docsys (en) 15 August 2016

Databridge
Host Programmer's Reference

Version 6.5

Legal Notices

Patents

This Micro Focus software is protected by the following U.S. patents: 6983315,7571180, 7836493, 8332489, and 8214884.

Trademarks

Third-Party Notices

Third-party notices, including copyrights and software license texts, can be found in a 'thirdpartynotices' file located in the root directory of the software.

Contents

About This Guide

This preface covers the following sections:

Audience

To use the Databridge application programming interface (API), you must have ALGOL programming experience. In addition, you should be thoroughly familiar with the following:

- Standard operations for Unisys® MCP-hosted servers
- Data Management System II (DMSII) databases and Data And Structure Definition Language (DASDL)
- Transferring files between your host and the system that uses the replicated DMSII data
- The Databridge host software

This guide does not define DBEngine or DBSupport, and it does not explain how Databridge operates. See the *Databridge Host Administrator's Guide* for details about Databridge operations.

Conventions

The following conventions and terms may be used in this guide.

This convention	Is used to indicate this
menu > sub menu 1 > sub menu 2 ... > menu item (item)	This font style/color shows mouse-clicks in the order required to access a specific function, window, dialog box, etc. The greater than symbol > indicates the next item to click in the series. The parentheses ( ) indicate the setting, option, or parameter being discussed. Note the font style reverts back to normal.
`this type style`	text that you type, filenames and directory names, onscreen messages
italic	variables, emphasis, document titles
square brackets ([ ])	optional items in a command. For example, [true | false]. (Do not type the brackets.) Buttons. For example, [OK], [Start], [Cancel]
pipe (|)	a choice between items in a command or parameter. When enclosed in braces ({ }), the choice is mandatory.
UPPERCASE	DMSII data set and data item names, buttons to click. For example, CANCEL, OK, APPLY.

This convention	Is used to indicate this
CAUTION:	The Caution note indicates that there is a possibility of losing data or corrupting files. When you see a Caution note, follow the instructions carefully.

This term	Is used to indicate this
MCP server host mainframe	Unisys MCP server
DBEngine	Databridge Engine
DBEnterprise	Databridge Enterprise Server
DBServer	Databridge Server

Abbreviations

The following abbreviations are used throughout this guide and are provided here for quick reference.

Abbreviation	Name
AA	Absolute Address
ABSN	Audit Block Serial Number
AFN	Audit File Number
API	Application Programming Interface
DASDL	Data and Structure Definition Language
DMSII	Data Management System II
IDX	Index
IPC	Inter-Process Communications
MCP	Master Control Program
RPC	Remote Procedure Call
RSN	Record Serial Number
SEG	Segment
WFL	Work Flow Language

Related Documentation

The following is a list of the documentation you might need to consult when using Databridge API:

Databridge product documentation	On the Databridge installation image, the DOCS folder contains guides for installation, error codes, and administrator's guides for each Databridge product. These documents require Adobe Reader for viewing, which you can download from the Adobe website (http://get.adobe.com/reader/). This documentation, and current technical notes, is also available on the Micro Focus Databridge support site (http://support.attachmate.com/manuals/databridge.html). Documentation for Databridge Enterprise Server and the Databridge Client Console is also available from the **Help** menu. A modern browser is required for viewing this documentation.
Unisys MCP server documentation	If you are not completely familiar with DMSII configuration, refer to your Unisys documentation.

1 Databridge API

In This Chapter

This chapter explains how you can use the Databridge API and provides an overview of it.

Databridge API Description

The Databridge API provides access to DBEngine and DBSupport to retrieve structural information, layout information, and data from audit files and a DMSII database.

All Databridge Accessories use the Databridge API. You can use the Databridge API to do any of the following:

- Write an Accessory (program) that calls DBEngine to perform cloning or tracking.
- Write an Accessory to retrieve the layout information for a DMSII database.
- Populate a virtual data set.

  See Chapter 3, “Virtual Data Sets,” on page 87 for information about virtual data sets.
- Reformat data items in an ALTERed data set.

  See Chapter 4, “Altered Data Sets,” on page 111 for information about ALTERed data sets.
- Write a formatting procedure to customize the format in which Databridge outputs data set records and use those custom formats with DBSpan, DBSnapshot, or a user-written Databridge Accessory.

  See Chapter 5, “Formatting Procedures,” on page 129 for information about formatting procedures.
- Write an error handling routine to analyze, log, and display errors and determine how Databridge Accessories respond to those errors.

  See Chapter 6, “Error Handling Routines,” on page 137 for information about error handling routines.

2 Using the Databridge API

In This Chapter

This chapter explains how to use the Databridge ALGOL API and provides information about the entry points and values that the API contains.

Databridge API Overview

Using the API, you can write an Accessory that uses entry points to request information from DBEngine or DBSupport. This information is usually structural and layout information about a DMSII database and data from the database and the audit trail. In addition, the DBSupport entry points can filter and format the data you request. To see a list of entry points grouped by their functions, read Entry Point Procedure Values.

The ALGOL API file (SYMBOL/DATABRIDGE/INTERFACE) contains all of the definitions an Accessory needs to call entry points in DBEngine or DBSupport. This file is installed with the Databridge Host software, and it includes brief descriptions of the expected parameters and the constants specific to DBEngine and DBSupport.

Databridge must be installed on your host before you can use the Databridge API. (If it is not, see the *Databridge Installation Guide* for instructions on installing Databridge.) Locate where Databridge has been installed and make sure it is visible to your Accessory, based on standard host security.

Information about the library entry points is divided into two sections in this chapter—one for DBEngine entry points and one for DBSupport entry points. Each of these sections include the following:

- Reference tables that list and briefly describe the DBEngine or DBSupport entry points as follows:
  - DBEngine Entry Points
  - DBSupport Entry Points
- Separate sections describing each entry point in detail

Sample Accessories

For sample Accessories that illustrate the API, see the following files installed with the Host software. Each sample Accessory also has an associated WFL job, WFL/DATABRIDGE/SAMPLE/*programname* for each sample Accessory that includes the necessary file equation.

- SYMBOL/DATABRIDGE/SAMPLE/SQLGEN

  This sample illustrates one way of generating structured query language (SQL) CREATE TABLE statements to build a relational database similar to a DMSII database. The DBSQLGen sample Accessory reads a DMSII database DESCRIPTION file and then generates SQL CREATE TABLE statements to build a relational database similar to the DMSII database.
- SYMBOL/DATABRIDGE/SAMPLE/DASDLGEN

  This sample generates a DASDL source for the data sets and sets in a given DESCRIPTION file. It does not provide the physical attributes, audit file attributes, parameters, etc.

- SYMBOL/DATABRIDGE/SAMPLE/AUDITCLOSE

  This sample causes an audit switch by closing the current audit file and switching to the next audit file. When you run this program, file-equate the DASDL to the title of the DESCRIPTION file. The following example shows how to do this for the BANKDB database description file:

```
RUN OBJECT/DATABRIDGE/SAMPLE/AUDITCLOSE;
FILE DASDL = DESCRIPTION/BANKDB;
```

- SYMBOL/DATABRIDGE/SAMPLE/COBOLGEN

  This sample generates a COBOL source for the data sets in a given DESCRIPTION file.

- SYMBOL/DATABRIDGE/SAMPLE/READDOC

  This sample reports on the programs and database events contained in the audit trail. When you run this program, file-equate the DASDL to the title of the DESCRIPTION file. The following example shows how to do this for the BANKDB database description file:

```
RUN OBJECT/DATABRIDGE/SAMPLE/READDOC;
FILE DASDL = DESCRIPTION/BANKDB;
```

Entry Point Procedure Values

All entry points are a specific type of procedure based on the kind of value returned. Procedure types can be one of the following:

Type	Description
BOOLEAN	BOOLEAN procedures return a value of TRUE or FALSE. See Boolean Callback Procedures for a description of BOOLEAN procedures and Callback Return Values for the meaning of these values.
DBMTYPE **NOTE:** The Databridge API defines DBMTYPE to be REAL.	DBMTYPE procedures return real numbers as their values, and the numbers correspond to the Databridge error and status messages. These numbers are listed in the API file. DBMTYPE values indicate the following: ◆ Success (for example, DBM_OK = 0) ◆ Action (for example, DBM_COMMIT = 1) ◆ Failure (for example, DBM_BAD_DSNAME = 5) Most of the entry points return DBMTYPE values. For a description of the messages associated with these values, see the *Databridge Host Administrator's Guide*. To retrieve a message that describes the DBMTYPE value, call the DBMESSAGE entry point in DBEngine. You can then display this message or write it to a log file.
EMATYPE	EMATYPE (error manager) procedures return EMATYPE values that indicate how a Databridge Accessory should handle errors EMATYPE procedures use AIDTYPE values, which are ID numbers that identify Databridge Accessories. See Error Manager Types for a description of EMATYPE and AIDTYPE values.

Using the DBMTYPE Values

One way to use DBMTYPE values is to program your Accessory to respond based on the DBMTYPE value returned. Following is an example of how you can do this.

NOTE: You must include SYMBOL/DATABRIDGE/INTERFACE before you can use DBMTYPE values in your Accessory. See Accessing the DBEngine and DBSupport Libraries for a description of how to include this file.

```
DBMTYPE DBRESULT;
  .
  .
  .
DBRESULT := engine_entry_point (parameter_1, . . . parameter_n);
IF DBRESULT NEQ DBM_OK THEN
  .
  .
  .
```

Boolean Callback Procedures

Many of the Databridge API entry points return a list or series of items, such as a list of data sets or a series of records. To accommodate these returned lists, Databridge uses callback procedures. The purpose of the callback procedure technique is to allow your Accessory to manipulate data, item by item, as it is returned from an entry point.

A callback procedure is a Boolean procedure that you name, declare, and write in your program. The callback procedures are referenced in the Databridge API, but they exist in your Accessory. You name the callback procedures whatever you prefer, and you define what the callback procedure does (for example, print, display, and so on). However, the Databridge API file determines the types of parameters the callback procedure receives; the callback procedure in your Accessory determines what to do with the received data. Therefore, you must write your callback procedure to accommodate the parameter values returned to it from the DBEngine entry point.

The Databridge API entry points pass values to the callback procedure based on the type of data that the API entry point retrieved. The Databridge API references your callback procedure as a formal parameter called `CALLBACK`. The program you are writing must supply the actual parameter for CALLBACK as it is defined in your program.

How Callback Procedures Work

Before you can call a Databridge entry point that has a callback procedure as a parameter, you must declare a Boolean procedure with the same parameter list as the callback procedure. The types of the parameters must match what is in the API, but the parameter names can be whatever you want. This Boolean callback procedure returns TRUE if the entry point should continue to return more items from the list. It returns FALSE to discard the rest of the list.

For example, suppose you want to enumerate the keys of a set using the entry point DBKEYS. The second parameter to DBKEYS is a callback procedure, which the API defines as follows:

```
boolean procedure Callback (ItemNum, DESCENDING);
   % Input: procedure to call back for each key item

        value   ItemNum, DESCENDING;
        integer ItemNum;
                % Input: item number
                %       (as in ITEM_INFO [II_ITEM_NUM])

   boolean DESCENDING;
     % Input: true if item is a descending key

   formal;
```

You would define a procedure matching that declaration, such as the following:

```
boolean procedure GetKey (KeyItemNbr, IsDescending);
   value   KeyItemNbr, IsDescending;
   integer KeyItemNbr;
   boolean IsDescending;
begin
   if IsDescending then
     display ("Key #" !! string (KeyItemNbr, *) !! " down")
   else
     display ("Key #" !! string (KeyItemNbr, *) !! " up");
GetKey := true;
end GetKey;
```

The program can then call the entry point, passing it the callback procedure. This example would use the following call:

```
if DBKEYS (SetStrNum, GetKey) NEQ DBM_OK then
   ... % an error occurred
```

This example callback procedure, GetKey, is called once for each of the key items of the set indicated by SetStrNum.

When you call an entry point with a callback procedure, the program follows this general sequence:

1. The program calls the entry point, passing it the callback procedure name as well as any other required parameters.
2. The entry point that your program calls then prepares the data it retrieves for a single item and calls the callback procedure in your program.
3. When your callback procedure exits (or finishes), control returns to the entry point.
4. The entry point retrieves and prepares the data for the next item and calls the callback procedure in your program.
5. Steps Step 2 on page 18–Step 4 on page 18 are repeated until there are no more items in the list. The entry point exits, and control returns to your program.

If the Databridge entry point retrieves 100 items, the callback procedure will be called 100 times.

Callback Return Values

Callback procedures return Boolean values as follows:

- TRUE—Continue calling the callback procedure
- FALSE—Stop calling the callback procedure. The entry point will typically return a `DBM_CB_CANCEL` result in this case.

DBEngine Entry Points That Use Callbacks

The following DBEngine entry points use the procedure callbacks:

- DBDATASETINFO
- DBDATASETS
- DBDIRECTORYSEARCH
- DBKEYINFO
- DBKEYINFOREMAP
- DBKEYS
- DBKEYSREMAP
- DBLAYOUT
- DBLINKS
- DBREAD
- DBREADTRANGROUP
- DBSETS
- DBSETSINFO
- DBSUBSETSINFO
- DBWAIT

DBSupport Entry Points That Use Callbacks

The following DBSupport entry points use procedure callbacks:

- DBFILTEREDDATASETS
- DBFILTEREDLAYOUT
- DBFILTEREDLINKS
- DBFILTEREDSETS
- DBFILTEREDSUBSETSINFO
- DBFILTEREDSETSINFO
- DBFILTEREDWRITE
- DBFORMAT
- DBPRIMARYKEY
- DBTRANSFORM

Accessing the DBEngine and DBSupport Libraries

Different options exist for accessing the libraries, depending on whether you are using entry points from one library or both.

Requirements for Both Libraries

Each ALGOL Accessory you write to access the DBEngine or DBSupport library must do the following:

- Set the appropriate $INCLUDE_ENGINE or $INCLUDE_SUPPORT options before including the interface. Be sure to include both if you are using entry points from both libraries.
- Include SYMBOL/DATABRIDGE/INTERFACE using the ALGOL $INCLUDE statement.
- Call the appropriate entry point to verify that your program was compiled against the same API file (SYMBOL/DATABRIDGE/INTERFACE) as DBEngine, and if applicable, DBSupport. See the table below for details.

If you are using	Then use this entry point
DBEngine only	DBINTERFACEVERSION or DBVERSION
DBEngine and DBSupport or DBSupport only	DBSUPPORTINIT

- Meet the requirements listed later in Additional DBEngine Requirements and, if applicable, in Additional DBSupport Requirements.

The following example shows the interface file $INCLUDE statement:

```
$SET INCLUDE_ENGINE
$INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
```

or

```
$SET INCLUDE_ENGINE INCLUDE_SUPPORT
$INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
```

Additional DBEngine Requirements

In addition to the requirements listed previously, you must also do the following to access DBEngine:

- Invoke the DBLINKENGINE define to link to DBEngine.
- If you are not using DBSupport, then call DBINTERFACEVERSION to verify that your program was compiled with the same version of the DBInterface as DBEngine was.
- Call the DBEngine entry point DBINITIALIZE, passing the title of the DMSII DESCRIPTION file to use.

Additional DBSupport Requirements

In addition to the requirements listed previously, you must also do the following to access DBSupport:

- Put the title of the DBSupport library in a string variable and call DBSupportTitle. For example:

```
DBMTYPE DMR;
string SupportTitle;
SupportTitle := "OBJECT/DATABRIDGE/SUPPORT/BANKDB";
DMR := DBSupportTitle (SupportTitle);
```

- Call DBSupportInit to do the following:
  - Verify your program was compiled with the same version of the DBInterface as DBEngine and DBSupport were

- Specify the names of the filter, format, and transform. For example:

```
if DBSupportInit (DBV_VERSION, "MyAccessory:",
      "MYFILTER", "COMMAFORMAT", "DBTRANSFORM") NEQ DBM_OK then
            begin
            display ("Interface version mismatch");
            end;
```

  These filter, format, and transform names can be ones you created or ones predeclared in DBSupport. Refer to the *Databridge Host Administrator's Guide* for more information about DBGenFormat.

Accessing DBEngine Only

The following example shows how to access DBEngine only (and not DBSupport). The declarations for many of the variables used in this example are not shown.

```
$ SET INCLUDE_ENGINE$
$ INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"

BOOLEAN PROCEDURE INITIALIZE;
%                 ----------
      BEGIN
      FILE    DASDL (DEPENDENTSPECS);  % for file-equating only
      FILE    DB;

      POINTER P;

      DBLINKENGINE;

      IF DBVERSION NEQ DBV_VERSION THEN
              BEGIN
              SetMsgParam1(DBVERSION, *);
              SetMsgParam2 (DBV_VERSION, *);
              DIE (DBM_VERSION_MISMATCH, MsgParam1c MsgParam2);
              END;

       REPLACE P:FILETITLE BY DASDL.TITLE;
       DASDLTITLE := STRING (FILETITLE, OFFSET (P) - 1);


       IF DB.FILEEQUATED THEN
               BEGIN
               REPLACE P:FILETITLE BY DB.FILENAME;
               DBNAME := STRING (FILETITLE, OFFSET (P) - 1);
               END;

      WRITE_IF_ERR (DBINITIALIZE (DASDLTITLE, DBNAME));

      INITIALIZE := NOT DONE;  % no fatal errors

      END INITIALIZE;
```

DBEngine Entry Points

Use the DBEngine entry points to request information from DBEngine.

The table below summarizes the DBEngine entry points and their functions, and each of these points is explained in detail later in this chapter. The Entry Point column in this table contains the name of the entry point, the Type column indicates what type of procedure the entry point is, and the Description column describes what the entry point does. See Entry Point Procedure Values for an explanation of the various procedure types and the values they return. DBMTYPE values are listed in the API file (SYMBOL/DATABRIDGE/INTERFACE) and the Databridge Host Administrator's Guide.

Entry Point	Type	Description
DBATTRIBUTE	DBMTYPE	Returns a specified file attribute value of a disk file
DBAUDITATTRIBUTE	DBMTYPE	Returns a specified file attribute value of an audit file
DBAUDITMEDIUM	DBMTYPE	Specifies where DBEngine looks for audit files
DBAUDITPACK	DBMTYPE	Specifies an alternate audit pack location
DBAUDITPREFIX	DBMTYPE	Specifies a non-standard file name prefix for audit files.
DBAUDITSOURCE	DBMTYPE	Specifies where to access remote audit files
DBAUDITSOURCEX	DBMTYPE	Specifies how to access remote audit files
DBCANCELWAIT	DBMTYPE	Cancels the wait (set by DBWAIT) for audit files to become available
DBCLOSEDATASET	DBMTYPE	Closes a dataset previously opened with DBOpenDataset.
DBCOMMENT	DBMTYPE	Copies the DASDL comment associated with a structure or data item into the caller's array
DBCOMPILESUPPORT	DBMTYPE	Compiles the DBSupport library
DBDATABASEINFO	DBMTYPE	Returns information about the database, such as the update level and timestamp
DBDATASETINFO	DBMTYPE	Returns layout information about the data set or remap as described in the DATASET_INFO array
DBDATASETNAME	DBMTYPE	Returns the structure name for the specified structure index
DBDATASETNUMS	DBMTYPE	Returns an array containing the structure numbers of all the data sets and remaps in the logical database
DBDATASETS	DBMTYPE	Lists the data set names, structure numbers, and other information about each data set
DBDATASETVFINFO	DBMTYPE	Loads the DATASET_INFO array with information about the data set, remap, or set
DBDATETIME	DBMTYPE	Converts a timestamp from TIME (6) format to a binary format time and date
DBDESELECT	DBMTYPE	Deselects a data set previously selected with DBSELECT so that it is not processed by subsequent DBREADs or DBWAITs

Entry Point	Type	Description
DBDIRECTORYSEARCH	DBMTYPE	Returns the file names within a specified directory and its subdirectories
DBDISPLAYFAULT	DBMTYPE	Displays a message describing a program fault, such as INVALID INDEX
DBDISPLAYMSG	DBMTYPE	Displays the error message associated with the result code returned from the call to the previous entry point
DBENGINEMISSINGENTRYPOINT	STRING	Returns the name of the first entry point missing from the library code file that the Accessory expected to be present based on the interface file
DBFAMILYINFO	DBMTYPE	Returns the date, time, and system serial number when the family was created.
DBFILEATTRIBUTE	DBMTYPE	Allows an Accessory to retrieve file attribute information about any file
DBGETFIRSTQPT	DBMTYPE	Finds the first quiet point in the audit trail beginning with the Audit File Number and Audit Block Serial Number given in STATE_INFO. STATE_INFO is updated to reflect the audit location of the quiet point.
DBINITFILTER	DBMTYPE	(not used)
DBINITIALIZE	DBMTYPE	Initializes DBEngine by specifying the title of the database DESCRIPTION file
DBINTERFACEVERSION	DBMTYPE	Validates the Accessory's DBInterface version against DBEngine's DBInterface version and returns an error if they are incompatible
DBIOERRORTEXT	DBMTYPE	Copies the error text describing the READ/WRITE result value into the caller's array
DBIORESULTTEXT	DBMTYPE	Copies the error text describing the I/O result value into the caller's array **NOTE:** This entry point is now called DBOPENRESULTTEXT.
DBITEMINFO	DBMTYPE	Returns information for a data item in a data set
DBITEMNUMINFO	DBMTYPE	Retrieves information about a single data item
DBKEYDATAREMAP	DBMTYPE	Enumerates the items of the KEY DATA for a set using the item descriptions of the designated data set or remap
DBKEYINFO	DBMTYPE	Returns information for each key item in a set
DBKEYINFOREMAP	DBMTYPE	Enumerates key items in a set using item information in a remap
DBKEYS	DBMTYPE	Lists the key items in the specified set
DBKEYSREMAP	DBMTYPE	Enumerates key items in a set using the item numbers of the specified data set of a remap
DBLAYOUT	DBMTYPE	Lists the data items in the specified data set
DBLIMITTASKNAME	DBMTYPE	Sets the processing limit task name

Entry Point	Type	Description
DBLIMITTIMESTAMP	DBMTYPE	Sets the processing limit timestamp
DBLINKS	DBMTYPE	Enumerates LINK items in a data set
DBMAKETIMESTAMP	DBMTYPE	Converts a date and time to a timestamp in TIME (6) format
DBMAXRECORDS	DBMTYPE	Returns the estimated maximum number of records currently in a data set
DBMAXRECORDSVF	DBMTYPE	Returns the estimated maximum number of records currently in a data set
DBMESSAGE	DBMTYPE	Copies the error message associated with a DBMTYPE value to the caller's array
DBMODIFIES	DBMTYPE	Specifies whether data set record updates should be returned as a DELETE/CREATE pair or as an update
DBMODIFYTIMESTAMP	DBMTYPE	Increments or decrements a timestamp by days, hours, minutes, and/or seconds
DBNULL	DBMTYPE	Returns the NULL value for the specified data item
DBNULLRECORD	DBMTYPE	Returns a record with all data items NULL
DBOLDESTAUDITLOC	DBMTYPE	Finds the oldest audit location on disk
DBOPENAUDIT	DBMTYPE	Opens an audit file and returns audit file information
DBOPENRESULTTEXT	DBMTYPE	Copies the error text describing the OPEN/CLOSE/RESPOND result value into the caller's array
DBPARAMETERS	DBMTYPE	Specifies various run-time processing parameter values
DBPRIMARYSET	DBMTYPE	Returns the structure number of the NO DUPLICATES set having the fewest key items for the given data set
DBPRIVILEGED	BOOLEAN	Indicates whether or not the caller is a privileged program or running under a privileged usercode
DBPUTMESSAGE	DBMTYPE	Sets the DBMESSAGE parameter values
DBREAD	DBMTYPE	Receives a transaction group (up to the next quiet point) of changes to data set records from the audit trail
DBREADAUDITREGION	DBMTYPE	Reads the audit file region, starting with the indicated audit block serial number (ABSN) and block offset
DBREADERPARAMETER	DBMTYPE	Allows an Accessory to specify the title of the FileXtract Reader library and the parameter string that is passed to the Reader library
DBREADTRANGROUP	DBMTYPE	Receives a transaction group (up to the next quiet point) of changes to data set records from the audit trail. If a transaction group is not available, it waits for a specified number of seconds before retrying.
DBRESETOPTION	DBMTYPE	Turns off DBEngine run-time options

Entry Point	Type	Description
DBSELECT	DBMTYPE	Selects a data set to be processed by a subsequent DBREAD or DBWAIT, and validates the client format level when a data set is selected for cloning
DBSELECTED	DBMTYPE	Checks to see if the specified data set has been selected
DBSETINFO	DBMTYPE	Returns information describing a set as given in the SET_INFO array
DBSETOPTION	DBMTYPE	Turns on DBEngine run-time options
DBSETS	DBMTYPE	Lists the names, structure numbers, and other information for the sets of the specified data set
DBSETSINFO	DBMTYPE	Returns information for each set of a given data set
DBSPLITTIMESTAMP	DBMTYPE	Converts a timestamp from TIME (6) format to a date and time in *yyyy, mm, dd, hh, mn, ss* form
DBSPLITTIME60	DBMTYPE	Splits a timestamp in TIME (60) format into separate components
DBSTATEINFOTODISPLAY	INTEGER	Converts state information into readable format
DBSTATISTICS	DBMTYPE	Returns statistics for the specified category **NOTE:** To accumulate statistics, DBEngine must be compiled with $ SET STATS, which is available as OBJECT/DATABRIDGE/ENGINE/STATS.
DBSTRIDX	DBMTYPE	Returns the structure index for a selected data set
DBSTRNUM	DBMTYPE	Returns the structure number for the specified structure name
DBSTRUCTURENAME	DBMTYPE	Returns a structure name for a DMSII structure
DBSUBSETSINFO	DBMTYPE	Enumerates information for each subset of a given data set
DBSWITCHAUDIT	DBMTYPE	Forces an audit file switch
DBTIMESTAMPMSG	DBMTYPE	Converts the timestamp from TIME (6) format to the following form: *month, day, year @ hh*:*mm*:*ss*
DBUPDATELEVEL	DBMTYPE	Returns the database update level and timestamp
DBVERSION	REAL	Returns the version number of the API file (SYMBOL/DATABRIDGE/INTERFACE) that DBEngine was compiled against.
DBWAIT	DBMTYPE	Receives the transaction group (up to the next quiet point) of changes to data set records from the audit trail If a transaction group is not available, it waits for a specified number of seconds before retrying.
DBWHEREDASDL	DBMTYPE	Returns the DASDL source expression associated with the WHERE clause of an automatic subset or the SELECT cause of a remap

Entry Point	Type	Description
DBWHERETEXT	DBMTYPE	Returns the ALGOL source code fragment associated with the WHERE clause of an automatic subset or the SELECT clause of a remap

DBATTRIBUTE

This entry point returns a specified file attribute value of a disk file. Contact Micro Focus for additional information.

DBAUDITMEDIUM

This entry point allows an Accessory to specify where DBEngine looks for audit files (on disk or tape) and whether to look for primary, secondary, or both.

Declaration

```
DBMTYPE procedure DBAuditMedium (AuditMedium,AuditType);
```

Input	Type	Definition
AUDITMEDIUM	INTEGER	The medium on which to look for the audit file, such as a disk or tape Possible values are as follows: ◆ DBV_AM_ORIGPACK—Tells the Accessory to look on the original pack(s) ◆ DBV_AM_TAPE—Tells the Accessory to look on the COPYAUDIT tapes ◆ DBV_AM_ALTERNATE—Tells the Accessory to look on the alternate pack as specified by DBAUDITPACK If this parameter is invalid, DBEngine returns a DBM_BAD_AUDMED (110) error.

Input	Type	Definition
AUDITTYPE	INTEGER	The type of audit file for which to look, such as primary or secondary Possible values are as follows: ◆ DBV_AM_NEITHER—Tells the Accessory not to look for audit files on the source ◆ DBV_AM_PRIMARY—Tells the Accessory to look only for the primary audit file ◆ DBV_AM_SECONDARY—Tells the Accessory to look only for the secondary audit file ◆ DBV_AM_BOTH—Tells the Accessory to look for both the primary and secondary audit files See DBAUDITMEDIUM Parameters for more information about these values. If this parameter is invalid, DBEngine returns a DBM_BAD_AUDTYPE (111) error.

DBAUDITATTRIBUTE

This entry point returns a specified file attribute value of an audit file, if the specified attribute number is VALUE (TITLE), VALUE (FILENAME), or VALUE (FAMILYNAME).

This function looks for the audit file first on the Alternate pack and then on the original audit pack. This allows DBEnterprise to access audit files in either location. Contact Micro Focus for additional information.

DBAUDITPACK

This entry point specifies where audit files may be located. DBEngine looks on the pack specified by the DMSII control file, the pack specified by DBAUDITPACK, or both, depending on DBAUDITMEDIUM.

Declaration

```
DBMTYPE procedure DBAUDITPACK (Packname);
```

Input	Type	Definition
PACKNAME	STRING	The name of the pack where DBEngine should look for normal DMSII audit files

DBAUDITPREFIX

This entry point can specify a non-standard file name prefix for audit files. Normally, the prefix for audit files is:

(*databaseusercode*)*databasename*

Using the DBAuditPrefix entry point, the prefix can have a different usercode, different first node, and/or additional nodes.

This entry point specifies where audit files may be located. DBEngine looks on the pack specified by the DMSII control file, the pack specified by DBAUDITPACK, or both, depending on DBAUDITMEDIUM.

If the prefix is badly formed, the entry point will return DBM_BAD_PREFIX (137) (Invalid audit file prefix: '*prefix*')

DBAUDITSOURCE

This entry point specifies where to access remote audit files.

Declaration

```
DBMTYPE procedure DBAuditSource (Host, SourceName, Protocol, Port);
```

Input	Type	Definition
HOST	STRING	The name or the IP address of the DBServer host where the audit files are located
SOURCENAME	STRING	A source identifier that is the name of a SOURCE in the DBServer parameter file
PROTOCOL	INTEGER	The network protocol value, such as TCPIP See Network Protocol Values for possible values.
PORT	INTEGER	The port number that matches PORT in the DBServer parameter file

DBAUDITSOURCEX

This entry point specifies how to access remote audit files. This entry point is identical to DBAUDITSOURCE except that the PORTNAME is a string rather than a number to allow names for BNA and HLCN ports.

Declaration

```
DBMTYPE procedure DBAuditSourceX (Host, SourceName, Protocol, PortName);
```

Input	Type	Definition
HOST	STRING	Name or IP address of the DBServer host where audit files are located
SOURCENAME	STRING	The name of a SOURCE in the DBServer parameter file
PROTOCOL	INTEGER	Network protocol values, such as TCPIP See the DBV_NET_xxx values in Network Protocol Values.
PORTNAME	STRING	The port name, such as 3000 or BNA350 This value must match the PORT in the DBServer parameter file.

DBCANCELWAIT

This entry point cancels the wait for more audit files to become available via DBWAIT or DBREADTRANGROUP.

Declaration

```
DBMTYPE procedure DBCANCELWAIT;
```

DBCLOSEDATASET

This entry point closes a dataset previously opened with DBOpenDataset. Contact Micro Focus for additional information.

DBCOMMENT

This entry point copies the DASDL comment associated with a structure or data item into the caller's array. These comments must have been declared in the DASDL using the double-quote Declaration. For example:

```
ACCT-YTD-INT "year-to-date interest" NUMBER (11, 2);
```

Declaration

```
DBMTYPE procedure DBComment (StrNum, ItemNum, pText, Len);
```

Input	Type	Definition
STRNUM	REAL	The structure number of the structure you are requesting Since no comment can be associated with the global record, a structure number of 1 is invalid.
ITEMNUM	REAL	The item number of the data item you are requesting Use 0 to request the comment associated with a data set, set, or remap.
PTEXT	POINTER	Destination for the comment text

Output	Type	Definition
LEN	REAL	The length, in bytes, of the text copied into PTEXT Possible values are as follows: ◆ If the array is too short, no text is copied, but LEN is set to the needed length, and the procedure value is DBM_SHORT_ARRAY (23). ◆ If no text is associated with the structure, such as when the structure number is a data set, set, or manual subset, the procedure returns DBM_OK, and LEN is set to 0.

DBCOMPILESUPPORT

This entry point compiles the DBSupport library. If an Accessory determines that DBSupport needs to be compiled with the current DESCRIPTION file, it can call this entry point. Any local patches to DBSupport are ignored if they are not specified in the DBGenFormat parameter file.

If DBEngine determines that a DBSupport library already exists for the desired update level, it copies the new title of the DBSupport library into the caller's array and then returns immediately without actually compiling it.

Declaration

```
DBMTYPE procedure DBCompileSupport (pTitle);
```

Input	Type	Definition
PTITLE	POINTER	The pointer to the title of the DBSupport library

Output	Type	Definition
PTITLE	VALUE	The new title of the DBSupport library (the update level node to be appended to the file name)

DBDATABASEINFO

This entry point returns layout information about the database, such as the update level and update timestamp.

Declaration

```
DBMTYPE procedure DBDATABASEINFO (DATABASE_INFO);
```

Output	Type	Definition
DATABASE_INFO	ARRAY	An array of information that describes the database For a description of the array, see DATABASE_INFO Layout.

DBDATASETINFO

This entry point returns layout information about a data set or remap. If the data set has any links, DBDATASETINFO sets the DATASET_INFO [DI_LINKS] = 1.

Declaration

```
DBMTYPE procedure DBDATASETINFO (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap
CALLBACK	BOOLEAN	The procedure that receives the data set information

BOOLEAN PROCEDURE CALLBACK

This is the procedure that receives information about the data set. For fixed?format data sets, this procedure is called once. For variable?format data sets, this procedure is called once for each format type.

Declaration

```
boolean procedure Callback (pDatasetName, Len, DATASET_INFO);
```

Parameter	Type	Definition
P_DATASETNAME	POINTER	The pointer to a data set name The caller must copy the actual data set name into its own local memory.
LEN	REAL	The length of the data set name
DATASET_INFO	ARRAY	Information about the data set For a description of the array, see DATABASE_INFO Layout.

DBDATASETNAME

This entry point returns a data set name corresponding to the specified structure index.

Declaration

```
DBMTYPE procedure DBDATASETNAME (StrIdx, pDSName,Len);
```

Input	Type	Definition
STRIDX	REAL	The structure index from UPDATE_INFO [UI_STRIDX]
P_DSNAME	POINTER	The pointer to the array that is to receive the data set name

Output	Type	Definition
LEN	REAL	The length of the data set name in bytes

DBDATASETNUMS

This entry point returns an array that contains the structure numbers of all of the data sets (except virtual data sets) and remaps in the logical database.

Declaration

```
DBMTYPE procedure DBDatasetNums (DSNums, LastIdx);
```

Output	Type	Definition
DSNUMS	ARRAY	Contains the structure numbers of all data sets (except virtual data sets) and remaps in the (logical) database
LASTIDX	REAL	The subscript of the last valid entry in the DSNUMS array (above) If the array is too small, it is automatically resized to the appropriate size.

DBDATASETS

This entry point provides data set names and their structure numbers. Use DBDATASETS to generate a pointer to a data set name, the data set name length, and an array that contains information about the data set, such as its structure number.

Declaration

```
DBMTYPE procedure DBDATASETS (Callback);
```

Input	Type	Definition
CALLBACK	BOOLEAN	The procedure that receives data set information

BOOLEAN PROCEDURE CALLBACK

For fixed-format data sets, DBDATASETS calls this procedure once. For variable-format data sets, DBDATASETS calls this procedure once for each format (record) type.

Declaration

```
boolean procedure Callback (pDatasetName, Len, DATASET_INFO);
```

Parameter	Type	Definition
P_DATASETNAME	POINTER	Points to a data set name The caller must copy the actual data set name into its own local memory.
LEN	REAL	The length of the data set name
DATASET_INFO	ARRAY	Information about the data set For a description of the array, see DATABASE_INFO Layout.

DBDATASETVFINFO

This entry point loads the DATASET_INFO array with information about the data set or remap, according to the record type number.

Declaration

```
DBMTYPE procedure DBDatasetVFInfo (DSStrNum, RecType, DATASET_INFO);
```

Input	Type	Definition
DSSTRNUM	REAL	Structure number of the data set or remap
RECTYPE	REAL	Record type number (0 for fixed format)

Output	Type	Definition
DATASET_INFO	ARRAY	Information about the data set as contained in the DATASET_INFO array See DATABASE_INFO Layout for a description of the array.

DBDATETIME

This entry point converts the timestamp in TIME (6) format to a binary format date and time.

Declaration

```
DBMTYPE procedure DBDATETIME (Timestamp, YYYYMMDD, HHMMSS);
```

Input	Type	Definition
TIMESTAMP	REAL	TIME (6) timestamp

Output	Type	Definition
YYYYMMDD	INTEGER	The date in binary format
HHMMSS	INTEGER	The time in binary format

DBDESELECT

This entry point deselects a data set that was previously selected by DBSELECT so that it is not processed by subsequent DBREADs or DBWAITs.

Declaration

```
DBMTYPE procedure DBDESELECT (StrIdx);
```

Input	Type	Definition
STRIDX	INTEGER	The unique number for this data set record type that was returned in a previous call to DBSELECT

DBDIRECTORYSEARCH

This entry point returns the file names within a specified directory and its subdirectories. DBEngine calls the Accessory-supplied callback procedure with the name of each file. This function will translate the filename or directory name to UPPERCASE, avoiding error DBM0125.

Declaration

```
DBMTYPE procedure DBDirectorySearch (pDirName, FilenameHandler);
```

Input	Type	Definition

PDIRNAME	POINTER	Pointer to the period-terminated directory name
		
		NOTE: Do not include /= in the name. For example, to get a list of files in SUMLOG/1234/= ON SYSTEM, use the following:
		
		`SUMLOG/1234`
		`ON SYSTEM.`

BOOLEAN PROCEDURE CALLBACK

This procedure receives the name of a file in the specified directory and is called once for each file. If the procedure returns TRUE (no more file names are available), DBEngine aborts the search.

Declaration

```
boolean procedure FilenameHandler (pFilename, FilenameLen);
```

Parameter	Type	Definition
PFILENAME	POINTER	Pointer to a file name in the directory
FILENAMELEN	INTEGER	Length of the file name in bytes

DBDISPLAYFAULT

This entry point displays a message describing a program fault, such as INVALID INDEX, along with the line number in the application program that caused the fault.

The sample reformatting routine (SYMBOL/DATABRIDGE/SAMPLE/REFORMAT) captures and identifies errors and uses this entry point to help sites debug their reformatting routines.

Declaration

```
DBMTYPE procedure DBDisplayFault (Prefix, FaultNbr, pFaultHistory);
```

Input	Type	Definition
PREFIX	STRING	The prefix of the message to be displayed This prefix is usually the program name followed by a colon. Example: `DBSpan:`
FAULTNBR	REAL	The fault number returned by the ALGOL ON statement
PFAULTHISTORY	POINTER	The stack history returned by the ALGOL ON statement

DBDISPLAYMSG

This entry point displays the error message associated with the result code returned from the call to the previous entry point.

Declaration

```
DBMTYPE procedure DBDisplayMsg (DBMResult);
```

Input	Type	Definition
DBMRESULT	DBMTYPE	The procedure value from a prior call to a DBEngine or DBSupport entry point

DBENGINEMISSINGENTRYPOINT

This entry point returns the name of the first entry point missing from the library code file that the Accessory expected to be present based on the interface file.

Declaration

```
string procedure DBENGINEMissingEntryPoint;
```

Example

```
string   MissingEP;
...
MissingEP := DBENGINEMissingEntryPoint;
if MissingEP NEQ empty then
      display ("Missing DBEngine entry point " !! MissingEP);
```

DBFAMILYINFO

This entry point returns the date, time, and system serial number when the family was created. Contact Micro Focus for additional information.

DBFILEATTRIBUTE

This entry point allows an Accessory to retrieve file attribute information about any file. The caller supplies the file title and a mask of desired file attributes. For example, to request the creation date and time, use the following mask:

```
0 & 1 [CREATIONDATEB:1] & 1 [CREATIONTIMEB:1]
```

DBEngine returns the values in the Attributes array, indexed by the attribute bit number. For example, to reference the creation date value after calling this entry point, use the following:

```
Attributes [CREATIONDATEB]
```

Declaration

```
DBMTYPE procedure DBFileAttribute (pFileTitle, AttrMask, Attributes);
```

Input	Type	Definition
PFILETITLE	POINTER	The pointer to the period-terminated file name. This function will translate the filename or directory name to UPPERCASE, avoiding error DBM0125.
ATTRMASK	REAL	Mask of desired file attributes See File Attribute Mask Bits for a list of attributes and their corresponding bits.

Output	Type	Definition
ATTRIBUTES	ARRAY	An array containing file attribute values

DBGETFIRSTQPT

This entry point finds the first quiet point (QPT) in the audit trail beginning with the audit file number (AFN) and ABSN given in the STATE_INFO array layout. STATE_INFO is updated to reflect the audit location of the quiet point. For a description of the STATE_INFO array layout, see STATE_INFO Layout.

Declaration

```
DBMTYPE procedure DBGETFIRSTQPT (STATE_INFO);
```

Input	Type	Definition
STATEINFO [SI_AFN]	REAL	The starting AFN
STATEINFO [SI_ABSN]	REAL	The starting ABSN This ABSN does not have to exist in the specified audit file. It simply functions as a lower bound.

Output	Type	Definition
STATEINFO [SI_AFN]	REAL	The AFN of the quiet point
STATEINFO [SI_ABSN]	REAL	The ABSN of the quiet point
STATEINFO [SI_SEG]	REAL	The segment of the quiet point
STATEINFO [SI_INX]	REAL	The word index of the quiet point
STATEINFO [SI_TIME]	REAL	The timestamp of the quiet point

DBGETINFO

This entry point returns individual values corresponding to the Info_Enginexxxx values listed in DBInterface.

Declaration

```
DBMTYPE procedure DBGetInfo (InfoId, InfoSelection, InfoValue);
```

Input	Type	Definition
InfoId	INTEGER	One of the Info_Enginexxxx or Info_Sourcexxxx values in DBInterface
InfoSelection	INTEGER	Unused

Output	Type	Definition
InfoValue	STRING	Value of the requested information

DBGETOPTION

This entry point returns the value of a Boolean run-time option. The options are named DBV_OP_xxxx and listed in DBInterface.

Declaration

```
DBMTYPE procedure DBGetOption (Option, Setting);
```

Input	Type	Definition
Option	INTEGER	One of the DBV_OP_xxxx options in DBInterface

Output	Type	Definition
Setting	BOOLEAN	Value of option: true or false

DBINITFILTER

This entry point will be removed.

DBINITIALIZE

This entry point initializes DBEngine by specifying the title of the database DESCRIPTION file (without the DESCRIPTION node). You must initialize DBEngine by calling DBINITIALIZE before you can use any of the other DBEngine API entry points that access database information either directly or indirectly. See Accessing the DBEngine and DBSupport Libraries to see what else you must do before using the DBEngine entry points.

Declaration

```
DBMTYPE procedure DBINITIALIZE (DBDescTitle, DB);
```

Input	Type	Definition
DBDESCTITLE	STRING	The title of the database DESCRIPTION file (without the DESCRIPTION node) Example: `"(PROD)PAYROLLDB ON SYSPACK."`
DB	STRING	Optional. The name of the logical or physical database associated with the DESCRIPTION file (the database to which you want access) If a logical database is specified, the Accessory cannot retrieve layout or update information for any data sets or remaps outside of the specified logical database.

Output	Type	Definition
DB	STRING	If you leave this parameter empty on input, the program returns the name of the physical database. See the Input parameter definition for DB above.

DBINTERFACEVERSION

This entry point validates the Accessory's DBInterface version against DBEngine's DBInterface version and returns a DBM_VER_MISMATCH (115) error if they are incompatible.

NOTE: If you are using DBSupport, you do not need to call DBINTERFACEVERSION explicitly because your call to DBSupportInit automatically calls DBINTERFACEVERSION to validate the DBInterface version of the Accessory against DBEngine.

For related information, see DBINITIALIZESUPPORT.

Declaration

```
DBMTYPE procedure DBInterfaceVersion (AccessoryVersion, AccessoryID);
```

Input	Type	Definition
ACCESSORYVERSION	REAL	The version of DBInterface that was used to compile the Accessory (DBV_VERSION)
ACCESSORYID	STRING	A descriptive string inserted in an error message that identifies the Accessory Example: `DBSpan:`

DBIOERRORTEXT

This entry point copies error text describing the READ/WRITE result value into the caller's array. The READ/WRITE result value is a Boolean value returned from a READ or WRITE function that was passed to the entry point by the calling program.

Declaration

```
procedure DBIOErrorText (IOResult, pText, TextLen);
```

Input	Type	Definition
IORESULT	BOOLEAN	Result value from the READ or WRITE
PTEXT	POINTER	Destination for error text

Output	Type	Definition
TEXTLEN	INTEGER	Length of the error text

DBIORESULTTEXT

This entry point is now called DBOPENRESULTTEXT (see DBOPENRESULTTEXT), and it is recommended that you use DBOPENRESULTTEXT. However, for compatibility, the name DBIORESULTTEXT still works.

DBITEMINFO

This entry point returns the ITEM_INFO array layout for a data item in a data set or remap. The only difference between this entry point and DBITEMNUMINFO is that DBITEMNUMINFO specifies the data item by name rather than number.

Declaration

```
DBMTYPE procedure DBITEMINFO (DSStrNum, RecType, ItemName, ITEM_INFO);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap that contains the data item
RECTYPE	REAL	The record type number (0 for fixed format)
ITEMNAME	STRING	The name of the data item for which you are requesting the array

Output	Type	Definition
ITEM_INFO	ARRAY	The information about the data item For a description of the array, see ITEM_INFO Array Layout.

DBITEMNUMINFO

This entry point retrieves the ITEM_INFO array layout for a single data item. The only difference between this entry point and DBITEMINFO is that DBITEMINFO specifies the data item by number rather than name.

Declaration

```
DBMTYPE procedure DBITEMNUMINFO (DSStrNum, ItemNum, ITEM_INFO);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap that contains the data item
ITEMNUM	INTEGER	The number of the data item for which you want information

Output	Type	Definition
ITEM_INFO	ARRAY	The information about the data item For a description of the array, see ITEM_INFO Array Layout.

DBKEYDATAREMAP

This entry point enumerates the items of the KEY DATA for a set using the item descriptions of the designated data set or remap. If the set does not have any KEY DATA, it returns the following DBM_NO_KEYDATA (122) message:

setname does not have key data

KEY DATA is not the same as the KEY of a set. The KEY determines the order of the set entries, while KEY DATA contains additional data items that are not part of the KEY.

The item descriptions are relative to a remap if the structure number of a remap is passed as the REMAPSTRNUM parameter. Otherwise they are relative to the original data set.

If you specify a remap and one or more of the KEY DATA items are not found in the specified remap, it returns the following DBM_BAD_ITEMNUM (31) message:

```
Invalid data item number: itemnum--not found in dataset
```

Declaration

```
DBMTYPE procedure DBKeyDataRemap (SetStrNum, RemapStrNum, Callback);
```

Input	Type	Definition
SETSTRNUM	REAL	The structure number of the set whose key items are to be returned
REMAPSTRNUM	REAL	The structure number of the data set or remap that contains the KEY DATA items If REMAPSTRNUM = 0, the original data set is assumed. Item numbers can very depending on whether the data set or remap is used.
CALLBACK	BOOLEAN	The procedure that receives information for each KEY DATA item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the KEY DATA item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Output	Type	Definition
ITEM_INFO	ARRAY	The information about the data item For a description of the array, see ITEM_INFO Array Layout.

DBKEYINFO

This entry point returns the key items in a set.

Declaration

```
DBMTYPE procedure DBKEYINFO (SetStrNum, Callback);
```

Input	Type	Definition
SETSTRNUM	REAL	The structure number of the set whose key items are to be returned
CALLBACK	BOOLEAN	The procedure that receives information for each key item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the key item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	ARRAY	An array of information that describes the key item For a description of the array, see ITEM_INFO Array Layout.

DBKEYINFOREMAP

This entry point lists items in a set using item information in a remap. The key items have the item number, name, and so on, as they are known in the remap.

Declaration

```
DBMTYPE procedure DBKeyInfoRemap (SetStrNum, RemapStrNum, Callback);
```

Input	Type	Description
SETSTRNUM	REAL	The structure number of the desired set
REMAPSTRNUM	REAL	The structure number of the remap (or data set) If this value is set to 0, the original data set is assumed.
CALLBACK	BOOLEAN	The procedure that receives information for each key item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the key item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	ARRAY	An array of information describing the key item For a description of the array, see ITEM_INFO Array Layout.

DBKEYS

This entry point returns the key items in a set.

Declaration

```
DBMTYPE procedure DBKEYS (SetStrNum, Callback);
```

Input	Type	Definition
SETSTRNUM	REAL	The structure number of the set whose keys are to be returned
CALLBACK	BOOLEAN	The procedure that receives information for each key item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the key item.

Declaration

```
boolean procedure Callback (ItemNum, DESCENDING);
```

Parameter	Type	Definition
ITEMNUM	INTEGER	The item number of the key, as in ITEM_INFO [II_ITEM_NUM]
DESCENDING	BOOLEAN	Returns a value of TRUE if the item is a descending key

DBKEYSREMAP

This entry point lists key items in a set using the item numbers of the specified data set or remap. If a key item is not found in the specified remap, the procedure returns the error DBM_BAD_ITEMNUM (31).

Declaration

```
DBMTYPE procedure DBKeysRemap (SetStrNum, RemapStrNum, Callback);
```

Input	Type	Description
SETSTRNUM	REAL	The structure number of the desired set
REMAPSTRNUM	REAL	The structure number of the data set or remap containing the key items The item numbers can vary depending on whether the data set or the remap is used. If this value is set to 0, the original data set is assumed.
CALLBACK	BOOLEAN	The procedure that receives information for each key item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the key items.

Declaration

```
boolean procedure Callback (ItemNum, Descending);
```

Parameter	Type	Definition
ITEMNUM	INTEGER	The item number, as in ITEM_INFO [II_ITEM_NUM]
DESCENDING	BOOLEAN	TRUE if the item is a descending key

DBLAYOUT

This entry point returns the ITEM_INFO array layout for a data items in a data set.

Declaration

```
DBMTYPE procedure DBLAYOUT (DSStrNum, RecType, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the desired data set
RECTYPE	REAL	The record type number (0 for fixed format)
CALLBACK	BOOLEAN	The procedure that receives information for each data item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about each data item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	ARRAY	Information describing each data item For a description of the array, see ITEM_INFO Array Layout.

Example

The following example calls DBLAYOUT:

```
ARRAY ITEMSINFO [0:99, 0:II_ENTRY_SIZE - 1];
BOOLEAN PROCEDURE GETITEMINFO (DIINFO);
        ARRAY DIINFO [0];
        BEGIN
        REAL IDX;

        IDX := DIINFO [II_ITEM_NUM];
        REPLACE POINTER (ITEMSINFO [IDX, 0], 0) BY
            POINTER (DIINFO, 0) FOR II_ENTRY_SIZE_WORDS;
        GETITEMINFO := TRUE;
        END GETITEMINFO;

REAL DSSTRNUM;
DSSTRNUM := 2;
DBLAYOUT (DSSTRNUM, 0, GETITEMINFO);
```

DBLIMITTASKNAME

This entry point sets the processing limit task name. When DBREADTRANGROUP reaches the specified task name in the audit trail, it stops processing at the next quiet point after the task's database close (if the type is AFTER) or the quiet point before the task's database open (if the type is BEFORE).

Declaration

```
DBMTYPE procedure DBLIMITTASKNAME (TaskName, LimitType);
```

Input	Type	Definition
TASKNAME	STRING	The name of the update program that limits processing
LIMITTYPE	INTEGER	The type of limit, either BEFORE or AFTER the specified timestamp Possible values are as follows: ◆ DBV_LIMIT_UNSPECIFIED—Indicates that no limit type is specified ◆ DBV_LIMIT_BEFORE—Tells the Accessory to stop at the QPT before the limit ◆ DBV_LIMIT_AFTER—Tells the Accessory to stop at the QPT after the limit

DBLIMITTIMESTAMP

This entry point sets the processing limit timestamp. When DBREADTRANGROUP reaches the specified time in the audit trail, it stops processing at the next quiet point (if the type is AFTER) or the previous quiet point (if the type is BEFORE).

Declaration

```
DBMTYPE procedure DBLIMITTIMESTAMP (Timestamp, LimitType);
```

Input	Type	Definition
TIMESTAMP	REAL	The limiting timestamp in TIME (6) format
LIMITTYPE	INTEGER	The type of limit, either BEFORE or AFTER the specified timestamp Possible values are as follows: ◆ DBV_LIMIT_UNSPECIFIED—Indicates that no limit type is specified ◆ DBV_LIMIT_BEFORE—Tells the Accessory to stop at the QPT before the limit ◆ DBV_LIMIT_AFTER—Tells the Accessory to stop at the QPT after the limit

DBLINKS

This entry point returns information about link items in a data set. The DBEngine option LINKS must be true, and the data set must be STANDARD, fixed-format, and unsectioned.

Declaration

```
DBMTYPE procedure DBLinks (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the data set. This procedure is called once for each link item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	BOOLEAN	The array of information describing the link item For a description of the array, see ITEM_INFO Array Layout.

DBMAKETIMESTAMP

This entry point converts a date and time to a timestamp in TIME (6) form.

Declaration

```
DBMTYPE procedure DBMAKETIMESTAMP (Year, MM, DD, HH, MN, SS, TS);
```

Input	Type	Definition
YEAR	INTEGER	The year in four digits, 1970–2035
MM	INTEGER	The month in two digits, 1–12
DD	INTEGER	The day in two digits, 1–31
HH	INTEGER	The hour in two digits, 0–23
MN	INTEGER	The minute in two digits, 0–59
SS	INTEGER	The second in two digits, 0–59

Output	Type	Definition
TS	REAL	The timestamp in TIME (6) form

DBMAXRECORDS

This entry point returns the estimated maximum number of records that are currently in the data set. The estimate is computed from the size of the file and the size of a fixed-format record. The actual number of records could be anywhere from 0 to the MAXRECORDS value.

Declaration

```
DBMTYPE procedure DBMAXRECORDS (DSStrNum, MaxRecords);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The DMSII structure number of the data set

Output	Type	Definition
MAXRECORDS	INTEGER	The estimated maximum number of records in the data set

DBMAXRECORDSVF

This entry point estimates the maximum number of records (of a certain record type) a data set can potentially hold. When making this estimate, the procedure assumes that all of the records in the data set are the record type you specified.

If the data set contains only fixed-format records (record type 0), DBMAXRECORDSVF returns the same value that DBMAXRECORDS returns.

The estimates returned for data sets containing variable-format records are smaller than those for fixed-format data sets since variable-format records are larger.

Declaration

```
DBMTYPE procedure DBMaxRecordsVF (DSStrNum, RecType, MaxRecords);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The data set structure number
RECTYPE	INTEGER	The record type number (0 for fixed format)

Output	Type	Definition
MAXRECORDS	INTEGER	The estimated maximum number of records of the specified record type in the data set

DBMESSAGE

This entry point copies the text that describes the error indicated in the DBM_RESULT code field to the caller's array.

Declaration

```
DBMTYPE procedure DBMESSAGE (DBMResult, pMessText, MessLen);
```

Input	Type	Definition
DBM_RESULT	DBMTYPE	The procedure value from a prior call to an entry point
P_MESSTEXT	POINTER	The pointer to the caller's array where the message text is copied **NOTE:** The caller must ensure that the array is large enough.

Output	Type	Definition
MESSLEN	REAL	The length of the output message in bytes

DBMODIFIES

This entry point specifies whether data set record modifications (updates) should be returned as a DELETE/CREATE pair instead of as an update. This is typically necessary for data sets that allow key changes.

The value of ALLOWED is the default value for all selected data sets. See DBSELECT and the SI_MODIFIES field in the STATE_INFO array (listed under STATE_INFO Layout) for information on setting this option for a single data set.

DBRESETOPTION can also be used to set this option.

Declaration

```
DBMTYPE procedure DBMODIFIES (Allowed);
```

Input	Type	Definition
ALLOWED	BOOLEAN	One of the following values: ◆ TRUE—Modifies (updates) are returned as modifies ◆ FALSE—Modifies (updates) are converted to a DELETE/CREATE pair

DBMODIFYTIMESTAMP

This entry point increments or decrements a timestamp by days, hours, minutes, and/or seconds. If the adjustment is negative (as indicated by a negative number, such as -3), the timestamp is decremented.

Declaration

```
DBMTYPE procedure DBMODIFYTIMESTAMP (Days, Hours, Minutes, Seconds, TS);
```

Input	Type	Definition
DAYS	INTEGER	The number of days adjustment
HOURS	INTEGER	The number of hours adjustment
MINUTES	INTEGER	The number of minutes adjustment
SECONDS	INTEGER	The number of seconds adjustment
TS	REAL	The original timestamp in TIME (6) format

Output	Type	Definition
TS	REAL	The modified timestamp in TIME (6) format

DBNULL

This entry point returns a NULL value for a given data item.

Declaration

```
DBMTYPE procedure DBNULL (DSStrNum, ItemNum, NullVal);
```

Input	Type	Definition
DSSTRNUM	REAL	The DMSII structure number of the data set or remap containing the data item
ITEMNUM	REAL	The DMSII data item number, as returned in ITEMINFO [II_ITEM_NUM] For a description of the array, see ITEM_INFO Array Layout.

Output	Type	Definition
NULLVAL	ARRAY	The binary image of the data item's NULL value The caller must ensure that the array is large enough to hold the NULL value. If it is too short, DBEngine resizes it so that it is just large enough.

DBNULLRECORD

This entry point returns a record with all data items NULL.

Declaration

```
DBMTYPE procedure DBNULLRECORD (DSStrNum, RecType, NullRec);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The DMSII structure number of the data set or remap
RECTYPE	INTEGER	The record type number (0 for fixed format)

Output	Type	Definition
NULLREC	ARRAY	The binary image of a data set record with all the data items set to NULL

DBOLDESTAUDITLOC

This entry point finds the oldest audit location on disk, searching among audit files that have the same update level as the current DESCRIPTION file. An audit location is a set of values that define a specific position in the audit trail. DBEngine starts with the current audit file and works backwards until it cannot find an earlier audit file. Then it retrieves the first audit location in that file. For example, suppose the current audit file is 100 and the following audit files are on disk: 89–92 and 96–100. DBOLDESTAUDITLOC returns the first audit location in audit file 96.

Declaration

```
DBMTYPE procedure DBOLDESTAUDITLOC (AFN, ABSN, Seg, Inx);
```

Output	Type	Description
AFN	REAL	The audit file number
ABSN	REAL	The audit block serial number
SEG	REAL	The segment number
INX	REAL	The word index within the audit block

DBOPENAUDIT

This entry point opens an audit file and returns audit file information.

Declaration

```
DBMTYPE procedure DBOpenAudit (AFN, AUDIT_INFO);
```

Input	Type	Definition
AFN	REAL	The number of the audit file to be opened

Output	Type	Definition
AUDIT_INFO	ARRAY	Information about the audit file For a description of the array, see AUDIT_INFO Layout.

DBOPENRESULTTEXT

NOTE: This entry point replaces the DBIORESULTTEXT entry point; however, you can still use DBIORESULTTEXT.

This entry point returns the error or warning message associated with an I/O result code from an OPEN, CLOSE, or RESPOND, and it copies the text describing the I/O result value into the caller's array.

Declaration

```
procedure DBOpenResultText (OpenResult, pText, TextLen);
```

Input	Type	Definition
OPENRESULT	INTEGER	The I/O result value from the OPEN, CLOSE, or RESPOND
PTEXT	POINTER	The destination array for the error text

Output	Type	Description
TEXTLEN	INTEGER	The length of the error text

DBPARAMETERS

This entry point allows the client to specify various run-time processing parameter values, such as the COMMIT frequency and maximum number of WORKER tasks during a clone.

Declaration

```
DBMTYPE procedure DBParameters (ParamType, ParamValue);
```

Input	Type	Definition
PARAMTYPE	REAL	The processing parameter type Possible values are as follows: ◆ DBV_CONCURR_EXTR—Specifies the number of maximum concurrent extracts ◆ DBV_TG_BLOCKS—Specifies the number of audit blocks per transaction group ◆ DBV_TG_UPDATES—Specifies the number of updates per transaction group ◆ DBV_TG_ELAPSED--Specifies the elapsed time per transaction group ◆ DBV_TG_TRANS--Specifies the number of transactions per transaction group ◆ DBV_THREADS--Specifies the maximum number of DBEnterprise threads to use during cloning For more information on these values, see DBPARAMETERS Processing Parameter Types.
PARAMVALUE	REAL	The processing parameter value If the value is less than 0, the entry point discards the change. Set the value to 0 to disable the processing parameter.

DBPRIMARYSET

This entry point returns the structure number of the NO DUPLICATES set that does not allow key changes and has the fewest key items for the given data set.

Declaration

```
DBMTYPE procedure DBPRIMARYSET (DSStrNum, SetStrNum);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The structure number of the data set or remap

Output	Type	Definition
SETSTRNUM	INTEGER	The structure number of the set

DBPRIVILEGED

This entry point returns true if the caller is a privileged program or running under a privileged usercode. Otherwise, it returns false.

Declaration

```
boolean procedure DBPrivileged;
```

DBPUTMESSAGE

This entry point sets the DBMESSAGE parameter values. Libraries that return standard DBMTYPE values can set the message parameter values so that DBMESSAGE fills in the parameter values correctly when an Accessory requests the message text. For related information, see DBMESSAGE.

Declaration

```
DBMTYPE procedure DBPUTMESSAGE (Subtype, Str1, Str2, Str3, Str4);
```

Input	Type	Definition
Subtype	STRING	This is the message subtype.
Str1	STRING	This is additional information supplied with the error. The information supplied varies with the type of error.
STR2	STRING	This is additional information supplied with the error. The information supplied varies with the type of error.
str3	STRING	This is additional information supplied with the error. The information supplied varies with the type of error.
STR4	STRING	This is additional information supplied with the error. The information supplied varies with the type of error.

DBREAD

This entry point receives a transaction group (up to a quiet point or super quiet point) of changes to data set records from the audit trail.

NOTE: The DBREAD entry point is compatible with older Accessories. Using DBREADTRANGROUP is the preferred method.

You should also be aware that you must have selected at least one data set with DBSELECT in order to use this entry point.

DBREAD defaults to committing at the first QPT after the CHECKPOINT interval specified in the DBEngine parameter file. If you want DBREAD to commit at every QPT, call DBSETOPTION (DBV_OP_QPT_GROUP). The DBSETOPTION entry point is explained in DBSETOPTION. If the next transaction group is not available, DBREAD returns immediately with an error.

Declaration

```
DBMTYPE procedure DBREAD (Callback);
```

Input	Type	Definition
CALLBACK	BOOLEAN	The callback procedure that receives information about data set record updates found in the current transaction group

BOOLEAN PROCEDURE CALLBACK

The procedure is called back for each data set record update (CREATE, DELETE, CHANGE) found in the current transaction group.

Declaration

```
boolean procedure Callback (Image, UPDATE_INFO);
```

Parameter	Type	Definition
IMAGE	ARRAY	This array contains the record image, as determined by [UI_UPDATE_TYPE] in the UPDATE_INFO layout. The [UI_UPDATE_TYPE] will be one of the following values. For more information, see Record Change Types. ◆ DBV_CREATE, DBV_DELETE, and DBV_MODIFY indicate that it is a before- or after-image. ◆ DBV_STATE indicates that state information has changed. See STATE_INFO Layout for a description of the array layout. ◆ DBV_DOC indicates that it's a DB_DOC_TYPE. See Documentation Records for the DB_DOC_TYPE value.
UPDATE_INFO	ARRAY	This array contains the description of the modification. See UPDATE_INFO Layout.

DBREADAUDITREGION

The entry point reads the audit file region, starting with the indicated ABSN and block offset. The region contains an integer that is equal to the number of audit blocks. The actual word offset of the region can be computed from NEXTREGIONOFS minus REGIONSIZE.

Declaration

```
DBMTYPE procedure DBReadAuditRegion (RegionABSN, RegionOfs, RegionSize, Region,
    NextRegionABSN, NextRegionOfs);
```

Input	Type	Definition
REGIONABSN	INTEGER	The ABSN of the first block in the region
REGIONOFS	INTEGER	The file-relative word offset of the first block in the region A -1 indicates that the value is unknown.
REGIONSIZE	INTEGER	Maximum size (in words) of the region to be returned

Output	Type	Definition
REGION	ARRAY	The buffer containing the audit region
NEXTREGIONABSN	INTEGER	ASBN of the first block in the next region (that is, following the region returned in the REGION array)
NEXTREGIONOFS	INTEGER	The file-relative word offset of the next region (that is, following the region returned in the REGION array)
REGIONSIZE	INTEGER	The size (in words) of the region returned in the REGION array

DBREADERPARAMETER

This entry point allows an Accessory to specify the title of the FileXtract Reader library and the parameter string that is passed to the Reader library. The parameter string typically contains a file name or directory name, but the individual Reader library determines the format of the string.

The string values specified in DBREADERPARAMETER override the values specified in the logical database comment in the DASDL. However, if you leave either DBREADERPARAMETER string parameter empty, the Accessory Reader library ignores the empty parameter, and the DASDL comment prevails.

Declaration

```
DBMTYPE procedure DBReaderParameter (LibraryTitle, Param);
```

Input	Type	Definition
LIBRARYTITLE	STRING	The title of the FileXtract Reader library If this string is empty, the default title specified in the DASDL logical database comment is used instead.
PARAM	STRING	The character string passed to the FILEREAD entry point of the FileXtract Reader library in the FileInfo array Refer to the *Databridge FileXtract Administrator's Guide* for more information about Reader libraries. If this string is empty, the default parameter specified in the DASDL logical database comment is used instead.

DBREADTRANGROUP

This entry point receives a transaction group (up to a quiet point) of changes to data set records from the audit trail.

NOTE: You must have selected at least one data set with DBSELECT in order to use this entry point.

By default, the CHECKPOINT interval specified in the DBEngine parameter file determines the size of the transaction group. If you want DBREADTRANGROUP to commit at every QPT, call DBSETOPTION (DBV_OP_QPT_GROUP). The DBSETOPTION entry point is explained in DBRESETOPTION. If a transaction group is not available, DBREADTRANGROUP waits up to the amount of time specified in MAXWAITSECS for the group to become available.

DBREADTRANGROUP responds if an Accessory's EXCEPTIONEVENT or ACCEPTEVENT is caused. This ensures that the AX command works immediately.

Declaration

```
DBMTYPE procedure DBReadTranGroup (Callback, RetrySecs, MaxWaitSecs);
```

Input	Type	Definition
CALLBACK	BOOLEAN	The callback procedure that receives each data set record update found in the current transaction group
RETRYSECS	REAL	The number of seconds between retries
MAXWAITSECS	REAL	The maximum number of seconds to wait for a transaction group to become available Values are as follows: ◆ DBV_WAIT_FOREVER—Retry for more audits indefinitely ◆ DBV_DONT_WAIT—Do not retry at all ◆ Positive integer—Specifies the number of seconds to wait

BOOLEAN PROCEDURE CALLBACK

This procedure receives each data set record update (CREATE, DELETE, or CHANGE), STATE_INFO update, or documentation record found in the current transaction group.

Declaration

```
boolean procedure Callback (UPDATE_INFO, BI, AI);
```

Parameter	Type	Definition
UPDATE_INFO	ARRAY	The UPDATE_INFO value describing the update For a description of the array, see UPDATE_INFO Layout.
BI	ARRAY	The before-image of the record This array is valid only for update types DBV_DELETE and DBV_MODIFY. See Record Change Types for a description of these types.
AI	ARRAY	The after-image of the record This array is not valid for update type DBV_DELETE. See Record Change Types for a description of this type.

DBRESETOPTION

This entry point resets (turns off) the DBEngine run-time options. To set run-time options, see DBSETOPTION.

Declaration

```
DBMTYPE procedure DBRESETOPTION (Option);
```

Input	Type	Definition
OPTION	INTEGER	This specifies the option to turn off. For a description of the options, see DBSETOPTION/DBRESETOPTION Run-Time Options.

DBSELECT

This entry point selects which data set(s) to process with subsequent DBREADTRANGROUPs, DBREADs, or DBWAITs.

NOTE: Since DBEngine returns data set records only for data sets that are specified here, you cannot able use DBREADTRANGROUP, DBREAD, or DBWAIT unless you specify a data set(s).

DBSELECT validates the data set's audit location (unless it is to be cloned) and the client format level, and it verifies that the filter allows the specified structure number and record type.

The parent of an embedded data set must be selected before selecting the embedded data set.

To deselect data sets, see DBDESELECT.

Declaration

```
DBMTYPE procedure DBSELECT (STATE_INFO, TableName, StrIdx);
```

Input	Type	Definition
STATE_INFO	ARRAY	The state of the client table, including the audit location The STATE_INFO array contains the DMSII structure number of the data set and the variable-format record type number. For a description of the array, STATE_INFO Layout.
TABLENAME	STRING	The name of the client table DBEngine uses this name in place of the data set name in any error messages. If TABLENAME is empty, Databridge updates it to the DMSII data set name implied by the structure number in the STATE_INFO array.

Output	Type	Definition
STRIDX	INTEGER	Unique index for this data set-record type, suitable for an array index **NOTE:** Remember this STRIDX because any entry point that returns the UPDATE_INFO array uses it.

DBSELECTED

This entry point checks to see if the specified data set has been successfully selected. For related information, see DBSELECT.

The procedure returns the value DBM_OK, which equates to a value of 0, if the data set you specify has been selected with DBSELECT. If it has not been selected, the procedure returns DBM_DS_NOTFOUND (10).

Declaration

```
DBMTYPE procedure DBSELECTED (DSStrNum, RecType);
```

Input	Type	Definition
DSSTRNUM	REAL	DMSII structure number of the data set
RECTYPE	REAL	Record type number (0 for fixed format)

DBSETINFO

This entry point retrieves information about a set.

Declaration

```
DBMTYPE procedure DBSETINFO (SetStrNum, SET_INFO);
```

Input	Type	Definition
SETSTRNUM	REAL	The structure number of the set

Output	Type	Definition
SET_INFO	ARRAY	The information about the set For a description of the array, see SET_INFO Layout.

DBSETOPTION

This entry point sets (enables) the DBEngine run-time options. To reset these options, see DBRESETOPTION.

Declaration

```
DBMTYPE procedure DBSETOPTION (Option);
```

Input	Type	Definition
OPTION	INTEGER	This specifies the option to turn on For a description of these options, see DBSETOPTION/ DBRESETOPTION Run-Time Options.

DBSETS

This entry point returns set names and their structure numbers for a given data set or remap.

Declaration

```
DBMTYPE procedure DBSETS (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the target data set or remap for which the sets are to be returned
CALLBACK	BOOLEAN	The procedure that provides information for each set

BOOLEAN PROCEDURE CALLBACK

This procedure is called once for each set it finds.

Declaration

```
boolean procedure Callback (pSetName, Len, SetStrNum, DuplicatesAllowed,
   KeyChangeAllowed);
```

Parameter	Type	Definition
P_SETNAME	POINTER	The pointer to a set name The caller is expected to copy the actual set name into local memory.
LEN	REAL	The length of the set name
SETSTRNUM	REAL	The structure number of the set
DUPLICATESALLOWED	BOOLEAN	Set to TRUE if duplicates are allowed
KEYCHANGEALLOWED	BOOLEAN	Set to TRUE if key changes are allowed

DBSETSINFO

This entry point returns information for each set of a given data set or remap.

Declaration

```
DBMTYPE procedure DBSETSINFO (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	This is the structure number of the target data set or remap for which the sets are to be returned
CALLBACK	BOOLEAN	The procedure that receives information for each set

BOOLEAN PROCEDURE CALLBACK

This procedure is called once for each set it finds.

Declaration

```
boolean procedure Callback (SET_INFO);
```

Parameter	Type	Definition
SET_INFO	ARRAY	This is the information describing the set For a description of the array, see SET_INFO Layout.

DBSPLITTIMESTAMP

This entry point converts a timestamp from TIME (6) format to a date and time in *yyyy,mm,dd,hh,mn,ss* form. Each component is in binary format.

Declaration

```
DBMTYPE procedure DBSPLITTIMESTAMP (TS, YYYY, MM, DD, HH, MN, SS);
```

Input	Type	Definition
TS	REAL	The timestamp in TIME (6) form, such as from the UI_TIME field of UPDATE_INFO See UPDATE_INFO Layout for a description of the UI_TIME field.

Output	Type	Definition
YYYY	INTEGER	The year in four digits, 1970–2035
MM	INTEGER	The month in two digits, 1–12
DD	INTEGER	The day in two digits, 1–31
HH	INTEGER	The hour in two digits, 0–23
MN	INTEGER	The minute in two digits, 0–59
SS	INTEGER	The second in two digits, 0–59

DBSPLITTIME60

This entry point splits a timestamp in TIME (60) format into separate components similar to DBSPLITTIMESTAMP.

Declaration

```
DBMTYPE procedure DBSPLITTIME60 (TS, YYYY, MM, DD, HH, MN, SS);
```

Input	Type	Definition
TS	REAL	Timestamp in TIME (60) format

Output	Type	Definition
YYYY	INTEGER	Four-digit year, 1970–2035
MM	INTEGER	Two-digit month, 1–12
DD	INTEGER	Two-digit day, 1–31
HH	INTEGER	Two-digit hour, 0–23
MN	INTEGER	Two-digit minute, 0–59
SS	INTEGER	Two-digit second, 0–59

DBSTATEINFOTODISPLAY

This entry point converts the STATE_INFO array, which includes the audit location, to a readable format. The value of this entry point is the length of the resulting message.

Declaration

```
integer procedure DBSTATEINFOTODISPLAY (STATE_INFO, pOut);
```

Input	Type	Definition
STATE_INFO	ARRAY	The state information as it is received from DBEngine For a description of the array, see STATE_INFO Layout.
POUT	POINTER	The destination of the readable format state information

DBSTATISTICS

This entry point returns statistics for the specified category. DBEngine prints a report of the statistics collected (at EOJ) if DBEngine is compiled with `$ SET STATS`, which is available as OBJECT/DATABRIDGE/ENGINE/STATS.

Declaration

```
DBMTYPE procedure DBStatistics (StatCategory, StatDescription, STATISTICS_INFO);
```

Input	Type	Definition
StatCategory	INTEGER	The statistics category number For a description of these values, see Statistics Category Values.

Output	Type	Definition
StatDescription	STRING	A brief description of the statistics category
STATISTICS_INFO	ARRAY	Statistical information For a description of the array, see STATISTICS_INFO Array Layout.

DBSTRIDX

This entry point returns the structure index of the specified data set or remap.

Declaration

```
DBMTYPE procedure DBSTRIDX (DSStrNum, RecType, StrIdx);
```

Input	Type	Definition
DSSTRNUM	REAL	The DMSII structure number of the data set or remap
RECTYPE	REAL	The record type number (0 for fixed format)

Output	Type	Definition
STRIDX	REAL	The structure index of the specified data set or remap. This is the same value returned by DBSELECT.

DBSTRNUM

This entry point returns a structure number for the specified structure name.

Declaration

```
DBMTYPE procedure DBSTRNUM (pStrName, StrNum);
```

Input	Type	Definition
PSTRNAME	POINTER	The pointer to a structure name Any illegal character, such as a space, terminates the name.

Output	Type	Definition
STRNUM	REAL	The DMSII structure number

DBSTRUCTURENAME

This entry point returns a structure name for a DMSII structure number and always uses the physical database, even if the caller specifies a logical database. It also returns the name of a virtual data set when given the structure number specified in DBGenFormat.

Declaration

```
DBMTYPE procedure DBSTRUCTURENAME (StrNum, pName, Len);
```

Input	Type	Definition
STRNUM	REAL	The DMSII structure number (data set, set, remap) from UPDATE_INFO [UI_STRNUM]
PNAME	POINTER	The pointer to the array that is to receive the structure name

Output	Type	Definition
LEN	REAL	The length of the structure name in bytes Characters beyond this point in the array are unchanged.

DBSUBSETSINFO

This entry point returns information about each subset of a given data set.

Declaration

```
DBMTYPE procedure DBSubsetsInfo (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set containing the subset

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the subset. It is called once for each subset.

Declaration

```
boolean procedure Callback (SET_INFO);
```

Parameter	Type	Definition
SET_INFO	BOOLEAN	The array of information describing the subset For a description of the array, see SET_INFO Layout.

DBSWITCHAUDIT

This entry point forces an audit file switch. If you write your own utility for periodically closing the audit file, you can use this entry point to actually perform the audit switch without having to give the database stack number an SM command.

NOTE: You must call the DBINITIALIZE entry point before you call DBSWITCHAUDIT.

This entry point does not take any parameters.

When DBSWITCHAUDIT retries a failed switch (such as after "`***VISIBLE DBS BUSY - TRY AGAIN`"), it increases the delay between retries until it is successful or until the maximum delay retry rate (120 seconds) is exceeded. The DBM_AUDITSWITCH (109) message appears when you exceed the maximum delay retry rate.

DBSWITCHAUDIT also performs an AUDIT CLOSE FORCE, rather than a simple AUDIT CLOSE, which makes the closed audit file available immediately instead of having to wait until two control points are generated normally.

Declaration

```
DBMTYPE procedure DBSWITCHAUDIT;
```

DBTIMESTAMPMSG

This entry point converts the timestamp from TIME (6) format to a date and time message in displayable format.

Declaration

```
DBMTYPE procedure DBTIMESTAMPMSG (TS, TSString);
```

Input	Type	Definition
TS	REAL	The timestamp in TIME (6) form

Output	Type	Definition
TSSTRING	STRING	The timestamp in displayable form, as follows: `month day, year @ hh:mm:ss November 25, 2009 @ 11:27:45`

DBUPDATELEVEL

This entry point returns the database update level and update timestamp. These values correspond to the last DASDL compile.

Declaration

```
DBMTYPE procedure DBUpdateLevel (Updatelevel, UpdateTimestamp);
```

Output	Value	Definition
UPDATELEVEL	REAL	The update level of the database
UPDATETIMESTAMP	REAL	The timestamp of the update

DBVERSION

This entry point provides the version number of the Databridge API file (SYMBOL/DATABRIDGE/INTERFACE) for which DBEngine was compiled. This number must match DBV_VERSION in the API file you include in your program, as in the following example:

```
IF DBVERSION NEQ DBV_VERSION THEN
    BEGIN
    DIE ("Databridge ENGINE software version
      mismatch");
    END;
```

Declaration

```
real procedure DBVERSION;
```

DBWAIT

This entry point receives a transaction group of changes to data set records from the audit trail. It waits up to the amount of time specified in MAXWAITSECS for the group to become available.

NOTE: The DBWAIT entry point is compatible with older Accessories. Using DBREADTRANGROUP is the preferred method.

You must have also selected at least one data set with DBSELECT in order to use this entry point.

DBWAIT responds if an Accessory's EXCEPTIONEVENT or ACCEPTEVENT is caused. This ensures that the AX command works immediately.

Declaration

```
DBMTYPE procedure DBWAIT (Callback, RetrySecs, MaxWaitSecs);
```

Input	Type	Definition
RETRYSECS	REAL	The number of seconds between retries For example, a value of five means to look for more available audits every five seconds.
MAXWAITSECS	REAL	The maximum number of seconds to wait for a transaction group to become available For example, a value of 100 means that this procedure waits a total of 100 seconds (which implies 20 retries when RETRYSECS is set to 5). Values are as follows: ◆ DBV_WAIT_FOREVER—Indicates to retry for more audits indefinitely ◆ DBV_DONT_WAIT—Indicates to not retry at all ◆ Positive integer—Indicates the numbers of seconds to wait
CALLBACK	BOOLEAN	The procedure to call back for each data set record update (CREATE, DELETE, CHANGE), STATE_INFO update, or documentation record found in the current transaction group

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the modification.

Declaration

```
boolean procedure Callback (Image, UPDATE_INFO);
```

Parameter	Type	Definition
IMAGE	ARRAY	This array contains the record image, as follows: ◆ Before-image if DELETE ◆ After-image if CREATE or MODIFY ◆ STATE_INFO image if the state (audit location) changes ◆ Documentation record for non-update information
UPDATE_INFO	ARRAY	This array contains the description of the modification For a description of the array, see UPDATE_INFO Layout.

DBWHEREDASDL

This entry point returns the DASDL source expression associated with the WHERE clause of an automatic subset or the SELECT cause of a remap.

Declaration

```
DBMTYPE procedure DBWhereDASDL (StrNum, pDASDLText, Len);
```

Input	Type	Definition
STRNUM	REAL	The structure number of the desired subset or remap
PDASDLTEXT	POINTER	Destination for the DASDL expression

Output	Type	Definition
PDASDLTEXT	POINTER	Destination for the DASDL expression
LEN	REAL	The length in bytes of the text copied into PDASDLTEXT Possible values are as follows: ◆ If the array is too short, no text is copied, but LEN is set to the needed length and the procedure value is DBM_SHORT_ARRAY (23). ◆ If no DASDL expression is associated with the structure, such as when the structure number is a data set, set, or manual subset, the procedure returns DBM_OK, and LEN is set to 0.

DBWHERETEXT

This entry point returns the ALGOL source code fragment associated with the WHERE clause of an automatic subset or the SELECT clause of a remap.

Declaration

```
DBMTYPE procedure DBWhereText (StrNum, pText, Len);
```

Input	Type	Definition
STRNUM	REAL	The structure number of the subset or remap
PTEXT	POINTER	The destination for the source code text

Output	Type	Definition
PTEXT	POINTER	Destination for the source code text
LEN	REAL	The length in bytes of the text copied into PTEXT Possible values are as follows: ◆ If the array is too short, no text is copied, but LEN is set to the needed length and the procedure value is DBM_SHORT_ARRAY (23). ◆ If no text is associated with the structure, such as when the structure number is a data set, set, or manual subset, the procedure returns DBM_OK, and LEN is set to 0.

DBSupport Entry Points

Use the DBSupport library to filter and format the data you receive from DBEngine.

Security Filtering

DBSupport provides the following levels of security through filtering:

- Data set-level security—For more information, see DBVIEWABLE.
- Record-level security—For more information, see DBFILTER.

Additional Filtering

The DBGenFormat utility can generate additional filtering routines using brief text descriptions in a parameter file. Refer to the *Databridge Host Administrator's Guide* for information.

In a non-tailored support library, you cannot use any data set or data item names, nor can you use any SELECT statements. Therefore, you must create a tailored support library to create effective filters.

You have access to the source code for the DBSupport library (SYMBOL/DATABRIDGE/SUPPORT) and can modify it in order to implement data filtering, data security, and other functions. (For instance, using DBFILTER.) It is strongly recommended, however, that you use DBGenFormat to provide these features if at all possible.

CAUTION: If you patch DBSupport directly rather than using declarations in DBGenFormat, make sure you observe the comments in the source that indicate where user-written patches should go. These lines are preserved from release to release; all other lines are subject to change and resequencing.

DBSupport Formatting

By default, the data records DBEngine returns to an Accessory are not formatted. In other words, they are the binary image of the corresponding record in the DMSII database as they would appear to a COBOL program. Often these records need to be reformatted into individual fields so that an Accessory can store the fields in a more suitable format.

The DBGenFormat utility generates additional formatting routines using brief text descriptions in a parameter file. Refer to the *Databridge Host Administrator's Guide* for description of all default formats.

Accessories can select a format routine by setting the ACTUALNAME of the DBFORMAT entry point to one of the defined formatting routines. For details, see DBFORMAT.

You have access to the source code for the DBSupport library (SYMBOL/DATABRIDGE/SUPPORT) and can modify it in order to implement custom formatting. We strongly recommend, however, that you use DBGenFormat to provide these features if at all possible.

CAUTION: If you patch DBSupport directly rather than using declarations in DBGenFormat, make sure you observe the comments in the source that indicate where user-written patches should go. These lines are preserved from release to release; all other lines are subject to change and resequencing.

Using the DBSupport Entry Points

Before you can use the entry points in the DBSupport library, you must complete the items listed in Accessing the DBEngine and DBSupport Libraries. One of these tasks is to specify a filter using DBSUPPORTINIT. When a description of a DBSupport entry point refers to a filter, it is referring to the specified filter.

The table below summarizes the DBSupport entry points and their functions, and each of these entry points is explained in detail later in this chapter. The Entry Point column in this table contains the name of the entry point, the Type column indicates the type of ALGOL procedure, and the Description column describes what the entry point does. See Entry Point Procedure Values for an explanation of the various procedure types and the values they return. DBMTYPE values are listed in the API file (SYMBOL/DATABRIDGE/INTERFACE) and the *Databridge Host Administrator's Guide*.

Entry Point	Type	Description
DBCLIENTKEY	EMATYPE	Reports errors You can write your own error handling routine to analyze the error and take appropriate action. If no error handler is defined in the DBGenFormat parameter file, this entry point displays the error message and lets the Accessory decide what to do, such as whether to terminate, keep going, or do an ACCEPT.
DBEXTRACTKEY	DBMTYPE	Extracts the primary key of a data set record
DBFILTER	BOOLEAN	Filters records
DBFILTEREDDATASETS	DBMTYPE	Enumerates the data set names and other information about each data set as restricted by a filter
DBFILTEREDITEMINFO	DBMTYPE	Returns information for a data item in a data set and applies the current filter and any ALTERs

Entry Point	Type	Description
DBFILTEREDITEMNAME	DBMTYPE	Returns information for a data item in a data set as restricted by a filter and any ALTERs
DBFILTEREDLAYOUT	DBMTYPE	Enumerates data items in a data set as restricted by a filter
DBFILTEREDLINKS	DBMTYPE	Returns the LINK items for a data set as allowed by the current filter
DBFILTEREDNULLRECORD	DBMTYPE	Returns a record with all data items set to NULL
DBFILTEREDSETS	DBMTYPE	Enumerates set names and their structure number for a data set as restricted by a filter
DBFILTEREDSETSINFO	DBMTYPE	Enumerates information for each set of a given data set available in the filter
DBFILTEREDSTRNUM	DBMTYPE	Returns the structure number for a data set name
DBFILTEREDSUBSETSINFO	DBMTYPE	Enumerates information for each subset of a given data set, provided it is available in the filter
DBFILTEREDWRITE	DBMTYPE	Performs all of the necessary filtering and formatting of an update received from DBReadTranGroup
DBFORMAT	BOOLEAN	Formats the data record for output
DBINITDATAERROR	DBMTYPE	Initializes data-error handling for the formatting routines
DBINITIALIZESUPPORT	DBMTYPE	**NOTE:** This entry point has been replaced by the DBINITIALIZESUPPORT entry point. Verifies that the Accessory is using the same version of DBInterface and allows the DBSupport library to link to DBEngine
DBPRIMARYKEY	DBMTYPE	Enumerates data items that form a unique key for a data set
DBSETUP	BOOLEAN	Verifies that the Accessory is using the same versions of the Databridge interface. This also allows the DBSupport library to initialize.
DBSUPPORTENGINE	DBMTYPE	Allows an Accessory to specify the title of the DBEngine library that DBSupport should link to
DBSUPPORTINIT	DBMTYPE	Required. Verifies that the Accessory is using the same version of DBInterface and allows the DBSupport library to link to DBEngine **NOTE:** This entry point replaces the DBINITIALIZESUPPORT entry point.
DBSUPPORTMISSINGENTRYPOINT	STRING	Returns the name of the first entry point missing from the DBSupport library code file that the Accessory expected to be present based on the interface file
DBUNREMAPITEMINFO	DBMTYPE	Takes a remap data item number and returns item information for the data item in the original data set
DBVIEWABLE	BOOLEAN	Determines whether a data set is viewable for user-defined data set filtering

DBCLIENTKEY

The client calls this entry point to indicate the primary key it is using for a structure. The formatting routines will then send both the BeforeImage and AfterImage of a modify if the key value changed.

Declaration

```
DBMTYPE procedure DBClientKey (StrIdx, KeyCount, KeyList);
```

Input	Type	Definition
StrIdx	INTEGER	Structure index of a selected data set
KeyCount	INTEGER	Number of key items in KeyList
KeyList	REAL ARRAY	LIst of item numbers of keys

DBERRORMANAGER

Accessories call this entry point to report errors. You can write your own error handling routine to analyze the error and take appropriate action. You must declare any error handler you create in the DBGenFormat parameter file (see Chapter 6, “Error Handling Routines,” on page 137 for more information about error handling routines). If no error handler is defined in the DBGenFormat parameter file, this entry point displays the error message and lets the Accessory decide what to do, such as whether to terminate, keep going, or do an ACCEPT.

Declaration

```
EMATYPE procedure DBErrorManager (AccessoryID, ErrNbr, pErrMsg, ErrMsgLen);
```

Input	Type	Definition
ACCESSORYID	AIDTYPE	The ID number of the Accessory AIDTYPE values are listed in Chapter A, “Types, Values, and Array Layouts,” on page 145.
ERRNBR	DBMTYPE	The error number Error numbers are listed in the *Databridge Host Administrator's Guide*.
PERRMSG	POINTER	The error message text Error messages are listed in the Databridge Host Administrator's Guide.
ERRMSGLEN	REAL	The length of the error message in bytes

Example

The following code shows how DBSpan calls DBErrorManager:

```
case DBErrorManager (DBV_Span, DMR, Msg, offset (pMsg)) of
    begin
    DBV_Default: % Accessory can decide
         ;

    DBV_Fatal:  % Accessory should terminate
        Fatal := true;

    DBV_Ignore:  % Accessory should continue
        Fatal := false;
        DMR := DBM_OK;

    DBV_Retry:  % Accessory should retry the operation
        Fatal := false;
    end;

if DMR ^= DBM_OK then % still an error
    begin
    WriteMsg (MSG_ERROR);

    if Fatal then
        begin
        InsertErrNbr (DBM_FATAL_ERROR);
        MESSAGESEARCHER (MessText [DBM_FATAL_ERROR], pMsg, MsgLen);
        display (Msg);
        MYSELF.STATUS := value (TERMINATED);
        end;
    end;
```

DBEXTRACTKEY

This entry point extracts the primary key of a data set record.

Declaration

```
DBMTYPE procedure DBEXTRACTKEY (DSStrNum, Record, Key);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The structure number of the data set whose primary key you want to extract
RECORD	ARRAY	Unformatted data set record from the audit trail from DBREADTRANGROUP, DBREAD, or DBWAIT

Output	Type	Definition
KEY	ARRAY	The primary key value for the record The caller must ensure that this array is large enough to hold the key value; otherwise, a SEG ARRAY ERROR may occur.

DBFILTER

This entry point allows you to apply user-defined record filtering. Use it for record security and selection.

The procedure value can be the following:

- TRUE—The record meets the criteria, so the caller should continue to process the record.
- FALSE—The caller should discard the record.
- Boolean (DBV_WRONGLEVEL)—The record has a different format level than the filter. Recompile the DBSupport library.
- Boolean (DBV_BAD_STRNUM)—The record is for an unknown data set. Recompile the DBSupport library.

Declaration

```
boolean procedure DBFILTER (UserRec, UI);
```

Input	Type	Definition
USERREC	ARRAY	Unformatted data set record from the audit trail
UI	ARRAY	Description of the modification For a description of the array, see UPDATE_INFO Layout.

You have access to the source code for the DBSupport library (SYMBOL/DATABRIDGE/SUPPORT) and can modify it in order to implement data filtering, data security, and other functions. We strongly recommend, however, that you use DBGenFormat to provide these features if at all possible. See the *Databridge Host Administrator's Guide* for more information on DBGenFormat.

CAUTION: Make sure you observe the comments in the source that indicate where user-written patches should go. These lines are preserved from release to release; all other lines are subject to change and resequencing.

DBFILTEREDDATASETS

This entry point returns data set names and other information about each data set in the filter. If the filter discards all records from a particular data set, that data set's information is not returned.

Declaration

```
DBMTYPE procedure DBFilteredDatasets (Callback);
```

Input	Type	Definition
CALLBACK	BOOLEAN	The procedure that receives information for each data set

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about each data set in the filter.

Declaration

```
boolean procedure Callback (pDSName, Len, DATASET_INFO);
```

Parameter	Type	Definition
PDSNAME	POINTER	The pointer to a data set name The calling program must copy this name into its local memory.
LEN	REAL	The length of the data set name
DATASET_INFO	ARRAY	Information about the data set For a description of the array, see DATABASE_INFO Layout.

DBFILTEREDITEMINFO

This entry point returns information for a data item in a data set or remap and applies the current filter and any ALTERs. See Chapter 4, “Altered Data Sets,” on page 111 for more information on ALTERs.

This entry point supports virtual data sets. For more information on virtual data sets, see Chapter 3, “Virtual Data Sets,” on page 87.

Declaration

```
DBMTYPE procedure DBFilteredItemInfo (DSStrNum, RecType, ItemNum, ITEM_INFO);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the desired data set or remap
RECTYPE	REAL	The record type of the desired data set or remap
ITEMNUM	INTEGER	The number of the data item for which to return information.

Output	Type	Definition
ITEM_INFO	ARRAY	The information for the data item For a description of the array, see ITEM_INFO Array Layout.

DBFILTEREDITEMNAME

This entry point returns information for a data item in a data set as restricted by a filter and any ALTERs.

Declaration

```
DBMTYPE procedure DBFilteredItemName (DSStrNum, RecType, ItemName, ITEM_INFO);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the desired data set
RECTYPE	REAL	The record type of the desired data set
ITEMNAME	STRING	The name of the data item whose information is to be returned

Output	Type	Definition
ITEM_INFO	ARRAY	The information for the data item For a description of the array, see ITEM_INFO Array Layout.

DBFILTEREDLAYOUT

This entry point returns data items in a data set or remap as restricted by a filter and any ALTERs.

Declaration

```
DBMTYPE procedure DBFilteredLayout (DSStrNum, RecType, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap that contains the data items you want to return
RECTYPE	REAL	The record type number (0 for fixed format)
CALLBACK	BOOLEAN	The procedure that receives information about each data item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about each data item in the data set or remap.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	ARRAY	Information about the data item For a description of the array, see ITEM_INFO Array Layout.

DBFILTEREDLINKS

This entry point returns the link items for a data set as restricted by the filter.

Declaration

```
DBMTYPE procedure DBFilteredLinks (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the desired data set

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the link items in a data set. This procedure is called once for each link item.

Declaration

```
boolean procedure Callback (ITEM_INFO);
```

Parameter	Type	Definition
ITEM_INFO	ARRAY	The array of information describing a link item in the data set For a description of the array, see ITEM_INFO Array Layout.

DBFILTEREDNULLRECORD

This entry point returns a record with all data items set to NULL. The record layout reflects the filter and any ALTERs.

Declaration

```
DBMTYPE procedure DBFilteredNullRecord (DSStrNum, RecType, NullRec);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The structure number of the data set or remap
RECTYPE	INTEGER	The record type number (0 for fixed format)

Output	Type	Definition
NULLREC	ARRAY	A binary image of a data set record with all data items set to NULL

DBFILTEREDSETS

This entry point returns set names and their structure numbers for a data set or remap as restricted by a filter. If a set contains a key that the filter does not allow, the set is not returned to the calling program.

Declaration

```
DBMTYPE procedure DBFilteredSets (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap for which you want to return sets
CALLBACK	BOOLEAN	The procedure that receives information for each set

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about each set of the specified data set or remap in the filter.

Declaration

```
boolean procedure Callback (pSetName, Len, SetStrNum, DuplicatesAllowed,
     KeyChangeAllowed);
```

Parameter	Type	Definition
P_SETNAME	POINTER	The pointer to the set name The calling program must copy this name to its memory.
LEN	REAL	The length of the set name
STRNUM	REAL	The structure number of the set
DUPLICATESALLOWED	BOOLEAN	One of the following: ◆ TRUE—The set allows duplicates. ◆ FALSE—The set does not allow duplicates.
KEYCHANGEALLOWED	BOOLEAN	One of the following: ◆ TRUE—The set allows an update to change the value of the key. ◆ FALSE—The set does not allow an update to change the value of the key.

DBFILTEREDSETSINFO

This entry point returns information for each set of a given data set or remap, as restricted by the filter. If a set contains a key that the filter does not allow, the set is not returned to the calling program.

Declaration

```
DBMTYPE procedure DBFilteredSetsInfo (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set or remap that is the target of the returned sets
CALLBACK	BOOLEAN	The procedure that receives information for each set

BOOLEAN PROCEDURE CALLBACK

This procedure receives information for each set of the data set or remap as restricted by the filter.

Declaration

```
boolean procedure Callback (SET_INFO);
```

Parameter	Type	Definition
SET_INFO	ARRAY	The information describing the set For a description of the array, see SET_INFO Layout.

DBFILTEREDSTRNUM

This entry point returns the structure number for a data set or remap name, including virtual data sets. If the filter does not allow the specified data set or remap, the entry point returns an error.

Declaration

```
DBMTYPE procedure DBFilteredStrNum (pDSName, DSStrNum);
```

Input	Type	Definition
PDSNAME	POINTER	The pointer to a data set name Any illegal character, such as a space, terminates the name.

Output	Type	Definition
DSSTRNUM	REAL	The structure number for the specified data set or remap

DBFILTEREDSUBSETSINFO

This entry point returns information about each subset of a given data set or remap, as restricted by the filter.

Declaration

```
DBMTYPE procedure DBFilteredSubsetsInfo (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	REAL	The structure number of the data set that is the target of the subsets
CALLBACK	BOOLEAN	The procedure that receives information for each subset

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the subset. This procedure is called once for each subset.

Declaration

```
boolean procedure Callback (SET_INFO);
```

Parameter	Type	Definition
SET_INFO	BOOLEAN	The array of information describing the subset For a description of the array, see SET_INFO Layout.

DBFILTEREDWRITE

This entry point performs all of the necessary filtering and formatting of an update received from DBREADTRANGROUP.

DBFILTEREDWRITE determines two things from UPDATE_INFO as follows:

- Whether to send only the after-image or both the before- and after-images to be modified.
- Whether or not a modify causes a change in the DBFILTER result and sends the appropriate update type. For example, if the update causes the DBFILTER result to change from FALSE to TRUE, DBFILTEREDWRITE sends the update as a CREATE. A change from TRUE to FALSE causes a DELETE.

Procedure values include:

- DBM_OK (0)—The record was written.
- DBM_FILTERED_OUT (104)—The record was not written because it did not satisfy the WHERE condition.
- DBM_FORMAT_ERROR (91)—The formatting routine encountered an error.
- DBM_COMP_SUPPORT (96)—DBSupport needs to be recompiled.

Declaration

```
DBMTYPE procedure DBFilteredWrite (UI, BI, AI, DBFormat, Writer)
```

Input	Type	Definition
UI	ARRAY	The UPDATE_INFO describing the update For a description of the array, see UPDATE_INFO Layout.
BI	ARRAY	The before-image of the record
AI	ARRAY	The after-image of the record
DBFORMAT	BOOLEAN	The formatting routine to call
WRITER	BOOLEAN	The procedure to call to return the formatted record

BOOLEAN PROCEDURE WRITER

This procedure receives information about the formatted record.

Declaration

```
boolean procedure writer (P, Chars);
```

Parameter	Type	Definition
P	POINTER	The pointer to the formatted record
CHARS	REAL	The length of the formatted record in bytes

DBFORMAT

This entry point formats a data record for output. This is the default format, which is a binary image of the corresponding record in the database as it would appear to a COBOL program. RAWFORMAT is an alias for this formatting routine.

The procedure value can be any DBMTYPE result code.

Declaration

```
DBMTYPE procedure DBFORMAT (UserRec, UI, Callback);
```

Input	Type	Definition
USERREC	ARRAY	An unformatted data set record from the audit trail
UI	ARRAY	A description of the modification For a description of the array, see UPDATE_INFO Layout.
CALLBACK	DBMTYPE	The procedure to call with the formatted record

DBMTYPE PROCEDURE CALLBACK

This procedure receives information about the formatted record.

Declaration

```
DBMTYPE procedure Callback (P, Chars);
```

Parameter	Type	Definition
P	POINTER	The pointer to the formatted record
CHARS	REAL	The length of the formatted record in bytes
UPDATE_INFO	ARRAY	A description of the modification. For a description of the array, see the UPDATE_INFO Layout
RawImage	ARRAY	The original unformatted record

Additional Options

Databridge includes several other formats, which DBGenFormat produces from its parameter file. The release object code (executable program) for the DBSupport library contains predefined formatting routines corresponding to the format declarations in DATA/GENFORMAT/SAMPLE/CONTROL.

For more information, see the *Databridge Host Administrator's Guide*.

An Accessory typically refers only to DBFORMAT. By changing the ACTUALNAME of DBFORMAT, however, you can redirect any calls to DBFORMAT to another formatting routine such as FIXEDFORMAT. This allows the Accessory to dynamically select the formatting routine while keeping a simple call to DBFORMAT.

Example

This example shows how to redirect calls from DBFORMAT to another formatting routine.

```
% get the format name
      FORMATNAME := YY_STRING (NAMELOC);

      REPLACE FILETITLE BY FORMATNAME, ".";

      IF SETACTUALNAME (DBFORMAT, FILETITLE) < 0 THEN
              BEGIN
              DIE (DBM_BAD_FORMATNAME, FORMATNAME);
              END;
```

Layout Information

The DBFORMAT routines in a non-tailored DBSupport library load new layout information as necessary. For example, if a data set is reorganized, DBFORMAT loads the new layout when it receives records with the new layout. The generic formatting routines check the DESCRIPTION file for layout information.

DBINITDATAERROR

This entry point initializes data-error handling for the formatting routines. When the formatting routines detect one of the specified error conditions during subsequent processing, they call the indicated procedure, DataError_Output. (The DBINITDATAERROR entry point itself does not call DataError_Output.)

Declaration

```
DBMTYPE procedure DBINITDATAERROR (DataError_Options, DataError_Output);
```

Input	Type	Definition
DATAERROR_OPTIONS	BOOLEAN	Each bit specifies the type of data error checking to perform For a description of these error types, see Data Error Types.
DATAERROR_OUTPUT	BOOLEAN	Procedure to call when a formatting routine detects a data error.

BOOLEAN PROCEDURE DataError_Output

This procedure receives information about the error message.

Declaration

```
boolean procedure DataError_Output (P, Chars);
```

Parameter	Type	Definition
P	POINTER	The pointer to the data error message
CHARS	REAL	The length of the data error message in bytes

DBINITIALIZESUPPORT

NOTE: You should use the DBSUPPORTINIT entry point instead of DBINITIALIZESUPPORT. This entry point is not the preferred method for initializing the DBSupport library.

If you use this entry point, you must first specify the filter and format names in the Accessory using the LIBPARAMETER attribute of DBSupport. (The Accessory cannot specify a transform using this entry point.) DBINITIALIZESUPPORT provides backward compatibility for existing Accessories. All new Accessories use DBSUPPORTINIT.

This entry point verifies that the DBInterface version of the Accessory, DBSupport, and DBEngine are all compatible. If the DBInterface versions used to compile DBEngine, DBSupport, or the Accessory do not match, it returns DBM_VER_MISMATCH (115). If the versions match, DBINITIALIZESUPPORT installs the designated filter and format and returns DBM_OK.

An Accessory must call this entry point (if not DBSUPPORTINIT) before calling any other DBSupport entry points. See Accessing the DBEngine and DBSupport Libraries for more information.

Before calling DBINITIALIZESUPPORT, the Accessory must specify the name of the filter and format DBSupport should use in all of its routines. To specify the filter and format names, set the LIBPARAMETER string library attribute of DBSupport to the filter name followed by a space and the format name as in the following example:

```
SUPPORT.LIBPARAMETER := "ONLYBANK1 BINARYFORMAT";
```

If you do not set LIBPARAMETER to the name of a filter, DBSupport defaults to the predefined DBFILTER, which allows everything.

Declaration

```
define  DBInitializeSupport (AccessoryVersion, AccessoryID) = DBSupportInit
   (AccessoryVersion, AccessoryID, head (Support.LIBPARAMETER, not " "),
    tail (tail (Support.LIBPARAMETER, not " "), " "), empty) #;
```

Input	Type	Definition
ACCESSORYVERSION	REAL	The version of the Databridge Interface used to compile the Accessory
ACCESSORYID	STRING	A description of the Accessory to insert in the error message

DBPRIMARYKEY

This entry point returns data items that form a unique key for a data set. The key is either user-defined (in DBGenFormat) or is the key of the set with the fewest key items that does not allow duplicates.

Declaration

```
DBMTYPE procedure DBPRIMARYKEY (DSStrNum, Callback);
```

Input	Type	Definition
DSSTRNUM	INTEGER	The structure number of the data set or remap for which you want a primary key
CALLBACK	BOOLEAN	The procedure that receives information for each key item

BOOLEAN PROCEDURE CALLBACK

This procedure receives information about the data item that forms the unique key.

Declaration

```
boolean procedure Callback (ItemNum, Descending);
```

Parameter	Type	Definition
ITEMNUM	INTEGER	The item number of the data item, as in ITEM_INFO [II_ITEM_NUM]
DESCENDING	BOOLEAN	TRUE if the item is descending

DBSETUP

NOTE: Use the DBSUPPORTINIT entry point instead of DBSETUP. This entry point is not the preferred method for initializing the DBSupport library. If you use this entry point rather than DBSUPPORTINIT, you cannot specify a filter or format name. They default to DBFILTER and DBFORMAT respectively.

This entry point checks the versions of the Databridge API and initializes the DBSupport library. Your program must call this entry point, if not DBSUPPORTINIT, before calling any other entry points in DBSupport. The success of the procedure is reflected in the Boolean procedure value, as follows:

- TRUE—The version number is correct, and the DBSupport library is initialized.
- FALSE—The initialization failed.

Declaration

```
boolean procedure DBSETUP (Caller_Version);
```

Input	Type	Definition
CALLER_VERSION	REAL	The version of the API file you used to compile your program

DBSUPPORTENGINE

This entry point allows an Accessory to specify the title of the DBEngine library that DBSupport should link to. Contact Micro Focus for additional information.

DBSUPPORTINIT

NOTE: This entry point replaces the DBINITIALIZESUPPORT entry point; however, DBINITIALIZESUPPORT is provided for backward compatibility.

An Accessory must call this entry point first to verify that the DBInterface version of the Accessory, DBSupport, and DBEngine are all compatible and to allow the DBSupport library to link to DBEngine.

If the Accessory, DBEngine, and DBSupport are not all compiled against the same version of DBInterface, this entry point returns a DBM_VER_MISMATCH message.

Declaration

```
DBMTYPE procedure DBSupportInit (AccessoryVersion, AccessoryID, FilterName,
   FormatName, TransformName);
```

Input	Type	Definition
ACCESSORYVERSION	REAL	The version of the API file you used to compile your program
ACCESSORYID	STRING	A string describing the Accessory that prefixes an error message
FILTERNAME	STRING	The name of the filter to use If you do not specify a filter, the default is DBFILTER.
FORMATNAME	STRING	The name of the format to use If you do not specify a format, the default is DBFORMAT.
TRANSFORMNAME	STRING	The name of the transform to use If you do not specify a transform, the default is DBTRANSFORM.

DBSUPPORTMISSINGENTRYPOINT

This entry point returns the name of the first entry point missing from the DBSupport library code file that the Accessory expected to be present based on the interface file.

Declaration

```
string procedure DBSUPPORTMissingEntryPoint;
```

Example

```
string   MissingEP;
...
MissingEP := DBSUPPORTMissingEntryPoint;
if MissingEP NEQ empty then
    display ("Missing DBSupport entry point " !!
    MissingEP);
```

DBUNREMAPITEMINFO

This entry point takes a remap data item number and returns item information for the data item in the original data set.

In the following example, if R remaps D, and you pass this procedure the structure number of R and the item number of R2, it returns ITEMINFO for D1. The item name in ITEMINFO, for example, will be D1.

If the item number is for RVIRT, the routine zeros out the ITEMINFO because it is a VIRTUAL and, therefore, has no original data item information.

```
D DATASET         (
         D1     ALPHA (6);
         D2     NUMBER (12);
         );
R REMAPS D         (
         R2 = D1;
         RVIRT  VIRTUAL NUMBER (2) = 99;
         );
```

Declaration

```
DBMTYPE procedure DBUnRemapItemInfo (RemapStrNum, RemapRecType,
   RemapItemNum, ITEM_INFO);
```

Input	Type	Definition
REMAPSTRNUM	INTEGER	The structure number of the remap
REMAPRECTYPE	INTEGER	The record type containing the remap item (0 for fixed-format)
REMAPITEMNUM	INTEGER	The number of the data item for which to return information

Output	Type	Definition
ITEM_INFO	ARRAY	The item information about the original data set item For a description of the array, see ITEM_INFO Array Layout.

DBVIEWABLE

This entry point determines if a structure is viewable (for user-defined data set filtering). The Boolean procedure values are as follows:

- TRUE—The caller can see the data set.
- FALSE—The caller cannot see the data set.

Declaration

```
boolean procedure DBVIEWABLE (DSStrNum);
```

Input	Type	Definition
DSSTRNUM	REAL	The DMSII structure number

3 Virtual Data Sets

In This Chapter

This chapter gives you programming tips and examples for creating virtual data sets.

Overview

A virtual data set is a collection of data that Databridge Accessories see as a DMSII data set, even though the virtual data set does not actually exist in the DMSII database. Databridge Accessories can clone and track virtual data sets in exactly the same way that they clone and track real data sets.

Virtual data can come from several sources, including sources external to the DMSII database, but something in the audit trail, such as an update or a documentation record, must cause Databridge to retrieve the external data.

Use virtual data sets when you want to create a structure that doesn't physically reside in the DMSII database but can be passed (via a Databridge Accessory) to a Databridge Client relational database or to another secondary database.

NOTE: If you want to convert the format of one or more data items within an existing data set individually, see Chapter 4, “Altered Data Sets,” on page 111.

If you are using a Databridge Client and want to join two or more data sets into a single data set, you should join the data sets in the client database using SQL rather than using a virtual data set.

Under certain circumstances, virtual data sets may be affected by DMSII reorganizations. For more information about how DMSII reorganizations may affect virtual data sets, see DMSII Reorganizations.

To create a virtual data set, you must declare the virtual data set in the DBGenFormat parameter file. The virtual data set declaration lists the data items that you want to include in the virtual data set and specifies other details about the virtual data set, such as the data set structure number.

You must also provide a transform procedure to populate the virtual data set and declare the transform procedure in the DBGenFormat parameter file. A single transform procedure populates all virtual data sets that you declare in the DBGenFormat parameter file. The transform procedure is compiled as a patch to the DBSupport Library (see step Step 9 on page 90 in Creating a Virtual Data Set).

Finally, you must enter the name of the tailored support library and the transform in the appropriate Accessory parameter file.

When this process is completed, the Accessory can clone or track the virtual data set(s).

Before You Begin

To create a virtual data set, complete the following steps:

1. Read this entire chapter so that you get an understanding of how the code you write for your virtual data set relates to the actual virtual data set declarations you make in the DBGenFormat file.

   For example, each virtual data set needs the following:

   - Data set name that follows DMSII data set naming conventions
   - Data item names that follow DMSII data item naming conventions
   - Data item types that adhere to DMSII data type conventions

2. Decide what data you want to use for your virtual data set.

   Virtual data sets may include data from a source external to the DMSII database, but something in the audit trail, such as an update or a documentation record, must cause Databridge to retrieve the external data. You can include any or all of the following:

   - Any DMSII data sets or remaps within one or more databases
   - Any flat file data
   - Any data generated by an external program or library

3. Create the virtual data set as explained in Creating a Virtual Data Set.

Sample Files

The following sections of this guide provide instructions, tips, and samples to help you create a virtual data set:

- Writing a Virtual Data Set Transform Procedure gives specific details about how to modify the virtual transform skeleton (an outline for a transform procedure), PATCH/DATABRIDGE/SAMPLE/SUPPORT/VIRTUAL.
- Sample ALGOL Virtual Transform Procedure contains the sample virtual transform procedure, PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS, and several corresponding parameter file declarations.

Creating a Virtual Data Set

To define a virtual data set, complete the following steps:

NOTE: It is recommended that you read through the section, Sample ALGOL Virtual Transform Procedure, before you create a virtual data set. The section that contains the sample transform also contains other helpful samples. For instance, DBGenFormat Parameter File Declarations contains a sample DBGenFormat declarations that correspond to steps Step 3–Step 5.

---

1. Use CANDE or another editor to retrieve the DBGenFormat parameter file DATA/GENFORMAT/SAMPLE/CONTROL.

   For a general description of the DBGenFormat parameter file, refer to the *Databridge Host Administrator's Guide*.

2. Rename the file, as follows:

   `DATA/GENFORMAT/`*`databasename`*`/CONTROL`

where *databasename* is the name of the database for which you are creating the tailored support library and from which you are creating part of your virtual data set.

3 Declare the virtual data set in the DBGenFormat parameter file (DATA/GENFORMAT/*databasename*/CONTROL) using the syntax in Syntax for Declaring a Virtual Data Set. Repeat this step for each virtual data set you want to declare.

4 Declare a primary key for each virtual data set you declared in the DBGenFormat parameter file if you plan to clone the virtual data sets.

Virtual data sets have no key, and Databridge needs a key to consolidate any fixup records with the extracted records.

NOTE: If you do not create a primary key, Databridge uses absolute address (AA) values to create a unique key for the virtual data set. The code you write for the transform must set the unique AA value of each virtual data set record.

Often, the transform can use the AA of the original ("trigger") record, but if your transform procedure produces more than one virtual data set record for each real data set record, you must create a unique AA value for each virtual data set record.

Refer to the *Databridge Host Administrator's Guide* for more specific information about when and why you need to declare a primary key and for PRIMARY KEY syntax.

5 (Optional) If you want to use the virtual data set definitions in the Transform Layouts section of PATCH/DATABRIDGE/SUPPORT/*databasename*/GENGLOBALS when you write your transform procedure, do the following.

Otherwise, skip this step and go to step Step 6 on page 90.

5a Save `DATA/GENFORMAT/databasename/CONTROL`.

5b Compile the tailored support library, as follows:

```
START WFL/DATABRIDGE/COMP ("SUPPORT", "databasename"
   ["logicaldatabasename"])
```

Where	Is
`"SUPPORT"`	The literal that represents the DBSupport program The quotation marks are required.
`"databasename"`	The name of the database for which you are creating the tailored support library The database name can include a usercode and pack, which are used to locate the database DESCRIPTION file, as follows: `"(usercode)databasename ON packname"` The quotation marks are required.
`"logicaldatabasename"`	The name of a logical database for which you are creating the tailored support library

This WFL compiles layout tables for each data set in the database designated by *databasename* or *logicaldatabasename*. This results in the new tailored support library titled as follows:

```
OBJECT/DATABRIDGE/SUPPORT/databasename
```

— or —

`OBJECT/DATABRIDGE/SUPPORT/`*`databasename/logicaldatabasename`*

These data set-specific layout tables contain the offsets and sizes of individual data items, including virtual data items.

CAUTION: If you have two databases with the same name under different usercodes, *and* you are running Databridge from a third usercode, be careful when you create a tailored support library. In this case, the second library you compile overwrites the first, because Databridge strips the usercode and pack name from the database name to create the tailored support library title.

---

6 Copy the virtual transform skeleton PATCH/DATABRIDGE/SAMPLE/SUPPORT/VIRTUAL as PATCH/DATABRIDGE/SUPPORT/*transformname*, where *transformname* is the name of the transform procedure.

7 Add your code to build virtual records in the sections of PATCH/DATABRIDGE/SUPPORT/*transformname* marked `% TO DO:` as follows:

7a (Optional) Study the declarations for the virtual dataset(s) in the Transform Layouts section of PATCH/DATABRIDGE/SUPPORT/*databasename*/GENGLOBALS.

If you declared any variables global to the transform procedure, initialize them in the InitializeVirtualTransform procedure, which is called the first time DBSupport calls the transform.

7b Write virtual data set transform routines as described in Writing a Virtual Data Set Transform Procedure.

8 Save your changes to PATCH/DATABRIDGE/SUPPORT/*transformname*.

9 Compile DBSupport with the transform as follows:

9a Declare the transform procedure in the DBGenFormat parameter file as shown in Syntax for Declaring a Transform.

9b Save DATA/GENFORMAT/*databasename*/CONTROL.

9c Compile the tailored support library as instructed in step Step 9 on page 90

10 Enter the name of the tailored support library and transform procedure in the appropriate Accessory parameter file, as follows:

For	Do this
Databridge Clients	In the DBServer parameter file, enter the tailored support library name for the SUPPORT option and enter the name of the transform procedure for the TRANSFORM option. For more information, refer to the *Databridge Host Administrator's Guide*.
DBSpan or DBSnapshot	In the DBSpan or DBSnapshot parameter file, enter the tailored support library name for the SUPPORT option and enter the transform name for the TRANSFORM option. For more information, refer to the *Databridge Host Administrator's Guide*.

What to Do Next

Repeat these steps for each virtual data set you want to create.

NOTE: You can declare any number of virtual data sets but you must have exactly one transform that handles all of them.

You can now use your virtual data set by running your Databridge Accessories as usual. If you encounter problems when creating or compiling your virtual data set, see Chapter B, “Troubleshooting,” on page 173 for troubleshooting information.

Chapter B, “Troubleshooting,” on page 173 provides specific troubleshooting tips for writing virtual data set transform procedures and working with virtual data sets.

Syntax for Declaring a Transform

To declare a transform, use the following syntax in the DBGenFormat parameter file:

```
TRANSFORM transformname IN "patchfiletitle"
```

where *transformname* is the transform procedure that you declared, and *patchfiletitle* is the title of the ALGOL patch file containing the transform procedure that you created.

Syntax for Declaring a Virtual Data Set

Use the following syntax to declare a virtual data set. This syntax is taken from DATA/GENFORMAT/SAMPLE/CONTROL.

```
VIRTUAL virtualdatasetname #strnum POPULATION estrecords DERIVED FROM datasetlist
(
dataitem     datatype;
 .
 .
 .
dataitem     datatype;
);
```

Where	Is
datasetlist	The names of one or more data sets from which the virtual data set obtains records. Use commas to separate multiple data set names (see Sample Virtual Data Set Declaration for an example that lists multiple data set names).
virtualdatasetname	The name you want to give to the virtual data set. **NOTE:** Do not use the underscore character.
#strnum	A structure number that you assign to this virtual data set. (The # symbol is required.) The structure number of the first virtual data set must be greater than the largest structure number assigned in the DMSII database. Before you select this number, however, allow room for adding more real structures to the database. For example, if the last structure number used in the DMSII database is 200, you might want to choose 400 as the structure number for the first virtual data set. This leaves room for you to add 199 new sets and data sets to the database. Once you choose a number for the first virtual data set, you can assign structure numbers one greater than the previous virtual data set. In this example, you would assign 400, 401, 402, and so on, to the virtual data sets. Structure numbers cannot exceed 4095.

Where	Is
POPULATION *estrecords*	An optional, but highly recommended, clause where estrecords is the estimate of the number of records that appear in the data set during a clone. This estimate helps Databridge Accessories to allocate space appropriately. The default value is 1000000.
DERIVED FROM *dataset*,…	*Required.* A list of the actual DMSII data sets from which you want to create your virtual data set. This declaration causes DBGenFormat to generate defines and variables in the GENGLOBALS patch that the transform can use to build virtual records.
dataitem	The list of data items you want to be included in this virtual data set. Name the data items the same way you would for a DMSII data set.
datatype	The DMSII data type for this data item. You may use the following data types: ◆ DMSII syntax and data types For example, you would use ALPHA(n) for a text data item. ◆ One of the data item types listed in Chapter B, “Troubleshooting,” on page 173 If you are declaring a signed numeric item, insert at least one space between the S and the number of digits (for example, `TRAN-AMT NUMBER (S 9, 2);`).

Sample Virtual Data Set Declaration

The following sample is the DBGenFormat declaration for a virtual data set:

```
VIRTUAL ADDRESS #79 POPULATION 100000
      DERIVED FROM BANK, CUSTOMER

     (
      ADDR-BANK-ID   NUMBER (4);
      ADDR-CUST-ID   NUMBER (8);
      ADDR-LINE-NBR  NUMBER (1);
      ADDR-LINE      ALPHA (30);
      );
```

Writing a Virtual Data Set Transform Procedure

This section provides additional information about writing a virtual data set transform procedure.

If you used the DERIVED FROM statement when you declared the virtual data set, you can use the `% Transform Layouts` section of PATCH/DATABRIDGE/SUPPORT/database/GENGLOBALS to build the virtual data set records.

NOTE: Compare the virtual transform skeleton (Virtual Transform Skeleton) with the sample transform procedure (ALGOL Source for the Sample Virtual Transform Procedure) to see how the code you must supply relates to the `% TO DO:` sections you modify in the virtual transform skeleton.

Initializing the Virtual Record

Before copying data into the virtual data set record, the transform procedure must initialize the whole virtual data set record area to high values (all bits on) because this is the value Databridge uses to recognize NULL data items. The following example illustrates how to do so:

```
replace VRec8 by real (not false) for size (VRec8);
```

Constructing an UPDATE_INFO Array

Transform procedures construct a virtual record based on real DMSII records and other sources of information. However, the transform procedures must also construct an UPDATE_INFO array to reflect an update to the virtual data set rather than the original (real) record. This includes setting the structure index (UI_STRIDX), structure number (UI_STRNUM), record type (UI_RECTYPE), record size (UI_RECSZ_WORDS), format level (UI_FORMAT_LEVEL), record address (UI_AA), and parent record address (UI_PARENT_AA).

The record type and the parent record address for virtual data sets are always 0. The transform must construct the record address. If the transform builds only one virtual record for each DMSII record, it can use the UI_AA of the DMSII record as the UI_AA of the virtual record.

If you use the DERIVED FROM clause in the virtual data set declaration, you can use the following variables and defines from the GENGLOBALS patch file for the other words of UPDATE_INFO:

*dataset*_StrNum
*dataset*_RecWords
*dataset*_FmtLvl
*dataset*_StrIdx

Calling a COBOL Library

You can code transform procedures in ALGOL and have them call COBOL libraries that actually create the data for the virtual data sets.

If your transform procedure calls a COBOL formatting program that is compiled with $ FEDLEVEL=5, then in the COBOL program's entry point declaration you must specify the ACTUALNAME to match the PROGRAM-ID name in the COBOL program where the library is invoked. For example, the sample COBOL program EXTRACTADDRESS has the following:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   EXTRACTADDRESS.
```

The declaration of the COBOL program's entry point in the ALGOL formatting routine would look like the following:

```
procedure ExtractAddress (...);
library ExtractAddressLib (ACTUALNAME = "EXTRACTADDRESS");
```

See the declaration of EXTRACTADDRESS in the section marked "`% Here's the COBOL program declaration`" in ALGOL Source for the Sample Virtual Transform Procedure.

In addition, if you are using a COBOL 85 compiler, you must set the following compiler options:

```
$$ SET BINARY EXTENDED
$$ SET LIBRARYPROG = TRUE
$$ SET SHARING = DONTCARE
$$ SET TEMPORARY
```

Virtual Transform Skeleton

This is the ALGOL source code for the virtual transform skeleton, PATCH/DATABRIDGE/SAMPLE/SUPPORT/VIRTUAL. Follow the instructions in Writing a Virtual Data Set Transform Procedure to modify and use this file to create a virtual data set.

```
                                                                        09000000
 $ SET OMIT                                                             09000100
------------------------------------------------------------------------09000200
                                                                        09000230
 Module: PATCH/DATABRIDGE/SAMPLE/SUPPORT/VIRTUAL                        09000240
                                                                        09000250
 Project: Databridge                                                    09000260
                                                                        09000270
 Description: Databridge Sample VIRTUAL Transform skeleton              09000280
                                                                        09000290
Copyright (c) 2012 - 2017 Attachmate Corporation, a Micro Focus company.09000390
                                                                        09000530
------------------------------------------------------------------------09000540
                                                                        09002000
       This is a sample skeleton patch to DBSupport for a virtual       09002100
       transform routine.                                               09002200
                                                                        09002300
       It should be used in conjunction with the declarations in        09002400
       PATCH/DATABRIDGE/SUPPORT/<database>/GENGLOBALS that              09002500
       DBGenFormat generates when a VIRTUAL dataset is declared with    09002600
       the DERIVED FROM ... syntax. These declarations follow the       09002700
       comment line " % Transform Layouts" in that patch file.          09002800
                                                                        09002900
       Copy this file as PATCH/DATABRIDGE/SUPPORT/<database>/VIRTUAL    09003000
       (or a name of your choosing). Add your code to build virtual     09003100
       records in the sections marked "TO DO:" below.                   09003200
                                                                        09003300
       Declare this file as a TRANSFORM in DBGenFormat, e.g.,           09003400
                                                                        09003500
       TRANSFORM VirtualTransform                                       09003600
              in "PATCH/DATABRIDGE/SUPPORT/<database>/VIRTUAL"          09003700
                                                                        09003800
              Modification history                                      09003900
              --------------------                                      09004000
                                                                        09004100
Version 41.471                                                          09004200
1      Initial release.                                                 09004300
       This is a sample skeleton patch to DBSupport for a virtual       09004400
       transform routine.                                               09004500
                                                                        09004600
End History                                                             09004700
 $ POP OMIT                                                             09004800
                                                                        70004900
                                                                        70005000
%-----------------------------------------------------------------------  70005100
                                                                        70005200
boolean VirtualTransformInitialized;                                    70005300
                                                                        70005400
DBMTYPE procedure InitializeVirtualTransform;                           70005500
%               -------------------------                               70005600
       begin_proc [InitializeVirtualTransform]                          70005700
                                                                        70005800
       % The following define will retrieve the structure index values 70005900
       % for the virtual datasets.                                      70006000
                                                                        70006100
       VirtualTransformSetup;                                           70006200
```

```
                                                                        70006300
        % TO DO:                                                        70006400
        %       Initialize user-defined variables                       70006500
                                                                        70006600
                                                                        70006700
        VirtualTransformInitialized := true;                            70006800
        end_proc [InitializeVirtualTransform];                          70006900
                                                                        70007000
                                                                        70007100
        % DBTransform-type routine                                      70007200
                                                                        70007300
DBTransformHead [VirtualTransform];                                     70007400
%               ----------------                                        70007500
        begin_proc [VirtualTransform]                                   70007600
                                                                        70007700
               % VirtualTransform will pass the original and generated 70007800
               % records to the formatting routine (DBFormat).         70007900
                                                                        70008000
        boolean FormatResult;                                           70008100
        DBMTYPE DBMResult;                                              70008200
                                                                        70008300
       integer DSStrNum;  % structure number of original dataset        70008400
                                                                        70008500
        define  ReturnIfNoVirtuals =                                   70008600
        %       ------------------                                      70008700
                if ^ IsBase (DSStrNum) then                             70008800
                       begin   % no virtuals derived from this dataset 70008900
                        return (DBM_OK);                                70009000
                        end #;                                          70009100
                                                                        70009200
       % array VRec [0 : ??];  % virtual record work area               70009300
        %                                                               70009400
        % EBCDIC array                                                  70009500
        %       VRec8 [0] = VRec;                                       70009600
        % HEX array                                                     70009700
        %       VRec4 [0] = VRec;                                       70009800
                                                                        70009900
        procedure BuildVirtual (DSRec);                                 70010000
        %         ------------                                          70010100
                array   DSRec [0];  % original dataset record          70010200
                                                                        70010300
                begin                                                   70010400
                EBCDIC array                                            70010500
                        DSRec8 [0] = DSRec;                             70010600
                HEX array                                               70010700
                        DSRec4 [0] = DSRec;                             70010800
                                                                        70010900
                % TO DO:                                               70011000
                %       Use DSStrNum to determine the original        70011100
                %       dataset, e.g.,                                70011200
                %                                                     70011300
                %               if DSStrNum = <dataset>_StrNum then   70011400
                %                       ...                           70011500
                %                                                     70011600
                %       and then build the virtual record(s)          70011700
                %       using the DSRec of the original dataset record. 70011800
                %       For each virtual record you want to send, use 70011900
                %       the <virtualdataset>_Send define, e.g.,       70012000
                %                                                     70012100
                %               <virtualdataset>_Send (<virtualrecord>);70012200
                %                                                     70012300
                %       which will take care of setting up the        70012400
                %       UpdateInfo for the virtual dataset and        70012500
```

```
        %       actually calling the formatting routine.        70012600
                                                                70012700
                                                                70012800
        end BuildVirtual;                                       70012900
                                                                70013000
                                                                70013100
if ^ VirtualTransformInitialized then                           70013200
        begin                                                   70013300
       return_if_error (InitializeVirtualTransform);            70013400
        end;                                                    70013500
                                                                70013600
DSStrNum := UI [UI_STRNUM];                                     70013700
                                                                70013800
       % Send the original record first and then build the     70013900
        % virtual records.                                      70014000
                                                                70014100
case UpdateType of                                              70014200
        begin                                                   70014300
        DBV_CREATE:                                             70014400
                        % first, AI of original                 70014500
                                                                70014600
               ReturnIfFormatError (AI);                        70014700
                                                                70014800
                       % now AI of virtuals                     70014900
                                                                70015000
               ReturnIfNoVirtuals;                              70015100
               BuildVirtual (AI);                               70015200
                                                                70015300
        DBV_MODIFY:                                             70015400
               if UI [UI_BI_AI] = 1 then                        70015500
                       begin                                    70015600
                              % first, BI-AI of original        70015700
                                                                70015800
                      UpdateType := DBV_MODIFY_BEFORE_IMAGE;    70015900
                       ReturnIfFormatError (BI);                70016000
                                                                70016100
                      UpdateType := DBV_MODIFY_AFTER_IMAGE;     70016200
                       ReturnIfFormatError (AI);                70016300
                                                                70016400
                              % now BI-AI of virtuals           70016500
                                                                70016600
                       ReturnIfNoVirtuals;                      70016700
                                                                70016800
                      UpdateType := DBV_MODIFY_BEFORE_IMAGE;    70016900
                       BuildVirtual (BI);                       70017000
                      UpdateType := DBV_MODIFY_AFTER_IMAGE;     70017100
                       BuildVirtual (AI);                       70017200
                       end                                      70017300
               else                                             70017400
                       begin                                    70017500
                              % first, AI of original           70017600
                                                                70017700
                       ReturnIfFormatError (AI);                70017800
                                                                70017900
                              % now AI of virtuals              70018000
                                                                70018100
                       ReturnIfNoVirtuals;                      70018200
                                                                70018300
                       BuildVirtual (AI);                       70018400
                       end;                                     70018500
                                                                70018600
        DBV_DELETE:                                             70018700
                        % first, BI of original                 70018800
```

```
                                                                    70018900
                ReturnIfFormatError (BI);                           70019000
                                                                    70019100
                        % now BI of virtuals                        70019200
                                                                    70019300
                ReturnIfNoVirtuals;                                 70019400
                                                                    70019500
                BuildVirtual (BI);                                  70019600
                                                                    70019700
            DBV_STATE:                                              70019800
                        % Since the Engine will send a StateInfo70019900
                        % for the virtual dataset separately, we70020000
                         % don't need to create one here.          70020100
                                                                    70020200
                ReturnIfFormatError (AI);                           70020300
                                                                    70020400
            else:                                                   70020500
                ReturnIfFormatError (AI);                           70020600
            end UpdateType;                                         70020700
                                                                    70020800
                                                                    70020900
        return (DBM_OK);                                            70021000
        end_proc [VirtualTransform];                                70021100
                                                                    70021200
% End of VirtualTransform transform patch                           70021300
                                                                    70021400
```

Sample ALGOL Virtual Transform Procedure

The sample in this section shows how to create a virtual data set called ADDRESS from data sets called BANK and CUSTOMER, which are part of the BANKDB database.

To illustrate how to create the ADDRESS virtual data set, this section provides the following:

- The various declarations you must make for the ADDRESS virtual data set, such as the declarations in the DBGenFormat parameter file
- A sample virtual transform procedure (modified transform skeleton), PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS, containing code to populate the ADDRESS virtual data set from the BANK and CUSTOMER data sets

Description

The FORMATADDRESS patch file contains a transform procedure called VirtualAddress that determines if the current record is from either the BANK or CUSTOMER data sets. When the transform procedure finds a BANK or CUSTOMER record, it calls a COBOL library, OBJECT/DATABRIDGE/SAMPLE/EXTRACTADDRESS, to extract the address information. Then, the transform procedure puts the data into the ADDRESS virtual data set. Finally, the transform procedure creates the UPDATE_INFO array for the ADDRESS virtual data set.

The source code for the COBOL library, EXTRACTADDRESS (OBJECT/DATABRIDGE/SAMPLE/EXTRACTADDRESS) is shown in Sample COBOL Library.

Sample DASDL Definition

The following sample shows the DASDL information for the BANK and CUSTOMER data sets:

```
 BANK            DATASET
   (
       BANK-ID         NUMBER (4)  NULL IS 0;
       BANK-NAME       ALPHA  (30) NULL IS "NO NAME";
       BANK-ADDR1      ALPHA  (30) NULL IS "N/A    ";
       BANK-ADDR2      ALPHA  (10);
       BANK-ADDR3      ALPHA  (30);
       BANK-ROUTE      NUMBER (9)  NULL IS 999;
       TS              REAL;
   );
.
.
.

CUSTOMER  COMPACT  DATASET
     (
       CUST-ID       NUMBER(8);
       BANK-ID       NUMBER(4);
       CUST-SSN      NUMBER(9);
       CUST-NAME     ALPHA (30) SIZE VARYING;
       CUST-LINES    NUMBER(1);
       CUST-ADDR     ALPHA (30) OCCURS 5 TIMES
                              DEPENDING ON CUST-LINES;
       CUST-DOB      NUMBER(8)  STORED OPTIONALLY;
       CUST-INFO     ALPHA (100) SIZE VARYING;
       TS            REAL;

     );
.
.
.
```

DBGenFormat Parameter File Declarations

The DBGenFormat parameter file for the BANKDB database (DATA/GENFORMAT/BANKDB/CONTROL) contains the following VIRTUAL and TRANSFORM declarations:

```
VIRTUAL ADDRESS #79 POPULATION 100000
       DERIVED FROM BANK, CUSTOMER

         (
          ADDR-BANK-ID   NUMBER (4);
          ADDR-CUST-ID   NUMBER (8);
          ADDR-LINE-NBR  NUMBER (1);
          ADDR-LINE      ALPHA (30);
          );
KEY ADDRESS (ADDR-BANK-ID, ADDR-CUST-ID, ADDR-LINE-NBR);

TRANSFORM VIRTUALADDRESS
   IN "PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS"
```

These declarations correspond to steps Step 3–Step 5 in Creating a Virtual Data Set.

Accessory Parameter File Declarations

The appropriate Accessory parameter file, such as the DBServer parameter file (DATA/SERVER/CONTROL), contains the following TRANSFORM declaration when you enter the name of the tailored support library:

```
SOURCE BANKDB:
        DATABASE    = DESCRIPTION/BANKDB ON DISK,
        TRANSFORM   = VIRTUALADDRESS,
        FILTER      = DBFILTER,
        PREFILTERED = FALSE,
        SUPPORT     = OBJECT/DATABRIDGE/SUPPORT/BANKDB ON DISK
        default;
```

This declaration corresponds to step Step 10 in Creating a Virtual Data Set.

GENGLOBALS Transform Layouts Section

In addition, when you make these particular VIRTUAL and TRANSFORM declarations in the DBGenFormat parameter file and compile a tailored support library, the `% Transform Layouts` section of PATCH/DATABRIDGE/SUPPORT/BANKDB/GENGLOBALS contains the following defines:

```
 % Transform Layouts
 % BANK
real    BANK_StrIdx;
define
    BANK_BANK_ID (Rec4) = Rec4 [0] #,
    BANK_BANK_ID_sz = 4 #,
    BANK_BANK_NAME (Rec8) = Rec8 [2] #,
    BANK_BANK_NAME_sz = 30 #,
    BANK_BANK_ADDR1 (Rec8) = Rec8 [32] #,
    BANK_BANK_ADDR1_sz = 30 #,
    BANK_BANK_ADDR2 (Rec8) = Rec8 [62] #,
    BANK_BANK_ADDR2_sz = 10 #,
    BANK_BANK_ADDR3 (Rec8) = Rec8 [72] #,
    BANK_BANK_ADDR3_sz = 30 #,
    BANK_BANK_ROUTE (Rec4) = Rec4 [204] #,
    BANK_BANK_ROUTE_sz = 9 #,
    BANK_TS (Rec8) = Rec8 [107] #,
    BANK_TS_sz = 6 #,
    BANK_StrNum = 10# ,
    BANK_RecWords = 28# ,
    BANK_FmtLvl = 6799# ,
    BANK_RecBytes = 168# ;

 % CUSTOMER
real    CUSTOMER_StrIdx;
define
    CUSTOMER_CUST_ID (Rec4) = Rec4 [0] #,
    CUSTOMER_CUST_ID_sz = 8 #,
    CUSTOMER_BANK_ID (Rec4) = Rec4 [8] #,
    CUSTOMER_BANK_ID_sz = 4 #,
    CUSTOMER_CUST_SSN (Rec4) = Rec4 [12] #,
    CUSTOMER_CUST_SSN_sz = 9 #,
    CUSTOMER_CUST_NAME (Rec8) = Rec8 [11] #,
    CUSTOMER_CUST_NAME_sz = 30 #,
    CUSTOMER_CUST_LINES (Rec4) = Rec4 [82] #,
    CUSTOMER_CUST_LINES_sz = 1 #,
    CUSTOMER_CUST_ADDR (Rec8, I1) = Rec8 [(84 + (I1-1)*60) div 2] #,
    CUSTOMER_CUST_ADDR_sz = 30 #,
    CUSTOMER_CUST_DOB (Rec4) = Rec4 [384] #,
    CUSTOMER_CUST_DOB_sz = 8 #,
    CUSTOMER_CUST_INFO (Rec8) = Rec8 [196] #,
    CUSTOMER_CUST_INFO_sz = 100 #,
    CUSTOMER_TS (Rec8) = Rec8 [296] #,
```

```
    CUSTOMER_TS_sz = 6 #,
    CUSTOMER_StrNum = 17# ,
    CUSTOMER_RecWords = 51# ,
    CUSTOMER_FmtLvl = 0# ,
    CUSTOMER_RecBytes = 306# ;

 % ADDRESS
real    ADDRESS_StrIdx;
define
    ADDRESS_ADDR_BANK_ID (Rec4) = Rec4 [0] #,
    ADDRESS_ADDR_BANK_ID_sz = 4 #,
    ADDRESS_ADDR_CUST_ID (Rec4) = Rec4 [4] #,
    ADDRESS_ADDR_CUST_ID_sz = 8 #,
    ADDRESS_ADDR_LINE_NBR (Rec4) = Rec4 [12] #,
    ADDRESS_ADDR_LINE_NBR_sz = 1 #,
    ADDRESS_ADDR_LINE (Rec8) = Rec8 [7] #,
    ADDRESS_ADDR_LINE_sz = 30 #,
    ADDRESS_StrNum = 79# ,
    ADDRESS_RecWords = 7# ,
    ADDRESS_FmtLvl = 25861# ,
    ADDRESS_Send (VRec) =
       VirtualSend (ADDRESS_StrNum, ADDRESS_StrIdx,
             ADDRESS_RecWords, ADDRESS_FmtLvl, VRec) #,
    ADDRESS_RecBytes = 42# ;
define  VirtualTransformSetup =
    begin
    GetStrIdx (BANK_StrNum, 0, BANK_StrIdx);
    GetStrIdx (CUSTOMER_StrNum, 0, CUSTOMER_StrIdx);
    GetStrIdx (ADDRESS_StrNum, 0, ADDRESS_StrIdx);
    end #;
define  IsBase (StrNum) = (
    if StrNum = 10 then true else
    if StrNum = 17 then true else
    false) #;
```

ALGOL Source for the Sample Virtual Transform Procedure

The ALGOL source code for PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS is as follows:

```
 $ SET OMIT                                                                  09000000
------------------------------------------------------------------------------09000100
                                                                             09000400
 Module: PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS                        09000500
                                                                             09000600
 Project: Databridge                                                         09000700
                                                                             09000800
 Description: Sample transform for VIRTUAL datasets                          09000900
                                                                             09001000
Copyright (c) 2012 - 2017 Attachmate Corporation, a Micro Focus company.09001100
                                                                             09001200
------------------------------------------------------------------------------09001300
                                                                             09001400
       Example transform routine for VIRTUAL datasets. This is a patch 09002000
       to SYMBOL/DATABRIDGE/SUPPORT and can be included by inserting   09002100
       the following declaration in DATA/GENFORMAT/<database>/CONTROL: 09002200
                                                                             09002300
               TRANSFORM VIRTUALADDRESS                                      09002400
                  IN "PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS"  09002500
                                                                             09002600
```

```
This transform routine illustrates how to populate a VIRTUAL    09002700
dataset from real dataset records. It extracts mailing addresses09002800
from BANK and CUSTOMER dataset records and puts them into a     09002900
VIRTUAL dataset called ADDRESS.                                 09003000
                                                                09003100
A COBOL program does the actual extraction of the address. The  09003200
transform routine below determines if the current record is     09003300
from either the BANK or CUSTOMER datasets, and if so, calls the 09003400
COBOL program to extract the address.                           09003500
                                                                09003600
The BANKDB DASDL has these definitions for BANK and CUSTOMER:   09003700
                                                                09003800
        BANK            DATASET                                 09003900
           (                                                    09004000
                BANK-ID         NUMBER (4)  NULL IS 0;          09004100
                BANK-NAME       ALPHA  (30) NULL IS "NO NAME";  09004200
                BANK-ADDR1      ALPHA  (30) NULL IS "N/A    ";  09004300
                BANK-ADDR2      ALPHA  (10);                    09004400
                BANK-ADDR3      ALPHA  (30);                    09004500
                BANK-ROUTE      NUMBER (9)  NULL IS 999;        09004600
                TS              REAL;                           09004700
           );                                                   09004800
                                                                09004900
        ...                                                     09005000
                                                                09005100
        CUSTOMER  COMPACT  DATASET                              09005200
           (                                                    09005300
                CUST-ID       NUMBER(8);                        09005400
                BANK-ID       NUMBER(4);                        09005500
                CUST-SSN      NUMBER(9);                        09005600
                CUST-NAME     ALPHA (30) SIZE VARYING;          09005700
                CUST-LINES    NUMBER(1);                        09005800
                CUST-ADDR     ALPHA (30) OCCURS 5 TIMES         09005900
                                    DEPENDING ON CUST-LINES;    09006000
                CUST-DOB      NUMBER(8)  STORED OPTIONALLY;     09006100
                CUST-INFO     ALPHA (100) SIZE VARYING;         09006200
                TS            REAL;                             09006300
           );                                                   09006400
                                                                09006500
        ...                                                     09006600
                                                                09006700
DATA/GENFORMAT/BANKDB/CONTROL has these declarations:           09006800
                                                                09006900
               VIRTUAL ADDRESS #79 POPULATION 100000            09007000
                       DERIVED FROM BANK, CUSTOMER              09007100
                                                                09007200
                       (ADDR-BANK-ID   NUMBER (4);              09007300
                        ADDR-CUST-ID   NUMBER (8);              09007400
                        ADDR-LINE-NBR  NUMBER (1);              09007500
                        ADDR-LINE      ALPHA (30);              09007600
                        );                                      09007700
                                                                09007800
       KEY ADDRESS (ADDR-BANK-ID, ADDR-CUST-ID, ADDR-LINE-NBR);09007900
                                                                09008000
        TRANSFORM VIRTUALADDRESS                                09008100
           IN "PATCH/DATABRIDGE/SAMPLE/SUPPORT/FORMATADDRESS"   09008200
                                                                09008300
And finally, DATA/SERVER/CONTROL has this declaration:          09008400
                                                                09008500
        SOURCE BANKDB:                                          09008600
               DATABASE    = DESCRIPTION/BANKDB ON DISK,        09008700
               TRANSFORM   = VIRTUALADDRESS,                    09008800
               FILTER      = DBFILTER,                          09008900
```

```
                        PREFILTERED = FALSE,                              09009000
                       SUPPORT     = OBJECT/DATABRIDGE/SUPPORT/BANKDB     09009100
                                        ON DISK                           09009200
                        default;                                          09009300
                                                                          09009400
       Notice that the TRANSFORM name matches the name of the routine     09009500
        below.                                                            09009600
                                                                          09009700
               Modification history                                       09009800
               --------------------                                       09009900
                                                                          09010000
Version 30.001                                                            09010100
       Initial release.                                                   09010200
                                                                          09010300
       This is a patch to SYMBOL/DATABRIDGE/SUPPORT that illustrates      09010400
       how to write a formatting routine to populate VIRTUAL datasets.    09010500
       It calls a COBOL program to extract addresses from other           09010600
        records.                                                          09010700
                                                                          09010800
Version 40.463                                                            09010900
1      Changed the ADDRESS record size calculation to match the COBOL     09011000
        record.                                                           09011100
                                                                          09011200
Version 41.471                                                            09011300
1      Changed the routine from a FORMAT to a TRANSFORM.                  09011400
                                                                          09011500
       The program SYMBOL/DATABRIDGE/SAMPLE/VIRTUALLIB has been           09011600
        deimplemented in favor of this patch.                             09011700
                                                                          09011800
Version 41.484                                                            09011900
2      The patch now uses the declarations generated in GENGLOBALS        09012000
       when a VIRTUAL dataset is declared with the DERIVED FROM           09012100
        syntax, such as,                                                  09012200
               ADDRESS_StrNum                                             09012300
               ADDRESS_StrIdx                                             09012400
               ADDRESS_RecWords                                           09012500
               ADDRESS_FmtLvl                                             09012600
               BANK_StrNum                                                09012700
               CUSTOMER_StrNum                                            09012800
                                                                          09012900
Version 50.491                                                            09013000
1      The code to retrieve the ADDRESS structure index is now            09013100
       deferred until the first BANK or CUSTOMER record is encountered.   09013200
       Previously, if the client did not select the BANK, CUSTOMER, and   09013300
       ADDRESS datasets, the transform would get an error when it tried   09013400
       to retrieve the structure index for ADDRESS when it received       09013500
        the first record (from some other dataset).                       09013600
                                                                          09013700
Version 50.504                                                            09013800
2      This transform will now tolerate the situation where the BANK      09013900
       and/or CUSTOMER datasets are selected but not the virtual          09014000
       ADDRESS is not. In such a case, the transform will not try to      09014100
       extract any addresses and will simply send the (real) dataset      09014200
        updates.                                                          09014300
                                                                          09014400
End History                                                               09014500
 $ POP OMIT                                                               09014600
                                                                          70000000
       % Here's the COBOL program declaration.                            70000100
                                                                          70000200
library ExtractAddressLib                                                 70000300
       (title = "OBJECT/DATABRIDGE/SAMPLE/EXTRACTADDRESS.");              70000400
                                                                          70000500
```

```
procedure ExtractAddress (WhichDS, DSRecord, NumAddrRecs, AddressRecs); 70000600
%        --------------                                                 70000700
                                                                        70000800
              % COBOL program that can extract addresses                70000900
               % from BANK and CUSTOMER                                 70001000
                                                                        70001100
       value   WhichDS;                                                70001200
       integer WhichDS;       % Input: Which dataset? (See below)     70001300
       ebcdic array                                                     70001400
              DSRecord [0];   % Input: record from audit trail        70001500
       integer NumAddrRecs;   % Output: number of addresses generated 70001600
       ebcdic array                                                     70001700
              AddressRecs [0];% Output: generated address records     70001800
                                                                        70001900
       library ExtractAddressLib                                        70002000
                                                                        70002100
              % if the COBOL program is compiled with $ FEDLEVEL = 5  70002200
              % then change "PROCEDUREDIVISION" to the value of the   70002300
               % PROGRAM-ID in the program                             70002400
                                                                        70002500
               (ACTUALNAME = "PROCEDUREDIVISION");                      70002600
                                                                        70002700
%-----------------------------------------------------------------------  70002800
                                                                        70002900
       % WhichDS values                                                 70003000
define  ItIsBank = 1#,                                                  70003100
       ItIsCust = 2#;                                                   70003200
                                                                        70003300
boolean VAInitialized;                                                  70003400
                                                                        70003500
integer AddrRecBytes;          % size of ADDRESS record in bytes       70003600
define  MaxAddressRecs = 10#;   % max number of ADDRESS records in      70003700
                               % AddressRecs                            70003800
                                                                        70003900
ebcdic array                                                            70004000
       AddressRecs [0:0];      % output ADDRESS records                 70004100
                                                                        70004200
array   AddressRec [0:0];      % single ADDRESS record to send          70004300
                                                                        70004400
interlock                                                               70004500
       AddressLock;                                                     70004600
                                                                        70004700
DBMTYPE procedure InitializeVA;                                         70004800
%                ------------                                           70004900
       begin_proc [InitializeVA]                                        70005000
       DBStrIdx (ADDRESS_StrNum, 0, ADDRESS_StrIdx);                    70005100
                                                                        70005200
       % We need the size of the record that the COBOL                 70005300
       % program actually uses. The easiest thing to do is to hard-    70005400
        % code the size...                                              70005500
                                                                        70005600
       AddrRecBytes := 37;                                              70005700
                                                                        70005800
       resize (AddressRecs, MaxAddressRecs * AddrRecBytes);             70005900
       resize (AddressRec, ADDRESS_RecWords);                           70006000
                                                                        70006100
       VAInitialized := true;                                           70006200
       end_proc [InitializeVA];                                         70006300
                                                                        70006400
                                                                        70006500
       % DBTransform-type routine                                       70006600
                                                                        70006700
DBTransformHead [VirtualAddress];                                       70006800
```

```
%               --------------                                          70006900
        begin_proc [VirtualAddress]                                     70007000
                                                                        70007100
               % VirtualAddress will pass the original and generated    70007200
               % records to the formatting routine (DBFormat).         70007300
                                                                        70007400
        ebcdic array                                                    70007500
               BI01 [0] = BI;    % dataset record before-image         70007600
        ebcdic array                                                    70007700
               AI01 [0] = AI;    % dataset record after-image          70007800
        pointer pAddress;         % points to an address              70007900
        boolean FormatResult;                                           70008000
        DBMTYPE DBMResult;                                              70008100
                                                                        70008200
        procedure GetAddresses (StrNum, Image01);                       70008300
        %         ------------                                          70008400
                value   StrNum;                                         70008500
                integer StrNum;                                         70008600
                ebcdic array                                            70008700
                        Image01 [0];                                    70008800
                                                                        70008900
                begin                                                   70009000
                integer NumAddresses;     % number of addresses found   70009100
                                                                        70009200
                        % look for any addresses                        70009300
                if StrNum = BANK_StrNum then                            70009400
                        begin                                           70009500
                        if ^ VAInitialized then                         70009600
                                begin                                   70009700
                                return_if_error (InitializeVA);         70009800
                                end;                                    70009900
                                                                        70010000
                        if ADDRESS_StrIdx > 0 then                      70010100
                                begin                                   70010200
                                ExtractAddress (ItIsBank, Image01,      70010300
                                                NumAddresses,           70010400
                                                AddressRecs);           70010500
                                end;                                    70010600
                                                                        70010700
                        end                                             70010800
                else                                                    70010900
                if StrNum = CUSTOMER_StrNum then                        70011000
                        begin                                           70011100
                        if ^ VAInitialized then                         70011200
                                begin                                   70011300
                                return_if_error (InitializeVA);         70011400
                                end;                                    70011500
                                                                        70011600
                        if ADDRESS_StrIdx > 0 then                      70011700
                                begin                                   70011800
                                ExtractAddress (ItIsCust, Image01,      70011900
                                                NumAddresses,           70012000
                                                AddressRecs);           70012100
                                end;                                    70012200
                                                                        70012300
                        end;                                            70012400
                                                                        70012500
                if NumAddresses > 0 then       % found some addresses    70012600
                        begin                                           70012700
                        % set up the UI for ADDRESS records             70012800
                                                                        70012900
                        UI [UI_STRNUM]         := ADDRESS_StrNum;       70013000
                        UI [UI_RECTYPE]        := 0;                    70013100
```

```
                UI [UI_STRIDX]         := ADDRESS_StrIdx;       70013200
                UI [UI_RECSZ_WORDS]    := ADDRESS_RecWords;     70013300
                UI [UI_FORMAT_LEVEL]   := ADDRESS_FmtLvl;       70013400
                 UI [UI_AA]            := 0;                    70013500
                 UI [UI_PARENT_AA]      := 0;                   70013600
                                                                 70013700
                        % send each virtual ADDRESS record       70013800
                                                                 70013900
                pAddress := AddressRecs;                         70014000
                while NumAddresses > 0 do                        70014100
                        begin                                    70014200
                        replace pointer (AddressRec) by          70014300
                                pAddress : pAddress              70014400
                                for AddrRecBytes;                70014500
                                                                 70014600
                        NumAddresses := * - 1;                   70014700
                        ReturnIfFormatError (AddressRec);        70014800
                         end;                                    70014900
                end;                                             70015000
                                                                 70015100
        end GetAddresses;                                        70015200
                                                                 70015300
                                                                 70015400
integer OrigStrNum;                                              70015500
                                                                 70015600
       % Since we use global arrays we have to single-thread.    70015700
epilog procedure ExitFormat;                                     70015800
        begin                                                    70015900
        unlock (AddressLock);                                    70016000
        end;                                                     70016100
                                                                 70016200
lock (AddressLock);                                              70016300
                                                                 70016400
OrigStrNum := UI [UI_STRNUM];                                    70016500
                                                                 70016600
       % Send the original record first and then get the        70016700
        % virtual address records.                               70016800
                                                                 70016900
case UpdateType of                                               70017000
        begin                                                    70017100
        DBV_CREATE:                                              70017200
                        % first, AI of original                  70017300
                                                                 70017400
                ReturnIfFormatError (AI);                        70017500
                                                                 70017600
                        % now AI of virtuals                     70017700
                                                                 70017800
                GetAddresses (OrigStrNum, AI01);                 70017900
                                                                 70018000
        DBV_MODIFY:                                              70018100
                if UI [UI_BI_AI] = 1 then                        70018200
                        begin                                    70018300
                                % first, BI-AI of original       70018400
                                                                 70018500
                        UpdateType :=                            70018600
                                DBV_MODIFY_BEFORE_IMAGE;         70018700
                        ReturnIfFormatError (BI);                70018800
                                                                 70018900
                        UpdateType :=                            70019000
                                DBV_MODIFY_AFTER_IMAGE;          70019100
                        ReturnIfFormatError (AI);                70019200
                                                                 70019300
                                % now BI-AI of virtuals          70019400
```

```
                                                                    70019500
                        UpdateType :=                               70019600
                               DBV_MODIFY_BEFORE_IMAGE;             70019700
                        GetAddresses (OrigStrNum, BI01);            70019800
                        UpdateType :=                               70019900
                               DBV_MODIFY_AFTER_IMAGE;              70020000
                        GetAddresses (OrigStrNum, AI01);            70020100
                        end                                         70020200
                 else                                               70020300
                        begin                                       70020400
                               % first, AI of original              70020500
                                                                    70020600
                        ReturnIfFormatError (AI);                   70020700
                                                                    70020800
                               % now AI of virtuals                 70020900
                                                                    70021000
                        GetAddresses (OrigStrNum, AI01);            70021100
                        end;                                        70021200
                                                                    70021300
          DBV_DELETE:                                               70021400
                        % first, BI of original                     70021500
                                                                    70021600
                 ReturnIfFormatError (BI);                          70021700
                                                                    70021800
                        % now BI of virtuals                        70021900
                                                                    70022000
                 GetAddresses (OrigStrNum, BI01);                   70022100
                                                                    70022200
          DBV_STATE:                                                70022300
                        % Since the Engine will send a StateInfo70022400
                        % for the virtual dataset separately, we70022500
                        % don't need to create one here.            70022600
                                                                    70022700
                 ReturnIfFormatError (AI);                          70022800
                                                                    70022900
          else:                                                     70023000
                 ReturnIfFormatError (AI);                          70023100
          end UpdateType;                                           70023200
                                                                    70023300
                                                                    70023400
   return (DBM_OK);                                                 70023500
   end_proc [VirtualAddress];                                       70023600
                                                                    70023700
% End of VirtualAddress transform patch                            70023800
                                                                    70023900
```

Sample COBOL Library

This sample library, EXTRACTADDRESS, extracts the address from individual BANK and CUSTOMER data set records and returns them to the transform procedure (VirtualAddress).

The transform procedure in the previous example calls this library.

NOTE: COBOL libraries called by virtual data set transform procedures can be affected by DMSII reorganizations. For more information on how COBOL libraries may be affected, see DMSII Reorganizations.

The source code (SYMBOL/DATABRIDGE/SAMPLE/EXTRACTADDRESS) for this sample is shown as follows:

```
000100
000200$$ SET BINARYEXTENDED                                                00
000300$$ SET LIBRARYPROG = TRUE                                            00
000400$$ SET SHARING = DONTCARE                                            00
000500$$ SET TEMPORARY                                                     00
000600                                                                     00
000700 IDENTIFICATION DIVISION.                                            00
000800 PROGRAM-ID.    EXTRACTADDRESS.                                      00
000900                                                                     00
001000$$ set omit
001010------------------------------------------------------------------
001020
001040
001050 Module: SYMBOL/DATABRIDGE/SAMPLE/EXTRACTADDRESS
001060
001070 Project: Databridge
001080
001090 Description: Databridge ExtractAddress Library Program
001100
001200 Copyright (c) 2017 Attachmate Corporation, a Micro Focus company.
001340
001350------------------------------------------------------------------
001360$$ pop omit
003000*       This is an example library program to demonstrate how        00
003100*       to extract mailing addresses from dataset records and        00
003200*       use them to populate a VIRTUAL dataset using Databridge.     00
003300*                                                                    00
003400                                                                     00
003500 ENVIRONMENT    DIVISION.                                           00
003600                                                                     00
003700 DATA           DIVISION.                                           00
003800 DATA-BASE      SECTION.                                            00
003900                                                                     00
004000*       We won't really access the DMSII database but we             00
004100*       want to use the 01s for the record layouts of the            00
004200*       records we will receive.                                     00
004300                                                                     00
004400 DB BANKDB.                                                          00
004500 01 BANK.                                                           00
004600 01 CUSTOMER.                                                       00
004700                                                                     00
004800 WORKING-STORAGE SECTION.                                            00
004900                                                                     00
005000 77 WS-INITIALIZE       PIC 9(01)       BINARY.                     00
005100    88 INITIALIZED      VALUE 1.                                    00
005200                                                                     00
005300 77 I                   PIC 9(10)       BINARY.                     00
005400                                                                     00
005500*       Length of the DMSII records (in bytes).                      00
005600                                                                     00
005700 77 BANK-REC-CHARS      PIC 9(06)       BINARY.                     00
005800 77 CUST-REC-CHARS      PIC 9(06)       BINARY.                     00
005900                                                                     00
006000                                                                     00
006100 LINKAGE        SECTION.                                            00
006200*       Parameters passed from the formatting routine.               00
006300                                                                     00
006400*       DMSII structure number of the input record.                  00
006500                                                                     00
006600 77 WHICH-DATASET       PIC 9(4)        BINARY.                     00
006700    88 BANK-DATASET     VALUE 1.                                    00
006800    88 CUST-DATASET     VALUE 2.                                    00
006900                                                                     00
```

```
007000* Record received from DMSII database.                                00
007100* Should be either from the BANK or CUSTOMER dataset.                  00
007200                                                                       00
007300 01 DB-RECORD.                                                        00
007400        03 DB-REC-WORD  PIC S9(11)      BINARY                       00
007500                                                                       00
007600*       The OCCURS value must make this record as large               00
007700*       or larger than the actual dataset record received.            00
007800                                                                       00
007900                        OCCURS 56.                                     00
008000                                                                       00
008100*       The following is the number of VIRTUAL ADDRESS records        00
008200*       returned.                                                     00
008300                                                                       00
008400 77 VIRTUAL-REC-COUNT   PIC 9(11)       BINARY.                       00
008500                                                                       00
008600*       Here is where we build the VIRTUAL ADDRESS records.           00
008700                                                                       00
008800 01 VIRTUAL-RECS.                                                     00
008900    02 ADDRESS-REC      OCCURS 10.                                     00
009000       03 ADDR-BANK-ID  PIC 9(4)        COMP.                        00
009100       03 ADDR-CUST-ID  PIC 9(8)        COMP.                        00
009200       03 ADDR-LINE-NBR PIC 9(1)        COMP.                        00
009300       03 ADDR-LINE     PIC X(30)       DISPLAY.                     00
009400                                                                       00
009500 PROCEDURE DIVISION USING WHICH-DATASET                               00
009600                          DB-RECORD                                   00
009700                          VIRTUAL-REC-COUNT                           00
009800                          VIRTUAL-RECS.                               00
009900 THE-ONLY       SECTION.                                              00
010000 GET-STARTED.                                                         00
010100                                                                       00
010200        IF NOT INITIALIZED                                            00
010300               PERFORM INITIALIZE-WS.                                 00
010400                                                                       00
010500        MOVE HIGH-VALUES TO VIRTUAL-RECS.                             00
010600                                                                       00
010700* Determine which dataset this record is from.                        00
010800                                                                       00
010900        IF BANK-DATASET                                               00
011000               PERFORM BANK-ADDRESS                                   00
011100        ELSE                                                          00
011200        IF CUST-DATASET                                               00
011300               PERFORM CUST-ADDRESS                                   00
011400        ELSE                                                          00
011500* Unrecognized dataset ...                                            00
011600               MOVE 0 TO VIRTUAL-REC-COUNT.                           00
011700                                                                       00
011800        EXIT PROGRAM.                                                 00
011900                                                                       00
012000 INITIALIZE-WS.                                                       00
012100                                                                       00
012200* Determine the size of the dataset records.                          00
012300                                                                       00
012400        COMPUTE BANK-REC-CHARS =                                      00
012500               FUNCTION FORMATTED-SIZE (BANK).                        00
012600        COMPUTE CUST-REC-CHARS =                                      00
012700               FUNCTION FORMATTED-SIZE (CUSTOMER).                    00
012800        MOVE 1 TO WS-INITIALIZE.                                      00
012900                                                                       00
013000 BANK-ADDRESS.                                                        00
013100                                                                       00
013200* Move the database record into the BANK user work area so            00
```

```
013300* that we can reference individual data items.                          00
013400                                                                        00
013500       UNSTRING DB-RECORD INTO BANK                                     00
013600              FOR BANK-REC-CHARS.                                       00
013700                                                                        00
013800* The address lines in the BANK dataset are in 3 separate               00
013900* data items: BANK-ADDR1, BANK-ADDR2, and BANK-ADDR3.                    00
014000                                                                        00
014100       MOVE BANK-ID OF BANK    TO ADDR-BANK-ID (1).                     00
014200       MOVE 0                 TO ADDR-CUST-ID (1).                      00
014300       MOVE 1                 TO ADDR-LINE-NBR (1).                     00
014400       MOVE BANK-ADDR1         TO ADDR-LINE (1).                        00
014500                                                                        00
014600       MOVE BANK-ID OF BANK    TO ADDR-BANK-ID (2).                     00
014700       MOVE 0                 TO ADDR-CUST-ID (2).                      00
014800       MOVE 2                 TO ADDR-LINE-NBR (2).                     00
014900       MOVE BANK-ADDR2         TO ADDR-LINE (2).                        00
015000                                                                        00
015100       MOVE BANK-ID OF BANK    TO ADDR-BANK-ID (3).                     00
015200       MOVE 0                 TO ADDR-CUST-ID (3).                      00
015300       MOVE 3                 TO ADDR-LINE-NBR (3).                     00
015400       MOVE BANK-ADDR3         TO ADDR-LINE (3).                        00
015500                                                                        00
015600       MOVE 3                 TO VIRTUAL-REC-COUNT.                     00
015700                                                                        00
015800 CUST-ADDRESS.                                                          00
015900                                                                        00
016000* Move the database record into the CUSTOMER user work area so          00
016100* that we can reference individual data items.                          00
016200                                                                        00
016300       UNSTRING DB-RECORD INTO CUSTOMER                                 00
016400              FOR CUST-REC-CHARS.                                       00
016500                                                                        00
016600       IF CUST-LINES > 5                                                00
016700              MOVE 5 TO CUST-LINES.                                     00
016800       PERFORM MOVE-CUST-ADDR                                           00
016900              VARYING I FROM 1 BY 1                                     00
017000              UNTIL I > CUST-LINES.                                     00
017100       MOVE CUST-LINES TO VIRTUAL-REC-COUNT.                            00
017200                                                                        00
017300 MOVE-CUST-ADDR.                                                        00
017400       MOVE BANK-ID OF CUSTOMER TO ADDR-BANK-ID (I).                    00
017500       MOVE CUST-ID            TO ADDR-CUST-ID (I).                     00
017600       MOVE I                 TO ADDR-LINE-NBR (I).                     00
017700       MOVE CUST-ADDR (I)      TO ADDR-LINE (I).                        00
017800$$ VERSION 61.532.0000                                                  00
```

4 Altered Data Sets

In This Chapter

This chapter gives you programming tips and examples for altering data sets.

Overview

An altered data set is a DMSII data set to which you apply a data item conversion routine (custom reformatting routine) to reformat data items in a data set to different layouts.

NOTE: The Databridge API is not involved in altering data sets; however, ALGOL programming is required.

You can accomplish any of the following tasks by altering a data set:

- Flatten OCCURS clauses—Involves changing an occurring item to a series of individual items.

  However, if you plan to clone the data set with the Databridge Clients, you may want to use the Databridge Clients to flatten OCCURS clauses. Refer to the *Databridge Clients Administrator's Guide* for more information.
- Subdivide compound items—Allows the secondary database to search and retrieve data for the individual items rather than the compound item as a whole.
- Convert or format dates—Involves changing the date from one format to another.

  The Databridge Clients, however, provide date formats that are often less expensive to implement. Refer to the *Databridge Clients Administrator's Guide* for more information on how the Databridge Clients format and convert dates.
- Expand compressed data—Allows you to expand data that has been stored in a compression format (such as a digital picture) or a delimited format.
- Convert data in a proprietary format to a well-known format—Involves changing the data from one kind of format to another.
- Merge a list data items in to a single data item

To alter a data set, you must modify one of the provided sample reformatting procedures or write your own reformatting procedure and declare it to be internal or external to your tailored support library.

In addition, you must list (declare) the data items you want to alter in the ALTER section of the DBGenFormat parameter file. By making the ALTER declaration, you indicate which reformatting routines you want DBGenFormat to apply to the data items you named. Then, whenever a DBGenFormat-generated formatting routine (such as COMMAFORMAT or BINARYFORMAT) encounters data items from the ALTER declaration data set statement list, that DBGenFormat formatting routine calls the particular reformatting routine indicated in the ALTER declaration. See ALTER Declaration Syntax for more information about how to determine what kind of a reformatting routine you want to write. For example, if you are reformatting similar items, such as timestamps, you would typically use the same reformatting routine to reformat all of them.

Finally, you must compile a tailored support library and enter the name of the tailored support library in the appropriate Accessory parameter file. When this process is completed, the Accessory can use your altered data set.

The provided sample reformatting procedures are listed as follows:

- PATCH/DATABRIDGE/SAMPLE/SUPPORT/REFORMAT for internal reformats—Sample Internal Reformatting Procedure shows how to use an internal reformatting procedure.
- SYMBOL/DATABRIDGE/SAMPLE/REFORMAT for external reformats—Sample External Reformatting Procedure shows how to use an external reformatting procedure.

The following sections include altered data set examples:

- ALTER Declaration Syntax contains an example ALTER declaration.
- Example Altered Data Set for Flattening OCCURS shows how to use altered data sets to flatten OCCURS.

Altering a Data Set

To alter data sets, complete the following steps:

1. Read this entire chapter so that you get an understanding of how the code you write for your altered data sets relates to the actual ALTER declarations you make in the DBGenFormat file.

   See ALTER Restrictions for more information about making ALTER declarations.
2. Look at the data items you want to convert so you can get an idea of how many reformatting routines you need to code. Keep in mind that several ALTER declarations can call the same reformatting routine.
3. Familiarize yourself with the following samples, which illustrate several ways to apply reformatting procedures:
   - Sample Internal Reformatting Procedure
   - Sample External Reformatting Procedure
   - Example Altered Data Set for Flattening OCCURS
4. Use CANDE or another editor to retrieve the DBGenFormat parameter file DATA/GENFORMAT/SAMPLE/CONTROL.

   For a general description of the DBGenFormat parameter file, refer to the *Databridge Host Administrator's Guide*.
5. Rename the file, as follows:

   `DATA/GENFORMAT/`*`databasename`*`/CONTROL`

   where *databasename* is the name of the database that contains the items you are altering and for which you are creating the tailored support library.
6. Make the following declarations in the DBGenFormat parameter file (DATA/GENFORMAT/*databasename*/CONTROL):
   - Declare the reformatting procedure.

     See Declaring Internal and External Reformatting Procedures for instructions.
   - Declare all of the altered data sets.

     See ALTER Declaration Syntax for instructions.
7. Save DATA/GENFORMAT/*databasename*/CONTROL.

8 Write the reformatting procedure as follows:

NOTE: The reformatting routines must be in ALGOL; however, you can write routines that invoke a COBOL library.

If you	Do this
Declared an internal reformatting procedure	Write the reformatting procedure patch file, *and give it the name you specified in the DBGenFormat parameter file.* **NOTE:** It is recommended that your reformatting procedure use a case statement to identify the individual reformatting routines.
Declared an external reformatting procedure	Write the reformatting procedure library source file. Compile the library containing the external reformatting procedure as the name you specified in the DBGenFormat parameter file. **NOTE:** It is recommended that your reformatting procedure use a case statement to identify the individual reformatting routines. You do not have to complete the external library file before going on to the next step. You must, however, finish writing and compiling it before you run an Accessory that uses the external reformat.

9 Compile the tailored support library as follows:

```
START WFL/DATABRIDGE/COMP ("SUPPORT", "databasename" ["logicaldatabasename"])
```

Where	Is
"SUPPORT"	The literal that represents the DBSupport program The quotation marks are required.
"*databasename*"	The name of the database for which you are creating the tailored support library The database name can include a usercode and pack, which are used to locate the database DESCRIPTION file, as follows: "(*usercode*)*databasename* ON *packname*" The quotation marks are required.
"*logicaldata basename*"	The name of a logical database for which you are creating the tailored support library

This WFL compiles layout tables for each data set in the database designated by *databasename* or *logicaldatabasename*. This results in the new tailored support library titled as follows:

```
OBJECT/DATABRIDGE/SUPPORT/databasename
```

— or —

```
OBJECT/DATABRIDGE/SUPPORT/databasename/logicaldatabasename
```

These data set-specific layout tables contain the offsets and sizes of individual data items, including those in the ALTER data set declaration.

CAUTION: If you have two databases with the same name under different usercodes, *and* you are running Databridge from a third usercode, be careful when you create a tailored support library. In this case, the second library you compile overwrites the first, because Databridge strips the usercode and pack name from the database name to create the tailored support library title.

10 Enter the name of the tailored support library in the appropriate Accessory parameter file, as follows:

For	Do this
Databridge Clients	In the DBServer parameter file, enter the tailored support library name for the SUPPORT option. For more information, refer to the *Databridge Host Administrator's Guide*.
Other Accessories	In the Accessory parameter file, enter the tailored support library name for the SUPPORT option. For more information, refer to the *Databridge Host Administrator's Guide*.

What to Do Next

You can now use your altered data set(s) and run your Databridge Accessories, as usual.

Repeat these steps for each data set you want to alter.

If you encounter problems when creating or compiling your altered data set, see Chapter B, “Troubleshooting,” on page 173 for troubleshooting information. Chapter B, “Troubleshooting,” on page 173 provides specific troubleshooting tips for writing reformatting procedures and working with altered data sets.

ALTER Restrictions

You must be aware of the following restrictions when you use the ALTER declaration:

- If you alter a GROUP item, the reformatting routine must format the entire group.
- You cannot alter an item subordinate to a GROUP OCCURS. In this case you must alter the entire GROUP.
- If you alter an item with an OCCURS clause, the reformatting procedure must reformat all occurrences at once. (The source length is the total size for all occurrences.)
- If you alter an item in a data set that has more than one data item with the same name, only the last item is altered. (This can happen if a data item is found in more than one of the variable-format parts of a variable format data set.)
- If the reformatted item is to be signed, you must have at least one space between the "S" and the declared size, as in the following example:

```
ACCT-BALANCE NUMBER (S 11, 2);
```

- If the reformatted item is an occurring GROUP (or FIELD), then the OCCURS clause must immediately follow the word GROUP (or FIELD), as in the following example:

```
  MONTHLY-SUMMARY GROUP OCCURS 12
(SALES  NUMBER (8);
...
);
```

- Items altered to type IMAGE are treated as ALPHA items by the Databridge Clients, except that the Databridge Clients do not translate or interpret IMAGE items.
- To merge data items, the data items must be adjacent and must exist in the same parent group.

ALTER Declaration Syntax

You must make one ALTER declaration for each data set that contains data items you want to alter. The following is the ALTER declaration for the DBGenFormat parameter file (DATA/GENFORMAT/*databasename*/CONTROL):

```
ALTER datasetname
(
[uservalue]  originaldataitemname newitemtype(n)
[uservalue]  originaldataitemname newitemtype(n)
[uservalue]  dataitemname1, dataitemname2, ... AS newdataitemname
newitemtype(n) <-- This syntax specifies data items to be merged into one.
...
);
```

Where	Is
datasetname	The name of the data set from which you want to convert data items The data set name must match the data set name in the DASDL.
[*uservalue*] or [DEFINE *uservalue*] or [REDEFINE]	A value that indicates the type of alteration made by the reformatting procedure You must include the brackets. The user value corresponds to a reformatting routine and can be any integer greater than or equal to 0. It is usually less than 1024 so that the reformatting procedure can use it as a case value (as in the example reformatting procedures). You can use the same integer (and therefore call the same formatting routine) for more than one data item. If you are reformatting similar items (for example, timestamps) you would typically assign the same user value to each one so that the reformatting procedure uses the same code (reformatting routine) to reformat all of them. ***Note for merging data items:*** If [*uservalue*] is an integer, the Reformat routine is called with the source offset of the first data item and the total length of all of the items. A value of DEFINE introduces a virtual data item into the altered data set. DEFINE does not apply to merging data items. A value of REDEFINE redefines the data in place rather than using the reformatting procedure. Use the REDEFINE command to subdivide elementary items and flatten OCCURS (see Example Altered Data Set for Flattening OCCURS). Two qualifications exist for using REDEFINE, as follows: ◆ The REDEFINE size must be equal to the original size. ◆ The original item must be on a byte boundary if the target data type is required to be byte-aligned.

Where	Is
originaldataitemname	The name of the data item as it appears in the DASDL
newitemtype(n)	The new data type for the changed data item or merged data items where *newitemtype* is the new type, such as ALPHA, IMAGE, or NUMBER, and *n* is the size of the field (NUMBER items can have a scaling factor and sign, as in NUMBER (S 6,2). See Chapter C, "Virtual and Alter Data Item Types," on page 177 for Databridge-specific data item types you can specify.
dataitemname1, dataitemname2, dataitemnamex,	A comma-separated list of adjacent data items in the same parent group that will be merged into a new data item
newdataitemname	The name of the merged data item. This is the name that Accessories will see in place of the listed data items.

REDEFINE Errors

If the REDEFINE size differs from the original size, DBGenFormat displays the following error:

dataitem original size: *origsize* but REDEFINE size: *newsize*

If the original data item was not on a byte boundary but the REDEFINE data type requires it to start on a byte boundary (as for GROUP items), DBGenFormat displays the following error:

REDEFINE of *dataitem* requires byte-alignment

Example 1

This example demonstrates how to merge data items using the REDEFINE command and a reformatting routine coded in the ALGOL procedure.

Assume that the original DASDL for the ORDERINFO data set contains the following:

```
ORDERINFO DATASET
(
ORD-YY    NUMBER   (4);
ORD-MM    NUMBER   (2);
ORD-DD    NUMBER   (2);
...
ORD-CITY    ALPHA  (16);
ORD-STATE   ALPHA   (2);
ORD-ZIP     NUMBER  (4);
...
);
```

However, you want to merge the year, month, and day data items into one date item, and you want to merge the city, state and zipcode into one alphanumeric item.

In the DBGenFormat parameter file, you could write the ALTER declaration for the ORDERINFO data set as follows:

```
ALTER ORDERINFO
(
[REDEFINE] ORD-YY,ORD-MM, ORD-DD AS
        ORD-YYMMDD      NUMBER (YYMMDD);

[2] ORD-CITY, ORD-STATE, ORD-ZIP AS
         ORD-ADDR-CSZ    ALPHA (45);
);
```

Example 2

This example demonstrates how to alter a data set using the DEFINE command, the REDEFINE command, and reformatting routines coded in the ALGOL procedure.

Assume that the original DASDL for the BANK data set contains the following:

```
BANK DATASET
(
BANK-ID    NUMBER (4);
BANK-NAM   ALPHA  (11)  INITIALVALUE "BRANCH NAME";
BANK-ADDR  ALPHA  (30);
BANK-TS    REAL;        % timestamp
BANK-ROUTE NUMBER (9);
);
```

However, you want to use the ALTER declaration to change the BANK data items as follows:

- Change BANK-ID from NUMBER (4) to NUMBER (6)
- Change BANK-ADDR from ALPHA (30) to a group containing three elementary data items
- Change BANK-TS (timestamp) from REAL to ALPHA (30), which contains a readable date and time
- Change BANK-ROUTE from NUMBER (9), to a group containing three data items
- Add a BANK-PRES virtual data item of ALPHA (40)

In the DBGenFormat parameter file, you would write the ALTER declaration for the BANK data set as follows:

```
ALTER BANK
(
[1]        BANK-TS    ALPHA (30);  % was REAL
[2]        BANK-ID    NUMBER (6);  % was NUMBER (4)
[DEFINE 4] BANK-PRES  ALPHA (40);  % virtual
[REDEFINE] BANK-ADDR  GROUP        % was ALPHA (30)
          (
           BR-CITY    ALPHA (18);
           BR-STATE   ALPHA (2);
           BR-ZIP     ALPHA (10);
           );
[3]        BANK-ROUTE GROUP
           (
           BR-1  NUMBER (2);
           BR-2  NUMBER (3);
           BR-3  NUMBER (4);
           );
);
```

In this example, the DBGenFormat formatting routines call the reformatting procedure to reformat TS, BANK-ID, and BANK-ROUTE.

Each [*uservalue*] in the ALTER declaration corresponds to a specific reformatting routine (that you have coded) in the reformat procedure. When the DBGenFormat formatting routines receive a BANK record, they call the reformatting procedure (once for each data item) with the following information:

- The value that corresponds to the specific reformatting routine in the reformat procedure (which is 1, 2, or 3 in this example)
- The original location and size of BANK-ID, BANK-ROUTE, and BANK-TS
- The location and size of where the reformatting procedure should place BANK-ID, BANK-ROUTE, and TS

When the formatting routines call the reformatting procedure for the virtual item BANK-PRES, they supply the 4 as the [*uservalue*], but the source offset and size is 0 because there is no source item. The reformatting routine must retrieve the data from some external source (such as another database, file, and so on) and copy it into the destination array.

The DBGenFormat formatting routines do not call any reformatting routines for BANK-ADDR because a REDEFINE command redefines the data in place.

NOTE: BANK-ROUTE cannot be a REDEFINE because GROUP items are multiples of whole bytes, and, therefore, are required to be byte-aligned.

Declaring Internal and External Reformatting Procedures

Reformatting procedures for altered data sets must be declared as internal and external reformats in the DBGenFormat parameter file. Declare the reformatting procedures in the DBGenFormat parameter file using the syntax below in Declaring Internal and External Reformatting Procedures" and Declaring External Reformats."

Consider the following information before you choose whether to declare an internal or external reformat:

Internal Reformat Description	External Reformat Description
Internal reformats are compiled as patches to your tailored support library. This requires that you recompile DBSupport via WFL/ DATABRIDGE/COMP each time you update the internal reformatting procedure.	External reformats are linked at run-time to a user-written format library, so they can be recompiled any time without having to recompile DBSupport.
Internal reformats do not have to specify how to link to DBEngine or DBSupport.	External reformats must link to DBSupport and DBEngine at the proper time.
Internal reformats do not have to verify that their interface version matches DBEngine.	External reformat libraries must ensure that their interface version matches DBEngine.

Declaring Internal Reformats

To declare the altered data set reformatting procedure as an internal reformat, use the following syntax in the DBGenFormat parameter file (the comments in the file indicate where this declaration should go):

```
INTERNAL REFORMAT reformattingprocedure IN "patchfiletitle"
```

where *reformattingprocedure* is the name of a reformatting procedure, and *patchfiletitle* is the title of the ALGOL file that you created as a patch for DBSupport.

Declaring External Reformats

If you want to write your own ALGOL library for a reformat, you can reference it in the tailored support library by using the following syntax in the DBGenFormat parameter file (the comments in the file indicate where this declaration should go):

```
EXTERNAL REFORMAT reformattingprocedure IN "objectfilename"
```

where *reformattingprocedure* is the name you have given to the reformatting procedure and *objectfilename* is the file title of your compiled ALGOL library code.

Writing Altered Data Set Reformatting Procedures

This section describes how to ensure that your reformatting procedure is communicating with the DBEngine and DBSupport libraries.

Writing an Internal Reformatting Procedure

If you declared an internal reformat in DBGenFormat, you must write an ALGOL patch file containing the reformatting procedure that converts altered data items. The patch file may include global declarations in addition to the reformatting procedure itself. The patch file should not include the EXPORT declaration for the reformatting procedure. DBGenFormat automatically generates the appropriate EXPORT declaration.

See PATCH/DATABRIDGE/SAMPLE/SUPPORT/REFORMAT in Sample Internal Reformatting Procedure as an example of an internal reformatting procedure.

Writing an External Reformatting Procedure

If you declared an external reformat in DBGenFormat, you must write your own library program that contains the reformatting procedure and does the following:

- Sets the $ INCLUDE_ENGINE option (and the $INCLUDE_SUPPORT option if you call any DBSupport entry points) and includes SYMBOL/DATABRIDGE/INTERFACE using the following ALGOL $INCLUDE statements:

  ```
  $SET INCLUDE_ENGINE
  $INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
  ```

  or

  ```
  $SET INCLUDE_ENGINE INCLUDE_SUPPORT
  $INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
  ```

  NOTE: Do not invoke the DBLINKENGINE define in your library. DBSupport automatically links your library to the correct instance of DBEngine. Do not attempt to call any DBEngine entry points before the library freezes. Otherwise, your library will link to a different instance of DBEngine than the Accessory, and it might return incorrect information and errors.

- Calls DBINTERFACEVERSION to verify that your program was compiled against the same API file (SYMBOL/DATABRIDGE/INTERFACE) as DBEngine.

NOTE: Do not call DBINITIALIZE. DBINITIALIZE will undo the initialization that the Accessory has already done.

See OBJECT/DATABRIDGE/REFORMAT in Sample External Reformatting Procedure as an example of an external reformatting procedure.

Sample Internal Reformatting Procedure

The ALGOL source code for this example is as follows:

```
 $ SET OMIT                                                                  09900000
------------------------------------------------------------------------------09900100
                                                                             09900130
 Module: PATCH/DATABRIDGE/SAMPLE/SUPPORT/REFORMAT                            09900140
                                                                             09900150
 Project: Databridge                                                         09900160
                                                                             09900170
 Description: Databridge Sample Reformatting Patch                           09900180
                                                                             09900190
Copyright (c) 2012 - 2017 Attachmate Corporation, a Micro Focus company.09900290
                                                                             09900430
------------------------------------------------------------------------------09900440
                                                                             09902000
        Modification history                                                 09902100
        --------------------                                                 09902200
                                                                             09902300
Version 30.001                                                               09902400
        Initial release.                                                     09902500
                                                                             09902600
        This is a sample patch to DBSupport for reformatting                 09902700
        data items in conjunction with the GenFormat ALTER construct.        09902800
        To include this patch in DBSupport put the following declaration09902900
        in the GenFormat parameter file:                                     09903000
                                                                             09903100
       INTERNAL REFORMAT IN "PATCH/DATABRIDGE/SAMPLE/SUPPORT/REFORMAT" 09903200
                                                                             09903300
Version 41.471                                                               09903400
1       Added cases 6 and 7 to illustrate handling virtual data items        09903500
        declared with the [DEFINE n] syntax in GenFormat.                    09903600
        Case 6 also illustrates the necessary code to handle formatting 09903700
        a null record when the reformatting routine normally stores a        09903800
        constant value.                                                      09903900
                                                                             09904000
Version 41.474                                                               09904100
2       Added defines for 8-bit offsets and 8-bit sizes and changed          09904200
        examples accordingly.                                                09904250
                                                                             09904260
Version 61.001                                                               09904300
1       The Reformat routine now returns false (indicating failure) if a09905000
        fault is detected.                                                   09906000
End History                                                                  09999990
 $ POP OMIT                                                                  09999999
                                                                             50000000
                                                                             50002000
                                                                             50005000
string  TSMsg;  % timestamp message                                          50006000
                                                                             50007000
real    FaultNbr;                                                            50007100
ebcdic array                                                                 50007200
        FaultHistory [0:79];                                                 50007300
```

```
                                                                        50009000
boolean procedure Reformat (UserValue, UpdateInfo,                      50010000
%                 --------                                              50011000
                              SourceRec,  SourceOfs, SourceSz,          50012000
                              DestRec,    DestOfs,   DestSz);           50013000
                                                                        50014000
       % Custom reformatting of a data item. This user-written          50015000
       % procedure converts a data item used in a non-standard way into50016000
       % a "standard" data item defined in the GenFormat parameter file50017000
        % using the ALTER declaration.                                  50018000
                                                                        50019000
       % For example, a "days-since" data item might be converted to    50020000
        % a YYYYMMDD date.                                              50021000
                                                                        50022000
       % Returns true if item successfully reformatted.                 50023000
                                                                        50024000
       value   UserValue, SourceOfs, SourceSz, DestOfs, DestSz;         50025000
                                                                        50026000
       integer UserValue;                                               50027000
              % Input: user-specified value associated with the data    50028000
               %       item (from GenFormat)                            50029000
                                                                        50030000
       array   UpdateInfo [0];                                          50031000
              % Input: information describing the update                50032000
                                                                        50033000
       array   SourceRec [0];                                           50034000
              % Input: dataset record containing source item           50035000
                                                                        50036000
       integer SourceOfs;                                               50037000
              % Input: offset of the source item in SourceRec           50038000
               %       (4-bit digits)                                   50039000
                                                                        50040000
       integer SourceSz;                                                50041000
              % Input: size of the source item in SourceRec             50042000
               %       (4-bit digits)                                   50043000
                                                                        50044000
       array   DestRec [0];                                             50045000
              % Output: reformatted dataset record                      50046000
                                                                        50047000
       integer DestOfs;                                                 50048000
              % Input: offset of the destination item in DestRec        50049000
               %       (4-bit digits).                                  50050000
                                                                        50051000
       integer DestSz;                                                  50052000
              % Input: size of the destination item in DestRec          50053000
               %       (4-bit digits)                                   50054000
                                                                        50055000
       begin                                                            50056000
       hex array                                                        50057000
              Source4 [0] = SourceRec,                                  50058000
              Dest4   [0] = DestRec;                                    50059000
                                                                        50060000
       ebcdic array                                                     50061000
              Source8 [0] = SourceRec,                                  50062000
              Dest8   [0] = DestRec;                                    50063000
                                                                        50063100
       define  SourceSz4       = SourceSz #;                            50063200
       define  SourceSz8       = (SourceSz / 2) #;                      50063300
                                                                        50063400
       define  DestSz4         = DestSz #;                              50063500
       define  DestSz8         = (DestSz / 2) #;                        50063600
                                                                        50064000
       define  SourceOfs4      = SourceOfs #;                           50064100
```

```
        define  SourceOfs8      = (SourceOfs / 2) #;                    50064200
                                                                        50064300
        define  DestOfs4        = DestOfs #;                            50064400
        define  DestOfs8        = (DestOfs / 2) #;                      50064500
                                                                        50066000
        own integer                                                     50066200
                BankIDOfs;                                              50066250
        own integer                                                     50066300
                BankIDSz;                                               50066350
        EBCDIC value array                                              50066400
                BankName (80"BANK");                                    50066600
                                                                        50066800
        own boolean                                                     50067000
                Initialized;                                            50067050
                                                                        50067100
        procedure Initialize;                                           50067200
%                 ----------                                            50067400
                begin                                                   50067600
                array   ITEM_INFO [0 : II_ENTRY_SIZE - 1];              50067800
                integer BankStrNum;                                     50068000
                                                                        50068200
                % Get the offset and size of BANK-ID.                   50068400
                                                                        50068600
                DBStrNum (BankName, BankStrNum);                        50068800
                DBItemInfo (BankStrNum, 0, "BANK-ID", ITEM_INFO);       50069000
                BankIDOfs := ITEM_INFO [II_OFFSET];                     50069200
                BankIDSz  := ITEM_INFO [II_SIZE];                       50069400
                                                                        50069500
                Initialized := true;                                    50069600
                end Initialize;                                         50069800
                                                                        50070000
                                                                        50070200
        if ^ Initialized then                                           50070400
                begin                                                   50070600
                Initialize;                                             50070800
                end;                                                    50071000
                                                                        50071200
        on anyfault [FaultHistory: FaultNbr],                           50072100
                begin                                                   50072200
               DBDisplayFault ("Reformat: ", FaultNbr, FaultHistory);   50072300
                Reformat := false;                                      50072350
                end;                                                    50072400
                                                                        50072500
        Reformat := true;                                               50073000
                                                                        50074000
        case UserValue of                                               50075000
                begin                                                   50076000
                1:      % timestamp                                     50077000
                                                                        50078000
                                % call Engine to convert                50079000
                        DBTIMESTAMPMSG                                  50080000
                               (real (Source4 [SourceOfs4], SourceSz4),50081000
                                TSMsg);                                 50082000
                                                                        50083000
                                % see if it will fit                    50086000
                        if length (TSMsg) > DestSz8 then                50087000
                                begin                                   50088000
                                TSMsg := take (TSMsg, DestSz8);         50089000
                                end;                                    50090000
                                                                        50091000
                               % copy the timestamp message into dest   50092000
                        replace Dest8 [DestOfs8] by                     50093000
                                TSMsg,                                  50094000
```

```
                        " " for DestSz8 - length (TSMsg);       50095000
                                                                50096000
                2:      % bigger branch id                      50097000
                                                                50098000
                        replace Dest4 [DestOfs4] by             50099000
                                4"00",                          50100000
                                Source4 [SourceOfs4] for 4;     50101000
                                                                50102000
                3:      % split out branch address              50103000
                                                                50104000
                        replace Dest8 [DestOfs8] by             50105000
                                Source8 [SourceOfs8] for 30,    50106000
                                "-" for 30,                     50107000
                                "your town here      ",         50108000
                                "your region    ";              50109000
                                                                50110000
               6:       % Bank president name virtual data item.        50110020
                        % If this is a null record then we want this to 50110040
                        % be null also so that the client isn't         50110060
                        % confused about what is the null value.        50110080
                                                                        50110100
                        if NullRecord (UpdateInfo) then % null record   50110120
                                begin                                   50110140
                                replace Dest4 [DestOfs4] by 4"F"        50110160
                                        for DestSz4;                    50110180
                                end                                     50110200
                         else                                           50110220
                                begin                                   50110240
                                replace Dest8 [DestOfs8] by             50110260
                                        " " for DestSz8;                50110280
                                replace Dest8 [DestOfs8] by             50110300
                                        "Pres. Greenspan";              50110320
                                end;                                    50110340
                                                                        50110360
                7:      % Bank phone number virtual data item           50110380
                        % pieced together from a constant and the       50110400
                         % BANK-ID.                                     50110420
                                                                        50110440
                        replace Dest8 [DestOfs8] by                     50110460
                                "202-555-",                             50110480
                                Source4 [BankIDOfs] for 4               50110500
                                with HEXTOEBCDIC;                       50110520
                                                                        50110540
                else:           % unrecognized UserValue                50111000
                                % copy data as-is                       50111100
                                                                        50111200
                        replace Dest4 [DestOfs4] by                     50111300
                                Source4 [SourceOfs4]                    50111400
                                for min (SourceSz4, DestSz4);           50111500
                                                                        50111600
                        Reformat := false;                              50112000
                end;                                                    50113000
                                                                        50114000
        end Reformat;                                                   50115000
                                                                        50116000
```

Sample External Reformatting Procedure

The ALGOL source code for this example is as follows:

```
 $ SET OMIT                                                                  09000000
------------------------------------------------------------------------------09000100
                                                                             09000110
 Copyright (c) 2017 Attachmate Corporation, a Micro Focus company.           09000120
                                                                             09000130
 Module: SYMBOL/DATABRIDGE/SAMPLE/REFORMAT                                   09000140
                                                                             09000150
 Project: Databridge                                                         09000160
                                                                             09000170
 Description: Databridge Sample Reformatting Library                         09000180
                                                                             09000190
                                                                             09000430
------------------------------------------------------------------------------09000440
                                                                             09002000
This is a sample reformatting library to illustrate how to reformat          09002100
data items in conjunction with the GenFormat ALTER construct.                09002200
                                                                             09002300
        Modification history                                                 09002400
        --------------------                                                 09002500
                                                                             09002600
Version 25.001                                                               09002700
1       Initial release.                                                     09002800
                                                                             09002900
Version 30.001                                                               09003000
1       Added fault-trapping code to handle SEG ARRAY ERR, INVALID           09003100
        INDEX, etc. faults caused by this library.                           09003200
                                                                             09003300
2       If this routine receives an unrecognized UserValue it will           09003400
        now copy the source data to the destination without                  09003500
        modification. Previously it did nothing in this situation.           09003600
                                                                             09003700
Version 41.484                                                               09003800
1       Added cases 6 and 7 to illustrate handling virtual data items        09003900
        declared with the [DEFINE n] syntax in GenFormat.                    09004000
        Case 6 also illustrates the necessary code to handle formatting      09004100
        a null record when the reformatting routine normally stores a        09004200
        constant value.                                                      09004300
                                                                             09004400
2       Added defines for 8-bit offsets and 8-bit sizes and changed          09004500
        examples accordingly.                                                09004600
                                                                             09004700
Version 41.485                                                               09004800
3       Added initialization code to check DBInterface version.              09004900
                                                                             09005000
Version 51.501                                                               09005100
End History                                                                  09005200
 $ POP OMIT                                                                  09005300
                                                                             09005400
                                                                             09005600
 $ VERSION 06.003.0000                                                      09999900Version
                                                                             46000000
begin                                                                        46000100
                                                                             46000200
 $ SET INCLUDE_ENGINE                                                        46000300
 $ INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"                                     46000400
                                                                             46000500
string  TSMsg;  % timestamp message                                          46000600
                                                                             46000700
real    FaultNbr;                                                            46000800
ebcdic array                                                                 46000900
        FaultHistory [0:79];                                                 46001000
                                                                             46001100
boolean procedure Reformat (UserValue, UpdateInfo,                           46001200
```

```
%               --------                                                46001300
                            SourceRec,  SourceOfs, SourceSz,            46001400
                            DestRec,    DestOfs,   DestSz);             46001500
                                                                        46001600
        % Custom reformatting of a data item. This user-written         46001700
        % procedure converts a data item used in a non-standard way into46001800
        % a "standard" data item defined in the GenFormat parameter file46001900
         % using the ALTER declaration.                                 46002000
                                                                        46002100
        % For example, a "days-since" data item might be converted to   46002200
         % a YYYYMMDD date.                                             46002300
                                                                        46002400
        % Returns true if item successfully reformatted.               46002500
                                                                        46002600
        value   UserValue, SourceOfs, SourceSz, DestOfs, DestSz;        46002700
                                                                        46002800
        integer UserValue;                                              46002900
               % Input: user-specified value associated with the data   46003000
                %       item (from GenFormat)                           46003100
                                                                        46003200
        array   UpdateInfo [0];                                         46003300
               % Input: information describing the update               46003400
                                                                        46003500
        array   SourceRec [0];                                          46003600
               % Input: dataset record containing source item           46003700
                                                                        46003800
        integer SourceOfs;                                              46003900
               % Input: offset of the source item in SourceRec          46004000
                %       (4-bit digits)                                  46004100
                                                                        46004200
        integer SourceSz;                                               46004300
               % Input: size of the source item in SourceRec            46004400
                %       (4-bit digits)                                  46004500
                                                                        46004600
        array   DestRec [0];                                            46004700
               % Output: reformatted dataset record                     46004800
                                                                        46004900
        integer DestOfs;                                                46005000
               % Input: offset of the destination item in DestRec       46005100
                %       (4-bit digits).                                 46005200
                                                                        46005300
        integer DestSz;                                                 46005400
               % Input: size of the destination item in DestRec         46005500
                %       (4-bit digits)                                  46005600
                                                                        46005700
        begin                                                           46005800
        hex array                                                       46005900
               Source4 [0] = SourceRec,                                 46006000
               Dest4   [0] = DestRec;                                   46006100
                                                                        46006200
        ebcdic array                                                    46006300
               Source8 [0] = SourceRec,                                 46006400
               Dest8   [0] = DestRec;                                   46006500
                                                                        46006600
        define  SourceSz4       = SourceSz #;                           46006700
        define  SourceSz8       = (SourceSz / 2) #;                     46006800
                                                                        46006900
        define  DestSz4         = DestSz #;                             46007000
        define  DestSz8         = (DestSz / 2) #;                       46007100
                                                                        46007200
        define  SourceOfs4      = SourceOfs #;                          46007300
        define  SourceOfs8      = (SourceOfs / 2) #;                    46007400
                                                                        46007500
```

```
        define  DestOfs4        = DestOfs #;                            46007600
        define  DestOfs8        = (DestOfs / 2) #;                      46007700
                                                                        46007800
        own integer                                                     46007900
                BankIDOfs;                                              46008000
        own integer                                                     46008100
                BankIDSz;                                               46008200
        EBCDIC value array                                              46008300
                BankName (80"BANK");                                    46008400
                                                                        46008500
        define  NullRecord (UI) = (UI [UI_STRIDX] = 0) #;               46008600
                        % true if null record                           46008700
                                                                        46008800
        own boolean                                                     46008900
                Initialized;                                            46009000
                                                                        46009100
        procedure Initialize;                                           46009200
%                 ----------                                            46009300
                begin                                                   46009400
                array   ITEM_INFO [0 : II_ENTRY_SIZE - 1];              46009500
                integer BankStrNum;                                     46009600
                                                                        46009700
                DBMTYPE DBRslt;                                         46009800
                                                                        46009900
               DBRslt := DBInterfaceVersion (DBV_VERSION, "Reformat:");46010000
                if DBRslt NEQ DBM_OK then                               46010100
                        begin                                           46010200
                        DBDisplayMsg (DBRslt);                          46010300
                        MYSELF.STATUS := value (TERMINATED);            46010400
                        end;                                            46010500
                                                                        46010600
                        % Get the offset and size of BANK-ID.           46010700
                                                                        46010800
                DBStrNum (BankName, BankStrNum);                        46010900
                DBItemInfo (BankStrNum, 0, "BANK-ID", ITEM_INFO);       46011000
                BankIDOfs := ITEM_INFO [II_OFFSET];                     46011100
                BankIDSz  := ITEM_INFO [II_SIZE];                       46011200
                                                                        46011300
                Initialized := true;                                    46011400
                end Initialize;                                         46011500
                                                                        46011600
                                                                        46011700
        if ^ Initialized then                                           46011800
                begin                                                   46011900
                Initialize;                                             46012000
                end;                                                    46012100
                                                                        46012200
        on anyfault [FaultHistory: FaultNbr],                           46012300
                begin                                                   46012400
               DBDisplayFault ("xReformat: ", FaultNbr, FaultHistory); 46012500
                Reformat := false;                                      46012550
                end;                                                    46012600
                                                                        46012700
        Reformat := true;                                               46012800
                                                                        46012900
        case UserValue of                                               46013000
                begin                                                   46013100
                1:      % timestamp                                     46013200
                                                                        46013300
                                % call Engine to convert                 46013400
                        DBTIMESTAMPMSG                                  46013500
                               (real (Source4 [SourceOfs4], SourceSz4),46013600
                                TSMsg);                                 46013700
```

```
                                                                46013800
                % see if it will fit                            46013900
              if length (TSMsg) > DestSz8 then                  46014000
                begin                                           46014100
                TSMsg := take (TSMsg, DestSz8);                 46014200
                end;                                            46014300
                                                                46014400
               % copy the timestamp message into dest           46014500
              replace Dest8 [DestOfs8] by                       46014600
                TSMsg,                                          46014700
                " " for DestSz8 - length (TSMsg);               46014800
                                                                46014900
2:            % bigger branch id                                46015000
                                                                46015100
              replace Dest4 [DestOfs4] by                       46015200
                4"00",                                          46015300
                Source4 [SourceOfs4] for 4;                     46015400
                                                                46015500
3:            % split out branch address                        46015600
                                                                46015700
              replace Dest8 [DestOfs8] by                       46015800
                Source8 [SourceOfs8] for 30,                    46015900
                "-" for 30,                                     46016000
                "your town here      ",                         46016100
                "your region    ";                              46016200
                                                                46016300
6:            % Bank president name virtual data item.          46016400
              % If this is a null record then we want this to   46016500
              % be null also so that the client isn't           46016600
              % confused about what is the null value.          46016700
                                                                46016800
              if NullRecord (UpdateInfo) then % null record     46016900
                begin                                           46017000
                replace Dest4 [DestOfs4] by 4"F"                46017100
                        for DestSz4;                            46017200
                end                                             46017300
               else                                             46017400
                begin                                           46017500
                replace Dest8 [DestOfs8] by                     46017600
                        " " for DestSz8;                        46017700
                replace Dest8 [DestOfs8] by                     46017800
                        "Pres. Greenspan";                      46017900
                end;                                            46018000
                                                                46018100
7:            % Bank phone number virtual data item             46018200
              % pieced together from a constant and the         46018300
               % BANK-ID.                                       46018400
                                                                46018500
              replace Dest8 [DestOfs8] by                       46018600
                "202-555-",                                     46018700
                Source4 [BankIDOfs] for 4                       46018800
                with HEXTOEBCDIC;                               46018900
                                                                46019000
else:           % unrecognized UserValue                        46019100
                % copy data as-is                               46019200
                                                                46019300
```

```
                    replace Dest4 [DestOfs4] by                    46019400
                           Source4 [SourceOfs4]                    46019500
                           for min (SourceSz4, DestSz4);           46019600
                                                                   46019700
                    Reformat := false;                             46019800
               end;                                                46019900
                                                                   46020000
                                                                   46020100
       end Reformat;                                               46020200
                                                                   46020300
export  Reformat;                                                  46020400
                                                                   46020500
freeze (temporary);                                                46020600
end.                                                               46020700
```

Example Altered Data Set for Flattening OCCURS

This section shows you the declarations you must make in order to flatten OCCURS using a REDEFINE command. Notice that no reformatting routines are used.

DASDL Declaration

This sample is the original DASDL declaration.

```
G DATA SET
       (G-1   GROUP OCCURS 2
              (G-ALPHA        ALPHA (10);
               G-NUM          NUMBER (5);
              );

      G-KEY  ALPHA (10);
       );
```

ALTER Declaration in DBGenFormat

Make the following declaration in DATA/GENFORMAT/*databasename*/CONTROL:

```
ALTER G
     (
      [REDEFINE] G-1 GROUP
                 (G-ALPHA-1    ALPHA (10);
                 G-NUM-1       NUMBER (5);
                 G-ALPHA-2     ALPHA (10);
                 G-NUM-2       NUMBER (5);
                 );
      );
```

5 Formatting Procedures

In This Chapter

This chapter explains how to customize the format in which Databridge outputs data set records and use those custom formats with DBSpan, DBSnapshot, or a user-written Databridge Accessory.

Overview

Record formatting procedures allow you to customize the format in which Databridge outputs data set records and use those custom formats with DBSpan, DBSnapshot, or a user-written Databridge Accessory. If you want to reformat data items in a data set to different layouts, see Chapter 4, “Altered Data Sets,” on page 111.

Sample Files

Sample ALGOL External Formatting Procedure contains the sample external formatting procedure, SYMBOL/DATABRIDGE/SAMPLE/ENCRYPT.

Using Custom Formatting Procedures

To customize how you want to output your data set records using a formatting procedure, complete the following steps:

1. Read this entire chapter so that you get an understanding of how the code you write for your formatting procedure relates to the actual FORMAT declarations you make in the DBGenFormat file.
2. Look at the data records you want to convert so you can get an idea of how many formatting routines you need to code.
3. Familiarize yourself with the sample in Sample ALGOL External Formatting Procedure, which illustrate how to write a formatting procedure.
4. Use CANDE or another editor to retrieve the DBGenFormat parameter file DATA/GENFORMAT/SAMPLE/CONTROL.

   For a general description of the DBGenFormat parameter file, refer to the *Databridge Host Administrator's Guide*.
5. Rename the file, as follows:

   `DATA/GENFORMAT/`*`databasename`*`/CONTROL`

   where *databasename* is the name of the database for which you are creating the tailored support library.
6. Declare the formatting procedure. See Declaring Internal and External Formatting Procedures for more information.
7. Save `DATA/GENFORMAT/`*`databasename`*`/CONTROL`.
8. Write the formatting routine as follows:

If you	Do this
Declared an internal formatting procedure	Write the formatting procedure patch file that contains the formatting procedure. For details, see Writing an Internal Formatting Routine.
Declared an external formatting procedure	Write the formatting procedure library source file that contains the formatting routine. For details, see Writing an External Formatting Routine. You do not have to complete the external library file before going on to the next step. You must, however, finish writing and compiling it before you run an Accessory that uses the external format.

9 If you wrote an external formatting procedure (SYMBOL/DATABRIDGE/*formattingroutine*), compile it as OBJECT/DATABRIDGE/*formattingroutine* or whatever you called it in the DBGenFormat parameter file.

10 Compile the tailored support library, as follows:

```
START WFL/DATABRIDGE/COMP ("SUPPORT",
"databasename" ["logicaldatabasename"])
```

Where	Is
"SUPPORT"	The literal that represents the DBSupport program The quotation marks are required.
"*databasename*"	The name of the database for which you are creating the tailored support library The database name can include a usercode and pack, which are used to locate the database DESCRIPTION file, as follows: "(*usercode*)*databasename* ON *packname*" The quotation marks are required.
"*logicaldatabasename*"	The name of a logical database for which you are creating the tailored support library

This WFL compiles layout tables for each data set in the database designated by *databasename* or *logicaldatabasename*. This results in the new tailored support library titled as follows:

OBJECT/DATABRIDGE/SUPPORT/*databasename*

— or —

OBJECT/DATABRIDGE/SUPPORT/*databasename*/*logicaldatabasename*

These data set-specific layout tables contain the offsets and sizes of individual data items.

CAUTION: If you have two databases with the same name under different usercodes, *and* you are running Databridge from a third usercode, be careful when you create a tailored support library. In this case, the second library you compile overwrites the first, because Databridge strips the usercode and pack name from the database name to create the tailored support library title.

11 In the Accessory parameter file, enter the tailored support library name for the SUPPORT option and enter the ALGOL formatting procedure name for the FORMAT option.

For more information, refer to the *Databridge Host Administrator's Guide*.

What to Do Next

You can now run your Databridge Accessories as usual.

Repeat these steps for each internal or external formatting procedure you want to use.

If you encounter problems, see Chapter B, “Troubleshooting,” on page 173 for troubleshooting information. Chapter B, “Troubleshooting,” on page 173 provides specific troubleshooting tips for writing formatting procedures.

Declaring Internal and External Formatting Procedures

Formatting procedures must be declared as an internal or external format in the DBGenFormat parameter file. Declare the formatting procedure in the DBGenFormat parameter file using the syntax below in Declaring Internal Formats" and Declaring External Formats."

Consider the following information before you choose whether to declare an internal or external formatting procedure:

Internal Format Description	External Format Description
Internal formats are compiled as patches to your tailored support library. This requires that you recompile DBSupport via WFL/ DATABRIDGE/COMP each time you update the internal formatting routine.	External formats are linked at run-time to a user-written format library, so they can be recompiled any time without having to recompile DBSupport.
Internal formats do not have to specify how to link to DBEngine or DBSupport.	External formats must link to DBSupport and DBEngine at the proper time.
Internal formats do not have to verify that their interface version matches DBEngine.	External format libraries must ensure that their interface version matches DBEngine.

Declaring Internal Formats

To declare the formatting procedure as an internal format, use the following syntax in the DBGenFormat parameter file:

```
INTERNAL FORMAT formattingprocedure IN "patchfiletitle"
```

where *formattingprocedure* is the formatting procedure that you declared, and *patchfiletitle* is the title of the ALGOL patch file containing the internal formatting procedure that you created.

Declaring External Formats

If you want to write your own ALGOL library for a format, you can reference it in the tailored support library by using the following syntax in the DBGenFormat parameter file:

```
EXTERNAL FORMAT formattingprocedure IN "objectfilename"
```

where *formattingprocedure* is the name you have given to the external formatting procedure and *objectfilename* is the file title of your compiled ALGOL library program.

Writing Formatting Routines

Although you must code the formatting procedure in ALGOL, you can code it to call a COBOL library that actually formats the data set record.

Initializing the Formatting Routine

You must initialize your formatting routine the first time it is called. Initializing your formatting routine allows you to obtain information, such as structure numbers and indexes. Your formatting routine uses this information to identify and format records. You can use the following entry points to obtain this information:

- DBSTRIDX.
- DBDATASETINFO.
- DBFILTEREDSTRNUM.

Writing an Internal Formatting Routine

If you declared an internal format in DBGenFormat, you must write an ALGOL patch file containing the formatting procedure. The patch file may include global declarations in addition to the formatting procedure itself. The patch file should not include the EXPORT declaration for the formatting routine. DBGenFormat automatically generates the appropriate EXPORT declaration.

Writing an External Formatting Routine

If you declared an external format in DBGenFormat, you must write your own library that contains the formatting procedure and does the following:

- Sets the $ INCLUDE_ENGINE option (and the $INCLUDE_SUPPORT option if you call any DBSupport entry points) and includes SYMBOL/DATABRIDGE/INTERFACE using the following ALGOL $INCLUDE statements:

  ```
  $SET INCLUDE_ENGINE
  $INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
  ```

  or

  ```
  $SET INCLUDE_ENGINE INCLUDE_SUPPORT
  $INCLUDE "SYMBOL/DATABRIDGE/INTERFACE"
  ```

  NOTE: Do not invoke the DBLINKENGINE define to link to DBEngine because DBSupport automatically links your library to the correct instance of DBEngine. If you invoke the DBLINKENGINE define before the library freezes, your library will link to a different instance of DBEngine than the Accessory, and it might return incorrect information and errors.

  Do not try to call any entry points before your library freezes because it gets linked to a different instance of DBEngine and/or DBSupport.

- Calls DBINTERFACEVERSION to verify that your program was compiled against the same API file (SYMBOL/DATABRIDGE/INTERFACE) as DBEngine.

  NOTE: Do not call DBINITIALIZE. DBINITIALIZE will undo the initialization that the Accessory has already done.

The following example shows how to call DBINTERFACEVERSION:

```
DBMTYPE DBRslt;

DBRslt := DBInterfaceVersion (DBV_VERSION, "MyFormat:");
if DBRslt NEQ DBM_OK then
        begin
        DBDisplayMsg (DBRslt);
        MYSELF.STATUS := value (TERMINATED);
        end;
```

See SYMBOL/DATABRIDGE/SAMPLE/ENCRYPT in Sample ALGOL External Formatting Procedure as an example of an external formatting routine.

Calling a COBOL Library

If your formatting routine calls a COBOL formatting program that is compiled with $ FEDLEVEL=5, then you must do the following where the library is invoked:

In the COBOL program's entry point declaration, specify the ACTUALNAME to match the PROGRAM-ID name in the COBOL program. For example, the sample COBOL program EXTRACTADDRESS has the following:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   EXTRACTADDRESS.
```

The declaration of the COBOL program's entry point in the ALGOL formatting routine would look like the following:

```
procedure ExtractAddress (...);
   library ExtractAddressLib (ACTUALNAME = "EXTRACTADDRESS");
```

See the declaration of EXTRACTADDRESS in Sample ALGOL Virtual Transform Procedure.

Sample ALGOL External Formatting Procedure

This formatting procedure (SYMBOL/DATABRIDGE/SAMPLE/ENCRYPT) illustrates how to write an external format that encrypts each update record using a translate table. The program that reads these records must reverse the translation to see the original values.

The DBGenFormat declaration for this example is as follows:

```
EXTERNAL FORMAT ENCRYPT IN "OBJECT/DATABRIDGE/SAMPLE/ENCRYPT"
```

The ALGOL source code (SYMBOL/DATABRIDGE/SAMPLE/ENCRYPT) for this example is as follows:

```
 $ SET OMIT                                                                 09000000
------------------------------------------------------------------------------09000100
                                                                            09000110
 Copyright (c) 2017 Attachmate Corporation, a Micro Focus company.          09000120
                                                                            09000130
 Module: SYMBOL/DATABRIDGE/SAMPLE/ENCRYPT                                   09000140
                                                                            09000150
 Project: Databridge                                                        09000160
                                                                            09000170
 Description: Databridge Encryption Format Library Program                  09000180
                                                                            09000190
                                                                            09000430
------------------------------------------------------------------------------09000440
                                                                            09002000
                                                                            09002100
        Modification history                                                09002200
        --------------------                                                09002300
                                                                            09002400
Version 41.473                                                              09002500
        Initial version.                                                    09002600
                                                                            09002700
       This sample Databridge library program illustrates how to write      09002800
       an external format. It uses a translate table to perform a           09002900
       simple encryption on each update record. A program reading such      09003000
       records would have to reverse the translation to see the             09003100
       original values.                                                     09003200
                                                                            09003300
Version 41.485                                                              09003400
1       Added DBInterface compatibility check.                              09003500
                                                                            09003600
End History                                                                 09003700
 $ POP OMIT                                                                 09003800
                                                                            09003900
                                                                            09004100
 $ VERSION 06.003.0000                                                     09999900Version
 $ SET SEQ 40000000                                                         40000000
                                                                            40001000
begin                                                                       40002000
                                                                            40003000
 $ set INCLUDE_ENGINE                                                       40004000
 $ include "SYMBOL/DATABRIDGE/INTERFACE"                                    40005000
                                                                            40006000
translatetable                                                              40007000
        Encryption                                                          40008000
                (                                                           40009000
               48"000102030405060708090A0B0C0D0E0F"                         40010000
               48"101112131415161718191A1B1C1D1E1F"                         40011000
               48"202122232425262728292A2B2C2D2E2F"                         40012000
               48"303132333435363738393A3B3C3D3E3F"                         40013000
               48"404142434445464748494A4B4C4D4E4F"                         40014000
               48"505152535455565758595A5B5C5D5E5F"                         40015000
               48"606162636465666768696A6B6C6D6E6F"                         40016000
               48"707172737475767778797A7B7C7D7E7F"                         40017000
               48"808182838485868788898A8B8C8D8E8F"                         40018000
               48"909192939495969798999A9B9C9D9E9F"                         40019000
               48"A0A1A2A3A4A5A6A7A8A9AAABACADAEAF"                         40020000
               48"B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF"                         40021000
               48"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF"                         40022000
               48"D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF"                         40023000
               48"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF"                         40024000
               48"F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF"                         40025000
                                                                            40026000
                to                                                          40027000
```

```
                                                                        40028000
                48"FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0"                    40029000
                48"EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0"                    40030000
                48"DFDEDDDCDBDAD9D8D7D6D5D4D3D2D1D0"                    40031000
                48"CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0"                    40032000
                48"BFBEBDBCBBBAB9B8B7B6B5B4B3B2B1B0"                    40033000
                48"AFAEADACABAAA9A8A7A6A5A4A3A2A1A0"                    40034000
                48"9F9E9D9C9B9A99989796959493929190"                    40035000
                48"8F8E8D8C8B8A898887868584838281 80"                   40036000
                48"7F7E7D7C7B7A79787776757473727170"                    40037000
                48"6F6E6D6C6B6A69686766656463626160"                    40038000
                48"5F5E5D5C5B5A59585756555453525150"                    40039000
                48"4F4E4D4C4B4A49484746454443424140"                    40040000
                48"3F3E3D3C3B3A39383736353433323130"                    40041000
                48"2F2E2D2C2B2A29282726252423222120"                    40042000
                48"1F1E1D1C1B1A19181716151413121110"                    40043000
                48"0F0E0D0C0B0A09080706050403020100"                    40044000
                );                                                      40045000
                                                                        40046000
boolean FirstTime;                                                      40047000
                                                                        40048000
DBFormatHead [Encrypt];                                                 40049000
%            -------                                                    40050000
        begin                                                           40051000
        real    Len;                                                    40052000
        pointer pRec;                                                   40053000
                                                                        40054000
        if FirstTime then                                               40055000
                begin                                                   40056000
                DBMTYPE DBRslt;                                         40057000
                                                                        40058000
               DBRslt := DBInterfaceVersion (DBV_VERSION, "Encrypt:"); 40059000
                if DBRslt NEQ DBM_OK then                               40060000
                        begin                                           40061000
                        DBDisplayMsg (DBRslt);                          40062000
                        MYSELF.STATUS := value (TERMINATED);            40063000
                        end;                                            40064000
                FirstTime := false;                                     40065000
                end;                                                    40066000
                                                                        40067000
        Len := 6 * UPDATE_INFO [UI_RECSZ_WORDS];                        40068000
        pRec := pointer (UserRec);                                      40069000
                                                                        40070000
        case UPDATE_INFO [UI_UPDATE_TYPE] of                            40071000
                begin                                                   40072000
                DBV_CREATE:                                             40073000
                DBV_DELETE:                                             40074000
                DBV_MODIFY:                                             40075000
                        replace pRec by pRec for Len                    40076000
                                with Encryption;                        40077000
                else:                                                   40078000
                end;                                                    40079000
                                                                        40080000
       Encrypt := Callback (pRec, Len, UPDATE_INFO, UserRec);           40081000
        end Encrypt;                                                    40082000
                                                                        40083000
export  Encrypt;                                                        40084000
                                                                        40085000
FirstTime := true;                                                      40086000
freeze (temporary);                                                     40087000
end.                                                                    40088000
```

6 Error Handling Routines

In This Chapter

This chapter explains how to write an ALGOL error handling routine.

Overview

DBGenFormat supports error handling routines that analyze, log, and display errors and determine how Databridge Accessories respond to those errors.

To use an error handling routine, you must write a patch file containing the error handling code and specify the error handling routine in the DBGenFormat parameter file.

Whenever an error occurs, the Databridge Accessory calls the DBERRORMANAGER entry point in DBSupport. The error manager procedure calls the user-written error handler to analyze the error and handle it. For example, the error handling routine might write the error to a log, send an e-mail message, or reload a missing audit file. It then returns an EMATYPE value that tells the Accessory what to do: ignore, retry, or fail.

NOTE: If you do not use an error handler patch, the default error handler in DBSupport returns DBV_Default to the Accessory, indicating that the Accessory can decide whether the error is fatal, should be retried, or should be ignored.

DBERRORMANAGER returns the following EMATYPE values:

EMATYPE	Description
DBV_Default	The Accessory decides what to do with the error.
DBV_Fatal	The Accessory terminates.
DBV_Ignore	The Accessory ignores the error and continues.
DBV_Retry	The Accessory retries the operation.

Error handling routines must use the DBErrorManagerHead heading defined in SYMBOL/DATABRIDGE/INTERFACE. While the patch file must be written in ALGOL, the error handling routine could call a COBOL program to perform the actual error handling.

The patch file can contain declarations global to the error manager procedure. See the sample error manager patch called PATCH/DATABRIDGE/SAMPLE/SUPPORT/ERRORHANDLER.

Writing an Error Handling Routine

To write an error handling routine, complete the following steps:

1. Read this entire chapter so that you get an understanding of what you can make your error handling routine do. For instance, the error handling routine can do the following:
   - Determine what errors the Accessory can handle
   - Determine what the Accessory can do in response to errors (analyze them, log them, display them, continue processing, terminate, and so on)
2. Familiarize yourself with the error handling sample in Sample Error Handling Routine.
3. Use CANDE or another editor to retrieve the DBGenFormat parameter file DATA/GENFORMAT/SAMPLE/CONTROL.

   For a general description of the DBGenFormat parameter file, refer to the *Databridge Host Administrator's Guide*.
4. Rename the file, as follows:

   ```
   DATA/GENFORMAT/databasename/CONTROL
   ```

   where *databasename* is the name of the database for which you are creating the error handling routine.

   NOTE: If your error handler does not use any database-specific information and you do not need a tailored DBSupport library for any other reason, (for example, because you use filler substitutions) you can put the error handler routine in the non-tailored DBSupport library.

   In this case, skip this step, save DATA/GENFORMAT/SAMPLE/CONTROL in step Step 6 on page 138, use START WFL/DATABRIDGE/COMP ("SUPPORT") in step Step 8 on page 138, and skip step Step 9 on page 139.

   ---
5. Declare the patch file that contains the error handling routine, as follows:

   ```
   ERROR MANAGER errormanagername IN "patchfiletitle"
   ```

   where *errormanagername* is the name of the error handling routine and *patchfiletitle* is the name of the patch file that contains the error handling routine.
6. Save `DATA/GENFORMAT/databasename/CONTROL`.
7. Write the error handling routine.

   A sample error handling routine is described in Sample Error Handling Routine.
8. Compile the tailored support library, as follows:

   ```
   START WFL/DATABRIDGE/COMP ("SUPPORT", "databasename" ["logicaldatabasename"])
   ```

Where	Is
"SUPPORT"	The literal that represents the DBSupport library The quotation marks are required.
"*databasename*"	The name of the database for which you are creating the tailored support library The database name can include a usercode and pack, which are used to locate the database DESCRIPTION file, as follows: "(*usercode*)*databasename* ON *packname*" The quotation marks are required.
"*logicaldatabasename*"	The name of a logical database for which you are creating the tailored support library

This WFL compiles layout tables for each data set in the database designated by databasename or logicaldatabasename. This results in the new tailored support library titled as follows:

OBJECT/DATABRIDGE/SUPPORT/*databasename*

— or —

OBJECT/DATABRIDGE/SUPPORT/databasename/*logicaldatabasename*

These data set-specific layout tables contain the offsets and sizes of individual data items.

CAUTION: If you have two databases with the same name under different usercodes, *and* you are running Databridge from a third usercode, be careful when you create a tailored support library. In this case, the second library you compile overwrites the first, because Databridge strips the usercode and pack name from the database name to create the tailored support library title.

9 In the Accessory parameter file, enter the tailored support library name for the SUPPORT opt ion as follows:

For	Do this
Databridge Clients	In the DBServer parameter file, enter the tailored support library name for the SUPPORT option. For more information, refer to the *Databridge Host Administrator's Guide*.
Other Accessories	In the Accessory parameter file, enter the tailored support library name for the SUPPORT option. For more information, refer to the *Databridge Host Administrator's Guide*.

What to Do Next

You can now use your error handling routine and run your Databridge Accessories as usual.

Sample Error Handling Routine

The sample error handling patch PATCH/DATABRIDGE/SAMPLE/SUPPORT/ERRORHANDLER, does the following:

1. Uses the DBErrorManager head included in SYMBOL/DATABRIDGE/INTERFACE and documented in Chapter A, “Types, Values, and Array Layouts,” on page 145.
2. Writes the error message to the system SUMLOG if the Accessory is privileged; otherwise, it displays the message.
3. Determines whether or not a missing audit file caused the error, and if that is the case, starts a WFL to recopy the missing audit file. Then the error handler returns `DBV_Retry` to the Accessory to indicate that it should retry whatever it was doing when it got the error message.

   The ALGOL source code for this example is as follows.

NOTE: Read through this patch carefully before implementing it. Its main purpose is to *illustrate* ways to handle errors as a basis for writing your own custom error handling routine.

This routine uses a job called WFL/RELOAD/AUDIT, to copy an audit file, but WFL/RELOAD/AUDIT is not included on the Databridge release medium.

```
                                                                      09000000
 $ SET OMIT                                                           09000100
------------------------------------------------------------------------09000200
                                                                      09000230
 Module: PATCH/DATABRIDGE/SAMPLE/SUPPORT/ERRORHANDLER                  09000240
                                                                      09000250
 Project: Databridge                                                  09000260
                                                                      09000270
 Description: Databridge Example Error Handler patch                   09000280
                                                                      09000290
Copyright (c) 2012 - 2017 Attachmate Corporation, a Micro Focus company.09000390
                                                                      09000530
------------------------------------------------------------------------09000540
                                                                      09002000
               Modification history                                   09002100
               --------------------                                   09002200
                                                                      09002300
Version 41.471                                                        09002400
1      Initial release.                                               09002500
                                                                      09002600
       Example error manager routine. This is a patch                 09002700
       to SYMBOL/DATABRIDGE/SUPPORT and can be included by inserting   09002800
       the following declaration in DATA/GENFORMAT/<database>/CONTROL: 09002900
                                                                      09003000
               ERROR MANAGER ERRORHANDLER                             09003100
                  IN "PATCH/DATABRIDGE/SAMPLE/SUPPORT/ERRORHANDLER"   09003200
                                                                      09003300
       Warning: Read through this patch carefully before implementing  09003400
       it. Its main purpose is to *illustrate* some different ways to  09003500
       handle errors, including writing to the SUMLOG and starting a   09003600
       job called WFL/RELOAD/AUDIT, which is not part of the release,  09003700
       to copy an audit file from somewhere.                          09003800
                                                                      09003900
       Use this patch as a basis for writing your own custom error     09004000
       handling routine. If all you want to do is display error        09004100
       messages then simply use the default error handler that is      09004200
       built into DBSupport.                                          09004300
                                                                      09004400
End History                                                           09004500
```

```
 $ POP OMIT                                                                09004600
                                                                           70004700
                                                                           70004800
boolean IAmPrivileged;                                                     70004900
                                                                           70005000
array   ErrScratch [0 : 29];                                              70005100
interlock                                                                  70005200
        ErrorHandlerLock;                                                  70005300
                                                                           70005400
DBErrorManagerHead [ErrorHandler];                                         70005500
%                  ------------                                            70005600
                                                                           70005700
        begin                                                              70005800
        own boolean                                                        70005900
                Initialized;                                               70006000
                                                                           70006100
        procedure Initialize;                                              70006200
        %         ----------                                               70006300
                begin                                                      70006400
                IAmPrivileged := DBPrivileged;                             70006500
                                                                           70006600
                Initialized := true;                                       70006700
                end Initialize;                                            70006800
                                                                           70006900
        procedure LogComment;                                              70007000
        %         ----------                                               70007100
                begin                                                      70007200
                % Write a comment to the SUMLOG.                           70007300
                % The log entry will look like:                           70007400
               % SETSTATUS CALL: LC : DBTwin: DBM116: Unknown filter       70007500
                %          name: EXAMPLE2.                                 70007600
                                                                           70007700
                define  LenLoc = 2 #;                                      70007800
               define  pLogStart = pointer (ErrScratch [LenLoc + 1]) #;    70007900
                define  Build = replace pLog : pLog by #;                  70008000
                pointer pLog;                                              70008100
                boolean SSResult;                                          70008200
                                                                           70008300
                pLog := pLogStart;                                         70008400
 $ set omit                                                                70008500
                case AccessoryID of                                        70008600
                        begin                                              70008700
                        DBV_Span:                                          70008800
                                Build "DBSpan";                            70008900
                        DBV_Snapshot:                                      70009000
                                Build "DBSnapshot";                        70009100
                        DBV_Server:                                        70009200
                                Build "DBServer";                          70009300
                        DBV_Tanker:                                        70009400
                                Build "DBTanker";                          70009500
                        DBV_Lister:                                        70009600
                                Build "DBLister";                          70009700
                        DBV_Twin:                                          70009800
                                Build "DBTwin";                            70009900
                        DBV_GenFormat:                                     70010000
                                Build "DBGenFormat";                       70010100
                        DBV_AuditTimer:                                    70010200
                                Build "DBAuditTimer";                      70010300
                        DBV_TwinInit:                                      70010400
                                Build "DBTwinInitialize";                  70010500
                        else:                                              70010600
                                Build "Accessory";                         70010700
                        end;                                               70010800
```

```
$ pop omit                                                              70010900
              Build ": ", pErrMsg for ErrMsgLen;                        70011000
                                                                        70011100
             ErrScratch [0] := LenLoc;  % location of length word      70011200
              ErrScratch [LenLoc] := offset (pLog)                      70011300
                                   - offset (pLogStart);                70011400
             SSResult := SETSTATUS (2, 26, 0, ErrScratch);              70011500
              end LogComment;                                           70011600
                                                                        70011700
                                                                        70011800
       EMATYPE procedure AuditUnavailable;                              70011900
       %                ----------------                                70012000
                                                                        70012100
              % If the problem is a missing audit file, we'll run a     70012200
              % WFL job to recopy it.                                   70012300
                                                                        70012400
              begin                                                     70012500
              pointer pAFN;                                             70012600
              integer AFN;                                              70012700
              real    Rem;                                              70012800
              real    Len;                                              70012900
              boolean MissingFile;                                      70013000
                                                                        70013100
             % Scan the error message to find the AFN and determine      70013200
              % if the problem is that the file is not found.           70013300
                                                                        70013400
              pAFN := pErrMsg;                                          70013500
              Rem := ErrMsgLen;                                         70013600
              while Rem > 0 do                                          70013700
                     begin                                              70013800
                     scan pAFN : pAFN + Len for Rem : Rem - Len          70013900
                            while = " ";                                70014000
                     scan pAFN for Len : Rem                            70014100
                            until = " ";                                70014200
                     Len := Rem - Len;                                  70014300
                     if Len > 0 then                                    70014400
                            begin                                       70014500
                            if pAFN in Numbers for Len then             70014600
                                   begin                                70014700
                                   AFN := integer (pAFN, Len);          70014800
                                    end                                 70014900
                            else                                        70015000
                            if pAFN = "not found" then                  70015100
                                   begin                                70015200
                                   MissingFile := true;                 70015300
                                    end;                                70015400
                             end;                                       70015500
                      end;                                              70015600
                                                                        70015700
              if MissingFile then                                       70015800
                     begin                                              70015900
                     replace pointer (ErrScratch) by                    70016000
                            "START WFL/RELOAD/AUDIT (",                 70016100
                            AFN for * digits, ")", 0;                   70016200
                     zip with ErrScratch;                               70016300
                                                                        70016400
                     % Tell Accessory to retry                          70016500
                                                                        70016600
                     AuditUnavailable := DBV_Retry;                     70016700
                      end;                                              70016800
              end AuditUnavailable;                                     70016900
                                                                        70017000
                                                                        70017100
```

```
        lock (ErrorHandlerLock);                                        70017200
                                                                        70017300
        if ^ Initialized then                                           70017400
                begin                                                   70017500
                Initialize;                                             70017600
                end;                                                    70017700
                                                                        70017800
               % If we are running under a privileged usercode, we      70017900
               % can write it directly to the SUMLOG, else we'll do     70018000
                % a simple display.                                     70018100
                                                                        70018200
        if IAmPrivileged then                                           70018300
                begin                                                   70018400
                LogComment;                                             70018500
                end                                                     70018600
        else                                                            70018700
                begin                                                   70018800
                display (pErrMsg);                                      70018900
                end;                                                    70019000
                                                                        70019100
        unlock (ErrorHandlerLock);                                      70019200
                                                                        70019300
               % Check for missing audit file.                          70019400
                                                                        70019500
        if ErrNbr = DBM_AUD_UNAVAIL then                                70019600
                begin                                                   70019700
               ErrorHandler := AuditUnavailable;                        70019800
                end;                                                    70019900
        end ErrorHandler;                                               70020000
                                                                        70020100
% End of ErrorHandler patch                                             70020200
                                                                        70020300
```

A Types, Values, and Array Layouts

In This Appendix

This appendix documents the section of SYMBOL/DATABRIDGE/INTERFACE called "Types, values, and array layouts".

Overview

DBEngine and DBSupport entry points return the "Types, Values, and Array Layouts" of SYMBOL/DATABRIDGE/INTERFACE. The tables in this appendix explain those various values and arrays.

Each array (such as the UPDATE_INFO array) or set of information (such as the Data Error Types) has its own section in this appendix, and that section appears here in the same order in which it appears in the file. Most of these sections provide a table further explaining the information in SYMBOL/DATABRIDGE/INTERFACE, and most of them are cross-referenced from the appropriate section of Chapter 2, “Using the Databridge API,” on page 15.

Each section is listed here for your reference:

- FileXtract FileInfo Array Layout
- DBOUTPUTHEAD Procedure Heading
- DBFORMATHEAD Procedure Heading
- DBTRANSFORMHEAD Procedure Heading
- DBFILTERHEAD Procedure Heading
- DBERRORMANAGERHEAD Procedure Heading
- DBFILEREADERHEAD Procedure Heading
- File Attribute Mask Bits

DBEngine Entry Point Result Values

These values are error and status messages, which give you information about how DBEngine or any Accessory linked to DBEngine is running. The *Databridge Errors and Messages Guide* provides a list of these values (by number) and briefly explains each value.

Record Change Types

The following table provides additional information about record change types as they are documented in SYMBOL/DATABRIDGE/INTERFACE.

These values are found in the UPDATE_INFO [UI_UPDATE_TYPE] and specify the type of update.

Value	Description
DBV_HEADER	This is for internal use only
DBV_CREATE	This indicates that the record was created.
DBV_MODIFY_AFTER_IMAGE	This is the after-image of a modify. Use IS rather than = as in the following: `IF UI [UI_UPDATE_TYPE] IS DBV_MODIFY_AFTER_IMAGE THEN...`
DBV_DELETE	This indicates that the record was deleted.
DBV_MODIFY_BEFORE_IMAGE	This is the before-image of a modify. Use IS rather than = as in the following: `IF UI [UI_UPDATE_TYPE] IS DBV_MODIFY_BEFORE_IMAGE THEN...`
DBV_MODIFY	This indicates that the record was modified.
DBV_STATE	This is the state information. For a description of the array layout, see STATE_INFO Layout.
DBV_DOC	This is a documentation record. For a description of possible values, see Documentation Records.
DBV_MODIFY_BI	This is the alternate value for DBV_MODIFY_BEFORE_IMAGE.
DBV_MODIFY_AI	This is the alternate value for DBV_MODIFY_AFTER_IMAGE.
DBV_LINK_BI	This is the before-image of a link.

Value	Description
DBV_LINK_AI	This is the after-image of a link.

Error Manager Types

An Error Manager procedure (see DBERRORMANAGERHEAD) returns the following EMATYPE values:

EMATYPE	Integer	Description
DBV_Default	0	The Accessory decides what to do with the error.
DBV_Fatal	1	The Accessory terminates.
DBV_Ignore	2	The Accessory ignores the error and continues.
DBV_Retry	3	The Accessory retries the operation.

Accessory ID Numbers

Databridge Accessories are identified by the following Accessory ID numbers:

AIDTYPE	Integer
DBV_Span	1
DBV_Snapshot	2
DBV_Server	3
(not used)	4
DBV_Lister	5
DBV_Twin	6
DBV_GenFormat	7
DBV_AuditTimer	8

Documentation Records

The following table provides additional information about documentation records from SYMBOL/DATABRIDGE/INTERFACE. This information is returned when an entry point, such as the DBREADTRANGROUP entry point, requests information about records in the current transaction group.

Field	Description
DB_DOC_TYPE_F	If UI_UPDATE_TYPE in the UPDATE_INFO array is set to DBV_DOC, then the value in this field is one of the values in the following table.

Value	Description
DB_DOC_TYPE	The type of documentation record Possible values are as follows: ◆ DBV_DOC_BEG_TRAN—Indicates begin transaction (BTR) For values, see Begin Transaction. ◆ DBV_DOC_END_TRAN—Indicates end transaction (ETR) ◆ DBV_DOC_BEG_DB—Indicates database stack initiate (DBSI) ◆ DBV_DOC_END_DB—Indicates database stack terminate (DBST) ◆ DBV_DOC_BEG_REC—Indicates begin recovery (RECOV-1) ◆ DBV_DOC_END_REC—Indicates end recovery (RECOV-2) ◆ DBV_DOC_BEG_CP—Indicates begin controlpoint (BCP) ◆ DBV_DOC_END_CP—Indicates end controlpoint (ECP) ◆ DBV_DOC_SYNC—Indicates syncpoint (SPT) ◆ DBV_DOC_OPEN—Indicates restart data set open (RDSO) For values, see Restart Data Set Open and Close. ◆ DBV_DOC_CLOSE—Indicates restart data set close (RDSC) For values, see Restart Data Set Open and Close. ◆ DBV_DOC_INIT—Indicates structure discontinuity (STRDC) (because of initialization) ◆ DBV_DOC_REORG—Indicates STRDC (because of reorganization) ◆ DBV_DOC_AF_HEADER—Provides audit file header information For the layout of this header, see Audit File Header.
DB_DOC_MAX_SZ	The maximum size of the documentation record
DB_DOC_LONGTRAN	Indicates a Long Transaction is in progress. Valid for the following types: DBV_DOC_BEG_TRAN, DBV_DOC_END_TRAN, DBV_DOC_BEG_CP, DBV_DOC_END_CP, and DBV_DOC_SYNC.

Begin Transaction

The following table provides information about the layout of the begin transaction:

Field	Description
DB_DOC_BEG_TRAN_TC	This field contains the active transaction count if the DB_DOC_TYPE is set to DBV_DOC_BEG_TRAN.

End Transaction

The following table provides information about the layout of the end transaction:

Field	Description
DB_DOC_END_TRAN_TC	This field contains the active transaction count if the DB_DOC_TYPE is set to DBV_DOC_END_TRAN.

Restart Data Set Open and Close

The following table provides information about the restart data set open (beginning of task, BOT) and restart data set close (end of task, EOT):

Field	Description
DB_DOC_TASK_JOBNBR	The job number of the job that started the task
DB_DOC_TASK_MIXNBR	The mix number of the task
DB_DOC_TASK_NAMELEN	The length of the task name in bytes
DB_DOC_TASK_NAME	The task name for *n* words

Audit File Header

The following table provides the layout of information in the audit file header. This header precedes the first update from an audit file and any messages associated with the new file.

Field	Description
DB_DOC_AF_REC_QPT	DMSII recovery policy Possible values are as follows: ◆ 1—Recover to QPT ◆ 0—Recover to a syncpoint or other super quiet point
DB_DOC_AF_UPDATE_LVL	The audit file update level
DB_DOC_AF_DBTS	The audit file database timestamp
DB_DOC_AF_RELEASE	The DMSII release level of the audit file
DB_DOC_AF_MARK	The DMSII system software release (SSR) level
DB_DOC_AF_CYCLE	The DMSII SSR cycle
DB_DOC_AF_AUD_LVL	The audit level number

Value	Description
DB_DOC_AF_HEADER_SZ	The size of the audit file header information in words
DB_DOC_MAX_SZ	The maximum size of the documentation record

DBSETOPTION/DBRESETOPTION Run-Time Options

The following table provides additional information about the DBSETOPTION/DBRESETOPTION run-time options as they are documented in SYMBOL/DATABRIDGE/INTERFACE. These options are turned on and off by the DBSETOPTION and DBRESETOPTION entry points.

Value	Description
DBV_OP_BI_AI	Requests that updated database records be sent as a pair of before? and after-images
DBV_OP_DOC	Requests that documentation records be sent to the Accessory in addition to normal CREATES, UPDATES, and DELETES
DBV_OP_UNGROUPED	Specifies no COMMITs or ABORTs
DBV_OP_MODELESS	Specifies no reorganization or purge errors
DBV_OP_NO_WAIT	Specifies whether to wait on NO FILE conditions when an audit file is unavailable. When reset or defaulting to FALSE, the Accessory will enter the Waiting Entries state when a NO FILE is encountered. If set to TRUE, the Accessory receives a DBM_AUD_UNVAIL(7) result code when a NO FILE condition occurs. **WARNING:** Warning: DBPlus does not support this feature.
DBV_OP_FILTERED	(No longer used)
DBV_OP_QPT_GROUP	Requests COMMITS at every QPT rather than the first quiet point following the CHECKPOINT frequency specified in the DBEngine parameter file
DBV_OP_UNFILT_OK	(No longer used)
DBV_OP_MAXRECS	Indicates that the Accessory wants record count upper bounds included in all DATASET_INFO arrays
DBV_OP_GLOBAL_SI	Indicates that the Accessory wants the global STATE_INFO record rather than individual STATE_INFO records for each data set when they are all at the same audit location
DBV_OP_EMB_EXTR	Allows an Accessory to request an extract of embedded data sets even if INDEPENDENTTRANS is reset. If this option is set but INDEPENDENTTRANS is reset, and the Accessory does a DBSELECT of an embedded data set with a mode=0, DBEngine extracts the embedded records but does not perform any fixup or normal tracking. Any attempt to DBSelect an embedded data set with a mode of 1 (fixup) or 2 (normal) results in a DBM_CANT_TRACK (113) error message.
DBV_OP_OFFLINE	Prohibits updates to the database during a clone
DBV_OP_ERROR_SI	Causes DBEngine to send a STATE_INFO update prior to returning an error in DBREADTRANGROUP
DBV_OP_LONGTRAN	Causes DBEngine to enable commits during long transactions at pseudo quiet points.
DBV_OP_LINKS	DMSII link items are included in record layouts and replication?
DBV_OP_READAHEAD	When retrieving audit regions from another system this option causes DBEngine to initiate the next read before the Accessory requests it.?
DBV_OP_STATS	Causes DBEngine to print a statistics report when it finishes replication.

Value	Description
DBV_OP_DBPLUS	Databridge Plus will be used to read the audit files if it is installed.
DBV_OP_ACTIVE	Allows DBEngine to read the active audit file.
DBV_OP_NO_REV	Converts reversals to normal updates so that both the original update and the reversal update are sent to the Accessory.?
DBV_OP_ITEMCOUNT	Enables item count integrity checking for detecting layout changes.
DBV_OP_IDLEDB	Causes DBEngine to commit updates when the database is idle
DBV_OP_SI_ULEVEL	Enables the use of the update level field in SI_HOST_INFO
DBV_OP_MANUALCOMP	Prevents the automatic compile of tailored software
DBV_OP_CHECKSUMDS	Causes DBEnterprise to verify the data set block checksums

DBPARAMETERS Processing Parameter Types

The following table provides additional information about the DBPARAMETERS processing parameter types as they are documented in SYMBOL/DATABRIDGE/INTERFACE. The DBPARAMETERS entry point allows the Databridge Clients and Accessories to specify these values.

In all cases, the specified value of each of these parameters must be within the range dictated by the corresponding option in the DBEngine parameter file. Otherwise, DBEngine will adjust the value accordingly.

For example, if the parameter file has CHECKPOINT CLIENT EVERY 50 (ALLOW 20 - 99999) AUDIT BLOCKS, you must specify a value between 20 and 99999 for DBV_TG_BLOCKS.

If the specified value for a parameter is 0 (and 0 is in the ALLOW range), DBEngine will disable the parameter and not use it.

If the specified value is less than 0, DBEngine will retain the current value for that parameter.

Value	Description
DBV_CONCURR_EXTR	The maximum number of concurrent extracts If this parameter value is less than 0 or not specified, the WORKERS option in the DBEngine parameter file determines the actual number of extract tasks. The minimum number of extract tasks is 1.
DBV_TG_BLOCKS	The number of audit blocks per transaction group If this value is less than 0 or is not specified, the CHECKPOINT...AUDIT BLOCKS option in the DBEngine parameter file determines the number of audit blocks per transaction group. If this value and the CHECKPOINT value are both less than or equal to 0, then the actual number of audit blocks per transaction group is 100.

Value	Description
DBV_TG_UPDATES	The number of updates per transaction group If this value is less than 0 or is not specified, the CHECKPOINT...RECORDS option in the DBEngine parameter file determines the number of updates per transaction group. If this value and the CHECKPOINT value are both less than or equal to 0, then the actual number of updates per transaction group is 1000.
DBV_ELAPSED	The number of seconds of elapsed time per transaction group. If this value is less than 0 or is not specified, the CHECKPOINT...SECONDS option in the DBEngine parameter file determines the number of seconds of elapsed time per transaction group. If this value and the CHECKPOINT value are both less than or equal to 0, then the elapsed time per transaction group is unlimited.
DBV_TG_TRANS	The number of transactions per transaction group. If this value is less than 0 or is not specified, the CHECKPOINT...TRANSACTIONS option in the DBEngine parameter file determines the number of transactions per transaction group. If this value and the CHECKPOINT value are both less than or equal to 0, then the actual number of transactions per transaction group is unlimited.
DBV_THREADS	The number of threads DBEnterprise can use during the extract phase of cloning. If this parameter value is less than 0 or not specified, the ENTERPRISE WORKERS option in the DBEngine parameter file determines the actual number of extract threads. The minimum number of extract threads is 1.

DBAUDITMEDIUM Parameters

The following table provides additional information about the DBAUDITMEDIUM parameters as they are documented in SYMBOL/DATABRIDGE/INTERFACE. These values are used by the DBAUDITMEDIUM to specify where DBEngine looks for audit files.

AUDITMEDIUM Value	Description
DBV_AM_ORIGPACK	Tells DBEngine to look on the original DASDL-specified pack(s)
DBV_AM_TAPE	Tells DBEngine to look for a COPYAUDIT tape
DBV_AM_ALTERNATE	Tells DBEngine to look on an alternate pack

AUDITTYPE Value	Description
DBV_AM_NEITHER	Tells DBEngine not to look on this source
DBV_AM_PRIMARY	Tells DBEngine to look for only the primary audit file
DBV_AM_SECONDARY	Tells DBEngine to look for only the secondary audit file
DBV_AM_BOTH	Tells DBEngine to look for both the primary and secondary audit file

Network Protocol Values

The following table provides additional information about the network protocol values as they are documented in SYMBOL/DATABRIDGE/INTERFACE. The DBAUDITSOURCE and DBAUDITSOURCEX use these values to determine which protocol the Accessory uses.

Value	Description
DBV_NET_NONE	Indicates that no network protocol is specified
DBV_NET_TCPIP	Indicates that the Accessory is using a TCP/IP protocol
DBV_NET_HLCN	Indicates that the Accessory is using an HLCN (NetBIOS) protocol
DBV_NET_BNA	Indicates that the Accessory is using a BNA protocol

MAXWAITSECS Values

The following table provides additional information about the MAXWAITSECS values as they are documented in SYMBOL/DATABRIDGE/INTERFACE and specified by several entry points.

Value	Description
DBV_WAIT_FOREVER	Indicates that DBEngine should retry for more audits indefinitely
DBV_DONT_WAIT	Indicates that DBEngine should not retry at all

ITEM_INFO Array Layout

The following table provides additional information about the ITEM_INFO array layout documented in SYMBOL/DATABRIDGE/INTERFACE and returned by several entry points.

Value	Description
II_ENTRY_SIZE	The size in words of the ITEM_INFO array

Field	Description
II_ITEM_NUM	The item number of the item

Field	Description
II_PARENT_NUM	The item number of the group that the item is in
II_DATA_TYPE	The data item type, such as ALPHA, REAL, and so on Possible types are as follows:

Type	Description	Type	Description
DBV_LINK	For link (IS IN...)	DBV_ALPH	For ALPHA
DBV_IMAG	For binary byte values	DBV_DECI	For NUMBER (I)
DBV_TYPE	For RECORD TYPE	DBV_DECF	For NUMBER (I,J)
DBV_BFLD	For FIELD of BOOLEANs	DBV_BINI	For REAL (I)
DBV_GRP	For GROUP	DBV_BINF	For REAL (I,J)
DBV_BOOL	For BOOLEAN	DBV_BFLT	For REAL
DBV_FLD	For FIELD	DBV_WIDE	For KANJI (16-bit characters)

II_REQUIRED	Indicates whether the data item is required or not If the item is required, it may not be NULL.
II_NAME_SIZE	The length of the name
II_DECL_LEN	The length of the item specified in the declaration
II_SCALING	The numeric scaling factor, which specifies how many digits are to the right of the assumed decimal point
II_RAWOFFSET	0 indicates II_OFFSET is relative to the user record. 1 indicates II_OFFSET is relative to the original, raw (nonuser) record. This is used primarily for link items, which are not in the user record.
II_OFFSET	The digit offset to data
II_SIZE	The digit size of the data
II_OCCURS	The declared occurrences
II_DIMS	The number of subscripts the data item requires
II_SIGNED	Indicates that the numeric item is signed
II_DESCENDING	Indicates that the item is a descending key This field is valid only if the Accessory is calling an entry point that returns information about key items.
II_DEPEND_NUM	The occurs-depending-on ITEM_NUM For example, if the DASDL has ADDR-LINES OCCURS 4 DEPENDING ON NUM-LINES, then this field would contain the ITEM_NUM of NUM-LINES.
II_DS_ITEM_NUM	The corresponding ITEM_NUM in the data set (if this is a remap)
II_LINK_DS_NUM	The target data set number of a link

Field	Description
II_LINK_SET_NUM	The target set number of a link If the item is a link, 0 indicates that the link is an AA. All other values indicate that the link is a foreign key of this set.

Field	Description		
II_FORMAT	IIF_DEFAULT—Default value, unspecified		
	IIF_DAYS_SINCE_1900—Number of days since 1/1/1900 as a NUMBER (n)		
	IIF_LINC_DATE—Number of days since 1/1/1957 as an ALPHA (n)		
	IIF_TIME_6—Timestamp		
	IIF_TIME_7—Day of the week, date, and time		
	IIF_TIME_60—Time zone, Julian date, time of day in hundredths of a second		
	Various date formats as follows:		
	NOTE: DDD is a number between 1–366 for Julian dates.		
	MMM is a three-letter abbreviation for the month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC).		
	Format	**Format**	**Format**

Field	Description		
	IIF_YYDDD	IIF_YYYYDDD	IIF_YYMMDD
	IIF_YYMMMDD	IIF_YYYYMMDD	IIF_DDMMYYYY
	IIF_DDMMYY	IIF_MMDDYY	IIF_DDMMYYYY
	IIF_DDMMYY		
	IIF_HHMMSS—Time of day		
	IIF_TIME_1—Time of day in sixtieths of a second		
	IIF_TIME_11—Time of day in 2.4 microseconds		
	Various combined date and time formats as follows:		
	NOTE: DDD is a number between 1–366 for Julian dates.		
	MMM is a three-letter abbreviation for the month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC).		
	Format	**Format**	**Format**
	IIF_YYDDDHHMMSS	IIF_MMDDYYHHMMSS	IIF_HHMMSSYYYYDDD
	IIF_DDMMYYHHMMSS	IIF_HHMMSSYYMMDD	IIF_YYYYMMDDHHMMSS
	IIF_MMDDYYYYHHMMSS	IIF_DDMMYYYYHHMMSS	IIF_YYYYDDDHHMMSS
	IIF_HHMMSSYYDDD	IIF_HHMMSSYYYYMMDD	IIF_HHMMSSMMDDYYYY
	IIF_HHMMSSMMDDYY	IIF_HHMMSSDDMMYY	
	IIF_YYMMDDHHMMSS		
	IIF_HHMMSSDDMMYYYY		
	IIF_NUMERIC—ALPHA containing only the characters 0–9		
	IIF_ALPHANUMERIC—NUMBER that should be represented as a character string on the client system		
II_NAME	The item name (for 3 words), else 0		

STATE_INFO Layout

The following table provides additional information about the STATE_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by several entry points.

Field	Description
SI_STRNUM	The DMSII structure number of the data set
SI_RECTYPE	The variable format record type
SI_AFN	The audit file number of the data set's current audit location
SI_ABSN	The audit block serial number of the data set's current audit location

Field	Description
SI_SEG	The audit file segment number of the data set's current audit location
SI_INX	The word index in the audit block of the data set's current audit location
SI_TIME	The timestamp (in TIME (6) format) of the data set's current audit location
SI_MODE	Possible values are as follows: ◆ DBV_MODE_CLONE—Indicates that the table needs to be cloned ◆ DBV_MODE_FIXUP—Indicates the fixup phase of cloning ◆ DBV_MODE_NORMAL—Indicates normal audit processing ◆ DBV_MODE_REORG—Indicates that the table needs to be reorganized ◆ DBV_MODE_PURGE—Indicates that the table needs to be purged
SI_FORMAT_LEVEL	The data set format update level
SI_TABLE_LEVEL	The client table format update level
SI_ITEM_COUNT	The number of data items
SI_OPTIONS	The SI_REC_OPTIONS record processing options
SI_NO_LINKS	The field that tells DBEngine not to return any updates to link values
SI_REORG_NOTIFY	The field that requests a documentation record from DBEngine after the Accessory encounters a reorganization discontinuity for the data set Possible values are as follows: ◆ DBV_REORG_DEFAULT—Sends a reorg document record if the DBV_OP_DOC global option is set ◆ DBV_REORG_IGNORE—Requests that no reorg document records be sent ◆ DBV_REORG_SEND—Requests that all reorg document records be sent
SI_MODIFIES	The field that indicates which images of modifies should be sent Possible values are as follows: ◆ DBV_MOD_DEFAULT—Indicates that the Accessory should use the "global" modifies setting ◆ DBV_MOD_AI_ONLY—Indicates that the Accessory should send only the after-image of the modification ◆ DBV_MOD_BI_AI—Indicates that the Accessory should send both the before- and after-images of the modification
SI_UI_MASK	The values returned in UPDATE_INFO [UI_MASK] on every update See UPDATE_INFO Layout for these values.
SI_HOST_INFO	Engine-defined information to be returned to DBSELECT

Value	Description
SI_ENTRY_SIZE	The size in words of the STATE_INFO array

DATABASE_INFO Layout

The following table provides additional information about the DATABASE_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by the DBDATABASEINFO entry point.

Field	Description
DBI_UPDATE_LEVEL	The database update level
DBI_UPDATE_TS	The database update timestamp
DBI_MAX_STRNUM	The last structure number used
DBI_TIMESTAMP	The database timestamp
DBI_NAME	The database name (for 3 words)
DBI_USERCODE	The database usercode (for 3 words)
DBI_OPTIONS	Various run-time options and values. DBI_FLAT indicates a FileXtract (flat file) database. DBI_RDB indicates an RDB (Remote Database) database. DBI_PLUS indicates DBEngine is using DBPlus. DBI_ACTIVE indicates database is being updated. DBI_ITRANS indicates setting of INDEPENDENTTRANS DASDL option. DBI_ADDRCHECK indicates setting of the ADDRESSCHECK DASDL option. DBI_MAXWORKERS indicates the maximum number of Extract Workers allowed.
DBI_CURR_AFN	The current (active) audit file number
DBI_CURR_ABSN	The current audit block serial number
DBI_MAX_RECSIZE	Maximum dataset record size in words
DBI_MAX_COMPACT	Maximum COMPACT data record size
DBI_MAXWORKERS	Maximum number of EXTRACT workers

Value	Description
DBI_INFO_SIZE	The size in words of the DATABASE_INFO array

DATASET_INFO Layout

The following table provides additional information about the DATASET_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by several entry points.

Field	Description
DI_STRNUM	The DMSII structure number

Field	Description
DI_RECTYPE	The DMSII record type number
DI_FORMAT_LEVEL	The data set format update level
DI_RECSZ_WORDS	The size of the data set record in words
DI_RECSZ_BYTES	The size of the data set record in bytes
DI_PARENT_STRNUM	The structure number of the parent data set if this data set is embedded, else 0
DI_NUM_CHILDREN	The number of supported (visible to the Databridge Clients) child (embedded) data sets that this data set has
DI_DS_STRNUM	The base data set structure number This is the same as DI_STRNUM unless this is a REMAP.
DI_SUBTYPE	The data set structure type Possible values are as follows: ◆ DI_STANDARD_V ◆ DI_RANDOM_V ◆ DI_ORDERED_V ◆ DI_UNORDERED_V ◆ DI_GLOBAL_V ◆ DI_DIRECT_V ◆ DI_COMPACT_V ◆ DI_RESTART_V ◆ DI_VIRTUAL_V
DI_ITEM_COUNT	The number of items in the record layout (relative to the active filter if the DBV_OP_ITEMCOUNT option is enabled)
DI_MISC_INFO	Contains DI_MISC_FLAGS, DI_TARGET_LINKS, and DI_MAX_RECORDS. See the remaining items in the table for more information about this field.
DI_VALID_AA	If TRUE, indicates that the absolute address (AA) values (record address) do not change due to normal updates. If this field is TRUE or the DI_STATIC_AA field is true then the value in UPDATE_INFO [UI_AA] is valid
DI_STATIC_AA	If TRUE, indicates that the UPDATE_INFO [UI_AA] value for the record is static In this case, the value does not change even if the data set is reorganized.
DI_ALTERED	If TRUE, indicates that the data set/remap was ALTERed using DBGenFormat
DI_LINKS	If TRUE, indicates that the data set has at least one link item; If FALSE, indicates that the data set has no link items
DI_TARGET_LINKS	The number of data sets with links to this data set
DI_ROW_FILTER	The filter has a WHERE clause
DI_COL_FILTER	The filter omits some columns
DI_INVALID_PAA	Parent records do not have valid absolute address (AA) values

Field	Description
DI_MAX_RECORDS	The upper bound on the population of the data set This field is 0 if the run-time option DBV_OP_MAXRECS is reset, which is the default.
DI_NAME	The data set name (for 17 bytes)
DI_NAME_SIZE	The length of the data set name given in DI_NAME
DI_MISC_INFO2	Miscellaneous information containing the remaining fields in this table
DI_RECTYPE_NUM	For variable format datasets item number of the record type item
DI_USERDATA_OFS	Offset to the start of the user-updateable data, in half-bytes
DI_IMAGE_WORDS	Size of an audit record image, in words

Value	Description
DI_INFO_SIZE	The size of the DATASET_INFO array

SET_INFO Layout

The following table provides additional information about the SET_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by several entry points.

Field	Description
XI_STRNUM	The DMSII structure number of the set
XI_DS_STRNUM	The DMSII structure number of the target data set
XI_FORMAT_LEVEL	The format update level
XI_KEYS_COUNT	The number of data items in the set's key
XI_KEYSZ_WORD	The size of the key in words
XI_KEYSZ_DIGITS	The size of the key in 4-bit digits
XI_SUBTYPE	The set structure type Possible values are as follows: ◆ XI_INDEXSEQ_V—Indicates that the index is sequential ◆ XI_RANDOM_V—Indicates that the index is random ◆ XI_ORDERED_V—Indicates an ordered list ◆ XI_UNORDERED_V—Indicates an unordered list
XI_NAME_SIZE	The length of the set name in bytes
XI_DUPLICATES	If TRUE, duplicates are allowed

Field	Description
XI_KEYCHANGEOK	This item is TRUE (key changes allowed) if any of the following options exist in the set: ◆ NO DUPLICATES KEYCHANGEOK ◆ DUPLICATES ◆ DUPLICATES FIRST ◆ DUPLICATES LAST
XI_NAME	The name of the set (for 3 words)

Value	Description
XI_INFO_SIZE	The size of the SET_INFO array

UPDATE_INFO Layout

The following table provides additional information about the UPDATE_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by several entry points.

Field	UI_MASK bit #	Description
UI_STRNUM	0	The DMSII structure number
UI_RECTYPE	1	The DMSII variable format record type
UI_RECSZ_WORDS	2	The size of the record image in words
UI_UPDATE_TYPE	3	The DBV_xxx indicating the type of change, where xxx is MODIFY, DELETE, and so on.
UI_MODIFY_F	N/A	A value of 1 indicates that the create or delete was originally a modify and the Accessory requested before- and after- images for modifies
UI_STRIDX	4	The unique index for the data set-recordtype
UI_MODE	6	The data set mode Possible values are as follows (from STATE_INFO [SI_MODE]): ◆ DBV_MODE_CLONE—Indicates that the table needs to be cloned ◆ DBV_MODE_FIXUP—Indicates the fixup phase of cloning ◆ DBV_MODE_NORMAL—Indicates normal audit processing ◆ DBV_MODE_REORG—Indicates that the table needs to be reorganized ◆ DBV_MODE_PURGE—Indicates that the table needs to be purged

Field	UI_MASK bit #	Description
UI_STACKNBR	5	The stack number of the program making the change
UI_FORMAT_LEVEL	7	The data set format update level
UI_AA	8	The record address If updates can cause the record address to change, such as in an ORDERED data set, this value is 0. The DATASET_INFO [DI_VALID_AA] for this data set is 1 if this field is valid. If the DATASET_INFO [DI_STATIC_AA] for this data set is 1, this field contains a static record number (RSN) rather than a record address.
UI_PARENT_AA	9	The record address of the parent record If the DATASET_INFO [DI_STATIC_AA] for the parent data set is 1, this field contains a static record number rather than a record address.
UI_AFN	10	The audit file number
UI_ABSN	11	The audit block serial number
UI_SEG	12	The audit file disk segment
UI_INX	13	The word offset within the audit block
UI_TIME	14	The approximate audit time in TIME(6) format For extracted (cloned) records, the time the record was read from the database
UI_OPTIONS	15	Specifies processing/formatting options See the remainder of this table for more information about this field.
UI_WORKER	N/A	Indicates the Extract Worker number that produced the record
UI_STATIC_AA	N/A	If 1, then the addresses in UI_AA and UI_PARENT_AA are static
UI_PREFILT	N/A	If 1, then the update was read from a filtered audit file
UI_BI_AI	N/A	If 1, then the Accessory requests both before- and after-images for this update and indicates a single abort reversal
UI_MASK	N/A	Tells DBServer which UPDATE_INFO fields to return to the client Each bit corresponds to a field.

Value	Description
UI_INFO_SIZE	The size in words of the UPDATE_INFO array

AUDIT_INFO Layout

The following table provides additional information about the AUDIT_INFO layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by the DBOPENAUDIT entry point.

Field	Description
AI_BLOCKSIZE	The maximum audit blocksize in words
AI_LASTSEG	The last segment number
AI_DBTIMESTAMP	The database timestamp
AI_OPENTIMESTAMP	The block 0 timestamp
AI_AUDITLEVEL	The audit level, for example, 7
AI_MAJORVERSION	The major part of SSR, for example, 45
AI_MINORVERSION	The minor part of SSR, for example, 1
AI_UPDATELEVEL	The database update level
AI_FLAGS	Miscellaneous audit flags, as follows: ◆ AI_PRIORCLOSEERROR—Indicates a possible error writing the last block of the prior file ◆ AI_RECOVERQPT—Indicates DMSII recovers to any QPT rather than to a super quiet point such as a syncpoint ◆ AI_CHECKSUMMED—Indicates that the audit blocks have a checksum

Value	Description
AI_FIRST_ABSN	The ABSN of the first data block in the audit file
AI_MAX_RECSIZE	The size of the largest dataset record, in words
AI_COMPACTSIZE	The size of the largest COMPACT dataset record
AI_INFO_SIZE	The size of the AUDIT_INFO layout

Link Update Info Layout

The following two fields comprise the first word of the data portion of a link update:

Field	Description
LNK_OFS_F	The original word offset
LNK_SZ_F	The size of the link update in words

Audit File Error Subtypes

The following table provides additional information about the audit file error subtypes as they are documented in SYMBOL/DATABRIDGE/INTERFACE.

Subtype	Description
UNAVAIL_NO_UPDATE	The database has never been updated
UNAVAIL_ACTIVE	The audit file is the active (current) audit file
UNAVAIL_NO_MORE	The active audit file has no more audit available
UNAVAIL_BAD_AFN	Invalid audit file number
UNAVAIL_EXCLUSIVE	Another program has opened the file with EXCLUSIVE=TRUE
UNAVAIL_NOT_FOUND	The audit file is not present
UNAVAIL_OFFLINE	The audit file was moved from disk to tape storage
UNAVAIL_NO_FAMILY	The audit file pack family is not present
UNAVAIL_UNKNOWN	The audit file is unavailable for an unknown reason

Data Error Types

The table below provides additional information about the data error types as they are documented in SYMBOL/DATABRIDGE/INTERFACE.

These errors occur when invalid data is entered.

Value	Description
DBV_DE_BAD_NUMBER	Indicates that the numeric item had undigits (4"ABCDEF").
DBV_DE_BAD_SIGN	Indicates that a signed numeric item had an invalid sign digit. The sign digit should be either 4"C" or 4"D".
DBV_DE_BAD_ALPHA	Indicates that an alpha item had control characters
DBV_DE_NULL_NUMBER	Indicates that a numeric item was NULL
DBV_DE_NULL_ALPHA	Indicates that an alpha item was NULL
DBV_DE_OVERFLOW	Indicates that the numeric item caused an integer overflow

Processing Limit Types

The following table provides additional information about the processing limit types as they are documented in SYMBOL/DATABRIDGE/INTERFACE. These values are used by entry points that set processing limits.

Value	Description
DBV_LIMIT_UNSPECIFIED	Indicates that no processing limits are specified
DBV_LIMIT_BEFORE	Indicates that processing stops at the QPT before the limit
DBV_LIMIT_AFTER	Indicates that processing stops at the QPT after the limit

Statistics Category Values

The following table provides additional information about the statistics category values as they are documented in SYMBOL/DATABRIDGE/INTERFACE.

Each of these values corresponds to a set of values in a STATISTICS_INFO array. All times are in units of 2.4 microseconds.

Value	Description
DBV_STAT_FIRST_TIME	The lowest time value
DBV_STAT_AF_OPEN	The time spent opening audit files
DBV_STAT_AF_CLOSE	The time spent closing audit files
DBV_STAT_AF_READ	The time spent reading audit files
DBV_STAT_CF_OPEN	The time spent opening DMSII CONTROL files
DBV_STAT_CF_CLOSE	The time spent closing DMSII CONTROL files
DBV_STAT_CF_READ	The time spent reading DMSII CONTROL files
DBV_STAT_DF_OPEN	The time spent opening DESCRIPTION files
DBV_STAT_DF_CLOSE	The time spent closing DESCRIPTION files
DBV_STAT_DF_READ	The time spent reading DESCRIPTION files
DBV_STAT_DB_OPEN	The time spent opening databases
DBV_STAT_DB_CLOSE	The time spent closing databases
DBV_STAT_DB_READ	The time spent reading databases
DBV_STAT_DB_WRITE	The time spent performing database stores/deletes
DBV_STAT_PF_OPEN	The time spent opening port files
DBV_STAT_PF_CLOSE	The time spent closing port files
DBV_STAT_PF_READ	The time spent waiting for port file reads
DBV_STAT_PF_WRITE	The time spent waiting for port file writes
DBV_STAT_BT_WAIT	The time spent waiting for begin transactions

Value	Description
DBV_STAT_ET_WAIT	The time spent waiting for end transactions
DBV_STAT_SP_WAIT	The time spent waiting for end transaction with syncpoint waits
DBV_STAT_CB_WAIT	The time spent waiting for callbacks
DBV_STAT_EX_WAIT	The time spent waiting for DMSII to return records during an extracts
DBV_STAT_SLEEP_WAIT	The time the program "slept" while waiting for a timer to expire
DBV_STAT_TG_WAIT	The time spend waiting while trying to read updates
DBV_STAT_LAST_TIME	The highest time value
DBV_STAT_FIRST_SIZE	The lowest size value
DBV_STAT_AA_SIZE	The size of the audit areas (regions) read in words
DBV_STAT_AB_SIZE	The size of the audit blocks read in words
DBV_STAT_AR_SIZE	The size of the audit records read in words
DBV_STAT_PR_SIZE	The size of the port messages read in bytes
DBV_STAT_PW_SIZE	The size of the port messages written in bytes
DBV_STAT_PA_SIZE	The size of port messages available for reading in bytes
DBV_STAT_COMMIT_BLOCKS	The number of commits caused by a "CHECKPOINT n BLOCKS" setting
DBV_STAT_COMMIT_RECORDS	The number of commits caused by a "CHECKPOINT n RECORDS" setting
DBV_STAT_COMMIT_TIME	The number of commits caused by a "CHECKPOINT n MINUTES" setting
DBV_STAT_COMMIT_TRANS	The number of commits caused by a "CHECKPOINT n TRANSACTIONS" setting
DBV_STAT_COMMIT_QPT	The number of commits caused by the need to commit at the next quiet point
DBV_STAT_COMMIT_EOF	The number of commits caused by reaching the end of an audit file
DBV_STAT_CREATES	The number of creates per commit
DBV_STAT_DELETES	The number of deletes per commit
DBV_STAT_MODIFIES	The number of modifies per commit
DBV_STAT_STATE	The number of StateInfo updates per commit
DBV_STAT_DOCS	The number of doc records per commit
DBV_STAT_UPDATES	The number of updates per commit
DBV_STAT_TRANS	The number of transactions per commit
DBV_STAT_LONGTRANS	The number of active Long Transacations per commit
DBV_STAT_LAST_SIZE	The highest size value

STATISTICS_INFO Array Layout

The following table provides additional information about the STATISTICS_INFO array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by the DBSTATISTICS entry point.

Field	Description
STAT_COUNT	The number of data points
STAT_MIN	The minimum time/size
STAT_MAX	The maximum time/size
STAT_TOTAL	The sum of all times/sizes

Value	Description
STAT_INFO_SIZE	The size of STATISTICS_INFO

FileXtract FileInfo Array Layout

The following table provides additional information about the FileInfo array layout as it is documented in SYMBOL/DATABRIDGE/INTERFACE and returned by the DBFILEREADER entry point.

Field	Description
FI_FILENBR	The file number The maximum value for this field is 65535
FI_RECLOC	The location of the record within the file The meaning and layout of this word varies from one FileXtract Reader to another.
FI_RECTS	The timestamp of the returned record
FI_STRIDX	The structure index specified by the FileXtract Reader
FI_PARAMLEN	The length of the FileXtract Reader parameter in bytes
FI_PARAM	The first word of the FileXtract Reader parameter string

Value	Description
FIV_PARAMLENMAX	The maximum length of the FileXtract Reader parameter in bytes
FIV_INFOSZ	The size of the FileInfo array in words

DBOUTPUTHEAD Procedure Heading

This define can be used as a heading for any output routines you write. Include the name of your output routine in brackets after DBOUTPUTHEAD. For example:

```
DBOUTPUTHEAD [MyWriter]
    begin
    …
    end MyWriter;
```

Formatting procedures call output routines with the formatted record.

If the output procedure encounters an error, it should return the appropriate DBMTYPE error code. Otherwise it should return DBM_OK.

Input	Type	Definition
P	POINTER	The pointer to the formatted record
CHARS	INTEGER	The length of the formatted record in bytes
UPDATE_INFO	ARRAY	A description of the modification For a description of the array, see the “UPDATE_INFO Layout.”
RawImage	ARRAY	The original unformatted record

DBFORMATHEAD Procedure Heading

You can use this define as the heading for any formats that you write. Include the name of your formatting routine in brackets after DBFORMATHEAD. For example,

```
DBFORMATHEAD [MyFormat]
    begin
    …
    end MyFormat;
```

If the formatting procedure encounters an error, it should return the appropriate DBMTYPE error code. Otherwise it should return DBM_OK.

Input	Type	Definition
USERREC	ARRAY	The unformatted data set record from the database or audit trail
UPDATE_INFO	ARRAY	A description of the modification For a description of the array, see the UPDATE_INFO Layout.
CALLBACK	DBMTYPE	The procedure that receives the formatted record The formatted record is usually written to a file or sent to a Databridge client.

DBTRANSFORMHEAD Procedure Heading

You can use this define as the heading for any update transform procedures you write.

Include the name of your transform routine in brackets after DBTRANSFORMHEAD. For example,

```
DBTRANSFORMHEAD [MyTransform]
    begin
    …
    end MyTransform;
```

If the transform procedure encounters an error, it should return the appropriate DBMTYPE error code. Otherwise it should return DBM_OK.

Parameter	Type	Definition
UI	ARRAY	A description of the modification For a description of the array, see UPDATE_INFO Layout.
BI	ARRAY	The before-image of the record This array is valid only for update types DBV_DELETE and DBV_MODIFY. See Record Change Types for a description of these types.
AI	ARRAY	The after-image of the record This array is not valid for update type DBV_DELETE. See Record Change Types for a description of this type.
DBFORMAT	PROCEDURE	The formatting procedure the transform procedure calls after the update is transformed See DBFORMATHEAD Procedure Heading.
WRITER	PROCEDURE	The procedure that writes the formatted record See DBOUTPUTHEAD Procedure Heading.

DBFILTERHEAD Procedure Heading

You can use this define as the header for any filters that you write. Include the name of your filtering routine in brackets after DBFILTERHEAD. For example,

```
DBFILTERHEAD [MyFilter]
    begin
    …
    end MyFilter;
```

A false return value indicates that the caller should discard the update because it did not satisfy the filter condition. Filter procedures return a Boolean result, but additional information can be placed in the DBMResultF field of that result. That field must contain a DBMTYPE value describing any error encountered.

Parameter	Type	Definition
USERREC	ARRAY	The unformatted data set record from the audit trail
UPDATE_INFO	ARRAY	A description of the modification For a description of the array, see UPDATE_INFO Layout.

DBERRORMANAGERHEAD Procedure Heading

You can use this define to declare an error handler procedure heading.

Include the name of your output routine in brackets after DBERRORMANAGERHEAD. For example,

```
DBERRORMANAGERHEAD [MyHandler]
    begin
    …
    end MyHandler;
```

The procedure should return an EMATYPE result code.

Parameter	Type	Definition
ACCESSORYID	AIDTYPE	The ID number of the Accessory
ERRNBR	DBMTYPE	The error message number
PERRMSG	POINTER	The error message text
ERRMSGLEN	REAL	The length of the error message

DBFILEREADERHEAD Procedure Heading

Use this define to declare a heading for a FileXtract Reader procedure. A Reader procedure reads a record from a non-DMSII file and returns it to DBEngine, which returns it to an Accessory for processing.

Include the name of your file reader routine in brackets after DBFILEREADERHEAD. For example,

```
DBFILEREADERHEAD [MyReader]
    begin
    …
    end MyReader;
```

If the file reader procedure encounters an error, it should return the appropriate DBMTYPE error code. Otherwise it should return DBM_OK.

Parameter	Type	Definition
FileInfo	ARRAY	See “FileXtract FileInfo Array Layout” for more information
FileRecord	ARRAY	Record contents to pass to DBEngine

File Attribute Mask Bits

Use the file attribute mask bits in this section when you call the DBFILEATTRIBUTE entry point. These bits correspond to GETSTATUS request type 3 calls.

For example, to request the creation date and time, use the following mask:

```
0 & 1 [CREATIONDATEB:1] & 1 [CREATIONTIMEB:1]
```

For a description of each attribute, refer to a Unisys GETSTATUS/SETSTATUS programmer's reference.

Attribute	Mask Bit
CREATIONDATEB	01
SIZESB	02
SAVEFACTORB	03
AREASECTORSB	05
ROWSINUSEB	07
FileOrDirB	09
AREASB	10
EOFSEGMENTSB	11
EOFLASTSEGBITSB	12
ACCESSDATEB	15
VERSIONB	19
CYCLEB	20
TIMESTAMPB	21
ACCESSTIMEB	25
USERINFOB	26
ALTERDATEB	27
ALTERTIMEB	28
CREATIONTIMEB	29
BASEUNITNBRB	30
ROWSLINKB	33
ORGANIZATION	34

Attribute	Mask Bit
STRUCTUREB	40
FILELENGTHB	41

B Troubleshooting

In This Appendix

This appendix explains general troubleshooting procedures and tells you how to contact Customer Support.

General Troubleshooting Procedures

If you have problems running Databridge, complete the following steps:

1. Check to see that your system meets the minimum hardware and software requirements necessary to use the product. Refer to the *Databridge Host Administrator's Guide* for information.
2. Check your configuration options. Most problems are caused by incorrect configuration.
3. Check the usercodes for your DMSII databases and the usercode for the Databridge files. Make sure the Databridge software can access the DMSII DESCRIPTION, CONTROL, DMSUPPORT, and audit files.
4. Check parameter file options for the Databridge Accessory you are using. Make sure all tailored support library transforms, filters, and formats are also entered into the DBGenFormat parameter file and are spelled correctly.
5. Check your system. You may be using peripheral equipment or other software that may not be compatible with this product.
6. Resolve errors. Refer to the *Databridge Errors and Messages Guide* for information on resolving error messages.
7. If you cannot identify and solve the problem without assistance, contact your product distributor. Call from a location where you have access to the problem.
8. Troubleshoot the problem using information available from Micro Focus Technical Support.

   http://www.attachmate.com/en-US/Support/

   This service directly links you to our internal help desk system, 24 hours a day, 7 days a week.
9. Contact Micro Focus Technical Support:

   http://support.attachmate.com/contact/

Troubleshooting for All Accessories

Following are two common things to check for in all Accessories:

- If an Accessory cannot find DBEngine, most likely the Accessory is running under a different usercode than where DBEngine is located. If this is unavoidable, use the ODT SL command to define the DBENGINE function name, as follows (assuming DBEngine is located under the DBA usercode on a pack called DBAPACK):

```
SL DBENGINE = (DBA)OBJECT/DATABRIDGE/ENGINE ON DBAPACK
```

- If an Accessory displays a message indicating a version mismatch and then terminates, most likely the Databridge software you are running was compiled using different versions of the Databridge API (SYMBOL/DATABRIDGE/INTERFACE). Make sure that all of the Databridge software is from the same release.

Outdated Filters and Formats

When a filter or format is out-of-date, Databridge Accessories attempt to recompile the DBSupport library. If that is unsuccessful, the Accessory displays an error message informing you that the support library must be recompiled. Use WFL/DATABRIDGE/COMP to recompile a tailored support library.

Troubleshooting External Filters and Formatting Procedures

If an external filtering or formatting procedure is unable to retrieve the correct structure indexes or data set information, and so on, be sure that it is not linking to a different copy of DBEngine than the Accessory. The external library must link to the same copy of DBEngine as the Accessory. The most common cause of linking to the wrong copy of DBEngine is invoking DBLINKENGINE or calling an entry point before the external library performs a FREEZE.

DBEngine is shared-by-run-unit, which means that the Accessory and any libraries it calls share a copy of DBEngine. Until the external library performs a FREEZE, it is considered a separate running program rather than a library, and it gets its own copy of DBEngine if it links to it either implicitly or explicitly.

The most common solution is to declare a flag that indicates whether the external library needs to perform its initialization. The first statement of the filtering or formatting procedure tests the flag and performs the initialization routine if it has not been executed yet.

Troubleshooting Virtual Data Set Transform Procedures

A common problem when creating virtual data sets is that data set record updates or STATE_INFO updates are missing because the virtual data set transform procedure ignores records that do not pertain to the virtual data set. The virtual data set transform procedure must call the formatting procedure passed to it as a parameter (usually BINARYFORMAT) for *all* records it receives, not just those related to the virtual data set.

Another potential problem is that data set record updates are corrupted. This could happen for either of the following reasons:

- The virtual data set transform procedure neglected to update the UPDATE_INFO with new values for the structure number, record type, record size, and structure index when it builds a record for the virtual data set. The UPDATE_INFO *must* contain the attributes of the virtual data set record.
- After sending a virtual data set record, the virtual data set transform procedure tried to send the original (real) data set record without first restoring the original UPDATE_INFO.

DMSII Reorganizations

The following table indicates changes you might need to make if any of the source data sets for your virtual data set were affected by a DMSII reorganization:

If	Then
Your virtual data set transform routine calls a COBOL library to create the data for your virtual data set	Recompile the COBOL library.
Any of your source data sets for the virtual data set increased in size	If you are calling a COBOL library, make sure that your COBOL program's Working?Storage variable is still large enough to accept the record images passed to it by the TRANSFORM routine. If the COBOL library uses DMSII user work areas (data set 01s), recompile the COBOL library to get the new layouts. For ALGOL routines, increase the size of the array holding the record images or dynamically resize the array based on the UI_RECSZ_WORDS value in the UPDATE_INFO.
The Accessory faults (F-DS) with a SEG ARRAY ERROR	Take the actions previously described for source data sets that increase in size.

Troubleshooting Reformatting Procedures

One kind of error associated with reformatting procedures is SEG ARRAY ERROR, which can be caused by using the offset values (SourceOfs and DestOfs) as byte offsets. These values are always digit (half-byte) offsets. Likewise, the size values (SourceSz and DestSz) are always in units of digits, not bytes.

For clarity, use the following defines from the sample in your reformatting procedure to distinguish 4-bit values from 8-bit:

```
SourceSz4
SourceSz8
DestSz4
DestSz8
SourceOfs4
SourceOfs8
DestOfs4
DestOfs8
```

Use the items ending in 4 with the arrays Source4 and Dest4. Use those ending in 8 with the arrays Source8 and Dest8.

Do not assume the destination area is initialized to any particular value. The reformatting procedure is responsible for the entire contents of the destination, starting at DestOfs for a length of DestSz digits.

When reformatting an OCCURing item, the reformatting procedure receives a SourceSz reflecting the total length of all occurrences. It must reformat all of the occurrences at once.

C Virtual and Alter Data Item Types

In This Appendix

This appendix lists Databridge-specific data item types for VIRTUAL and ALTER declarations.

Additional Databridge Data Item Types

Databridge supplies several data item types in addition to the regular DMSII types that you can use when you specify a VIRTUAL or ALTER in the DBGenFormat parameter file.

The additional TIME, NUMERIC, and ALPHA data item types are grouped throughout this section by the following formats:

- TIME_*n* formats
- Combined date and time formats
- Specially-defined formats

The Databridge Client also provides a way to change the data item type. For more information, refer to the *Databridge Client Administrator's Guide* for more information about how you can use the various date, time, and combined date/time formats available with the Databridge Client.

TIME_*n* Formats

The following TIME_*n* data types are all one-word (6 byte) data items corresponding to the TIME (*n*) function, as follows:

Type	Description
TIME_1	Time of day in sixtieths of a second
TIME_6	Timestamp
TIME_7	Day of week, date, time
TIME_11	Time of day in 2.4 microseconds
TIME_60	Time zone, Julian date, time of day in hundredths of a second

Combined Date and Time Formats

The following NUMBER and ALPHA declarations allow you to specify a date and time format rather than a size, as follows:

```
NUMBER (datetimeformat)
```

or

```
ALPHA (datetimeformat)
```

where *datetimeformat* is one of the following:

Format	Description
YYDDD	ALPHA (5) or NUMBER (5) with a two-digit year *YY* and with days *DDD* where *DDD* is a number between 1–366 for Julian dates
HHMMSS	ALPHA (6) or NUMBER (6) time of day
YYMMDD MMDDYY DDMMYY	ALPHA (6) or NUMBER (6) with two-digit *YY* (1900–1999)
YYYYDDD	ALPHA (7) or NUMBER (7) with four-digit year *YYYY* and with days *DDD* where *DDD* is a number between 1–366 for Julian dates
YYMMMDD	ALPHA (7) with a two-digit year *YY* (1900–1999) and a three-character month abbreviation Months are abbreviated JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC.
YYYYMMDD MMDDYYYY DDMMYYYY	ALPHA (8) or NUMBER (8) with four-digit year *YYYY*
YYYYMMMDD	ALPHA (9) with a four-digit year *YYYY* and a three-character month abbreviation Months are abbreviated JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC.
HHMMSSYYDDD YYDDDHHMMSS	ALPHA (11) or NUMBER (11) with two-digit year *YY* (1900–1999), days *DDD* where *DDD* is a number between 1–366 for Julian dates, and a six-digit time
YYMMDDHHMMSS MMDDYYHHMMSS DDMMYYHHMMSS	ALPHA (12) or NUMBER (12) with two-digit year *YY* representing dates in both the 20th and 21st centuries, followed by a six-digit time
HHMMSSYYMMDD HHMMSSMMDDYY HHMMSSDDMMYY	ALPHA (12) or NUMBER (12) with two-digit year *YY* (1900–1999) preceded by a six-digit time
YYYYDDDHHMMSS	ALPHA (13) or NUMBER (13) with four-digit year *YYYY* and with days *DDD* where *DDD* is a number between 1–366 for Julian dates followed by a six-digit time
HHMMSSYYYYDDD	ALPHA (13) or NUMBER (13) with four-digit year *YYYY* and with days *DDD* where *DDD* is a number between 1–366 for Julian dates preceded by a six-digit time
YYYYMMDDHHMMSS MMDDYYYYHHMMSS DDMMYYYYHHMMSS	ALPHA (14) or NUMBER (14) with four-digit year followed by a six-digit time
HHMMSSYYYYMMDD HHMMSSMMDDYYYY HHMMSSDDMMYYYY	ALPHA (14) or NUMBER (14) with four-digit year preceded by a six-digit time

Specially-defined Formats

The following formats allow you to represent NUMBER, ALPHA, and integer types in predetermined ways, as follows:

Type	Description
DAYSCOUNT (**n**)	Number of days since 1/1/1900 as a NUMBER (*n*) The DAYSCOUNT ALTER triggers the Databridge Client to generate a Miser date, which is a dms_subtype of 1. The Miser date is fixed at 01.01.1900 and cannot be changed. To modify and use a custom base year, the dms_subtype must be 3, which is a LINC date.
LINCDATE (*n*)	Number of days since 1/1/1957 as an ALPHA (n)
NUMERIC ALPHA (*n*)	An ALPHA data item of *n* bytes containing a numeric value in alphanumeric EBCDIC form For example, a NUMERIC ALPHA (4) having the value 123 would contain "0123", that is, 4"F0F1F2F3".
BITS (*n*)	A field of *n* bits containing a binary integer value This type is similar to a DASDL FIELD (*n*).
IMAGE (*n*)	An IMAGE data item of *n* bytes containing binary data (such as a scanned image or bitmap) rather than EBCDIC characters
REAL	A one-word floating point value
REAL (*n*, *m*)	A one-word binary integer with optional scaling
NUMBER (*Sn*, *m*)	A NUMBER data item with an optional sign digit followed by *n* 4-bit digits with optional scaling
FIELD (*booleanlist*)	A field of named bits (Booleans) **NOTE:** The format applies to ALTER declarations only.
GROUP (*dataitems*)	A group of related data items that could be called collectively by the group name rather than by the individual item names **NOTE:** The format applies to ALTER declarations only.

Glossary

This glossary includes terms that are unique to Databridge, as well as terms that are standard for DMSII databases. Complete, detailed definitions for Unisys MCP-hosted mainframes and DMSII terms can be found in Unisys documentation. The purpose of this glossary is to explain how these terms relate to Databridge.

Accessory. An Accessory is any program (including user-written programs) that links to a support library, such as DBEngine, DBSupport, or a user-written library.

Accessroutines. The Accessroutines program is a DMSII utility that controls access to the database, reads and writes records, and creates the audit trail.

audit file. The audit file is created by DMSII and contains the raw format of changes made to the DMSII database by update programs. Audit file records contain the deletes, adds, and modifies that were made to the various structures. It can contain, for example, hours, days, or weeks worth of information.

When an audit file is closed, DMSII creates the next one in the series. Audit files are closed for several reasons, including the following:

- An operator closes the audit file with the *mixnumber* SM AUDIT CLOSE command
- The audit file reaches the file size set in its DASDL
- There is an I/O error on the audit file
- There is not enough disk space for this audit file
- The database update level changes due to database definition changes
- Databridge performed an online clone without DBPlus
- The current audit file could not be found
- A file reorganization was executed to modify a DMSII structure

Databridge uses the audit file for the raw data of each database change to exactly replicate the primary database. Databridge records the audit location (AFN, ABSN, SEG, IDX) between runs, so it can restart without losing any records.

If you have Databridge Plus, Databridge can access up to and including the current audit file. If you do not have Databridge Plus, Databridge can access audit information only up to and including the current audit file minus one. Additionally, the audit file contains the update level at the time the audit file was created. The update level in the audit file and the update level in the DESCRIPTION file used by Databridge must match before Databridge will update a replicated database.

audit trail. The audit trail consists of all of the audit files generated for a database. It can contain recovery records, which indicate that there was a failure such as a HALT/LOAD that caused the Accessroutines to rollback the DMSII database to a quiet point. If DBEngine encounters one of these recovery records, it notifies the caller to rollback the replicated data accordingly.

The audit trail consists of the audit files named as follows:

databasename/AUDITnnnn

where *databasename* is the name of the DMSII database, AUDIT is a literal, and *nnnn* is the AFN (Audit File Number), a number between 1 and 9999. For example, if you have a database named BANKDB, an audit file would be named similar to the following:

BANKDB/AUDIT7714

client. The client is the computer system that will receive DMSII records from the primary database. The client could be a PC, a UNIX computer, or a mainframe. The client can have a relational or a DMSII database.

cloning. Cloning is the process of generating a complete snapshot of a data set to another file. Cloning creates a static picture of a dynamic database. Databridge uses the DMSII data sets and the audit trail to ensure that the cloned data represents a snapshot of the data sets at a quiet point, even though other programs may be updating the database concurrently. Databridge clones only those data sets you specify.

Cloning is one phase of the database replication process. The other phase is tracking (or updating), which is the integration of database changes since the cloning. For more details, see the definition for tracking.

Databridge Accessories are available for cloning, as follows:

- DBSnapshot Accessory uses a batch method that provides a one?time snapshot only.
- DBSpan Accessory uses a dynamic method that provides a one?time extraction and fixup followed by ongoing tracking.
- Databridge Clients perform an initial clone of a DMSII database and then subsequent tracking of the changes made to the DMSII database. Databridge Clients connect to Databridge Server or Databridge Enterprise to get the DMSII data.

compound item. An elementary item that could be altered into a GROUP item containing multiple elementary items.

For instance, assume that CUST-NAME ALPHA (30) has a 20?character last name, followed by a 9-character first name, and a 1?character middle initial. CUST-NAME could be altered to be a GROUP containing CUST-LAST-NAME ALPHA (20), CUST?FIRST-NAME ALPHA (9), and CUST-MID-INITIAL ALPHA (1).

consolidated file. A file created by DBSpan that contains records for all selected data sets.

CONTROL file. The DMSII CONTROL file is the run-time analog of the DESCRIPTION file. The DESCRIPTION file is updated only when you compile a modified DASDL. The CONTROL file controls database interlock. It stores audit control information and verifies that all database data files are compatible by checking the database timestamp, version timestamp, and update level. The CONTROL file is updated each time anyone opens the database for updates. The CONTROL file contains timestamps for each data set (when the data set was defined, when the data set was updated). It contains parameters such as how much memory the Accessroutines can use and titles of software such as the DMSUPPORT library (DMSUPPORT/*databasename*).

Databridge uses the CONTROL file for the following information:

- Timestamps
- INDEPENDENTRANS option
- AFN for the current audit file and ABSN for the current audit block
- Data set packnames
- Audit file packname
- Database usercode

DASDL. Data And Structure Definition Language—This is the language that defines DMSII databases. The DASDL must be compiled to create a DESCRIPTION file.

Databridge Plus. Databridge Plus is an optional utility that enables Databridge to access and retrieve information from up to and including the current audit file. If you do not have Databridge Plus, the most recent audit file Databridge can read is the current audit file minus one. For example, if the current audit file number is 23, Databridge can access audit file number 22 (23 - 1).

data set. A file (structure) in DMSII in which records are stored. It is similar to a table in a relational database. You can select the data sets you want to store in your replicated database.

DESCRIPTION file. The DESCRIPTION file contains the structural characteristics of a database (physically and logically). It is created from the DASDL source by the DASDL compiler and contains the layout (physical description), timestamp, audit file size, update level, logical database definition, and any static information about the database. It contains information about the database, not the data itself.

There is only one current DESCRIPTION file for each DMSII database. Databridge must have access to the DESCRIPTION file before it can replicate a database. Additionally, Databridge uses the DESCRIPTION file information for consistency checks between the primary database and the secondary or replicated database.

The DESCRIPTION file corresponds to the schema in a relational database.

extraction. The process of reading through a data set sequentially and writing those records to a file (either a secondary database or flat file).

file format conversion. A DMSII file format conversion affects file size values (for example, AREASIZE, BLOCKSIZE, or TABLESIZE), but it does not change the layout of the DMSII database.

flattening OCCURS. Changing an occurring item into a series of individual items.

formatting procedure. A procedure residing either directly or indirectly in DBSupport that contains routines for formatting the data items of a data set record. DBGenFormat generates formatting procedures such as COMMAFORMAT and BINARYFORMAT based on the setting of certain options in the DBGenFormat parameter file. You can write custom formatting procedures in ALGOL or COBOL to satisfy specific formatting requirements.

formatting routine. A section of code in a formatting procedure that formats a specific type of data item, such as an ALPHA.

garbage collection reorganization. A garbage collection reorganization moves records around, but it doesn't change the layout of the DMSII database. Its primary function is to improve disk and/or I/O efficiency by eliminating the space occupied by deleted records. Optionally, a garbage collection reorganization reorders the remaining records in the same sequence as one of the sets.

null text. The value defined in the DASDL to be NULL for that ALPHA data item. If the DASDL does not explicitly specify a NULL value for a data item, the NULL value is all bits turned on.

primary database. This is the original DMSII database that resides on the ClearPath NX/LX or A Series host. Databridge replicates from the primary database to one or more client databases. The client databases can be another DMSII database or one of several relational databases. Compare this to the replicated (or secondary) database.

quiet point (QPT). A point in time when no program is in transaction state. This can occur naturally, or it can be forced by a DMSII syncpoint. The quiet point is a point in time in the audit trail that Databridge uses as a reference point to help synchronize cloning or tracking of the DMSII database. Databridge uses the quiet points to ensure an accurate snapshot of the data. Audit addresses of these quiet points are stored in the replicated database for database synchronization purposes.

reformatting procedure. An ALGOL procedure that allows you to alter or convert data items to different layouts using custom written reformatting routines. The ALTER declaration in the DBGenFormat parameter file indicates which data items will be converted by the reformatting procedure. The procedure itself must call individual reformatting routines to convert the data items.

reformatting routine. An ALGOL conversion routine that alters the layout of a data item.

replicated database. This is the database that resides on the client (also called the client or secondary database) and that contains all of the records cloned from the DMSII database you specified for cloning. The replicated database is updated periodically with changes made to the primary (original) DMSII database. The periodic update (or tracking process) is explained later in this section. Compare this to the primary database.

replication process. The ongoing process of cloning and tracking a DMSII database. With the DBSnapshot Accessory, you can clone a database as a one time snapshot to flat files. With the DBSpan Accessory, however, you can extract the database to flat files and then subsequently update it by tracking. The DBSpan Accessory performs extraction as well as tracking.

secondary database. See replicated database.

set. An index into a data set.

structure. A data set, set, subset, access, or remap. Each structure has a unique number called the structure number.

table. A data structure in the client database corresponding to a data set or remap in the host DMSII database.

tracking. Retrieving only the changes from the audit trail to apply to the replicated database. Tracking is an ongoing process for propagating changes made to records in the DMSII primary database to the replicated database. The DBSpan and DBServer Accessories perform extraction as well as tracking.

Tracking is one phase of the database replication process. The other phase is cloning.

undigits. A NUMBER data item containing values from 10 to 15. The NUMBER data item should contain values from 0 to 9; however, it is possible for NUMBER data item to contain values 0 to 15. Because values 10 to 15 are not valid digit values, NUMBER data items containing values from 10 to 15 are called undigits.