           02 WS-MONTH              PIC 99.
           02 WS-DAY                PIC 99.
        01 WS-TIME.
           02 WS-HOUR               PIC 99.
           02 WS-MINUTE             PIC 99.
           02 WS-SECOND             PIC 99.
           02 WS-HUNDREDTH          PIC 99.
        01 ER-DATE.
           02 ER-CENTURY            PIC 99.
           02 ER-YEAR               PIC 99.
           02 FILLER                PIC X    VALUE '/'.
           02 ER-MONTH              PIC 99.
           02 FILLER                PIC X    VALUE '/'.
           02 ER-DAY                PIC 99.
           02 FILLER                PIC X(4) VALUE ' '.
        01 ER-TIME.
           02 ER-HOUR               PIC 99.
           02 FILLER                PIC X    VALUE ':'.
           02 ER-MINUTE             PIC 99.
           02 FILLER                PIC X    VALUE ':'.
           02 ER-SECOND             PIC 99.
           02 FILLER                PIC X    VALUE ':'.
           02 ER-HUNDREDTH          PIC 99.
*****************************************************************
* VIDCONV AND PARAMETER LIST VARIABLES
*****************************************************************
 01 VIDCONV                    PIC X(8) VALUE 'VIDCONV ' .
 01 WS-FILE-LAST-UPDATE-TM     PIC X(10)                 .
 01 DB2-TO-VSAM-PARMLIST                .
    02 DB2-ROUTINE-NO          PIC S9(8)  COMP VALUE 50.
    02 DB2-SOURCE-FIELD        PIC X(8)                 .
    02 DB2-SOURCE-FIELD-LEN    PIC S9(8)  COMP VALUE  8.
    02 DB2-SOURCE-FIELD-PIC    PIC S9(8)  COMP VALUE  0.
    02 DB2-DEST-FIELD          PIC S9(7)  COMP-3       .
    02 DB2-DEST-FIELD-LEN      PIC S9(8)  COMP VALUE  4.
    02 DB2-DEST-FIELD-PIC      PIC X(6)  VALUE 'HHXXSS'.
    02 DB2-PIC-FIELD-LEN       PIC S9(8) COMP VALUE   6.
 01 VSAM-TO-DB2-PARMLIST .
    02 VS-ROUTINE-NO           PIC S9(8) COMP VALUE  20.
    02 VS-SOURCE-FIELD         PIC S9(7) COMP-3        .
    02 VS-SOURCE-FIELD-LEN     PIC S9(8) COMP VALUE   4.
    02 VS-SOURCE-FIELD-PIC     PIC X(6) VALUE 'HHXXSS' .
    02 VS-DEST-FIELD           PIC X(08)               .
    02 VS-DEST-FIELD-LEN       PIC S9(8) COMP VALUE   8.
    02 VS-DEST-FIELD-PIC       PIC S9(8) COMP VALUE   0.
    02 VS-PIC-FIELD-LEN        PIC S9(8) COMP VALUE   6.
*****************************************************************
* DB2 COMUNICATION AREA
*****************************************************************
   EXEC SQL
       INCLUDE SQLCA
   END-EXEC.
*****************************************************************
* DB2 TABLES GENERATED BY DCLGEN
```

```
*****************************************************************
     COPY ITEMFLTC .
 LINKAGE SECTION             .
 01 VSAM-FIELD       PIC X(6) .
 01 DB2-FIELD        PIC X(6) .
 COPY VIDFBEC                .
 COPY ITEMFL .
 COPY ITEMFLTB .
 01 DB2-RECORD-KEY   PIC X(6) .
*
 PROCEDURE DIVISION USING VSAM-FIELD, DB2-FIELD, EXITPARMS.
 MAIN-SECTION.
     SET ADDRESS OF ITEM-FILE-RECORD TO EXVSAIO   .
     SET ADDRESS OF DB2-RECORD-KEY   TO EXDB2IO   .
     SET ADDRESS OF VSAM-FIELD       TO EXVSAFLD  .
     SET ADDRESS OF DCLHLL-ITEM      TO EXDB2IO   .

 EVALUATE EXFUNCT
    WHEN 'V' PERFORM BUILD-VSAM-FIELD
    WHEN 'D' PERFORM BUILD-DB2-FIELD
 END-EVALUATE .
 MAIN-SECTION-END.
     GOBACK.
     EXIT.

 BUILD-VSAM-FIELD SECTION.
 10-BUILD-VSAM-FIELD.
     IF DB2-FIELD NOT = LOW-VALUES    THEN
         MOVE DB2-FIELD TO VSAM-FIELD
         GO TO 10-BUILD-VSAM-FIELD-END
     END-IF.

     IF EXVSABLD NOT = 'Y'            THEN
         MOVE DB2-FIELD TO VSAM-FIELD
         GO TO 10-BUILD-VSAM-FIELD-END
     END-IF.
*****************************************************************
* WE DROP THROUGH HERE IF WE ARE PROCESSING A GET TYPE CALL
* FOR THE CONTROL RECORD (EXFUNCT=V & EXVSABLD = Y).
*****************************************************************
     EXEC SQL
          SELECT
                 ITEMUP_NUMBER
                ,ITEMUP_COMPLETE
                ,ITEMUP_PROGRAM
                ,ITEMUP_JOBNAME
                ,ITEMUP_LAST_DATE
                ,ITEMUP_LAST_TIME
                ,ITEMUP_REC_DELETES
                ,ITEMUP_REC_INSERTS
                ,ITEMUP_REC_UPDATES
                ,ITEMUP_REMARKS
           INTO
                 :ITEMUP-NUMBER
                ,:ITEMUP-COMPLETE
                ,:ITEMUP-PROGRAM
                ,:ITEMUP-JOBNAME
                ,:WS-FILE-LAST-UPDATE-DT
                ,:ITEMUP-LAST-TIME
                ,:ITEMUP-REC-DELETES
                ,:ITEMUP-REC-INSERTS
                ,:ITEMUP-REC-UPDATES
                ,:ITEMUP-REMARKS
              FROM    HLL_ITEM_CONTROL
             WHERE ITEMUP_NUMBER      = :DB2-FIELD
     END-EXEC.
```

```
     IF SQLCODE NOT = 0   THEN PERFORM SQL-ERROR           .

     MOVE ITEMUP-NUMBER         TO ITEM-FILE-KEY           .
     MOVE ITEMUP-PROGRAM        TO ITEM-FILE-UPDATE-PROG   .
     MOVE ITEMUP-JOBNAME        TO ITEM-FILE-UPDATE-JOB-NM .
     MOVE ITEMUP-REC-DELETES    TO ITEM-FILE-RECORDS-DELETED.
     MOVE ITEMUP-REC-INSERTS    TO ITEM-FILE-RECORDS-INSERTD.
     MOVE ITEMUP-REC-UPDATES    TO ITEM-FILE-RECORDS-UPDATED.
     MOVE ITEMUP-REMARKS        TO ITEM-FILE-CONTROL-REMARKS.

     IF ITEMUP-COMPLETE      = 'YES' THEN
          MOVE '0'              TO ITEM-UPDATE-IN-PROGRESS
     ELSE MOVE '1'              TO ITEM-UPDATE-IN-PROGRESS
     END-IF.
*****************************************************************
* CONVERT DB2 DATE FORMAT TO JULIAN DATE FORMAT YYYYDDD
*****************************************************************
     MOVE DB2-YYYY               TO TEMP-YYYY       .
     MOVE DB2-MM                 TO TEMP-MM         .
     MOVE DB2-DD                 TO TEMP-DD         .
     COMPUTE WS-VSAM-FIELD   =
           FUNCTION INTEGER-OF-DATE(WS-DB2-TEMP)
     COMPUTE ITEM-FILE-LAST-UPDATE-DT   =
           FUNCTION DAY-OF-INTEGER(WS-VSAM-FIELD).
*****************************************************************
* CONVERT DB2 TIME FORMAT TO PACKED DECIMAL
*****************************************************************
      MOVE ITEMUP-LAST-TIME     TO DB2-SOURCE-FIELD.
      CALL VIDCONV        USING DB2-ROUTINE-NO
                                DB2-SOURCE-FIELD
                                DB2-SOURCE-FIELD-LEN
                                DB2-SOURCE-FIELD-PIC
                                DB2-DEST-FIELD
                                DB2-DEST-FIELD-LEN
                                DB2-DEST-FIELD-PIC
                                DB2-PIC-FIELD-LEN.
      MOVE DB2-DEST-FIELD       TO  ITEM-FILE-LAST-UPDATE-TM.
      MOVE 'Y'                  TO EXRET .
 10-BUILD-VSAM-FIELD-END.
      EXIT.
*
 BUILD-DB2-FIELD SECTION.
 10-BUILD-DB2-FIELD .

    IF VSAM-FIELD NOT = LOW-VALUES THEN
       MOVE VSAM-FIELD TO DB2-FIELD
       GO TO 10-BUILD-DB2-FIELD-END
    END-IF.

    IF EXDB2BLD     =  'N'  THEN
       MOVE VSAM-FIELD TO DB2-FIELD
       GO TO 10-BUILD-DB2-FIELD-END
    END-IF.
     IF EXCALL        = 'LOAD' GO TO DUMMY-CONTROL-RECORD.
*****************************************************************
* WE CAN ONLY GET HERE IF WE ARE PROCESSING AN UPDATE CALL
* FOR THE CONTROL RECORD (EXFUNCT=D & EXDB2BLD = Y)
*****************************************************************
    IF ITEM-UPDATE-IN-PROGRESS = '0' THEN
         MOVE 'YES'     TO WS-UPDATE-IN-PROGRESS
    ELSE MOVE 'NO '     TO WS-UPDATE-IN-PROGRESS
    END-IF.
*****************************************************************
* CONVERT JULIAN DATE FORMAT YYYYDDD TO DB2 DATE FORMAT
*****************************************************************
     COMPUTE WS-DB2-FIELD     =
```

```
            FUNCTION INTEGER-OF-DAY(ITEM-FILE-LAST-UPDATE-DT)
     COMPUTE WS-DB2-TEMP      =
            FUNCTION DATE-OF-INTEGER(WS-DB2-FIELD).
     MOVE    TEMP-YYYY      TO DB2-YYYY         .
     MOVE    TEMP-MM        TO DB2-MM           .
     MOVE    TEMP-DD        TO DB2-DD           .
     MOVE    '-'            TO DASH1 OF
                            WS-FILE-LAST-UPDATE-DT-TEMP.
     MOVE    '-'            TO DASH2 OF
                            WS-FILE-LAST-UPDATE-DT-TEMP.
*****************************************************************
* CONVERT PACKED DECIMAL TIME VALUE TO DB2 TIME FORMAT
*****************************************************************
    MOVE ITEM-FILE-LAST-UPDATE-TM TO VS-SOURCE-FIELD.
    CALL VIDCONV       USING VS-ROUTINE-NO
                             VS-SOURCE-FIELD
                             VS-SOURCE-FIELD-LEN
                             VS-SOURCE-FIELD-PIC
                             VS-DEST-FIELD
                             VS-DEST-FIELD-LEN
                             VS-DEST-FIELD-PIC
                             VS-PIC-FIELD-LEN .
    MOVE VS-DEST-FIELD TO    WS-FILE-LAST-UPDATE-TM.
    EXEC SQL
         UPDATE    HLL_ITEM_CONTROL
            SET
                 ITEMUP_COMPLETE    = :WS-UPDATE-IN-PROGRESS
                ,ITEMUP_PROGRAM     = :ITEM-FILE-UPDATE-PROG
                ,ITEMUP_JOBNAME     = :ITEM-FILE-UPDATE-JOB-NM
                ,ITEMUP_LAST_DATE   = :WS-FILE-LAST-UPDATE-DT
                ,ITEMUP_LAST_TIME   = :WS-FILE-LAST-UPDATE-TM
                ,ITEMUP_REC_DELETES = :ITEM-FILE-RECORDS-DELETED
                ,ITEMUP_REC_INSERTS = :ITEM-FILE-RECORDS-INSERTD
                ,ITEMUP_REC_UPDATES = :ITEM-FILE-RECORDS-UPDATED
                ,ITEMUP_REMARKS     = :ITEM-FILE-CONTROL-REMARKS
           WHERE ITEMUP_NUMBER      = :VSAM-FIELD
    END-EXEC.
    IF SQLCODE NOT = 0    THEN PERFORM SQL-ERROR
    ELSE                  MOVE 'Y'     TO EXRET
    END-IF .
 10-BUILD-DB2-FIELD-END .
     GOBACK .
     EXIT.
*
 DUMMY-CONTROL-RECORD    SECTION.
 10-DUMMY-CONTROL-RECORD.
*****************************************************************
* THIS SECTION IS ONLY EXECUTED AT INITIAL DATA MIGRATION AND
* BUILDS THE DUMMY CONTROL RECORD WHICH MUST EXIST IN THE MAIN
* DB2 TABLE. (THE ACTUAL CONTROL RECORD MUST BE MANUALLY
* INSERTED INTO THE CONTROL TABLE).
*****************************************************************
     INITIALIZE DCLHLL-ITEM    .
     MOVE LOW-VALUES                   TO DB2-FIELD       .
     MOVE '0001-01-01'                 TO ITEM-DATE-FSHIP .
     MOVE 'Y'                          TO EXRET.
 10-DUMMY-CONTROL-RECORD-END.
     GOBACK               .
    EXIT.
 SQL-ERROR    SECTION.
 99-SQL-ERROR   .
     ACCEPT WS-DATE             FROM DATE YYYYMMDD .
     ACCEPT WS-TIME             FROM TIME          .
     MOVE WS-CENTURY            TO    ER-CENTURY   .
     MOVE WS-YEAR               TO    ER-YEAR      .
     MOVE WS-MONTH              TO    ER-MONTH     .
     MOVE WS-DAY                TO    ER-DAY       .
```

```
    MOVE WS-HOUR              TO    ER-HOUR      .
    MOVE WS-MINUTE            TO    ER-MINUTE    .
    MOVE WS-SECOND            TO    ER-SECOND    .
    MOVE WS-HUNDREDTH         TO    ER-HUNDREDTH .
    DISPLAY '*********************************************'
    DISPLAY 'CICS VT: ' ER-DATE, '           ' ER-TIME
    DISPLAY 'CICS VT: INVALID SQL CODE FOR DIM ' EXDIMNAM
    DISPLAY 'CICS VT: PROCESSING ITEM CONTROL TABLE'
    DISPLAY 'CICS VT: SEE VIDDMPD DD STATEMENT FOR DETAILS'
    DISPLAY '*********************************************'
    MOVE     'E' TO EXRET.
    SET EXSQLCA TO ADDRESS OF SQLCA.
99-SQL-ERROR-END .
    GOBACK               .
    EXIT.
```

Chapter 7. Record level reengineering

You need to write an IRD to migrate a VSAM file to more than one DB2 table.

There are two typical cases where this is required:

- When a VSAM file contains multiple records types that you want to migrate to multiple tables.
- To enable you to create multiple rows in the same table from the same VSAM record. This is a solution to a multi-dimension array, such as a group field with an OCCURS clause.

These two scenarios are the most commonly used in CICS VT. There are sample exits to handle multiple tables for multiple record types. There is also a description of the variations in MULTFBE and MULTIRD that are required to handle other circumstances.

Relationship between an IRD and an FBE

There are three particular aspects to consider for IRDs:

- It is very seldom that an IRD is a complete solution, and you nearly always require an FBE in conjunction with an IRD.
- When you are migrating to multiple DB2 tables, your IRD must include code to handle the initial data migration process.
- One table must be mapped in CICS VT, and this table must contain one DB2 record for every record in the VSAM file. In some cases, a DB2 table containing just key columns may be required.

For a multiple table solution, you write the IRD before the FBE. This reasons for this become clear as you read this section.

There is potential overlap between the functionality in an FBE and an IRD for a multiple table migration but there are two important facts to keep in mind:

- An FBE is invoked for every VSAM call
- An IRD is only invoked for an update-type call

Because an IRD is not involved in a retrieval call, the conversion of data from its DB2 form to the equivalent VSAM form must be handled by the FBE. For update calls, conversion from VSAM form to DB2 form can be done in either the FBE or the IRD.

In general, the FBE includes SQL SELECT statements, sometimes using a cursor. The IRD includes INSERT and UPDATE SQL statements, and in some occasions SELECT statements. DELETE processing is normally achieved by DB2 referential integrity cascading deletes.

DB2 table design

Every VSAM file that you migrate must be mapped in CICS VT to a DB2 table.

When a file is migrated to a single table, CICS VT performs all the appropriate SQL calls. When you map a file to multiple tables, the CICS VT DDM performs the appropriate SQL calls to the table that you map to the file. All SQL calls to the additional tables must be performed in your FBE and IRD.

DB2 primary table

When CICS VT intercepts a retrieval call in your program, the DDM accesses the DB2 table that the file maps to. This table is referred to as the primary table.

Figure 11 shows the mapping for a file called VSAM01, and the primary table is called CICSVT.TB_VSAM01.

```
 ------------------- CICS VT: List of fields for VSAM01      Row 1 to  4 of 4
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CICSVT                               +
Data set length: 00074    Table name: TB_VSAM01                          +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                 Exit     Pic Par
- -------- ----- ----- ---- ------------------------------   -------- --- ---
_ VSAMKEY  00025 00001   C                                 +
_ KEYPART1 00020 00001   C  DB2_RECORD_KEY                 +
_ KEYPART2 00005 00021   C  DB2_RECORD_TYPE                +
_ VSAMDATA 00049 00026   C                                 + MYFBE
****************************** Bottom of data ********************************
```

Figure 11. Primary table mapping

Assume that an application program issues a START BROWSE for VSAM01. The DDM issues an SQL call to retrieve the appropriate row from DB2. If the row doesn't exist, CICS VT returns an appropriate *not found* condition to your application. The FBE called MYFBE won't be invoked in this case.

This means that the primary table must contain one DB2 row for every record in the VSAM file.

In some cases, the primary table may contain just those columns that correspond to the VSAM key field. This is the case when your VSAM file copybook redefines the entire record, excluding the key field. The table CICSVT.TB_VSAM01 in Figure 12 is an example and just contains the columns DB2_RECORD_KEY and DB2_RECORD_TYPE.

DB2 secondary tables

When a START BROWSE for VSAM01 in Figure 11 specifies a key which does exist, CICS VT invokes MYFBE. The tables accessed by your FBE and IRD are referred to as secondary tables. MYFBE uses the DB2_RECORD_TYPE column value to decide which secondary to access. It also uses both key columns from the primary table to access the appropriate row in the secondary table.

This means that the primary key columns in the primary table must exist in each secondary table. The primary table is effectively the parent of the secondary tables. You should enforce this relationship using DB2 referential integrity.

Recommended approach

Exits on a file play a key part in the initial migration of data from VSAM to DB2. The CICS VT migration utility VIDLOAD assumes that each VSAM record becomes a single record in DB2. For a file containing multiple record types, there are normally at least two DB2 records for each VSAM record. One record is for the primary table and one is for the appropriate secondary table.

Assume that a file has a group OCCURS field and you want each occurrence to become a separate row in DB2. There are potentially many DB2 secondary table records for each record in VSAM.

In both situations, CICS VT relies on parameters provided by IRD to create the records for loading to DB2. You must code the IRD before you can perform initial data migration.

Code the IRD first and include functionality to perform any data transformation between VSAM and DB2. This approach means that you don't necessarily need to code an FBE prior to data migration. Testing is simplified because you can separate IRD testing from FBE testing.

Multiple record type solution

A sample IRD and FBE to handle migration to multiple tables is provided.

The file is called APPLCTL and the field mapping is shown in Figure 12. There are four different record types in the copybook, and five DB2 tables are required to implement this in CICS VT. The copybook and DDL for the five tables is shown in Appendix B, "Copybook and DDL for APPLCTL," on page 79.

These exits are described in detail in the sections that follow.

Mapping for APPLCTL

Figure 12 shows the field mapping for APPLCTL. The record length of the file is 263 bytes and it has a 17-byte key. The field called FILLER in the mapping corresponds to a 233-byte redefined area. It is managed by the exit MULTFBE.

```
 ------------------- CICS VT: List of fields for APPLCTL    -- Row 1 to 7 of 7
Command ===> _____________________________________________ Scroll ===> CSR

VSAM file type : KSDS     Creator   : CIRSP                                  +
Data set length: 00263    Table name: TB_APPLCTL                           +

Actions: S Display, U Update, I Insert, D Delete

A Field    Bytes Start Type DB2 column name                  Exit     Pic Par
- -------- ----- ----- ---- ------------------------------   -------- --- ---
_ KY       00017 00001   C                                 +
_ OBJECT   00006 00001   C  OBJECT_ID                       +
_ RECTYPE  00003 00007   C  REC_TYPE                        +
_ USERID   00008 00010   C  USER_ID                         +
_ FILLER   00233 00018   C                                 + MULTFBE
_ ULSTCHG  00008 00251   C  USER_ID_LAST_CHG                +
_ CHGDTE   00005 00259   P  LAST_CHG_DATE                  + DATEFBE
****************************** Bottom of data ********************************
```

Figure 12. Field mapping for sample file APPLCTL

The 17-byte key is split into three fields. The middle field is mapped to the DB2 column REC_TYPE. The value in this field indicates the record type, and therefore the layout of the record.

All four record types have two common fields, and these are mapped to columns USER_ID_LAST_CHG and LAST_CHG_DATE. These columns are in the table CICSVT.TB_APPLCTL. The FBE DATEFBE builds the final field. This exit is detailed in "Reformatting a date field" on page 20.

The table CICSVT.TB_APPLCTL contains only the columns shown in the mapping in Figure 12 on page 43. The FBE MULTFBE builds the bytes 18 to 250 of the VSAM record from one of four DB2 tables, based on the value of REC_TYPE.

Mapping for IRD exit

The IRD for the APPLCTL file is called MULTIRD.

The mapping of the IRD is shown in Figure 13 .

```
 -------------------- CICS VT: Update data set information -------------------
Command ===> ____________________________________________________________

DIM name                          ===> APPLCTL_

Field build user exit name        ===> ________ If selected, you must build all
                                               fields for the data set with
                                               the specified exit

I/R/D user exit name              ===> MULTIRD_

I/R/D user exit processing order  ===> A        (B=Before,A=After,' '=Both)

I/R/D user exit language          ===> L        (A=Assembler,L=LE enabled)

DIM ready to be generated         ===> Y




Enter=Update  PF3=Exit
```

Figure 13. IRD mapping for APPLCTL

MULTIRD is a COBOL *after* exit.

MULTIRD exit

This section looks at the processing performed by MULTIRD.

You can download the entire exit from the following link: This link opens in a new window

WORKING STORAGE SECTION

Because MULTIRD contains SQL, the SQLCA and the DB2 tables accessed by the exit are defined in working storage as follows:

```
EXEC SQL
     INCLUDE SQLCA
END-EXEC.
```

```
COPY APPLCTL1 .
COPY APPLCTL2 .
COPY APPLCTL3 .
COPY APPLCTL4 .
```

The copy members are created by the DB2 DCLGEN utility, which is an option in the DB2I application in ISPF. MULTIRD accesses the four secondary tables. The primary table is accessed by the DDM.

Other working storage variables are used for error messages.

LINKAGE SECTION

The parameters used by MULTIRD are addressed in the linkage section of the program, as follows:

```
LINKAGE SECTION                     .
COPY VIDIRDC                        .
COPY APPLCTL .
*
01 DB2-RECORD-KEY .
   02 DB2-OBJECT-ID         PIC X(6) .
   02 DB2-REC-TYPE          PIC X(3) .
   02 DB2-USER-ID           PIC X(8) .
```

VIDIRDC is the copybook that defines the parameters passed by CICS VT. APPLCTL is the copybook for the APPLCTL file.

Main logic

At the start, the exit establishes addressability to the VSAM record area and the DB2 record area. The only data in the DB2 record area is the key of the record.

The first test performed in MULTIRD is for IRDFUNCT = D. This indicates a DELETE/ERASE call. Because MULTIRD is an *after* exit, the DDM has already performed an SQL DELETE to the primary table. DB2 referential integrity deletes the appropriate record from the secondary table, so MULTIRD has nothing to do and ends at this point.

For all other calls, MULTIRD establishes the record type being processed, as follows:

```
EVALUATE RECORD-TYPE
    WHEN '010' PERFORM BUILD-010-RECORD
    WHEN '020' PERFORM BUILD-020-RECORD
    WHEN '030' PERFORM BUILD-030-RECORD
    WHEN '040' PERFORM BUILD-040-RECORD
    WHEN OTHER PERFORM INVALID-RECORD-TYPE
END-EVALUATE .
```

The processing at each BUILD-nnn-RECORD is the same. Each field is moved from the incoming VSAM record area to the appropriate DB2 table working storage variable.

Test for the call type being processed

Once the appropriate DB2 table working storage record has been built, there is a test for the type of call being processed.

```
EVALUATE IRDFUNCT
     WHEN    'L'      PERFORM LOAD-RECORD
     WHEN    'I'      PERFORM INSERT-0n0-RECORD
     WHEN    'R'      PERFORM UPDATE-0n0-RECORD
END-EVALUATE .
```

For INSERT and UPDATE calls, the appropriate SQL call is issued. Assuming that the SQL code is 0, the exit ends.

Error handling

If an SQL error occurs, an error message is issued.

The message includes the DIM name and name of the DB2 table that was being processed when the SQL error occurred. In this error routine, the following is also performed:

- IRDSQLCA is set to the address of the exit SQLCA so that formatted SQL error messages are issued.
- IRDRET is set to E

This causes a U3108 abend in batch or a 3108 transaction abend in CICS.

Build a load record

The IRDFUNCT parameter is L when the VIDLOAD utility is running.

There are four key IRD parameters that are set for processing this type of call:

IRDB2DAT
: You set this pointer to the address of the DB2 record area that your exit has built.

IRDTYPE
: This is a user-specified 2-byte field that indicates the record type. For MULTIRD, the record types are 10, 20, 30, and 40. See "Loading the DB2 data" on page 47 for an illustration of how this is used.

IRDRPTGR
: This is the number of DB2 records that have been built in the IRDB2DAT area. This parameter is related to an IRD for a file containing a repeating group field where each group item becomes a separate DB2 row. This scenario is discussed in "Handling repeating groups" on page 51. For MULTIRD, the value in this field is 1.

IRDB2DLN
: This is the length of the DB2 record. In MULTIRD, each of the 4 secondary tables is a different length. This field has particular relevance in "Handling repeating groups" on page 51.

No SQL statement can be processed when the IRDFUNCT parameter is L because there is no connection to DB2.

General notes for MULTIRD

MULTIRD builds DB2 column values prior to issuing an SQL call or building a load record. No field level reengineering needs to be performed so the DB2 column values are built by MOVE statements.

When field level reengineering is required in an IRD, there is no field level parameter provided by CICS VT.

Running VIDLOAD

The first time MULTIRD is tested is when you run VIDLOAD. Because multiple output records are written for each VSAM record, you must specify the DD statement LOADOUTM in the VIDLOAD JCL.

Figure 14 shows the JCL for migrating APPLCTL.

```
//jobcard
//*
//VIDLOAD EXEC PGM=VIDLOAD,PARM='APPLCTL',REGION=8M
//STEPLIB  DD DISP=SHR,DSN=appl.DRIVERS.LOAD
//         DD DISP=SHR,DSN=VID.SVIDLODE
//SYSPRINT DD SYSOUT=*
//LOADIN   DD DISP=SHR,DSN=your.VSAM.UNLOAD
//LOADOUTM DD DSN=your.APPLCTL.LOADDB2,
//         DISP=(,CATLG,DELETE),
//         UNIT=SYSDA,SPACE=(CYL,(150,35),RLSE),
//         RECFM=FB,LRECL=242,
```

Figure 14. VIDLOAD JCL for multiple tables

The record length of the output file is 242. This is the length of the longest DB2 row plus 2 bytes. This is explained next.

Loading the DB2 data

The output file *your*.APPLCTL.LOADDB2 from the VIDLOAD utility is loaded into DB2. Because APPLCTL is converted to multiple tables, the records in this file are for multiple tables in DB2.

MULTIRD specifies a value for IRDTYPE when IRDFUNCT is L. When you use the VIDLOAD utility with the LOADOUTM DD, each record has a 2-byte prefix. The prefix for records for the primary table is always 00. The prefix for the secondary tables is whatever you set in IRDTYPE. Here is an extract of the load file for APPLCTL:

```
----+----1----+----2----+----3----+----4----+----5----+----6
00AD2021010THOMSONJOSBORNEI1998-04-25.....................
10AD2021010THOMSONJAPYACCTS PAYABLE BLENKINSOPFIONA 2
00FL3055020PALLINMJHALLAJ 2000-10-03.......................
20FL3055020PALLINMJAPYCHEQUES ACCCHQ ACCCPAT1
00FL3055020THOMSONJHALLAJ 2000-10-03.......................
20FL3055020THOMSONJAPYRECEIPTS ACCRCPT ACCCPAT1
00JB8383040MCGRAWA MCGURND 2005-11-15.......................
40JB8383040MCGRAWA APYPAPD0205MONTHLY REPORTS YYYNNYMACCOUNT
00PG5267030STEWARTHRITCHIEA2004-06-28.......................
30PG5267030STEWARTHAPYA407DLG LEDGER DAILY RECONCILE 00005
```

You must add a statement to your DB2 load utility control cards to select the appropriate records. For example, to load the primary table use the control card:

```
WHEN (1:2) = '00'
```

Because of DB2 referential integrity, you must load the primary table first.

Loading large tables

The DB2 load utility reads every record in the input sequential data set to satisfy the WHEN statement. If you are migrating a very large VSAM file, it is often more efficient to split the VIDLOAD output file into a separate data set for each table.

You can achieve this with DFSORT , like the example in Figure 15 .

```
//jobcard
//*
//COPY1    EXEC PGM=SORT
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SORTIN   DD DISP=SHR,DSN=your.APPLCTL.LOADDB2
//TYPE00   DD DSN=your.TYPE00.FILE,DISP=(,CATLG,DELETE),
//            SPACE=(CYL,(350,100)),UNIT=SYSDA,AVGREC=U,
//            DCB=your.APPLCTL.LOADDB2
//TYPE10   DD DSN=your.TYPE10.FILE,DISP=(,CATLG,DELETE),
//            SPACE=(CYL,(150,75)),UNIT=SYSDA,AVGREC=U,
//            DCB=your.APPLCTL.LOADDB2
//TYPE20   DD DSN=your.TYPE20.FILE,DISP=(,CATLG,DELETE),
//            SPACE=(CYL,(750,200)),UNIT=SYSDA,AVGREC=U,
//            DCB=your.APPLCTL.LOADDB2
//TYPE30   DD DSN=your.TYPE30.FILE,DISP=(,CATLG,DELETE),
//            SPACE=(CYL,(1550,300)),UNIT=SYSDA,AVGREC=U
//            DCB=your.APPLCTL.LOADDB2
//TYPE30   DD DSN=your.TYPE40.FILE,DISP=(,CATLG,DELETE),
//            SPACE=(CYL,(450,100)),UNIT=SYSDA,AVGREC=U,
//            DCB=your.APPLCTL.LOADDB2
//SYSIN    DD *
 OPTION COPY
 OUTFIL FNAMES=TYPE00,INCLUDE=(1,2,CH,EQ,C'00')
 OUTFIL FNAMES=TYPE10,INCLUDE=(1,2,CH,EQ,C'10')
 OUTFIL FNAMES=TYPE30,INCLUDE=(1,2,CH,EQ,C'30')
 OUTFIL FNAMES=TYPE40,INCLUDE=(1,2,CH,EQ,C'40')
```

Figure 15. JCL to create multiple load files

MULTFBE exit

The FBE to handle APPLCTL is described in this section.

You can download the entire exit from the following link: This link opens in a new window

WORKING STORAGE SECTION

Because MULTFBE contains SQL, the SQLCA and the DB2 tables accessed by the exit are defined in working storage. The same copy members used in MULTIRD are used in MULTFBE.

```
EXEC SQL
    INCLUDE SQLCA
END-EXEC.

COPY APPLCTL1 .
COPY APPLCTL2 .
COPY APPLCTL3 .
COPY APPLCTL4 .
```

The copy members are created by the DB2 DCLGEN utility.

LINKAGE SECTION

The parameter list that CICS VT passes to an FBE is different to an IRD.

For MULTFBE, the linkage section is:

```
01 VSAM-FIELD       PIC X(233).
01 DB2-FIELD        PIC X(1)  .
COPY VIDFBEC                  .
```

```
COPY APPLCTL                  .
01 DB2-RECORD-KEY             .
02 DB2-OBJECT-ID     PIC X(6) .
02 DB2-REC-TYPE      PIC X(3) .
02 DB2-USER-ID       PIC X(8) .
```

In the mapping shown in Figure 12 on page 43, MULTFBE is mapped against the field called FILLER, which is 233 bytes. This field is not mapped to a DB2 column. The second parameter which CICS VT passes to an FBE is the address of the DB2 column. Because no column is mapped to FILLER, no address is passed and the linkage section field DB2-FIELD is a placeholder.

Main Logic

At the start, the exit establishes addressability to the VSAM field and the DB2 record.

The first test performed in MULTFBE is for EXFUNCT. If EXFUNCT = D, the call being processed is either a WRITE/REWRITE or the VIDLOAD is executing. The IRD MULTIRD is responsible for building the DB2 row. If EXFUNCT is not V, MULTFBE ends.

The next test in MULTFBE is to establish the record type. The field RECORD-TYPE is part of the key that is passed to the exit in the linkage section.

```
EVALUATE RECORD-TYPE
    WHEN '010'    PERFORM PROCESS-010-RECORD
    WHEN '020'    PERFORM PROCESS-020-RECORD
    WHEN '030'    PERFORM PROCESS-030-RECORD
    WHEN '040'    PERFORM PROCESS-040-RECORD
END-EVALUATE .
```

The processing at each PROCESS-0n0-RECORD is the same. MULTFBE issues an SQL SELECT to the appropriate table, using a WHERE clause specifying the values in the key of the VSAM record.

Each field is moved from the DB2 host variables in the select statement to the appropriate field in the VSAM record, using the appropriate VSAM copybook definition. The exit ends when the VSAM record is built.

No data reengineering is required.

Error handling

Error handling in MULTFBE is equivalent to error handling in MULTIRD, except that different parameter names are used.

The significant fields for MULTFBE are as follows:

- EXSQLCA is set to the address of the exit SQLCA so that formatted SQL error messages are issued.
- EXRET is set to E.

EXRET = E causes a U3107 abend in batch or a 3107 transaction abend in CICS.

General notes for MULTFBE

MULTFBE is a simple exit, and most of the coding is for the SQL SELECT statements and the subsequent MOVE statements. There are separate MOVE

statements for each discrete field, and this is largely for illustration purposes. Because there is no data transformation, these moves could be combined into a single block move.

Testing exits like MULTIRD and MULTFBE

There is no CICS VT stand-alone test facility for an IRD or FBE.

The initial testing occurs during data migration. The migration steps are as follows:

Unload the VSAM file

Use the standard VIDUNLOD utility to unload your VSAM file to a sequential data set. The exits are not executed during this process.

Run the VIDLOAD utility

The VIDLOAD utility creates the file for loading to DB2. The IRD is invoked to build the records to be loaded to the secondary tables. If you have field level FBEs, these will be tested as well.

You can use CICS VT tracing to help to debug field level FBEs, but the trace will not show any of the data built by an IRD like MULTIRD. You can browse the LOADOUTM data set to review the effects of the IRD.

If you save the VIDLOAD input file, you can rerun it multiple times to help with debugging. You can also use COBOL DISPLAYs or PL/I PUTs.

Run the DB2 load utility

This is the biggest test for an IRD like MULTIRD. If the exit is not building the data correctly, packed decimal and date fields may be in error. The DB2 load utility abends in this case.

Below are the typical error messages you will see if you have invalid packed decimal or date data:

DSNU334I with error code = 04 indicates an invalid packed decimal value

DSNU334I with error code = 14 indicates an invalid date or time value

Run the VIDREAD utility with DMF

When the data is loaded to DB2, run the VIDREAD utility with dual mode facility (DMF) active. This tests exits like MULTFBE. It also tests any field level FBEs.

You can use CICS VT tracing to help with debugging.

If you are dealing with large data volumes, limit the volume of tracing with the TRACSET trace parameter.

Verify DB2 access paths

When you have successfully migrated the data to DB2, and successfully read it with VIDREAD, check that the SQL in your exits is efficient. REBIND the DB2 packages for exits that contain SQL and specify EXPLAIN(YES).

Review the output that is generated to the PLAN_TABLE. A tablespace scan or any form of sorting indicates an issue with either your SQL or a DB2 table or index definition. You should also check that the attributes of host variable fields are consistent with the associated DB2 column data type.

Handling repeating groups

A repeating field or structure is anathema in a relational database but commonplace in VSAM files.

A number of different approaches can be considered for implementation in CICS VT:

- If there is a single repeating field, each element can be a separate column with a numeric suffix. If this DB2 design is acceptable, no exits are required and the file can be mapped very quickly using the automated mapping facility. Be aware that the maximum number of columns in a single table is limited by DB2 to 750.
- The entire array structure can be mapped as either a single DB2 column, a series of columns if the structure size is more than 255 bytes, or a single VARCHAR column.
- For a repeating group field, each instance can be a separate row in an additional table. This is the only normalized design.

There are obvious limitations with the first two approaches although their advantages is that the data is migrated to a single table and the migration effort is minimized. The runtime performance is also minimized because data is processed in a single I/O operation.

Implementing a normalized DB2 design

Assume that you decide to implement a normalized DB2 design with each group entry becoming a separate DB2 row. To illustrate this consider the following copybook extract:

```
05 RETL-PRICE-GROUP OCCURS 10 TIMES
 10 RETL-PRICE-HIGH-SIZE-NUM PIC S999 COMP-3
 10 RETL-PRICE-LOW-SIZE-NUM  PIC S999 COMP-3
 10 STK-RETL-SLS-PRICE       PIC S9(5)V99 COMP-3
 10 SKU-PRICE-CHNG-PCT       PIC S9V9999 COMP-3
 10 SKU-PRICE-CHNG-AMT       PIC S999V99 COMP-3.
```

Figure 16. Repeating group field

A normalized design uses a secondary table containing 10 DB2 rows for each VSAM record. The key of the secondary table is the same as the key of the primary table, but with an additional column to achieve uniqueness. This can be a timestamp column, or a numeric column which indicates the relative record position within the group. The additional column value must be generated by the IRD.

The secondary table is related to the primary tables through DB2 referential integrity.

There are no sample CICS VT high-level language exits for this solution. The following section highlights the variations in MULTIRD and MULTFBE that are required.

MULTIRD variations

There are no OCCURS clauses that have to be handle by the MULTIRD IRD exit. Each update call accesses a single table. When an IRD exit has to handle an OCCURS clause, insert processing, update processing, and initial load processing are all different. These are discussed in this section.

Insert processing

If there are a fixed number of occurrences in each group, insert processing is straightforward. Multiple rows are inserted by the IRD.

In some cases, an OCCURS clause may indicate the maximum number of items in the group. Unused array items often have predefined values based on field attributes, such as zero or low values. In this case, the exit has to determine how many array items contain valid data to be inserted to DB2.

Update processing

A key factor affecting the complexity of the IRD is the update process. For example, assume that RETL-PRICE-GROUP in the copybook extract in Figure 16 on page 51 holds information for the last 10 variations in the price of an item, stored chronologically. The first array element always contains latest price information. When the price of an item changes, the oldest price information drops off, and the nine remaining array elements are moved within the array. This is illustrated in the VSAM record show in Figure 17, where 0 represents the latest price information and -9 the oldest.

```
+------+------+------+------+------+------+------+------+------+------+
|  0   |  -1  |  -2  |  -3  |  -4  |  -5  |  -6  |  -7  |  -8  |  -9  |
+------+------+------+------+------+------+------+------+------+------+
```

Figure 17. OCCURS field in VSAM record

The application program issues a PUT/REWRITE call to add new price information. The latest price information is always written to the first element, so when this update takes place the remaining elements shift to the next element along.

When you implement this as multiple rows in a DB2 table, you must ensure that the same is achieved in your CICS VT exits. You could add a TIMESTAMP column to the DB2 table, and use it as part of the ORDER BY cursor statement in the FBE. If the column was defined as NOT NULL WITH DEFAULT, DB2 generates a value corresponding to the date and time the insert takes place. This would also ensure uniqueness of the primary key.

If a TIMESTAMP column is not desirable, the column value has to be generated by the IRD.

Testing for an update

The effects of an update call for a normalized design vary. If the OCCURS clause defines the maximum number of occurrences, an update call may mean one or more SQL INSERTS. It can also mean one or more SQL UPDATES. SQL is the most expensive part of an exit and minimizing SQL activity minimizes the overhead.

It is essential that your IRD considers the implications of an update call on the OCCURS table.

There are several different ways to approach this:

1. You can assume that the group OCCURS fields changes on every update call. In this case, you may decide to delete all the rows in DB2 and insert new rows.
2. You can compare the updated data in the group field against the data in DB2. If there is any difference, issue SQL delete calls and then insert new rows.
3. You can compare each group field value and issue SQL update calls for those which have changed.

The greater the OCCURS number, the greater the SQL overhead to process the OCCURS table. One way to eliminate unnecessary SQL calls is described in "Exit work area" on page 54.

Initial data migration

The significant IRD parameters for initial data migration are described in "Build a load record" on page 46. For a repeating group IRD, pay particular attention to IRDRPTGR and IRDB2DLN.

To illustrate, consider the example of a group field with OCCURS 10. When IRDFUNCT = L, the IRD must build all 10 records in a single contiguous area in working storage or in the exit work area. Set the address of this area to IRDB2DAT. You specify the number of DB2 records that you have built in this area in the parameter IRDRPTGR. In this case, the value is 10.

The output data set created by the VIDLOAD utility contains records for the primary DB2 table as well as the repeating group table. You must specify a value in IRDTYPE to identify the repeating group table records. The area addressed by IRDB2DAT looks like Figure 18.

```
+-+--------+---------------+-+--------+---------------+-+--------+------\
|a|key+TS  | data          |a| key+TS | data          |a| key+TS | data  \_
+-+--------+---------------+-+--------+---------------+-+--------+---------\
```

Figure 18. Repeating group record built by IRD

In Figure 18, the boxes containing "a" correspond to the 2-byte IRDTYPE parameter. The box containing "key+TS" refers to the concatenation of the DB2 primary key plus a TIMESTAMP column to make each row in the repeating group table unique. The box containing "data" refers to each occurrence of the group field.

Calculate the length of the entire area - a+key+TS+data - and store it in IRDB2DLN.

MULTFBE variations

MULTFBE retrieves a single row from the appropriate table. When a group OCCURS field is migrated to multiple rows, the FBE uses a cursor to retrieve all of the DB2 rows to build the group OCCURS fields. An SQL code +100 is returned when all rows have been fetched.

If your IRD is checking for updates to the group field, your FBE must store the group field data when a get-for-update call is processed. EXCALL = GETU in this case. This technique is described in "Exit work area" on page 54.

Initial data migration to multiple tables

You use the VIDLOAD utility to convert the VSAM records into the appropriate format for loading into DB2. When you are migrating to multiple DB2 tables using a combination of an FBE and an IRD, you must specify the output DD statement LOADOUTM. An example of the JCL is shown in Figure 14 on page 47.

Loading large tables

When a VSAM file is migrated to multiple DB2 tables, you use the LOADOUTM DD statement in the VIDLOAD utility. The output sequential data set contains records for all the tables associated with the file, and you must use the WHEN statement in the DB2 load utility control cards.

The DB2 load utility reads every record in the input sequential data set to satisfy the WHEN statement. If you are migrating a very large VSAM file, it is often more efficient to split the VIDLOAD output file into a separate data set for each table. You can achieve this with DFSORT, like the example in Figure 15 on page 48.

Exit work area

The exit work area is explained in "Passing data between exits" on page 27. You can use this area to optimize the SQL that your IRD needs to perform for an update call.

Assume that you have a file containing a 10-element array, and each array occurrence corresponds to a separate row in DB2. When you use a secondary table for handling repeating groups, a REWRITE call may update none, some, or all of the array data. This means that the IRD exit has to perform a combination of SQL INSERT, UPDATE, and DELETE calls.

To avoid potentially unnecessary SQL calls, add code to the FBE to store the array in the exit work area on a get-for-update (GETU) call. On an update call, compare the data and only issue SQL calls if the data has changed.

This technique is valid only if your programs update the last record read. If there is a possibility that a program reads several records and then performs an update against any previously read record, this technique is not valid.

Other IRD considerations

FBEs may be generic and reused in many fields across many VSAM files. IRDs tend to be very specific to a particular VSAM file.

The two different uses of IRDs illustrated earlier in this section are the two most common uses. There are features of IRD capability that are not used in these two situations.

IRDTYPE parameter

This 2-byte field is used to identify secondary table records created by the VIDLOAD utility. The sample MULTIRD explains one way to use it.

There are two other potential situations you should be aware of and these are discussed next.

Suppressing IRDTYPE

In some cases, secondary table records may contain a field that uniquely identifies each different record type.

For example, the extract of the file produced by the VIDLOAD utility shown in "Loading the DB2 data" on page 47 shows record types of 10, 20, 30, and 40 in position 1. MULTIRD uses the IRDTYPE parameter to set these record types. The field RECORD-TYPE is at position 9 in these records, and it contains values of 010, 020, 030, and 040.

There is a pair of records for each record type. The first instance corresponds to the record to be loaded to the primary table. The second instance corresponds to the record to be loaded to the secondary table. Without the IRDTYPE parameter in bytes 1 and 2, you are unable to code a WHEN (x:y) = '*record_type*' parameter in the DB2 load utility to distinguish between records for the primary and secondary tables.

When your data already has a field which uniquely identifies it, you can suppress the 2-byte record identifier by specifying IRDTYPE = ??.

Generating records for multiple secondary table records

In MULTIRD, each VSAM record was split between the primary table and one of four secondary tables. You may want to split a single VSAM record into several different DB2 tables.

To illustrate this, assume that you want to split each VSAM record into one primary table and three different secondary tables. CICS VT handles the data in the primary table. Your IRD is responsible for processing the data in the three secondary tables. The main consideration is handling the initial data migration.

The VIDLOAD utility creates fixed length output records. You specify the length of the output record in IRDB2DLN. This is described in "Initial data migration" on page 53. To migrate a VSAM record to three secondary tables, set IRDB2DLN to the length of the longest of the three DB2 records. Pad shorter records with spaces.

IRDTYPE cannot be used to distinguish different record types in this situation. Your IRD must build record identifier fields. For example, assume that the working storage area your IRD builds that contains the data for the secondary tables is as follows:

```
+-+---+--------+-----+-+---+--------------+-+---+----------+---+
|X|key| T1 data| pad |Y|key|T2 data       |Z|key| T3 data  |pad|
+-+---+--------+-----+-+---+--------------+-+---+----------+---+
```

This shows the area addressed by IRDB2DAT. The data for the first table (T1) has a record identifier of X. The data for the second table (T2) has a record identifier of Y and the data for the third table (T3) has a record identifier of Z. The T2 table has the longest row, so the value your IRD sets in IRDB2DLN is T2 row length plus the length of the record identifier. The data for tables T1 and T3 are padded to the length of T2.

In this situation, always set IRDTYPE=??.

The DB2 load cards for the table T1 should include WHEN(1:n) = 'X', and the equivalent values for loading tables T2 and T3.

Handling nullable columns in an IRD

The sample FBE CNULLCL shows you how CICS VT supports nullable DB2 columns.

You can only use an FBE for a field that is mapped to CICS VT. MULTIRD shows that when you migrate a file to multiple DB2 tables, the IRD is responsible for building DB2 column values for the secondary tables. The secondary tables are not mapped to CICS VT, so your IRD is responsible for maintaining nullable secondary table columns.

SQL statement for nullable columns

The IRD issues SQL insert and update calls.

When you are processing a nullable column, you must specify a null indicator variable in the VALUES clause of your insert statement. Here is an example:

```
EXEC SQL
       INSERT INTO TB_MYFILE (
              MYFILE_KEY,
              MYFILE_COL1,
              MYFILE_COL1,
              MYFILE_COL3)
       VALUES (
              :HV-KEY,
              :HV-COL1,
              :HV-COL2,
              :HV-COL3:HVNULL-IND)
```

The column MYFILE_COL3 is nullable. The working storage variable HVNULL-IND is the null indicator field. A value less than 0 will result in a null value for this insert.

Data migration

The null indicator is not relevant for the initial data migration.

Your IRD decides if a nullable column value is valid or should become null. When you want to specify a null value, move a predefined character, such as '@' to the first value of the field in the VIDLOAD output record. In your DB2 load utility control cards, add the statement

```
NULLIF POS1:POS1 = '@'
```

to the utility control cards.

Chapter 8. Exit parameter lists

The exit parameter lists that are passed to FBEs and IRDs provide parameters and storage areas to drive the logic.

For assembler exits, use the macros VIDFBEP and VIDIRDP in *my* .SVIDSAMP. *custom* . COBOL versions are provided as copybooks in members VIDFBEC and VIDIRDC.

The parameter list for FBEs is explained in "FBE parameter list" . The parameter list for IRDs is explained in "IRD parameter list" on page 62 . In both cases, field attributes are shown in COBOL.

Accessing exit parameters

When an FBE is invoked by CICS VT, addressability is provided to three parameters:

- The address of the VSAM field the exit maps to.
- The address of the DB2 column that maps to the VSAM field. If the VSAM field is not mapped to a DB2 column, the address is 0.
- The address of the FBE parameter list.

In a COBOL FBE, include appropriate variables in the linkage section. For a PLI FBE, include them as parameters in the PROCEDURE statement. In an assembler FBE, register 1 points to the three-word parameter list.

The following three parameters are passed to an IRD:

- The address of the VSAM record area
- The address of the DB2 record area
- The address of the IRD parameter list.

You address these parameters using the same techniques that are described for FBE parameters.

FBE parameter list

The names, short descriptions, and attributes of the parameters are shown in Table 3, followed by a more detailed explanation of each parameter.

Table 3. VIDFBE parameters

Parameter	Description	Attribute
EXDIMNAM	DIM name	PIC X(8)
EXDDMNAM	DDM name	PIC X(8)
EXFLDNAM	VSAM field name	PIC X(8)
EXCONVAD	Address of VIDCONV	USAGE IS PTR
EXVSAIO	Address of VSAM record area	USAGE IS PTR
EXDB2IO	Address of the DB2 record area	USAGE IS PTR
EXRSVD1	Reserved	PIC X(4)

Table 3. VIDFBE parameters (continued)

Parameter	Description	Attribute
EXRSVD2	Reserved	PIC X(4)
EXRSVD3	Reserved	PIC X(4)
EXRSVD4	Reserved	PIC X(4)
EXVSAFLD	Address of VSAM field	USAGE IS PTR
EXDB2FLD	Address of DB2 column	USAGE IS PTR
EXWRKA	Address of 16k work area	USAGE IS PTR
EXFLDNUM	Field build order	PIC S9(4) COMP
EXVSAFLN	VSAM field length	PIC X
EXDB2FLN	DB2 column length	PIC X
EXFUNCT	Function to be performed	PIC X
EXRET	Exit return code	PIC X
EXNULOFF	Offset into null pool for column	PIC S9(4) COMP
EXNULLS	Address of null pool	USAGE IS PTR
EXRTNTAB	Address of field routine table	USAGE IS PTR
EXFILLCH	VSAM fill character	PIC X
EXFLDTYP	VSAM field type	PIC X
EXFLDMAP	Field is mapped	PIC X
EXFLDNL	DB2 column is nullable	PIC X
EXDB2TYP	DB2 column data type	PIC X
EXCICS	Exit running in CICS	PIC X
EXFILLER	Reserved	PIC X(2)
EXSQLCA	Address of SQLCA	USAGE IS PTR
EXWRKDS	Reserved	USAGE IS PTR
EXVSABLD	VSAM record area being built	PIC X
EXRSVD5	Reserved	PIC X
EXSTAT	Return code from exit	PIC X(4)
EXVSAFLV	VSAM field length	PIC S9(4) COMP
EXDB2FLV	DB2 column length	PIC S9(4) COMP
EXDB2BLD	DB2 record area being built.	PIC X
EXRSVD6	Reserved	PIC X(3)
EXCALL	VSAM call type	PIC X(4)

Description and usage of FBE parameters

The description of some of the parameters refers to the sample exits in Chapter 5, “Coding FBEs for field level reengineering,” on page 17 , Chapter 6, “FBE for managing a control record,” on page 31 and “Multiple record type solution” on page 43 .

EXDIMNAM

This is the name of the DIM being processed. This parameter is useful when your exit is used by multiple DIMs, and the processing varies in each case.

EXDDMNAM
This is the name of the DDM being processed.

EXFLDNAM
This is the name of the field that the exit is mapped to. This parameter is useful when your exit is used by multiple fields and the processing varies in each case.

EXCONVAD
This is the address of the VT module VIDCONV that contains the field conversion routines. You use this in assembler FBEs to branch to VIDCONV.

EXVSAIO
This is the address of the VSAM record area. If your FBE is operating at an individual field level, like the samples CNULLCL or CJULGREG, this parameter is not required. FBEs at a record level such as MULTFBE and CTLRECF use this parameter.

EXDB2IO
This is the address of the DB2 record area and its usage is similar to EXVSAIO.

EXRSVD1-4
Reserved.

EXVSAFLD
This is the address of the VSAM field the FBE is mapped to. Use this parameter when your FBE is operating at an individual field level, like the samples CNULLCL or CJULGREG.

EXDB2FLD
This is the address of the DB2 column the FBE is mapped to. This is only valid for fields that are mapped to a column. If the field is not mapped to a column, this address is zero. Usage is similar to EXVSAFLD.

EXWRKA
This is the address of the 16k work area that is shared between all the exits in a single DIM. For assembler FBEs, use this to store the save area to enable your FBE to be re-entrant. For high-level language FBEs, use this to share data areas between exits.

EXFLDNUM
This is the build order number of this field in the mapping. If your FBE handles multiple fields, you can use this parameter instead of EXFLDNAM.

EXVSAFLN
This contains the length of the VSAM field. This is a 1-byte field and is intended for use by an EXECUTE instruction in an assembler exit. (See EXVSAFLV for high-level language FBEs).

EXDB2FLN
This is the length of the DB2 column. This is a 1-byte field and is intended for use by an EXECUTE instruction in an assembler exit. (See EXDB2FLV for high-level language FBEs).

EXFUNCT
This is the function being performed when the exit is invoked. Possible values are:

V Building VSAM field

D Building DB2 field

If your FBE is mapped to a key field, use EXFUNCT in conjunction with EXVSABLD and EXDB2BLD.

EXRET

This is the return code that is set by the FBE. Possible values are:

"blank"
Good return code. VT continues building the remaining mapped fields.

Y Good return code. The IO area is fully built. This is used in CTLRECF.

P Bad return code. Control returns to the application program.

E Bad return code. The VSAM call abends U3017 in batch and 3107 transaction abend code in CICS .

When an FBE is invoked, EXRET is blank. See EXSTAT.

EXNULOFF

VT maintains a table of null variables for every column in the table that the DIM maps to. This field contains the offset in the table to the column that is mapped to the field being processed by the exit.

EXNULLS

This is the address of the nulls table. To locate the null indicator variable for any given field, add EXNULOFF to the address in EXNULLS. An example of how to do this in COBOL is in CNULLCL.

EXRTNTAB

This is the address of the field routine table. This is for use by an assembler FBE only.

EXFILLCH

This is the value of the filler character for unmapped areas of the VSAM record area. This is user-specified in the mapping.

EXFLDTYP

This is the VSAM field type that is specified in the mapping. Possible values are:

C The VSAM field contains character (or zoned decimal) data

P The VSAM field contains signed packed decimal data

U The VSAM field contains unsigned packed decimal data

H The VSAM field contains hexadecimal data

B The VSAM field is 2 fullwords and contains hexadecimal data

EXFLDMAP

This indicates if the VSAM field is mapped to a column. Possible values are:

X'00' The field is mapped to a column

X'FF' The field is not mapped to a column

EXFLDNL

This indicates if the DB2 column in nullable. Possible values are:

N The DB2 column is not nullable

Y The DB2 column is nullable

EXDB2TYP
This is the DB2 column type that the field maps to. Possible values are:

F INTEGER

H SMALLINT

N DECIMAL

C CHAR

B BIGINT

V VARCHAR

D DATE

T TIME

S TIMESTAMP

EXCICS
This parameter is set to `C` if the exit is being invoked in a CICS environment.

EXFILLER
Reserved

EXSQLCA
This is the address of the SQLCA. Set this to the address of the FBE SQLCA to enable CICS VT to format and display the SQL code in the VIDDMPD DD statement.

EXWRKDS
Reserved

EXVSABLD
This indicates that the VSAM record area is being built. It is only relevant when EXFUNCT = `V`. Possible values are:

Y The VSAM IO area is being built

N A VSAM key value is being built for a call using a key

This parameter is required when the FBE is on a field that is part of the key or an alternate index path. It is used in CTLRECF.

EXRSVD5
Reserved

EXSTAT
This is the status code returned by the exit. This is used in conjunction with EXRET. It is set by CICS VT to 0000 when the FBE is invoked. If your FBE sets EXRET = `P`, CICS VT sets EXSTAT to 0020.

EXVSAFLV
This is the length of the VSAM field. It is equivalent to EXVSAFLN but is for use by high-level language exits.

EXDB2FLV
This is the length of the DB2 column. It is equivalent to EXDB2FLN but is for use by high-level language exits. If you are processing a file containing variable length records and your DB2 table has a VARCHAR column, this field contains the length of the variable data.

EXDB2BLD
This indicates that the DB2 record area is being built. It is only relevant when EXFUNCT = 'D'. Possible values are:

Y The DB2 record area is being built

N A DB2 key value is being built for a call using a key

This parameter is required when the FBE is on a field that is part of the key or an alternate index path. It is used in CTLRECF.

EXRSVD6
Reserved

EXCALL
This field shows the VSAM call being processed. Possible values are:

OPEN
CLOS
GETU
GET
LOAD
PUTI
PUTR
ERAS
POIN
ENDR

IRD parameter list

The names, short descriptions, and attributes of the parameters are shown in Table 4, followed by a more detailed explanation of each.

Table 4. VIDIRDP parameters

Parameter	Description	Attribute
IRDIMNAM	DIM name	PIC X(8)
IRDDMNAM	DDM name	PIC X(8)
IRDCNVAD	Address of VIDCONV	USAGE IS PTR
IRDVSAIO	Address of VSAM record area	USAGE IS PTR
IRDDB2IO	Address of DB2 record area	USAGE IS PTR
IRDRSVD1	Reserved	PIC X(4)
IRDRSVD2	Reserved	PIC X(4)
IRDRSVD3	Reserved	PIC X(4)
IRDRSVD4	Reserved	PIC X(4)
IRDWRKA	Address of 16k work area	USAGE IS PTR
IRDFUNCT	Function being performed	PIC X
IRDRET	Return code	PIC X
IRDBA	When exit is invoked	PIC X
IRDSTAT	Status code	PIC X(2)
IRDCICS	Exit running in CICS PIC X	PIC X
IRDRPTGR	Number of repeating groups	PIC S9(4) COMP

Table 4. VIDIRDP parameters (continued)

Parameter	Description	Attribute
IRDTYPE	Redefined record type	PIC X(2)
IRDB2DLN	Repeat group record length	PIC S9(4) COMP
IRDB2DAT	Address of repeat group	USAGE IS PTR
IRDSQLCA	Address of SQLCA	USAGE IS PTR

Description and usage of IRD parameters

The explanation of the usage of some parameters refers to the sample exits in Chapter 5, "Coding FBEs for field level reengineering," on page 17, Chapter 6, "FBE for managing a control record," on page 31 and "Multiple record type solution" on page 43.

IRDIMNAM
This is the name of the DIM.

IRDDMNAM
This is the name of the DDM.

IRDCNVAD
This is the address of the VT module VIDCONV that contains the field conversion routines. You use this in assembler IRDs to branch to VIDCONV.

IRDVSAIO
This is the address of the VSAM record area.

IRDDB2IO
This is the address of the DB2 record area.

IRDRSVD1-4
Reserved

IRDWRKA
This is the address of the 16k work area that is shared between all the exits in a single DIM. For assembler IRDs, use this to store the save area to enable your IRD to be re-entrant. For high-level language IRDs, use this to share data areas between exits.

IRDFUNCT
This is the function being performed when the exit is invoked. Possible values are:

I A new record is being inserted

D A record is being deleted

R An existing record is being updated

L The record is being processed by the VIDLOAD utility

X An exclusive table lock has been obtained (see PK14457)

IRDRET
This is the return code from the exit. Possible values are:

blank The exit has ended normally

Y The exit has ended normally and the call is complete

N The exit has ended and the call is not complete

P The exit has ended and a bad return code should be set

E CICS VT abends the call with U3018

X The exit has ended normally and obtained an exclusive table lock

IRDBA
This is the value specified in the mapping for the IRD processing sequence. Possible values are:

B The exit is called before the DDM has issued an SQL call

A The exit is called after the DDM has issued an SQL call

blank The exit is called before and after the DDM has issued an SQL

IRDSTAT
The status code that CICS VT sets for the call. Possible values are:

OK A normal return code will be set

NO A bad return code will be set

If you set IRDRET = `Y` and IRDSTAT=`NO`, CICS VT sets a return code of 0020 for the call and writes the formatted SQLCA to VIDDMPD.

IRDCICS
This parameter is set to `C` if the exit is being invoked in a CICS environment.

IRDRPTGR
This parameter is only used for the initial data migration. It is significant when you have a repeating group or array and each group item becomes a single DB2 row. You use it to define the number of DB2 records to be written. It is explained further in "IRD parameters for data migration."

IRDTYPE
Use this parameter to identify the output record type. It is only used for the initial data migration and is explained further in "IRD parameters for data migration."

IRDB2DLN
Your exit defines the length of the output area that is built. This parameter is only used for the initial data migration and is explained further in "IRD parameters for data migration."

IRDB2DAT
This parameter contains the address of the output area your exit builds. It is only used for the initial data migration and is explained further in "IRD parameters for data migration."

IRDSQLCA
This is the address of the SQLCA. Set this to the address of the FBE SQLCA to enable CICS VT to format and display the SQL code in the VIDDMPD DD statement.

IRD parameters for data migration

You must use a combination of an IRD and FBE to migrate a dataset to more than one DB2 table. Your IRD must take account of the initial data migration, when IRDFUNCT=`L`. The IRD parameters IRDGPTR, IRDTYPE, IRDB2DLN, and IRD2DAT are only used when IRDFUNCT=`L`.

To understand the use of these parameters, the following sections describe their use in various scenarios.

Migrating a file with multiple record types

The sample exit MULTIRD details the FBE and IRD for migrating a file with multiple record types. For each VSAM record, there are two DB2 records created during initial migration; a record for the primary DB2 table and a record for the appropriate record type DB2 table.

The values of the data migration parameters in this case are shown in Table 5:

Table 5. Data migration parameter settings

Parameter Name	Parameter Value
IRDRPTGR	A single record is written for each secondary table so the value is 1.
IRDTYPE	The appropriate 2-byte record type indicator.
IRDB2DLN	The length of the DB2 record excluding IRDTYPE.
IRDB2DAT	The area in your exit where the DB2 record is built.

Migrating a file with a repeating group

When your file has a repeating group field, you may decide that the group field data is migrated to a separate table containing one DB2 row for each group entry. The values of the data migration parameters in this case are shown in Table 6.

Table 6. Data migration parameter settings

Parameter Name	Parameter Value
IRDRPTGR	The number of records to be written for the secondary table. If you have an OCCURS DEPENDING ON clause then this parameter value may vary.
IRDTYPE	The appropriate 2-byte record type indicator.
IRDB2DLN	The length of each DB2 record excluding IRDTYPE.
IRDB2DAT	The area in your exit where the DB2 records are built.

Chapter 9. Generic assembler FBEs

Generic exits written in assembler are available for some common scenarios. These have all been used in live customer situations.

The generic exits are as follows:

PACKC2F
Handles the conversion between unsigned packed decimal fields in VSAM and signed decimal columns in DB2.

PACKDEC
Handles invalid packed decimal field values in a START/STARTBR/ POINT call.

NULLCOL
Enables a field containing a predefined value in VSAM to be stored as a null value in DB2.

NULLCOLS
Enables a field containing common values such as spaces or binary zeros to become null values in DB2.

BIT2CHAR
Stores bit data as discrete CHAR(1) values in DB2.

JULGREG
Transforms Julian date field values in VSAM to DB2 date column format.

These exits are available for downloading from the following link: This link opens in a new window

Minor modifications may be required to BIT2CHAR and JULGREG if your field attributes or lengths differ from those that these exits were initially written for. Comments in the exits define the characteristics of the supported fields.

There has been no benchmarking to compare the performance of assembler exits with the equivalent exits in COBOL.

PACKC2F

DB2 stores packed decimal data in signed format. In COBOL, a packed decimal field can be defined as signed or unsigned as follows:

Signed:
PIC S999V99 COMP-3

Unsigned:
PIC 99999V99 COMP-3

Assume you have a field defined as PIC 999V99 COMP-3 (unsigned). In your VSAM file, a value of 123.45 is physically stored as X'12345F'. DB2 treats all decimal column values as signed. If you map this unsigned field to a decimal column, CICS VT would return a field value of X'12345C' to your application.

Unless the field is part of the key or an alternate index path, it may not affect your application program but it would be highlighted by the CICS VT dual mode

facility (DMF) as a difference, and therefore an error. To avoid this error use the generic exit PACKC2F. It resets the sign bits from b'1100' to b'1111' when the field is retrieved from DB2.

If the field is part of the base cluster key or an alternate index key, values retrieved from DB2 must be converted to unsigned values using an exit such as PACKC2F.

PACKDEC

When a copybook field is mapped to a decimal or date DB2 column, invalid field values from your existing application programs will result in an error, such as an S0C7 abend or -310 SQL code. In most cases, invalid data is identified during initial data migration, either during the VIDLOAD utility or during the DB2 load process.

When a date or decimal column maps to a field or subfield which is the base cluster or alternate index key, a data error may occur in calls such as START, STARTBR, or POINT. For example, assume you have a group key field and the subfields are a combination of zoned and packed decimal. If the key field is initialized in a single COBOL MOVE statement, either the zoned or packed decimal field will contain in invalid value. This is not an issue in VSAM but is an issue when the data is in DB2.

The PACKDEC exit is used for a packed decimal key or alternate index field that is mapped to a DEC column with the same length. If the exit is being called for a direct read it tests the value of the first byte. If the first byte value is X'F0', the exit overlays the entire field with packed decimal zero. Without the exit, an S0C7 abend occurs.

NULLCOL

This exit translates predefined hexadecimal strings into nulls in DB2, and translates a null back to the specified hexadecimal value on retrieval from DB2.

For example, your program could insert either high values or low values in a packed decimal field to represent that a valid field value is not known. In a DB2 application, nulls are often used to represent column values that are not known at insert time.

The field value must be coded as a constant in line 23 in the sample exit source as follows:

```
NULLVAL  DC    X'FFFFFFF'        NULL VALUE
```

No other changes to the sample NULLCOL exit are required.

The NULLCOL exit should be copied to a new source member with an 8-byte name and the appropriate nulls value specified at the NULLVAL label. The new exit must be assembled and link-edited, and mapped, The mapping is shown in Figure 19 on page 69:

```
----------------------------- CICS VT: Update field -----------------------------
 Command ===> _____________________________________________________ Scroll ===> CSR_

 Data set name  : ORDERFL
 Creator        : PROD                                                          +
 Table          : ORDER_TAB                                                     +
 Data set length: 00352

 Field name . . . . . . : VIDF0019
 Field length . . .  ===> 00004             (In bytes)
 Field type . . . .  ===> P                 ("C", "P", "U", "X", "F", "H")
 Column name  . . .  ===> CREATE_DATE..................... + (Look-up available)
 Starting position   ===> 00259             ("1" = Beginning of data set)
 Picture or FBE . .  ===> EXITL=NULLSMP1____________  (example  HH.XX.SS.NNNNNN)
                                                        (or MMDDYY)
                                                        (or EXITx=exit name)
 Parameters . . . .  ===> _____________________________  Optional user parameters
 Special function .  ===> ___               ("KEY", "PTH", "BKY", or blank)
 Mapped from table   ===> P                 ("P"=Prim, "X"=Not mapped)
 Build order  . . .  ===> 00019             ("1"=first, "2"=second and so on)


 Press: Enter=Update  PF3=Exit  PF1=Help
```

Figure 19. Specifying an FBE exit

Note that the exit compares the value of the entire VSAM field to the hard-coded nulls value.

NULLCOLS

This generic exit is for mapping fields to nullable DB2 columns. Optional user parameters specified in the mapping define the VSAM field values that are to become nulls in DB2. The default version of the exit handles the conversion of repeating bytes of binary zeros, binary ones, and spaces into nulls, and conversion to any of these values on retrieval from DB2.

Much of the function of this exit is superseded by the built-in support for nullable columns.

The exit must be mapped as follows:

```
EXITA=NULLCOLS,input,output
```

The parameter `input` defines the VSAM field value that is to become null in DB2. The parameter `output` defines the field value to be built by CICS VT and returned to VSAM. The value you specify must be repeated in each byte for the entire field. Otherwise it is not treated as null.

Possible parameter values for `input`are:

L Low values (binary zeros)

S Spaces

H High values (binary ones)

1-5 User defined

The `output` parameter is optional. If omitted, it is assumed to be the same as the input parameter. Multiple input values can be specified if more than one VSAM field value is to become null. In this case, if no output parameter is specified, the exit assumes that the first input parameter represents the output parameter.

Example 1:

```
EXITA=NULLCOLS,S
```

An input field value of spaces becomes a null value in DB2. On retrieval from DB2, the field will be spaces.

Example 2:

```
EXITA=NULLCOLS,LS
```

An input field value of binary zeros or spaces become null in DB2. On retrieval, the field value becomes binary zeros.

Example 3:

```
EXITA=NULLCOLS
```

If no parameters are specified, the default is L. Binary zeros become null in DB2, and the retrieved field value is binary zeros.

Example 4:

```
EXITA=NULLCOLS,,S
```

No input parameter is supplied, so the default is L. Field values of binary zeros become nulls in DB2. On retrieval, the field value becomes spaces.

You can define up to five custom values to be handled as null bytes. This involves modifying the exit source and assembling a new load module. There are potentially 10 lines of assembler code near the start of the program to be modified.

```
SETNULL1     DC    X'6F'          /* INPUT PARAMETER VALUE 1
SETNULL2     DC    X'4B'          /* INPUT PARAMETER VALUE 2
SETNULL3     DC    X'60'          /* INPUT PARAMETER VALUE 3
SETNULL4     DC    X'00'          /* INPUT PARAMETER VALUE 4
SETNULL5     DC    X'00'          /* INPUT PARAMETER VALUE 5

FILLVAL1     DC    X'4B'          /* OUTPUT PARAMETER VALUE 1
FILLVAL2     DC    X'FF'          /* OUTPUT PARAMETER VALUE 2
FILLVAL3     DC    X'00'          /* OUTPUT PARAMETER VALUE 3
FILLVAL4     DC    X'00'          /* OUTPUT PARAMETER VALUE 4
FILLVAL5     DC    X'00'          /* OUTPUT PARAMETER VALUE 5
```

The `SETNULLx` fields define `input` parameters and the `FILLVALn` fields define `output` parameters. Custom value parameters can be combined with supplied parameter values.

Example 5:

```
EXITA=NULLCOLS,1SH
```

Input field values of X'6F', spaces, or binary ones become null values. No output parameter is specified, so the first input parameter value is used, resulting in field values x'6F' returned from DB2.

Example 6:

```
EXITA=NULLCOLS,1,2
```

VSAM field values of X'6F' become nulls, and are set to X'FF' on retrieval from DB2.

BIT2CHAR

This generic exit converts bits in a binary string to DB2 character data. Each bit becomes a byte. There are no parameters required.

The exit handles input field sizes of 1, 2, 3, or 4 bytes. The output DB2 columns can be 1-byte columns, or 8, 18, 32, or 64 bytes. If the output is 1-byte columns they must be adjacent in the DB2 table. For example:

VSAM Field	DB2 Row
8 bits	Either 8 x 1-byte columns Or 2 x 4-byte columns Or 1 x 8-byte column
32 bits	Either 32 x 1-byte columns Or 4 x 8-byte columns Or 1 x 32-byte column

JULGREG

DB2 does not have the capability to store Julian dates (unless you use a LOCAL date format). The exit JULGREG transforms a Julian date field value in VSAM to the appropriate date value in DB2. Windowing enables you to set the appropriate century. There are a number of different variations in fields that use Julian dates, and the version currently supported by JULGREG has a VSAM field format of YYDDDHHMM.

Chapter 10. Built-in conversion routines

There are many built-in data conversion routines provided with CICS VT.

"Converting from VSAM to DB2" on page 74 shows all of the routines for converting a VSAM field to a DB2 column. "Converting from DB2 to VSAM" on page 75 shows all of the routines for converting from DB2 to VSAM.

You can call any of the built-in conversion routines from within an exit. This can be particularly useful when you are processing DATE, TIME, and TIMESTAMP columns in DB2.

Do not statically link VIDCONV with your exit. Include VIDHLIPI in the link-edit.

Calling a VT conversion routine in COBOL

The format for calling the conversion routines in COBOL is as follows:

```
CALL VIDCONV        USING vidconv-routine-number
                          source-field
                          source-field-length
                          source-field-picture
                          destination-field
                          destination-field-length
                          destination-field-picture
                          picture-field-length
```

Sample COBOL working storage variables

Here is an example of the working storage variables for a COBOL program. This is taken from the CTLRECF sample exit described in Chapter 6, "FBE for managing a control record," on page 31.

```
* VIDCONV AND PARAMETER LIST VARIABLES
 01 VIDCONV                    PIC X(8) VALUE 'VIDCONV ' .
 01 DB2-TO-VSAM-PARMLIST                .
    02 DB2-ROUTINE-NO          PIC S9(8) COMP VALUE 50.
    02 DB2-SOURCE-FIELD        PIC X(8) .
    02 DB2-SOURCE-FIELD-LEN    PIC S9(8) COMP VALUE  8.
    02 DB2-SOURCE-FIELD-PIC    PIC S9(8) COMP VALUE  0.
    02 DB2-DEST-FIELD          PIC S9(7) COMP-3 .
    02 DB2-DEST-FIELD-LEN      PIC S9(8) COMP VALUE  4.
    02 DB2-DEST-FIELD-PIC      PIC X(6) VALUE 'HHXXSS'.
    02 DB2-PIC-FIELD-LEN       PIC S9(8) COMP VALUE  6.
 01 VSAM-TO-DB2-PARMLIST                          .
    02 VS-ROUTINE-NO           PIC S9(8) COMP VALUE 20.
    02 VS-SOURCE-FIELD         PIC S9(7) COMP-3       .
    02 VS-SOURCE-FIELD-LEN     PIC S9(8) COMP VALUE  4.
    02 VS-SOURCE-FIELD-PIC     PIC X(6) VALUE 'HHXXSS'.
    02 VS-DEST-FIELD           PIC X(08)              .
    02 VS-DEST-FIELD-LEN       PIC S9(8) COMP VALUE  8.
    02 VS-DEST-FIELD-PIC       PIC S9(8) COMP VALUE  0.
    02 VS-PIC-FIELD-LEN        PIC S9(8) COMP VALUE  6.
```

Using conversion routines

CICS VT uses the conversion routines based on mapping information and stores it in the DIM. For example, if you have a field defined as type P for packed decimal and you map it to a DECIMAL column, CICS VT will use the PACKDEC routine for update and insert calls and DECPACK for retrieval calls.

Most of the conversion routines are provided for user-written FBE exits. If you are writing CICS VT exits that use the VIDCONV routines, you can test that your parameter list is correct and that the correct conversion is occurring using the VIDFCTST program in the VID.SVIDLODE library. Input into the program consists of an 80-byte record describing the name of the field conversion routine that is to be invoked, the source and destination lengths, the picture, and the source data.

Output from the program is an 80-byte record that tests both conversion routines. For example, if a test is performed for CHARDATE, the output also includes a test for DATECHAR.

Member VIDCONVT in *my*.SVIDSAMP.*custom* contains sample JCL and instructions on how to use this testing facility.

Converting from VSAM to DB2

Routine number	Routine name and function	Source length required	Dest length required	Source PIC allowed	Dest PIC allowed	Note
1	CHARINT Character to Integer	Y	N(4)	N	N	
2	CHARSINT Character to SMALLINT	Y	N(2)	N	N	
3	CHARDEC Character to Packed	Y	Y	N	N	
4	CHARVCH Character to VARCHAR	Y	N	N	N	
5	CHARDATE Character to DATE	Y	N(10)	Y (req'd)	N	2
6	CHARTIME Character to TIME	Y	N(8)	Y (req'd)	N	2
7	CHARDTIM Character to TIMESTAMP	Y	N(26)	Y (req'd)	N	2
8	ZONEINT Zoned to INTEGER	Y	N(4)	N	N	
9	ZONESINT Zoned to SMALLINT	Y	N(2)	N	N	
10	ZONEDEC Zoned to Packed	Y	Y	N	N	
11	ZONECHAR Zoned to CHARACTER	Y	Y	N	N	1
12	ZONEVCH Zoned to VARCHAR	Y	N	N	N	
13	ZONEDATE Zoned to DATE	Y	N(10)	Y (req'd)	N	2
14	ZONETIM Zoned to TIME	Y	N(8)	Y (req'd)	N	2
15	ZONEDTIM Zoned to TIMESTAMP	Y	N(26)	Y (req'd)	N	2
16	PACKINT Packed to INTEGER	Y	N(4)	N	N	
17	PACKSINT Packed to SMALLINT	Y	N(2)	N	N	
18	PACKDEC Packed to DECIMAL	Y	Y	N	N	
19	PACKDATE Packed to DATE	Y	N(10)	Y (req'd)	N	2
20	PACKTIME Packed to TIME	Y	N(10)	Y (req'd)	N	2
21	PACKDTIM Packed to TIMESTAMP	Y	N(26)	Y (req'd)	N	2
22	BINSINT Binary to INTEGER	Y	Y	N	N	

Routine number	Routine name and function	Source length required	Dest length required	Source PIC allowed	Dest PIC allowed	Note
23	BINDEC Binary to DECIMAL	Y	Y	N	N	

1. A picture clause is required.
2. The ZONECHAR and CHARZONE routines require that the source and destination lengths are the same.

Converting from DB2 to VSAM

Routine number	Routine name and function	Source length required	Dest length required	Source PIC allowed	Dest PIC allowed	Note
31	INTCHAR Integer to Character	N(4)	Y	N	N	
32	SINTCHAR SMALLINT to Character	N(2)	Y	N	N	
33	DECCHAR Packed to Character	Y	Y	N	N	
34	VCHCHAR VARCHAR to Character	N	N	N	N	
35	DATECHAR DATE to Character	N(10)	Y	N	Y (req'd)	1
36	TIMECHAR TIME to Character	N(8)	Y	N	Y (req'd)	1
37	DTIMCHAR TIMESTAMP to Character	N(26)	Y	N	Y (req'd)	1
38	INTZONE INTEGER to Zoned	N(4)	Y	N	N	
39	SINTZONE SMALLINT to Zoned	N(2)	Y	N	N	
40	DECZONE Packed to Zoned	Y	Y	N	N	
41	CHARZONE Character to Zoned	Y	Y	N	N	2
42	VCHZONE VARCHAR to Zoned	N	Y	N	N	
43	DATEZONE Date to Zoned	N(10)	Y	N	Y (req'd)	1
44	TIMEZONE TIME to Zoned	N(8)	Y	N	Y (req'd)	1
45	DTIMZONE TIMESTAMP to Zoned	N(26)	Y	N	Y (req'd)	1
46	INTPACK INTEGER to Packed	N(4)	Y	N	N	
47	SINTPACK SMALLINT to Packed	N(2)	Y	N	N	
48	DECPACK Packed to Packed	Y	Y	N	N	
49	DATEPACK DATE to PACKED	N(20)	Y	N	Y (req'd)	1
50	TIMEPACK TIME to PACKED	N(8)	Y	N	Y (req'd)	1
51	DTIMPACK TIMESTAMP to PACKED	N(26)	Y	N	Y (req'd)	1
52	SINTBIN SMALLINT to Binary	Y	Y	N	N	
53	DECBIN Packed to Binary	Y	Y	N	N	

1. A picture clause is required.
2. The ZONECHAR and CHARZONE routines require that the source and destination lengths are the same.

Appendix A. JCL to compile COBOL exit

```
//COB     EXEC PGM=IGYCRCTL,REGION=2M,
//        PARM=(QUOTE,NODYNAM,ADV,'BUF(12288)',SOURCE,XREF,LIST,MAP)
//STEPLIB   DD DSN=IGY320.SIGYCOMP,DISP=SHR
//SYSLIB    DD DSN=DFH320.CICS.SDFHCOB,DISP=SHR
//          DD DSN=DFH320.CICS.SDFHMAC,DISP=SHR
//          DD DSN=DFH320.CICS.SDFHSAMP,DISP=SHR
//          DD DSN=CICSVT.HLL.COPYBOOK,DISP=SHR
//          DD DSN=my.SVIDSAMP.custom,DISP=SHR
//SYSIN     DD DSN=CICSVT.HLL.EXIT.SOURCE(CTLRECF),DISP=SHR
//DBRMLIB   DD DSN=CICSVT.HLL.EXIT.DBRMLIB(CTLRECF),DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSLIN    DD DSN=&&LOADSET,DISP=(MOD,PASS),
//             UNIT=SYSDA,SPACE=(80,(250,100))
//SYSUT1    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT2    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT3    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT4    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT5    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT6    DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT7    DD UNIT=SYSDA,SPACE=(460,(350,100))
//LKED    EXEC PGM=IEWL,REGION=2M,
//             PARM='LIST,XREF',COND=(5,LT,COB)
//SYSLIB    DD DSN=DFH320.SDFHLOAD,DISP=SHR
//          DD DSN=CEE.SCEELKED,DISP=SHR
//VS2LIB    DD DSN=CVT120.SVIDLODE,DISP=SHR
//SYSLMOD   DD DSN=CICSVT.HLL.EXIT.LOAD(CTLRECF),DISP=SHR
//SYSUT1    DD UNIT=SYSDA,DCB=BLKSIZE=1024,SPACE=(1024,(200,20))
//SYSPRINT  DD SYSOUT=*
//SYSLIN    DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//SYSIN     DD *
   INCLUDE VS2LIB(VIDHLIPI)
//BIND    EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB   DD DISP=SHR,DSN=CICSVT.HLL.EXIT.DBRMLIB
//SYSTSPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSTSIN   DD *
   DSN SYSTEM(DB2B)
   BIND PACKAGE(VIDCOLL)                    +
        MEMBER(CTLRECF)                     +
        ISOLATION(CS)                       +
        VALIDATE(BIND)                      +
        RELEASE(COMMIT)
```

Note that the DB2 precompiler is invoked implicitly in this example.

Appendix B. Copybook and DDL for APPLCTL

Here is the DDL for the APPLCTL file that is discussed in Chapter 7, "Record level reengineering," on page 41. The primary table is called TB_APPLCTL. The four secondary tables are called TB_APPLCTL_010, TB_APPLCTL_020, TB_APPLCTL_030, and TB_APPLCTL_040.

```
CREATE TABLE TB_APPLCTL     (
       OBJECT_ID            CHAR(06)         NOT NULL
      ,REC_TYPE             CHAR(03)         NOT NULL
      ,USER_ID              CHAR(08)         NOT NULL
      ,USER_ID_LAST_CHG     CHAR(08)         NOT NULL
      ,LAST_CHG_DATE        DATE             NOT NULL
      ,PRIMARY KEY (OBJECT_ID, REC_TYPE, USER_ID))
    IN CVTDB.APPLCTL0;

CREATE TABLE TB_APPLCTL_010(
       OBJECT_ID            CHAR(06)         NOT NULL
      ,REC_TYPE             CHAR(03)         NOT NULL
      ,USER_ID              CHAR(08)         NOT NULL
      ,APPL_CODE            CHAR(03)         NOT NULL
      ,APPL_NAME            CHAR(30)         NOT NULL
      ,APPL_OWNER_LNAME     CHAR(20)         NOT NULL
      ,APPL_OWNER_FNAME     CHAR(10)         NOT NULL
      ,APPL_OWNER_PHONE     CHAR(10)         NOT NULL
      ,APPL_OWNER_DEPT      CHAR(150)        NOT NULL
      ,PRIMARY KEY (OBJECT_ID, REC_TYPE, USER_ID)
      ,FOREIGN KEY (OBJECT_ID, REC_TYPE, USER_ID)
       REFERENCES TB_APPLCTL ON DELETE CASCADE)
    IN CVTDB.APPLCTL1 ;

CREATE TABLE TB_APPLCTL_020(
       OBJECT_ID            CHAR(06)         NOT NULL
      ,REC_TYPE             CHAR(03)         NOT NULL
      ,USER_ID              CHAR(08)         NOT NULL
      ,APPL_CODE            CHAR(03)         NOT NULL
      ,FILE_NAME            CHAR(18)         NOT NULL
      ,DD_NAME              CHAR(08)         NOT NULL
      ,PATH_NAME1           CHAR(08)         NOT NULL
      ,PATH_NAME2           CHAR(08)         NOT NULL
      ,PATH_NAME3           CHAR(08)         NOT NULL
      ,PATH_NAME4           CHAR(08)         NOT NULL
      ,PATH_NAME5           CHAR(08)         NOT NULL
      ,PATH_NAME6           CHAR(08)         NOT NULL
      ,KEY_LEN              DEC(3,0)         NOT NULL
      ,REC_LEN              DEC(5,0)         NOT NULL
      ,READ_FLG             CHAR(01)         NOT NULL
      ,DELETE_FLG           CHAR(01)         NOT NULL
      ,INSERT_FLG           CHAR(01)         NOT NULL
      ,UPDATE_FLG           CHAR(01)         NOT NULL
      ,MAXIMUM_CT           DEC(13,0)        NOT NULL
      ,MINIMUM_CT           DEC(13,0)        NOT NULL
      ,AVERAGE_CT           DEC(13,0)        NOT NULL
      ,SHR_OPT_1            DEC(1)           NOT NULL
      ,SHR_OPT_2            DEC(1)           NOT NULL
      ,DISK_VOL1            CHAR(06)         NOT NULL
      ,DISK_VOL2            CHAR(06)         NOT NULL
      ,DISK_VOL3            CHAR(06)         NOT NULL
      ,DISK_VOL4            CHAR(06)         NOT NULL
      ,DISK_VOL5            CHAR(06)         NOT NULL
      ,PRIMARY KEY (OBJECT_ID, REC_TYPE, USER_ID)
      ,FOREIGN KEY (OBJECT_ID, REC_TYPE, USER_ID)
       REFERENCES TB_APPLCTL ON DELETE CASCADE)
    IN CVTDB.APPLCTL2 ;
```

```
CREATE TABLE TB_APPLCTL_030(
       OBJECT_ID            CHAR(06)         NOT NULL
      ,REC_TYPE             CHAR(03)         NOT NULL
      ,USER_ID              CHAR(08)         NOT NULL
      ,APPL_CODE            CHAR(03)         NOT NULL
      ,PROGRAM_NAME         CHAR(08)         NOT NULL
      ,PROGRAM_DESCR        CHAR(58)         NOT NULL
      ,K_LOC                DEC(9,0)         NOT NULL
      ,ONLINE_FLG           CHAR(01)         NOT NULL
      ,BATCH_FLG            CHAR(01)         NOT NULL
      ,RUN_CYCLE            CHAR(01)         NOT NULL
      ,LANG                 CHAR(01)         NOT NULL
      ,COMPILE_OPTIONS      CHAR(50)         NOT NULL
      ,PRIMARY KEY (OBJECT_ID, REC_TYPE, USER_ID)
      ,FOREIGN KEY (OBJECT_ID, REC_TYPE, USER_ID)
       REFERENCES TB_APPLCTL ON DELETE CASCADE)
    IN CVTDB.APPLCTL3 ;
CREATE TABLE TB_APPLCTL_040(
       OBJECT_ID            CHAR(06)         NOT NULL
      ,REC_TYPE             CHAR(03)         NOT NULL
      ,USER_ID              CHAR(08)         NOT NULL
      ,APPL_CODE            CHAR(03)         NOT NULL
      ,JOB_NAME             CHAR(08)         NOT NULL
      ,JOB_DESCR            CHAR(25)         NOT NULL
      ,RERUNNABLE           CHAR(01)         NOT NULL
      ,CRIT_PATH            CHAR(01)         NOT NULL
      ,CALLOUT              CHAR(01)         NOT NULL
      ,MUST_COMPLETE        CHAR(01)         NOT NULL
      ,PREREQ               CHAR(01)         NOT NULL
      ,POSTREQ              CHAR(01)         NOT NULL
      ,RUN_CYCLE            CHAR(01)         NOT NULL
      ,REPORT_TITLE_01      CHAR(20)         NOT NULL
      ,REPORT_TITLE_02      CHAR(20)         NOT NULL
      ,REPORT_TITLE_03      CHAR(20)         NOT NULL
      ,REPORT_TITLE_04      CHAR(20          NOT NULL
      ,REPORT_TITLE_05      CHAR(20)         NOT NULL
      ,RESTART_INSTR        CHAR(75)         NOT NULL
      ,PRIMARY KEY (OBJECT_ID, REC_TYPE, USER_ID)
      ,FOREIGN KEY (OBJECT_ID, REC_TYPE, USER_ID)
       REFERENCES TB_APPLCTL ON DELETE CASCADE)
   IN CVTDB.APPLCTL4 ;
```

Here is the COBOL copybook for the APPLCTL file.

```
01 APPL-CONTROL-REC .
   02 APPL-CONTROL-KEY .
      03 MULTREC-KEY .
         05 OBJECT-ID              PIC X(6) .
         05 RECORD-TYPE            PIC X(3) .
            88 OWNER-RECORD         VALUE '010' .
            88 FILE-RECORD          VALUE '020' .
            88 PROGRAM-RECORD       VALUE '030' .
            88 JOB-RECORD           VALUE '040' .
         05 USER-ID                PIC X(8) .

   02 TYPE-010-REC .
      03 APPL-CODE                 PIC X(3) .
      03 APPL-NAME                 PIC X(30) .
      03 APPL-OWNER-LNAME          PIC X(20) .
      03 APPL-OWNER-FNAME          PIC X(10) .
      03 APPL-OWNER-PHONE          PIC X(10) .
      03 APPL-OWNER-DEPT           PIC X(150) .
      03 FILLER                    PIC X(10) .
      03 USER-ID-LAST-CHG          PIC X(8) .
      03 LAST-CHG-DATE             PIC S9(9) COMP-3 .

   02 TYPE-020-REC REDEFINES TYPE-010-REC .
```

```
   03 APPL-CODE                PIC X(3) .
   03 FILE-NAME                PIC X(18) .
   03 DD-NAME                  PIC X(8) .
   03 PATH-NAME1               PIC X(8) .
   03 PATH-NAME2               PIC X(8) .
   03 PATH-NAME2               PIC X(8) .
   03 PATH-NAME3               PIC X(8) .
   03 PATH-NAME4               PIC X(8) .
   03 PATH-NAME5               PIC X(8) .
   03 PATH-NAME6               PIC X(8) .
   03 KEY-LEN                  PIC S9(3) COMP-3 .
   03 REC-LEN                  PIC S9(5) COMP-3 .
   03 READ-FLG                 PIC X .
   03 DELETE-FLG               PIC X .
   03 INSERT-FLG               PIC X .
   03 UPDATE-FLG               PIC X .
   03 MAXIMUM-CT               PIC S9(13) COMP-3 .
   03 MINIMUM-CT               PIC S9(13) COMP-3 .
   03 AVERAGE-CT               PIC S9(13) COMP-3 .
   03 SHR-OPT-1                PIC 9 .
   03 SHR-OPT-2                PIC 9 .
   03 DISK-VOL1                PIC X(6) .
   03 DISK-VOL2                PIC X(6) .
   03 DISK-VOL3                PIC X(6) .
   03 DISK-VOL4                  PIC X(6) .
   03 DISK-VOL5                  PIC X(6) .
   03 FILLER                     PIC X(94) .
   03 USER-ID-LAST-CHG           PIC X(8) .
   03 LAST-CHG-DATE              PIC S9(9) COMP-3 .

02 TYPE-030-REC REDEFINES TYPE-010-REC .
   03 APPL-CODE                  PIC X(3) .
   03 PROGRAM-NAME               PIC X(8) .
   03 PROGRAM-DESCR              PIC X(58) .
   03 K-LOC                      PIC 9(9) .
   03 ONLINE-FLG                 PIC X .
   03 BATCH-FLG                  PIC X .
   03 RUN-CYCLE                  PIC X .
      88 DAILY                 VALUE 'D' .
      88 WEEKLY                VALUE 'W' .
      88 MONTHLY               VALUE 'M' .
      88 QUARTERLY             VALUE 'Q' .
      88 YEARLY                VALUE 'Y' .
      88 ON-REQUEST            VALUE 'O' .
      88 SPECIAL               VALUE 'S' .
   03 LANG                        PIC X .
      88 COBOL-LANG            VALUE 'C' .
      88 COBOL2                VALUE '2' .
      88 COBOL-LE              VALUE 'L' .
      88 ASSEMBLER             VALUE 'A' .
      88 OTHER-LANG            VALUE 'O' .
   03 SPECIAL-COMPILE-OPTIONS     PIC X(50) .
   03 DATA-CHANGE-FLG             PIC X .
   03 FILLER                      PIC X(100) .
   03 USER-ID-LAST-CHG            PIC X(8) .
   03 LAST-CHG-DATE               PIC S9(9) COMP-3 .

02 TYPE-040-REC REDEFINES TYPE-010-REC .
   03 APPL-CODE                  PIC X(3) .
   03 JOB-NAME                   PIC X(8) .
   03 JOB-DESCR                  PIC X(25) .
   03 RERUNNABLE-FLG             PIC X .
      88 YES                   VALUE 'Y' .
      88 NO-RERUN              VALUE 'N' .
   03 CRITICAL-PATH-FLG          PIC X .
      88 YES                   VALUE 'Y' .
      88 NOT-CRITICAL          VALUE 'N' .
```

```
03 CALL-OUT-FLG              PIC X .
   88 YES                  VALUE 'Y' .
   88 NO-CALLOUT           VALUE 'N' .
03 MUST-COMPLETE-FLG         PIC X .
   88 YES                  VALUE 'Y' .
   88 NO-COMPLETE          VALUE 'N' .
03 PREREQ-FLG                PIC X .
   88 YES                  VALUE 'Y' .
   88 NO-PREREQ            VALUE 'N' .
03 POSTREQ-FLG               PIC X .
   88 YES                  VALUE 'Y' .
   88 NO-POSTREQ           VALUE 'N' .
03 JOB-RUN-CYCLE             PIC X .
   88 DAILY                VALUE 'D' .
   88 WEEKLY               VALUE 'W' .
   88 MONTHLY              VALUE 'M' .
   88 QUARTERLY            VALUE 'Q' .
   88 YEARLY               VALUE 'Y' .
   88 ON-REQUEST           VALUE 'O' .
   88 SPECIAL              VALUE 'S' .
03 REPORT-TITLE              PIC X(20)
   OCCURS 5 .
03 SPECIAL-RESTART-INSTR      PIC X(75) .
03 FILLER                     PIC X(15) .
03 USER-ID-LAST-CHG           PIC X(8) .
03 LAST-CHG-DATE              PIC S9(9) COMP-3 .
```

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain CICS VSAM Transparency in one of these ways:

- Using a 3270 emulator logged on to CICS
- Using a 3270 emulator logged on to TSO
- Using a 3270 emulator as an MVS™ system console

IBM Personal Communications provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need for CICS VSAM Transparency.

Notices

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 in 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 publication 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.

Licensees of this program who want 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 United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Readers' Comments — We'd Like to Hear from You

CICS VSAM Transparency for z/OS
Version 2 Release 1
Data Reengineering and Customization Guide

Publication No. SC34-7250-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44 1962 816151
- Send your comments via email to: idrcf@uk.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-7250-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP189)
Hursley Park
Winchester
Hampshire
United Kingdom
SO21 2JN

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

SC34-7250-00