Databridge Client Administrator's Guide

7.0 Service Pack 1

Table of contents

1. Databridge Client Administrator's Guide

This guide contains instructions for configuring and using the Micro Focus Databridge Client.

1.1 About this guide

While this guide was originally written for command-line Client operations, **it is applicable to both command-line or service-controlled Client operations**. To facilitate the transition to using service-based operations we provide instructions on how to perform the tasks described in this document using the Administrative Console and information on where to find configuration parameters in the Administrative Console's **Configure** command dialog pages.

For Administrative Console users this manual contains a lot of detailed information that they do not need to know; particularly in the area of writing user scripts. The Administrative Console's **Customize** command hides a lot of these details from you and lets you concentrate on what you are trying to do (usually the operations involve a single mouse-click plus a selection among the possible data formats in DMSII). Using the Administrative Console to control the running of the Client gives you the ability to fully automate Client operations and do comprehensive error recovery. For information specific to the service-controlled Client, see the following topics in this chapter or go to the **Help** in the Databridge Administrative Console:

- Using the Client Console and the Service

- Automating Client Operations with the Service

To install, configure, and run Databridge, you should be a system administrator familiar with the following:

- Standard Unisys® operations for MCP-hosted mainframes such as the CS7xxx series, Libra series, ClearPath® NX/LX or A Series

- DMSII databases and Data And Structure Definition Language (DASDL)

- File layouts and the description of those layouts for the files you will be replicating

1.2 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
bold	Folder names and program names. For example **logs**, **DBClient**.
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.

This term	Is used to indicate this
MCP server host mainframe	Unisys ClearPath NX, LX or A Series mainframe
DBEngine	Databridge Engine on the mainframe
DBEnterprise	Databridge Enterprise Server
DBServer	Databridge Server on the mainframe
Service	For UNIX Clients, consider this term synonymous with "daemon"

1.3 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

1.4 Related Documentation

When using Databridge, you may need to consult the following resources.

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 in PDF format. This documentation and current Knowledge Base articles are available on the Databridge Support Resources site. Documentation for Databridge Enterprise Server and the Databridge Client Console is also available from the Help menu and from the Databridge Documentation site.
Unisys MCP server documentation	If you are not completely familiar with DMSII configuration, refer to your Unisys documentation.

2. Introducing Databridge Client

Micro Focus Databridge is a combination of host and (optional) client software that provides automated replication of DMSII databases and flat files. All replications occur while the DMSII database is active. After the initial clone, Databridge updates the secondary database, copying only the DMSII data changes from the audit trail.

2.1 Choosing the Client Manager Service vs. the Command-Line Client

The Databridge Client provides two modes of operation. One mode lets you configure and run the Client from the Administrative Console, where the Client Manager service launches the Client and automates much of the replication process. The other mode uses a command prompt session (or terminal session in the case of UNIX) to run the command-line Client (**dbutility**). For a comparison of these modes of operation, see Comparing the Databridge Client Manager Service to Command-Line Operations.

While this guide can be used with either the command-line or service-controlled Client, **it is intended primarily for command-line Client operations.** For information specific to the service-controlled Client, see the following topics or refer to the **Help** in the Databridge Administrative Console:

2.2 Using the Administrative Console and the Service

The Administrative Console is an easy-to-use browser-based graphical interface that lets you access Clients on different platforms. You can view multiple data sources (defined by Databridge Server or Enterprise Server) and monitor all Client activity via onscreen messages and status updates.

Clients
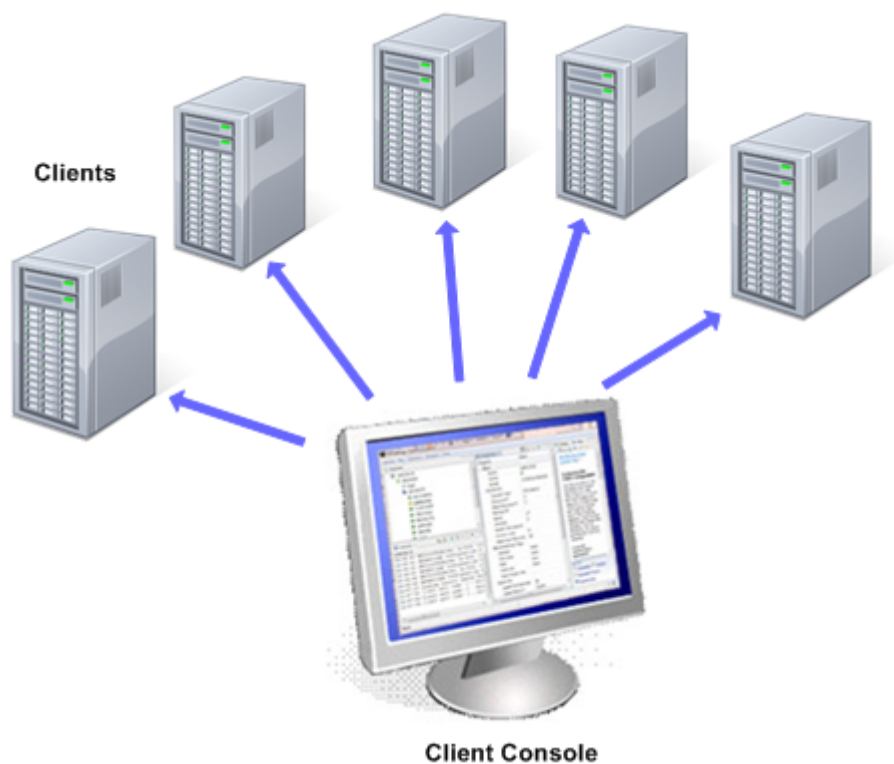
Client Console

The Administrative Console communicates directly with the Client Manager service, which starts Client runs in the background and completely automates Client operations. You decide when and how often Client runs are initiated by using the service scheduling settings in the Administrative Console. You can also use command files (shell scripts in UNIX) to interact with the service and automate processes. For more information, see Chapter 8, Automating Client Operations with the Service.

Enhanced Security

Because the service always starts the Client in the background, the Administrative Console is the only interface to service-initiated Client runs. Neither the service nor the background runs interact with the desktop or require that a user be signed on to the server. This makes service-initiated Client operations more secure than command-line operations and prevents the Client runs from being terminated, whether accidentally or maliciously. The service starts at system startup (by default), which ensures that replication can continue in the event of system failure without outside intervention.

In addition to launching Client runs, the service routes all log and informational messages to the Administrative Console and facilitates operator tasks using the **DBClient** and **DBClntCfgServer** programs. The first program, **DBClient**, performs DMSII processing and cloning tasks. The second program, **DBClntCfgServer**, handles Administrative Console requests that require access to the relational database (such as `define`, `generate` and `reorganize` commands). The activity and output of these programs is displayed in the Administrative Console's console window.

The following diagram shows the Client architecture for the two types of clients: the command-line Client (**dbutility**) and the service-controlled client (**DBClient** and other components).
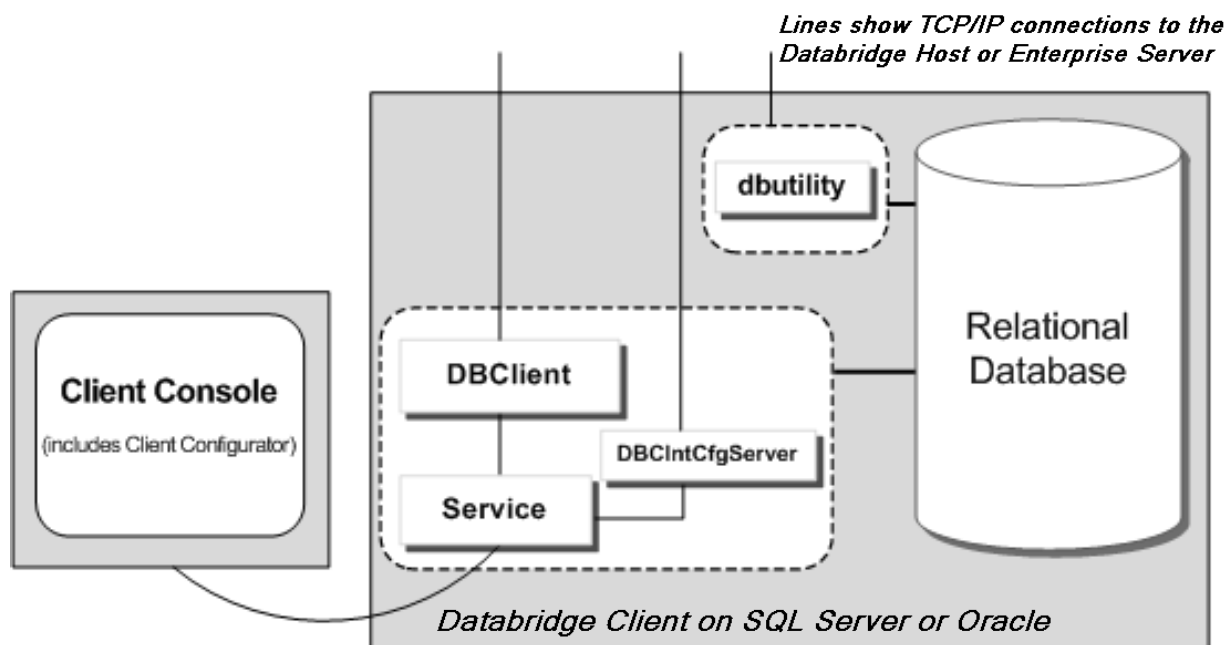
The Administrative Console can perform data source customizations by clicking on **Settings > Customize** after selecting the data source. The **Customize** command lets you easily customize your data sources without any knowledge of SQL or how your Client control tables work. Instead of hand-coding SQL user scripts, you can select options to configure your data sources and map DMSII data to your relational database tables. To use the **Customize** command with existing Client configurations that employ user scripts, you must first upgrade your Client control tables using the **dbscriptfixup** utility. (See the *Databridge Installation Guide*.)

> ✎ **Note**
>
> To customize a specific data source, you must first navigate to the data sources page for the Client Manager in question by using the provided links (**Databridge Servers > Client Managers**). If you are in the monitor page click on the link for the Client Manager on upper left hand corner of the group of entries for the Client Manager in question. Place your mouse over the desired data source name and click on it to select it (the line should turn light blue). Click on the **Settings > Customize** to start the customizer.

2.3 How Replication Works

The Databridge Client controls the replication process of DMSII databases and flat files. It initiates connections to Databridge Server on the host and maintains the state information necessary for resuming replication in case operations are interrupted. At the heart of the host-based Databridge system is the Databridge Engine---a system library that retrieves structural information and data from the DMSII database and passes it to Databridge Server. When you opt to use Enterprise Server with the Databridge Client instead, Enterprise Server takes over much of the functionality of the Databridge Engine and Databridge Server.

The Databridge Support Library, also installed on the host, provides filtering, formatting, and reformatting services for the Databridge Server. See Client vs. Host Filtering.

After the Databridge Server receives data from the Databridge Engine, it calls the Support Library to determine if the data should be replicated, and if so, it passes the data to the Support Library for formatting.

Replication involves three discrete phases, as described below. These three phases are tracked for each data set in the `ds_mode` column of the DATASETS control table as values 0, 1, and 2.

Data Extraction	This phase (identified by a mode of 0) applies only to data sets that have not yet been cloned. During this phase, the Databridge Engine sequentially reads all of the records from the data sets being cloned and passes them to the Databridge Client. Using the appropriate bulk loader utility, the Client populates the relational database tables and creates indexes for the tables.
Fixup	During this phase (identified by a mode of 1), the Databridge Engine processes audit files and passes all of the DMSII updates that occurred while data extraction was taking place to the Client, which updates the relational database. This phase is fully restartable. During this phase the relational database is not fully consistent, as the not all the tables are in-sync. The only difference between the Fixup Phase and the Update (or Tracking) Phase is that the Client has to deal with conditions caused by the fact that the tables from which records were extracted were changing as the extraction was taking place. Until the audit file processing gets past the point in the audit trail where the data extraction ends, the Client behaves somewhat differently in order to handle such issues as updates to records that are not in the tables, deletions of records that are not in the tables, and inserts of records that are already in the tables.
Update	During this phase (identified by a mode of 2), the Client processes audit files and then passes all of the DMSII database updates to the Client, which updates the relational database. This phase is also referred to as the change tracking phase. Databridge uses `quiet points` to synchronize the `replicated database` with the DMSII database and ensure accuracy of the data. Quiet points mark the start of a group of updates, which is referred to as a transaction. When the Databridge Engine reaches the end of the last DMSII audit file (or encounters a program that did a rollback), it usually rolls back the transaction and instructs the Client to roll back updates. The Client stores quiet point information with other state information in a set of control tables, referred to as the Client control tables, and uses it to restart the replication process. If near real-time replication is required, set the parameter `use_dbwait` to true. This causes the Engine to enter a wait-and-retry loop for a configurable amount of time, instead of returning an audit file unavailable status, which normally occurs when no more Audit Files are available.

2.4 Client vs. Host Filtering

Use the following guidelines to determine when to use the host instead of the Databridge Client to perform filtering.

Filtering Columns

On the host side, you can filter columns by creating a filtering routine with the DBGenFormat utility. On the Databridge Client side, you can filter columns the same way you can filter data sets, which is to set the `active` column to 0 for the corresponding entry in the DMS_ITEMS Client control table.

The advantage of performing the filtering on the Databridge Client side is that you save on host resources. However, there are a few cases where you should consider filtering on the host side, as follows:

- If you plan to filter *many* columns, consider filtering on the host side to reduce TCP/IP traffic. The best way to determine this is to try the filtering both ways and see which gives you the best throughput.
- If you plan to filter columns with confidential or sensitive information, it is best to perform the filtering on the host.

Filtering Data Sets

You can filter data sets on the host side by using a logical database or by creating a filtering routine with the DBGenFormat program. On the Databridge Client side, you can filter data sets by setting the `active` column to 0 for the corresponding entry in the DATASETS Client control table.

If you want to filter data sets that contain confidential or sensitive information, consider using a logical database or a filtering routine in the DBGenFormat utility. In this case, the Databridge Client will have no record that these data sets exist.

Filtering Rows

Row filtering limits data to certain ranges; you can accomplish this via the WHERE clause of filtering routines created with the DBGenFormat program on the host. For more information, see Chapter 4 in the *Databridge Host Administrator's Guide*.

Filtering OCCURS Tables

OCCURS tables are secondary tables generated by the Databridge Client when OCCURS clauses for items (or GROUPs) are not flattened. Frequently, not all rows in such tables contain meaningful data, for this reason it is desirable to filter such rows to reduce the storage requirements and improve performance. Starting with version 6.5, the Databridge Client implements row filtering for OCCURS tables. For more information, refer to OCCURS Table Row Filtering.

2.5 Databridge Components

The following table lists all of the Databridge products and components that can have a role when replicating data with the Databridge Client.

Databridge Host (installed on the mainframe)

Component	Description
Databridge Engine (DBEngine)	The main component of the Databridge software, DBEngine is a host library program that retrieves structural information, layout information, and data from the DMSII database and passes the information to the Databridge Server. Additionally, it retrieves updates by reading the audit files on the host and sends the changes to the Client.
Databridge Server (DBServer)	An accessory that provides communications between DBEngine and the Databridge Client, and also between DBEngine and Databridge Enterprise Server. DBServer responds to Databridge Client requests for DMSII data or DMSII layout information.
Support Library (DBSupport)	A library that provides formatting and filtering to the DBServer and other accessories. After DBServer receives data from the DBEngine, it calls the Support Library to determine if the data should be replicated, and if so, passes the data to the Support Library for formatting.
DBGenFormat	A host utility that creates filter and format routines. The DBGenFormat utility interprets the DBGenFormat parameter file to generate ALGOL source code patches, which are included in the tailored Support Library.

Databridge Enterprise Server

A Windows-based product that provides the same functionality as the Databridge Engine (DBEngine) and Databridge Server (DBServer) on the host. Enterprise Server offloads much of the replication workload from the Unisys mainframe to a Windows computer, reducing mainframe resource utilization and initial load time.

Databridge Clients can connect directly to Enterprise Server, which in turn connects to DBServer on the mainframe. If MCP disks are directly accessible from the Windows server, Enterprise Server extracts the DMSII data directly. Enterprise Server reads the audit trail on the host to retrieve updates that occurred during the extraction and sends the changed information from the audit file to the Client. If MCP disks are not directly accessible, Enterprise Server uses DBServer to retrieve blocks of data from DMSII data sets or the audit files. Enterprise Server provides high-speed file transfer between the host and the Windows environment and audit file mirroring.

Component	Description
DBEnterprise	The executable file for Enterprise Server, frequently used interchangeably with Enterprise Server.
Databridge Director (DBDirector)	A Windows Service that listens for Client connection requests and starts DBEnterprise whenever a connect request is received.

> ✏️ **Note**
>
> We use terms "Databridge Server" and "Databridge Engine" throughout the rest of this manual as generic terms that apply to either "DBServer" and "DBEngine" on the mainframe or to the equivalent component in "Databridge Enterprise Server".

Databridge Client

The Client initiates a connection with the Databridge Server and then specifies the DMSII data sets to be replicated from a DMSII database.

Component	Description
Client Manager (**DBClntControl**)	The service (Windows) or daemon (UNIX) that automates most Client operations. It handles operator requests from the Administrative Console and routes all log and informational messages to the administrative consoles.
DBClient	A Client program that is launched by the service. **DBClient** handles the processing of DMSII data and updates the same as **dbutilty**, except that it runs as a background run and uses the Administrative Console to display its output and interact with the operator.
DBClntCfgServer	A program that handles all requests specific to a data source from the Administrative Console. These requests include updating the Client configuration file, providing access to the Client control tables, and handling the **Customize** command. Like **DBClient**, this program is run by the service as a background run.
dbutilty	A program that runs the Databridge Client from a command line.
Batch Console (**bconsole**)	A program that allows Windows command files (UNIX shell scripts) to issue console requests to the Databridge Client Manager. The Batch Console executes console commands in script files that are written in a language that vaguely resembles Visual Basic.
Administrative Console	A browser-based application that is controlled by a server which connects to the Databridge Client Managers. The Administrative Console, besides providing a replacement for the Eclipse based Client Console and the Client Configurator, also implements a monitor that allows you to monitor the operations of the various Client Managers and the Clients they control from a single screen. To connect to the Administrative Console server from the browser use the following URL https://*hostname*:7445/ where *hostname* is the name or IP address of the Administrative Console server.

Databridge FileXtract

An application that allows you to clone and update **Flat Files** that reside on Unisys ClearPath NX, LX, or A Series mainframes. You can also use FileXtract with the Databridge Client to replicate this data. From the Client perspective, FileXtract data sources look like DMSII data sources.

FileXtract is bundled with Databridge Host software and includes several Reader libraries and other associated files.

> ✏️ **Note**
>
> A flat file is a plain text or mixed text and binary file which usually contains one record per line. Within the record, individual fields may be separated by delimiters, such as commas, or have a fixed length and be separated by padding. An example of a flat file is an address list that contains fields for Name and Address.

Databridge Flat File Client

The Flat File Client (also known as PCSPAN) is a Windows implementation of the DBSPAN accessory on the MCP. As is the case with DBSPAN, rather than update the secondary database, the Flat File Client creates data files that contain the data records for the updates. This approach is useful when a Databridge Client does not exist for a particular database or platform or when the data has to be transformed before being loaded into a secondary database. The Flat Client has a very similar architecture to the relational database clients, such as the SQL Server and the Oracle Clients.

Databridge Kafka Client

The Databridge Client for Kafka, which is implemented on Linux platforms only, enables the ability to utilize the Kafka messaging system within the Databridge architecture. The Kafka messaging system is a scalable fault-tolerate data management system that provides efficient real-time data processing.

2.6 Comparing the Databridge Client Manager Service to Command-Line Operations

The Databridge Client Manager service performs the same operations as the command-line Client, **dbutility**. We refer to these operations using the term "service" whether the Client Manager service is running on Windows platforms (service) or UNIX/Linux platforms (daemon). Each machine has its own service (Windows) or daemon (UNIX/Linux). The primary advantage to using the service is its ease of use and the ability to automate most of your Client processes. Additionally, Client runs initiated by the service can't be interrupted or tampered with as they occur as background runs.

The following table can give you a better idea of how the two modes of operations compare when performing Client-specific tasks.

To do this	With this `dbutility` command	With the Databridge Client Manager (via Administrative Console)
Create Client control tables	`configure`	The Client control tables are automatically created if they don't already exist when you run a `define` or `redefine` command or you run a **Customize** command from **Settings** menu for the data source.
Clone the data sets specified on the command line	`clone`	To clone selected data sets, click on the **Clone Data Sets** command in the **Advanced** menu for the data source. The resulting dialog allows you to select the data sets to clone, and it also allows you to add command line options.
Populate the Client control tables with information about the DMSII data set layouts and the corresponding relational database table layouts	`define`	Click on the **Define/Redefine** command in the **Actions** menu for the data source. **DBClntCfgServer** executes the appropriate command (`define` or `redefine`).
Apply changes from the primary database to the relational database structure while preserving existing information	`redefine`	Click on the **Define/Redefine** command in the **Actions** menu for the data source. **DBClntCfgServer** executes the appropriate command (`define` or `redefine`). To run a `redefine` command with the `-R` option (i.e. redefine all data sets) from the Administrative Console, click ON **Redefine (with options)** in the **Advanced** menu for the data source and check the "Redefine All Data Sets" checkbox.
To write the Client control table entries to the log file	`display`	To write control tables to the log file, click on the **Log Control Tables** command in the **Advanced** menu for the data source.
To create script files	`generate`	Click on the **Generate Scripts** command in the **Actions** menu for the data source. This is the equivalent of running the `generate` command with no command-line options. To force all script files to be recreated in the **dbscripts** subdirectory, click on the **Generate All Scripts** command in the **Advanced** menu for the data source. NOTE: This is equivalent to using the –u option in the command line Client to create and place all of the script files in the **dbscripts** subdirectory.

To do this	With this `dbutilty` command	With the Databridge Client Manager (via Administrative Console)
To perform the initial clone or process DMSII database updates	`process`	Click on the **Process** command in the **Actions** menu for the data source. The service, which controls scheduling for all `process` commands, starts **DBClient** at the scheduled time (if specified) and terminates **DBClient** when the `process` command finishes. You can run this command anytime. To add command line options to `process` commands for runs initiated from the Console, click on the **Process (with options)** command in the **Advanced** menu for the data source, then choose the options you need from the provided set of checkboxes.
To recreate the stored procedures for tables associated with a given data set in the specified data source (for example, after a DMSII reorganization)	`refresh`	Click on the **Refresh Data Set** command in the **Advanced** menu for the data source. You can either refresh a specific data set or all data sets.
To alter the relational database using the scripts created by the `redefine` command. The command automatically refreshes the scripts and stored procedures associated with the tables whose layouts have changed.	`reorg` or `reorganize`	Click on the **Reorganize** command in the **Actions** menu for the data source..
To run user scripts or Databridge Client scripts	`runscript`	Click on the **Run Script** command in the **Advanced** menu for the data source. This command runs the script in the user script directory (`user_script_dir`), the name and location of which is defined in the Client configuration file. If you start the filename with a backslash for a Windows Client or a slash for a UNIX Client, this command uses the -n option, which overrides the directory specification.
To close an audit file on the host	`switchaudit`	Not supported.
To back up the Client control tables	`unload`	Click on the **Unload Data Source** command in the **Advanced** menu for the data source.
To restore the Client control tables using the backup file	`reload`	Click on the **Reload Data Source** command in the **Advanced** menu for the data source.
To export the binary configuration file to an editable text file	`export`	Click on the **Export Client Configuration** command in the **Advanced** menu for the data source. You can only execute the `export` command with the default command-line options from the Administrative Console.

To do this	With this dbutilty command	With the Databridge Client Manager (via Administrative Console)
To import a text configuration file (and convert it to a binary file) for use with the Client	`import`	Not available. The configuration file is updated directly from the Administrative Console.
To create user scripts to back up customizations made by the **Customize** command	`createscripts`	Click on the **Create User Scripts** command in the **Advanced** menu for the data source.

For more information about **dbutilty** commands, see dbutility Commands. For more information about the Administrative Console, see Using the Console and Service and the Help included in the Administrative Console.

2.7 Switching from Command-Line to Service Operations

Use this procedure if you currently run the Databridge Client from a command line and want the Client Manager to run it, or if you need to add existing data sources to the Client Manager.

To operate the Client Manager on the Client machine, you'll need a specific directory structure, referred to as the *service's working directory*. You can use the migrate utility to create this directory.

> 🔥 **Important**
>
> If you're in the process of upgrading your Databridge software, use the instructions in the Databridge Installation Guide for upgrading the Client.

## 2.7.1 To switch to the service based client (Windows)

> ✏️ **Note**
>
> Using the 6.6 console with the 7.0 Client software will not work well. The 7.0 Admin Console will not work with the 6.6 Client software. You should always run matching software for the Administrative Console and the Client.

1. Set up the service's working directory. If you use the Client on Windows, you can run the Migrate program to do this. For more information, see **The Working Directory** in the *Databridge Installation Guide*.

2. Install the Administrative Console that matches the version of Databridge Client software you use. For instructions, see the Databridge Installation Guide.

3. Do one of the following:

   • If you use a text Client configuration file, proceed to step 4.

   • If you use a binary Client configuration file (that is, you have not exported your configuration file to a text file to edit it), skip to step 6.

4. From the data source directory, locate the **config** folder, and copy the text configuration file "dbridge.cfg" to a file named "dbridge.ini".

   The following image shows an example of a working directory for a data source named "zdatabase".
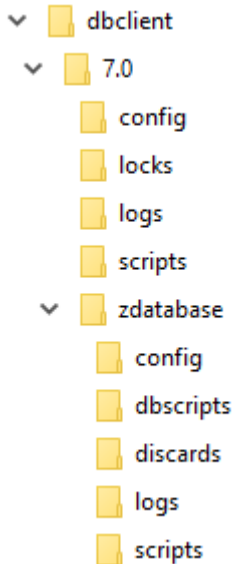
5. From a command prompt, go to the working directory for your data source and run the **dbutilty** `import` command. This creates a binary configuration file required by the service.

6. To make sure that the Client Manager service is running, from the **Start** menu, click **Control Panel** > **Administrative Tools** > **Services**, double-click **Micro Focus Databridge Client Manager 7.0**, then click **[Start]**.

7. Using a browser such as Microsoft Edge or Google chrome enter the following URL https://*hostname*:7445/ where *hostname* is the name or IP address of the Administrative Console server.

8. From the Databridge Server page click **+Add** and provide the hostname or IP address of the Client Manager and the port it listen on (typically 8001). This will add the node to the list of servers and detect the fact that it is a Client Manager.

9. Repeat step 8 for the Databridge server you wish the data source be associated with. This will add the node to the list of servers and detect the fact that it is a Databridge server on the MCP or a Databridge Enterprise Server. It will detect if the connection to the Databridge Server on the MCP is encrypted or not.

10. Go to the page for the Client Manager you just added by Clicking on Client Managers link and click on the Client Manager in question. Click on the **+Add** menu and click on **New** to open the "Add new data source" dialog.

11. Select the Databridge server from the dropdown list provided. Note that if you skipped Step 9, you can add the server by pushing the **Add new server** radio button and providing the server name or IP address and the port it listens on. When done push the continue button which move you to the next page of the dialog.

12. The Administrative console retrieves the data source names configured for the server in question and provides you with a list box from which you can select the desired data source.

13. Fill in the rest of the boxes on the page with the various relational database information that is needed. This includes the ODBC data source name for the SQL Server Client and the database name for the Oracle Client and the relational database signon information user/password and in the case of the SQL Server you can use Integrated Windows Authentication. When done click the **Add** button on the lower right corner of the display.

2.7.2 To switch to the daemon based client (UNIX/Linux)

1. Log on as the userid specified in the USERID parameter of the file `globalprofile.ini` . This is the same userid you currently use to run the command-line Client.

2. Set the current directory to the home or other directory to which you have write access and copy the script `dbdaemon` from the install directory to it. This allows you to make changes to the script.

3. In an editor, open the script `dbdaemon` .

4. Make sure that the environment variables (such as INSTALLDIR, WORKING_DIR, ORACLE_HOME, LD_LIBRARY_PATH) are correct for your system and edit them as needed.

5. Save and close the script.

6. Start the daemon by typing the following:

```
dbdaemon start
```

7. To verify that the daemon is running use the `ps` command, which should produce output similar to the following:

```
databridge@VMOPENSUS114-64:~> ps -ef | grep DBC
databridge   1110     1  0 12:00 ?        00:00:00
/opt/dbridge70/DBClntControl
```

8. If the daemon doesn't start, in the script `dbdaemon` , make sure that the WORKING_DIR and INSTALLDIR environment variables are correct. Also, check the Client Manager's working directory to determine if a file "dbdaemon.log" was created and if it contains any clues.

9. Using a browser such as Microsoft Edge or Google chrome enter the following URL https://*hostname*:7445/ where *hostname* is the name or IP address of the Administrative Console server.

10. From the Databridge Server page click **+Add** and provide the hostname or IP address of the Client Manager and the port it listen on (typically 8001). This will add the node to the list of servers and detect the fact that it is a Client Manager.

11. Repeat step 10 for the Databridge server you wish the data source be associated with. This will add the node to the list of servers and detect the fact that a Databridge server on the MCP or a Databridge Enterprise Server. It will detect if the connection to the Databridge Server on the MCP is encrypted or not.

12. Go to the page for the Client Manager you just added by Clicking on Client Managers link and click on the Client Manager in question. Click on the **+Add** menu and click on **New** to open the "Add new data source" dialog.

13. Select the Databridge server from the dropdown list provided. Note that if you skipped Step 11, you can add the server by pushing the **Add new server** radio button and providing the server name or IP address and the port it listens on. When done push the continue button which move you to the next page of the dialog.

14. The Administrative console retrieves the data source names configured for the server it question and prides you with a list box from which you can select the desired data source.

15. Fill in the rest of the boxes on the page with the various relational database information that is needed. This includes the database name for the Oracle Client and the relational database signon information user/password. When done click the **Add** button on the lower right corner of the display.

16. Type `su` to switch to the root user and then copy the script `dbdaemon` from the location you specified in step 5 to the following location:

   • (Linux/Solaris) `/etc/init.d`

   • (AIX) `/etc/rc.d/init.d`

   • (HP-UX only) `/sbin/init.d`

17. To make the operating system automatically start the daemon whenever the system starts, consult the documentation for your operating system.

2.8 Command-Line Client Operations

This section describes the tasks required to clone a DMSII database and track changes using the Databridge Client from a command line. This sequence assumes that the relational database is properly configured and the required Databridge software is installed and running.

1	*Update the configuration file*	Because the Client reads the Client configuration file before writing the script files and the log file, you may need to modify this file. To edit the configuration file, you must export it to an editable format and then import it after you make your changes. For more information, see Export or Import a Configuration File.
2	*Run the dbutilty `configure` command*	In a first-time installation, you must run the **dbutility** `configure` command. This creates the Databridge Client control tables that will hold the layout information for the DMSII database and corresponding relational database tables.
3	*Run the dbutilty `define` command*	This command creates an entry in the DATASOURCES Client control table that contains the data source name, the host name and the port number on which Databridge Server listens. It also populates the Client control tables with the DMSII layout and creates the corresponding table layout for the relational database. `**dbutilty define` *datasourcename hostname portnumber*
4	*Create user scripts*	To customize the Client control tables, you must create user scripts. Or, you can use the default mapping that the Databridge Client produces. Creating user scripts is an iterative process that involves writing scripts to select and map data sets and customizing data tables, and then testing and updating the scripts until the resulting data table layout meets your specifications. When you are satisfied with the results of your user scripts, re-run the `define` command with the `-u` option.
5	*Run the `generate` command*	This command creates scripts that the Databridge Client uses to create and populate the data tables from the data sets that you selected for cloning. You can modify the configuration of your data tables by adding optional parameters to the end of the create table and create index statements. Do not directly modify scripts created by the `generate` command; your changes will be lost the next time your run this command. **dbutilty** `generate` *datasourcename*
6	*Review the bulk loader options*	Review the sections on the bulk loader utilities and their related options, making changes to the bulk loader parameters in the configuration file as needed. If you make changes to the Client configuration file, you will need repeat step 5 with the `-u` option added, as the scripts will otherwise not reflect these changes.
7	*Clone the database*	To clone the DMSII database, run the `process` command. If you want to clone only a few data sets, use the `clone` command. This step populates the Databridge data tables in the relational database with actual DMSII data. **dbutilty** `process` *datasourcename*
8	*Update the database*	To update the relational database with changes made to the DMSII database, run the `process` command. Or, to make the Databridge Engine wait for updates when it reaches the end of the audit trail, set the configuration parameter `use_dbwait` to true or use the command-line switch `-w`.

2.9 Before You Run the Command-Line Client

Before you use **dbutility**, do the following:

- Set up the relational database.

- Install the Databridge software on the host and set up and start Databridge Server (see the Databridge Host Administrator's Guide). If Databridge Server requires signon parameters, include these in the **dbutilty** configuration file. If necessary, you can also provide these parameters in the environment variables or in **dbutilty** command-line options.

- Install or upgrade the Databridge Client (see the Databridge Installation Guide) and create a separate working directory for each DMSII database to be cloned. In most cases, you'll add the relational database logon parameters to the data source configuration file. You can do this by supplying the signon parameters to the `import` command using command line switches (such as `-U -P -O -W`) when you create a new data source. Passwords are automatically encoded.

> 🔥 **Important**
>
> When you execute **dbutility** commands for different data sources, make sure that you have set the current directory to be the directory created for that data source. This ensures that the `process` or `clone` command can locate the configuration files and the scripts created by the `generate` command.

2.9.1 Controlling and Monitoring dbutility

The Databridge Client includes a command-line console to help control and monitor **dbutility**. Commands are entered in the same command prompt session where **dbutility** is running. When **dbutility** is running, make sure that the focus is on the command prompt window and then use the keyboard to enter the command. You need to type only enough of the command to make it unique.

The configuration file parameter `inhibit_console` allows you to disable the command-line console, and the command-line option -C toggles this parameter. The available commands are as follows:

This Command	Allows You To
co[mmit] a[bsn] <number>	Dynamically changes the value of the parameter `commit_absn_inc`. The change only takes effect at the next commit. The allowable range of values is the same as that of the parameter commit_absn_inc.
co[mmit] ti[me] <number>	Dynamically changes the value of the parameter `commit_time_inc`. The change only takes effect at the next commit. The allowable range of values is the same as that of the parameter commit_time_inc.
co[mmit] tr[ans] <number>	Dynamically changes the value of the parameter `commit_txn_inc`. The change only takes effect at the next commit. The allowable range of values is the same as that of the parameter commit_txn_inc.
co[mmit] s[tats]	Displays the commit statistics.
co[mmit] u[pdate] <number>	Dynamically changes the value of the parameter `commit_update_inc`. The change only takes effect at the next commit. The allowable range of values is the same as that of the parameter commit_update_inc.
h[elp]	Displays a list of available commands.
l[ogswitch]	Closes the current log file and starts a new one.
p[stats]	Displays the performance statistics. These statistics are the same as those shown after an audit file switch when the `show_perf_stats` configuration file parameter is set to True.
q[uit] [at hh:mm | after afn]	Terminates the program at the next quiet point. Issuing this command during data extraction has no effect until the fixup phase starts. Optionally, you can specify either to terminate the program at a particular time or to terminate it after a particular audit file is processed.
q[uit] now	Terminates the program immediately by first closing the TCP/IP connection to the server. This command is particularly useful when using DBWAIT and there are no updates available. This command will not take effect immediately if the Client is waiting for an SQL operation to complete.
sc[hed] {on | off}	Disables or enables update scheduling. For instructions on scheduling updates, see Scheduling dbutility Updates.
ss[tats]	Displays statistics for Databridge Server at the next quiet point.
st[atus]	Displays a variety of status information, such as the current stateinfo during audit file processing, the trace status, and the program status (for example, waiting for TCP for 100 ms).
tr[ace] trace_mask	Dynamically alters the trace mask; for more information, see Enabling a Trace.
ts[witch]	Closes the current trace file and starts a new one.
v[erbose] {on | off}	Alters the setting of the verbose flag; see -v in dbutilty Command-Line Options.

2.9.2 Databridge Client Exit Status Values

After each command, the Databridge Client command-line program, **dbutilty**, returns an exit status value.

exit_status Value	Description
0	Run completed successfully.
1	This value identifies generic Databridge Client errors.
2	Client `process` or `clone` command exited because of a DMSII reorganization. **NOTE:** An exit_status of 2 can occur only with the `process` and `clone` commands.
0*nnn*	Client exited because of a Databridge Engine error. The error is listed in Appendix A of the *Databridge Host Administrator's Guide*.
10*nn*	Client exited because of a DBServer error. The error is listed in Appendix A of the *Databridge Host Administrator's Guide*.
11*nn*	Client exited because of a DBEnterprise error. The error is listed in Appendix A in the *Databridge Enterprise Server Guide*.
20*nn*	Client exited because of a recognized Databridge Client error. The error is listed in Appendix E.
97*nn*	Client exited because of a recognized TLS error. The error is listed in Appendix E.

For detailed information about Client exit codes, see Appendix E: Client Exit Codes.

2.9.3 Testing for Exit Status

The following examples show how you can test for the exit status (`exit_status`).

Windows Example

This example tests for the `exit_status` in a Windows .cmd file:

```
dbutility process datasource
if errorlevel 2 if not errorlevel 3 goto reorg
exit
:reorg
echo DMSII reorganization has occurred
sendmail "message"
```

where <sendmail> is a hypothetical user application that sends a notification to you.

UNIX Example

The following example, written for the UNIX Korn shell (ksh), determines whether or not the value of the exit status (that appears after you run the **dbutility** program) indicates that a DMSII reorganization was detected. If a reorganization was detected, it echoes a message and runs the sendmail program:

```
dbutility process datasource
if [$? -eq 2]
then
echo "DMSII reorganization has occurred"
sendmail "message"
fi
```

where *sendmail* is a hypothetical user application that sends a notification to you.

3. Getting Started

Use the topics in this section to prepare to clone a DMSII database.

Before you clone a DMSII data source, you must create Client control tables. Once they're created, you can populate them with DMSII information sent by the Databridge Engine and create the actual data tables that will store your cloned data. The resulting data tables are based on information from the Client control tables and any additional customizations you make.

To customize how DMSII data sets get mapped to their corresponding relational database tables, you can use the Administrative Console's **Customize** command in the **Settings** menu or you can write user scripts in SQL. The **Customize** command is compatible with command-line operations and can help make reorganizations easier to handle. For more information on the **Customize** command, see the **Help** in the Administrative Console. Writing and testing user scripts is typically time-consuming and requires a bit of trial and error. For instructions, see Customizing with User Scripts, in this chapter.

3.1 Creating Client Control Tables

Use this procedure to create a set of control tables in a new installation when using the command line Client. If you are using the Administrative Console, the Client control tables are created automatically when you define a new data source, unless the tables already exist.

When you upgrade your Databridge Client software, the **dbfixup** program updates your control tables to the current version. It first alters the control tables and performs any required updates to them. Then, it unloads the Client control tables to a file, recreates them using a **configure** command, and restores them from the unload file. If a problem occurs and you need to revert to the older version of the Client, the unload file will let you reload the Client control tables using the older version of the Client.

Client control tables contain information about the data sets in a DMSII database and information about the corresponding tables and their layout in the relational database. These tables are stored and maintained in the relational database.

To run a `configure` **command**

1. Make sure that the database software and the appropriate Windows services or UNIX processes are running. For more information, see your relational database documentation.

2. Make sure that the current directory is the working directory for the data source (DMSII database) that you are cloning. For details about the working directory, see the *Databridge Installation Guide.*

3. From a Windows Command Prompt or UNIX terminal session, enter the following:

   dbutility [*signon_options misc_options*] **configure**

Where	Is
signon_options	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** `[-U userid] [-P password] [-D database]` **SQL Server:** `[-U userid] [-P password] [-W] [-O ODBCdatasource ]`
misc_options	Any of the following miscellaneous command-line options: `-T` forces the Client to use a new trace file for this run, if tracing is enabled. `-f` filename lets you specify a configuration file other than the default "dbridge.cfg" file in the **config** subdirectory of the Client's working directory. `-L` forces the Client to use a new log file for this run. `-u` , if you want to override conditions that **dbutility** would otherwise interpret as a possible user error. For example, you may want to create a second set of control tables within one relational database (these must be owned by a different user), or you might want to drop and re-create the control tables, removing all of the state information associated with the user tables. In most cases you do not need the `-u` option.

   You only need to run **dbutility** `configure` once, unless you drop your tables via the **dbutility** `dropall` command and have to start over.

4. (Optional) To display the tables created by **dbutility** `configure` , use a utility appropriate for your relational database. For example, for Microsoft SQL Server you can use the Query window of SQL Server Management Studio to enter the following:

   ```
   select name from sysobjects where type = "U" order by name
   ```

   For Oracle, you can use the SQL*Plus utility and enter the following:

   ```
   select table_name from user_tables order by table_name
   ```

3.1.1 Creating a Second Set of Tables

Occasionally, sites create a test environment that includes a second set of Client control tables. Even though the second set of tables may coexist with the production environment, this type of test environment can negatively impact the performance of your production database and is not recommended.

Creating a second set of Client control tables in the same relational database requires a separate database user ID. You must keep the working directories for the two user IDs separate. Because table name duplications are allowed when using different user IDs, this can lead to script files with the same names. If you create a set of Client control tables by running a `configure` command under the user ID "usera", those tables will be owned by usera (for example, USERA.DATASOURCES). If you later use the user ID "userb" to run a `configure` command.

Databridge Client creates a new set of Client control tables owned by userb (for example, USERB.DATASOURCES). Usera and userb tables are treated independently of each other.

3.1.2 Dropping and Re-Creating Client Control Tables

If you drop and create the Client control tables, you must re-clone everything. All Databridge tables that contain DMSII data will remain in the relational database. However, all of the information required from the Client control tables to request updates from the Databridge Engine will be lost.

Under normal circumstances, use the `dropall` command before running another `configure` command. This ensures the following:

- Tables and stored procedures created by the Databridge Client are dropped from the relational database
- All scripts created by the Databridge Client are deleted from the **dbscripts** subdirectory of the working directory

In some rare cases where the `dropall` command cannot drop all the tables, you may want to drop and create the Client control tables directly by running **dbutility** `configure` as follows:

```
dbutility -u configure
```

The -u option bypasses the checks that prevent you from doing this when the Client control tables exists.

If you attempt to execute a **dbutility** `configure` command after the Client control tables have been populated, without adding the -u option, you get the following message:

```
ERROR: Databridge control tables are not empty, use dropall command first
- To bypass this test use the 'u' option for the configure command
```

This message is provided as a safeguard so that you do not inadvertently drop and create the Client control tables.

3.1.3 Updating Client Control Tables

You can update some values in the Client control tables. For best results, use the Administrative Console's **Customize** command to customize the table layout for the relational database instead of using user scripts. Avoid using tools like SQL Server Management Studio or SQL*Plus to directly update the control tables.

> ✏️ **Note**
>
> If you previously used user scripts to update control tables and want to switch to using the **Customize** command, you'll need to update your Client control tables first. For instructions, see the *Next Steps* section in the *Databridge Installation Guide*.

Where	Is
Values in the `active` column of all tables	Set cloning on or off for data sets or DMS items.
Values in the `dms_subtype` column of the DMS_ITEMS tables	Specify the format of a field that is to be cloned as a date.
The `set_name` column in the DATASETS table and the `item_key` columns in the DMS_ITEMS table	Create a composite key.
The columns `ds_options` in the DATASETS table, `di_options` and `di_options2` in DMS_ITEMS, `dt_options` in DATATABLES and `da_options` in DATAITEMS	Set a value. Make sure that you do not clear any existing bits. You should use the logical OR and AND operator (BITOR or BITAND for the Oracle Client)
The `sql_type`, `sql_length` and `sql_scale` columns in the DATAITEMS table	Force the `define` command to remap values.
The `dms_concat_num` column in the DMS_ITEMS table	Set the value of the `dms_item_number` you want concatenated to this item.
The `table_name` and `index_name` columns in the DATATABLES table	Rename.
The `item_name` column in the DATAITEMS table	Rename.
The `item_number` column in the DATAITEMS table	Reorder columns.

> ✏️ **Note**
>
> BITOR and BITAND are functions needed to set and clear bits in user scripts used by the Oracle Client in the various xx_options columns of the Client control tables. When you run a `define` or `redefine` command, the Client creates the BITOR function, while the BITAND function is part of SQL language of Oracle. BITAND (a,b) returns the bitwise AND of a and b while BITOR (a,b) returns the bitwise OR of a and b. This means that you can use the BITOR function as if it was part of the Oracle SQL functions.
>
> The following example shows BITOR setting a bit:
>
> ```
> update DATASETS set ds_options=BITOR(ds_options, 4096) where dataset_name='CUSTOMER
> ```
>
> SQL Server Transact-SQL uses & and | to perform these functions. In the case of SQL Server the above example would look like:
>
> ```
> update DATASETS set ds_options=ds_options | 4096 where dataset_name='CUSTOMER'
> ```
>
> All scripts generated by the Oracle Client using the `createscripts` command use the BITOR function for settings bits and the BITAND function for clearing bits.

3.1.4 Primary and Secondary Data Tables

The Databridge data tables hold the cloned DMSII data. You will see two types of Databridge data tables:

- Primary data tables, which are the relational database equivalent of the DMSII data sets. Primary table names are derived from the corresponding DMSII data set name by converting it to lowercase and replacing hyphens (-) with underscores (_).
- Secondary data tables, which are additional tables that need to be generated to represent a DMSII structure that does not have a relational database equivalent (for example, items with OCCURS clauses that are not flattened). Secondary table names are constructed using the primary table name with an appropriate suffix.

3.2 Defining a Data Source

A data source is the DMSII database or FileXtract file that you want the Client to replicate. The DBServer control file (on the host) identifies each data source by name in the section that uses the key word SOURCE. A SOURCE has a FORMAT, FILTER and SUPPORT specification.

If you use Enterprise Server, each data source will be associated with a SOURCE in the Enterprise Server configuration file. This SOURCE is based on a base data source that matches a SOURCE in DBServer. If you use the base source without any additional filtering applied by Enterprise Server, the DBServer and Enterprise Server sources are identical and completely interchangeable.

Each data source has an entry in the DATASOURCES Client control table. The `hostname` column identifies the Databridge server by the domain name or IP address. The `hostport` column identifies the port on which the server listens for incoming connections. You can switch the server from DBServer to Enterprise Server simply by changing the values of these two columns.

> ✏ **Note**
>
> You may use Databridge FileXtract sources for Client operations. These sources are made to look like data sources for DMSII databases.

3.2.1 Using the Define Command

Follow these steps to define a data source and populate the Client control tables. You can also perform this action from the Administrative Console by navigating to the data sources page for the Client Manager in question and clicking **Actions > Define/ Redefine**.

To define a data source

> ✏️ **Note**
>
> This procedure assumes that Databridge Server is running and the signon parameters are configured appropriately.

1. Because the following `dbridge.cfg` parameters are difficult to change later without redefining and re-cloning, make sure that they're set appropriately before you run the `define` command:

```
allow_nulls
automate_virtuals
auto_mask_columns (SQL Server only)
bracket_tabnames (SQL Server only)
clr_dup_extr_recs
convert_ctrl_char
default_user_columns
dflt_history_columns
enable_dms_links
enable_dynamic_hist
enable_extended_types
external_column
extract_embedded
flatten_all_occurs
force_aa_only
history_tables
inhibit_required_opt
maximum_columns
min_varchar
minimize_col_updates
miser_database
optimize_updates
read_null_records
reorg_batch_size
sec_tab_column_mask
split_varfmt_dataset
strip_ds_prefixes
suppress_dup_warnings
suppress_new_columns
suppress_new_datasets
use_bigint (SQL Server only)
use_binary_aa
use_clob (Oracle only)
use_clustered_index (SQL Server only)
use_column_prefixes
use_date (SQL Server only)
use_datetime2 (SQL Server only)
use_decimal_aa
use_internal_clone
use_nullable_dates (Miser databases only)
use_primary_key
use_stored_procs
use_time (SQL Server only)
use_varchar
```

For information on setting these parameters, see Appendix C: Client Configuration.

2. Enter the following command:

```
dbutility [signon_opts misc_opts] define datasource hostname portnum
```

> ✏️ **Note**
>
> If you previously used user scripts to update control tables and want to switch to the **Customize** command, you'll need to update your Client control tables first. For instructions, see the *Next Steps* section in the *Databridge Installation Guide*.

Where	Is
signon_opts	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** `[-U userid] [-P password][-D database]` **SQL Server:** `[-U userid] [-P password] [-W] [-O ODBCdatasource]`
misc_opts	Any of the following miscellaneous command-line options: `-L` forces the Client to use a new log file for this run. `-T` forces the Client to use a new trace file for this run, if tracing is enabled. `-f` filename to specify a configuration file other than the default dbridge.cfg file in the working directory. `-u` allows the command to delete Client control table entries for a data source that already exists.
datasource	**For DBServer:** The name that matches the entry for SOURCE in the DBServer control file. You can enter the data source name in uppercase or lowercase. **For DBEnterprise:** The name of a source (base or filtered) defined in Enterprise Server.
hostname	The domain name or IP address of the Databridge server.
portnum	The TCP/IP port number on which the appropriate Databridge server listens for incoming calls.

For DMSII databases that have a large number of data sets and data items, the process of retrieving the layout information may take several minutes.

3. Read the following section, Results of the Define Command, and then specify which data sets and data items you do not want to be cloned or updated, as explained in Customizing with User Scripts.

Example

Assuming the DBServer control file contains SOURCE ORDDB and PORT=5001 on the host "OURHOST.CIN.AAA.COM", you would enter the following:

```
dbutility define ORDDB OURHOST.CIN.AAA.COM 5001
```

The Databridge Client makes remote procedure calls to DBServer to get DMSII database layout information. DBServer returns the DMSII layout information to the Client. The Client populates the control tables with the DMSII layout information and creates the corresponding relational database table layout.

The empty control tables (that were built during the **dbutility** `configure` command) are now populated.

For example, this SQL statement

```
select data_source, hostname, hostport from DATASOURCES
```

would yield a table similar to the following. Only the selected columns are shown.

```
data_source    hostname              hostport
-----------    --------------------  ------
ORDDB          OURHOST.CIN.AAA.COM   5001
```

3.2.2 Results of the Define Command

The `define` command automatically does the following with table names and column names:

- (Typically) converts data set, data item, and set names to lowercase and changes dashes to underscores for their equivalent relational database table, column, and index names. For more details on how this actually occurs, see Relational Database Table and Column Names.

- Constructs secondary table names by appending an underscore followed by the lowercase data item name (for which the table is constructed) to the primary table name. For example, if a DMSII data item named SALES, which has an OCCURS clause, appears in a data set named CUSTOMERS, the relational database table generated for the OCCURS item is named customers_sales. For more details, see Handling DMSII OCCURS.

- Appends the suffix _x to all object names that are relational database reserved words. For example, if a DMSII data set is named ORDER, which is a relational database reserved word, the table generated for the ORDER data set is named order_x. Likewise, for a DMSII data item named COUNT, which is also a relational database reserved word, the corresponding column would be named count_x.

- Adds two-character prefixes to table names (i_, u_, and d_) when constructing the names of the stored procedures it uses to insert, update, and delete records from these tables. The result is that table names are limited to 28 characters, even though some relational databases limit table and index names to 30 characters.

- Checks table and index names to see if they duplicate existing table and index names that Databridge previously created. Databridge recognizes only those relational database objects that it has created. When the Databridge Client finds a duplicate name, it makes the name unique in one of the following ways:

  - Appending a numeric suffix. For a data set named ITEM that must be split into three tables, the resulting table names would be as follows: item, item1, item2.

  - If the name is too long to add a suffix, overwriting as many of the last characters as necessary with numeric characters to make the name unique.

3.2.3 Cloning from Multiple Data Sources

If you are cloning multiple data sources to the same relational database and you have duplicate data set names, Databridge modifies the table name for those duplicates to avoid creating multiple tables with the same name.

For example, if you have two data sources (DSA and DSB), both of which include a data set named PRODUCTS, Databridge clones the data set from DSA into a table named "products". When Databridge clones DSB, it clones DSB's data set PRODUCTS into a table named "products1".

> 🔥 **Important**
>
> To avoid potential errors, rename any tables that have duplicate names. For example, rename the "products" table to "products_a" for data source DSA and to "products_b" for data source DSB. You can rename tables during the relational database customization phase of the `define` command using the `script.user_define.` `primary_tablename`. For a sample script, see Renaming a Table.

The Databridge Client renames duplicate table names across data sources as a precaution against accidentally removing a table that contains good data. If you do not drop either of the data sources, rerunning the `define` command for either data source does not cause any problems.

For example, if you execute another `define` command for DSA because DMSII database A was reorganized, the `define` command looks for the table name "products" in the DATATABLES Client control table that belongs to data sources other than DSA. Because the name "products" belongs to DSA only, the `define` command does not find "products" as a table name under any other data source. Thus the table corresponding to the data set PRODUCTS will be named "products", as was the case earlier.

Similarly, if you execute a `define` command for DSB, the `define` command looks for the name "products" in the DATATABLES Client control table that belongs to data sources other than DSB. Because the name "products" belongs to DSA, the `define` command will find "products" as a table name used by another data source and it will resolve the conflict by renaming the table. Thus the table corresponding to the data set PRODUCTS will be named "products1" as was the case before the `define` command was run.

If you drop either of the data sources, however, the results may be different because the table name is no longer a duplicate. For example, if you drop DSA and then execute a `define` command for data source DSB, the table will be named "products", not "products1", because it is no longer a duplicate.

Similarly, if you do a `dropall` command and then execute a `define` command for data source DSB first, the tables will be named "products" for data source DSB and "products1" for data source DSA.

Add a Prefix to Duplicate Data Set Names

If you replicate two or more databases, which have many data set names in common, you can make the program add a prefix to all the table names for a data source. You must define the prefixes, which can be 1--8 characters long, before you create the relational database layout. To do this, assign a value, such as X1, to the `tab_name_prefix` column of the corresponding entry in the DATASOURCES Client control table using the script `script.user_datasets.datasource`. Using different prefixes for each data source makes the table names unique and eliminates the need to rename tables.

If you are using multiple data sources that have data sets or indexes that have the same name, we strongly recommend that you write user scripts to resolve this issue by forcing such a table to use a different name for one (or more if the name occurs in more than two data sources). This will ensure that you have a consistent naming convention. Without this, you could run into problems if you reorganize these data sets.

Example script

```
script.user_define.customer:
update DATATABLES set table_name='customer_demodb'
where data_source='DEMODB' and dataset_name='CUSTOMER'
/***/
update DATAITEMS set table_name='customer_demodb'
where data_source='DEMODB' and table_name='customer'
```

This example script forces the table 'customer' in data source DEMODB to always be renamed. If another data source also has a data set named CUSTOMER, it will then be able to always use the name 'customer' for the corresponding table. It also makes sure that all the items in the renamed table point to the renamed table. The line `/***/`, which separates the two SQL statements in the script, tells the Client to execute the first SQL statement before moving on to the second one.

3.3 Customizing with User Scripts

User scripts are files that contain SQL statements for modifying the Client control tables. They provide a convenient way of automating the customization changes that are applied to the control tables. The Databridge Client looks for user scripts in the directory specified by the configuration file parameter `user_script_dir`. If you do not set the `user_script_dir` parameter in the configuration file, the Databridge Client uses the **scripts** directory. It automatically executes user scripts when certain commands are run, provided they exist.

The main purpose of user scripts is to preserve changes to the control tables by having the program run these scripts to restore the changes whenever necessary. To view sample data set layout and data table customization scripts, see Appendix D: Customization Scripts.

> ✏ **Note**
>
> You can customize the Client control tables easily by using the **Customize** command instead of writing user scripts.
>
> Customizing a DMS item is very simple. Click on the data set name in the data sets view of the **Customize** command to open the DMS item view. Then click the wrench to the left of the DMS item that you want to customize. This open up the properties page for the item where you can then click on the appropriate radio button option (e.g. "Replicate as date"). You may also need to change additional properties of the item when the `dms_subtype` needs to be set.
>
> You can find a complete description of the additional requirements for user scripts that are compatible with the **Customize** command in Appendix D: Customization Scripts. For information about using the **Customize** command to customize your data source, see the Databridge Administrative Console Help.

3.3.1 Types of User Scripts

The Databridge Client supports the following types of user scripts:

Script/Filename	Description
Session initialization script	`script.user.session` This script allows you to change session parameters without changing the database settings. Use this script to alter session parameters whose default values are not suitable for Client operations. For example, when the NLS_LANGUAGE for Oracle is a European language, ALTER the NLS_LANGUAGE parameter to set the language to AMERICAN and the NLS_TERRITORY to AMERICA. The Databridge Client executes these scripts when running any command that connects to the relational database. **NOTE:** The Oracle Client will automatically execute the SQL to ALTER the SESSION when the language is not AMERICAN or when the character set of the database is UTF8. However, it will only do this if there is no session script present in the user scripts directory. This allows you to override the actions taken by the Client by providing a session script to use instead.
Data set global mapping customization script	`script.user_datasets.datasource` where datasource is the name of the data source (in lowercase) as defined in the DATASOURCES Client control table. Use this script to disable mapping for unwanted data sets or to enable mapping for data sets that are not mapped by default. Create only one of these scripts for each data source. The Client processes the user script `script.user_datasets.datasource` before the DMSII mapping phase of both the `define` and `redefine` commands. This script can contain global script commands that allow you to make changes to multiple columns in the DATASETS and DMS_ITEMS layouts with a single SQL statement. For instance, if a DMSII database has a time value item called TS in almost every data set, you can use a single SQL statement to update the dms_subtype value for every occurrence of TS. For an example script, see Sample Data Set Global Mapping Customization Script.
Data set mapping customization script	`script.user_layout.primary_tablename` where datasource is the name of the data source (in lowercase) as defined in the DATASOURCES Client control table. This script is run after the relational database layout has been created during a `define` or `redefine` command. It allows you to make global changes to DATATABLES and DATAITEMS. You can use this script to insert common scripts into a single file rather than having to duplicate the SQL in each of the define scripts for the individual data sets. For an example, see Sample Data Table Global Customization Script.

Script/Filename	Description
Data table customization script	`script.user_define.primary_tablename` where *primary_tablename* is the name of the primary table mapped from the data set. These scripts make changes to the DATATABLES and DATAITEMS tables for changing table or column names, changing SQL data types, and so on. Create one of these scripts for each data set that has one or more tables which need to be customized. When you change the name of the table within the script, you must use the original primary table name in the script filename. All changes related to tables mapped from a data set are contained in the data table customization script for the primary table specified by *tablename*. The Databridge Client runs these scripts after the relational database layout has been created by the `define` and `redefine` commands.

Data table creation user script	`script.user_create.tablename`
	
	where *tablename* is the name of the relational database table. Use this script for the following:
	
	  • To define default values for non DMSII columns
	
	  • To alter the table and add a column that the Databridge Client does not need to be aware of, add a "ALTER TABLE xxx ADD COLUMN yyy" SQL statement to these scripts instead of adding SQL statements to the table creation scripts.
	
	CAUTION: Do not use this script to create columns for specific types of data generated by the Client. This script creates a type of user column that the Client is unaware of. To create user columns that the Client is aware of, see Adding a Non DMSII Column.

Index creation user script	`script.user_index.tablename` where *tablename* is the name of the relational database table. Use this script to add SQL statements to the index creation scripts (`script.index.`*tablename*) created by the **dbutility** `generate` command. Do not modify the scripts created by the generate command, as your changes will be lost the next time a generate command is run. These scripts are executed immediately after the related Databridge Client script named `script.index.`*tablename* during the `process` or `clone` command. If you set the `check_user_scripts` parameter, the Databridge Client returns a warning if it cannot find the script.
Data table cleanup user script	`script.user_cleanup.tablename` where *tablename* is the name of the relational database table Use these scripts to undo any actions, such as creating a secondary index, that are done in the `script.user_index.table` user script. These scripts are run during the `process` or `clone` command, prior to executing the cleanup scripts created by the `generate` command. The scripts are only used in cases where the relational database tables are not dropped when a data set is re-cloned, such as when deleted records are to be preserved.

Script/Filename	Description
Stored procedure creation user script	`script.user_create_sp.` *tablename* where *tablename* is the name of the relational database table. This new type of script allows the user to split updates to stored procedures from other actions taken when a table is created by the Client. Unlike the user scripts `script.user_create.` *tablename*, this script is also run when the table is refreshed during a `refresh` or `reorg` command. This allows the user to alter the stored procedures without requiring any manual intervention.

3.3.2 User Script Syntax

Use the syntax you would typically use for SQL statements; however, separate each statement with the following separator:

`/***/`

In addition, be aware of the following:

- You must begin the separator line with the characters `/***/` and no leading spaces . Trailing blanks or carriage returns are ignored.
- Do not end the script with the `/***/` separator.
- Do not use a semicolon or GO as you would if you were using a relational database query tool.
- You can add comments to the end of any line (including a blank line) by using " `//` " to start a comment. This causes the Client to ignore the rest of the line, including these two characters. If you add a comment to a separator line, the separator must be followed by at least one space.

3.3.3 Writing and Testing User Scripts

Following is a recommended method for creating user scripts. Typically, you would start writing your user scripts after you have run `configure` and `define` for the first time. This procedure does *not* cover the data table creation user script or the index creation user script.

> ⚠️ **Caution**
>
> If you have already used the Databridge Client to clone a database, we highly recommend that you test your scripts using a *test version* of the Client control tables, *not* your production version of the Client control tables.

Follow these guidelines as you develop your user scripts:

- Store your user scripts in the directory pointed to by the `user_script_dir` parameter of the Client configuration file (by default, the `scripts` subdirectory of the data source's working directory). Storing them in the global working directory ensures that they are protected by file security, if enabled.
- Use the `runscript` command to test each script. This command executes the scripts as a transaction. If an error occurs in a script, the Databridge Client rolls back all changes. You then have the opportunity to fix the error and rerun the script.
- If you make a mistake and change the Client control tables in a way you did not intend to, remove or rename the offending script and then run **dbutility** `define` again. This creates a fresh set of Client control tables.

To write and test user scripts

1. Do one of the following:

    • If you are already using Client control tables in production, run `configure` to create a test version of the Client control tables or `unload` to create a backup copy of the tables.

    • If you haven't created Client control tables yet, run `configure` .

2. Run `define` to populate the Client control tables.

3. Run `display` to create a report of your Client control tables. This report gives you a record of table names, column names, and so on, that you can use as a reference as you write your user scripts.

4. Create your data set mapping customization scripts, as follows:

    • Create the data set selection script for selecting/deselecting data sets. See Sample Data Set Selection Script.

    • Create a data set mapping customization script for each data set that requires that its mapping be customized. These user scripts can contain several SQL statements that perform different types of mapping customizations (for example, flatten OCCURS clauses, specify that items should be cloned as dates, and disable the cloning of some DMSII items). See Tips for More Efficient Cloning.

    • Test each script as follows:

    dbutility [`-n`] **runscript** *scriptfilename*

    where *scriptfilename* is the name of the script you're testing and `-n` is a command line option that overrides your entry for `user_script_dir` by allowing you to specify a complete path for the script.

    > ✏️ **Note**
    >
    > The `runscript` command runs the script in transaction mode. If an error occurs during script execution, the Databridge Client rolls back all changes. This allows you to safely rerun the script after correcting it.

5. Fix any errors uncovered by running the scripts, and rerun the script until it is correct.

    If the script gets corrupted beyond repair, rerun the `define` command as described in step 2. You must add the `-u` command line option to force the program to allow you to rerun the `define` command.

6. When you are satisfied with the script, repeat the `define` command.

    You can also set bit 8 of the `status_bits` column of the DATASETS Client control table to inform **dbutility** that the data set needs to be redefined. To set this value, run the following within a relational database query tool:

    ```
    update DATASETS set status_bits = 8
    where dataset_name = 'DSNAME' and data_source = 'SOURCE'
    ```

    Then execute a `define` command to refresh the mapping.

7. Repeat step 3 at this point to view the effect of your data set mapping customization.

8. Create a data table customization script for each data set whose tables need to be customized.

    These user scripts can contain several SQL statements that perform different types of customizations for any of the tables mapped from the data set (for example, renaming a table, renaming a column, changing the `sql type` column of a data item, inserting a non DMSII item into a tables). See Sample Data Table Customization Scripts.

9. Test each script as described in step 6.

    > ⚠️ **Caution**
    >
    > Include all changes that affect the tables derived from a data set in that data set's script. For example, after a reorganization, the Databridge Client runs your data table customization user scripts after the relational database layout has been created by a `define` command. If some scripts are missing, or if a data table customization script does not include all the changes for its tables, the Databridge Client creates tables that have different layouts than the original ones.

10. Fix any errors uncovered by running the scripts, and rerun the script until it is correct.

    If the script gets corrupted, rerun the `define` command as described in step 2. You must add the `-u` command line option to force the program to allow you to rerun the `define` command.

11. Run **dbutility** `define` again, using the `-u` option. If you don't use the `-u` option, the `define` command will tell you the data source already exists. Enter the following:

```
dbutility -t0x801 -u datasource hostname portnumber
```

The Databridge Client automatically runs your user scripts and updates the Client control tables accordingly. The -t 0x801 option produces a trace of all SQL commands that execute as part of user scripts. These are followed by row counts for update or insert statements. If you do not enable tracing, you will only see the row counts in the log file.

The next phase of the `define` command executes the mapping of the DMSII data sets to relational database tables for data sets whose `active` column is set to 1. Finally, the Databridge Client runs the data table customization scripts for all the data sets whose `active` column is set to 1. The `-t 0x801` options also produce a trace of all SQL commands in these scripts.

The Databridge Client runs the data set selection scripts and all the data set mapping customization scripts as well as the data table customization scripts in a single transaction group. If there is an error, the Databridge Client does not commit any of the changes; instead, it rolls back all changes and the command terminates.

> ✏️ **Note**
>
> If you created table creation or index creation user scripts, the Databridge Client runs those immediately after running its own table creation or index creation scripts.

12. If you decide to clone a data set or data item that you did not previously clone or if a DMSII reorganization occurs, you will need to update your scripts.

3.3.4 Using Scripts to Disable Data Sets

To disable cloning by writing user scripts, do the following:

- Disable data set cloning via `script.user_datasets.datasource`
- Disable DMSII item cloning via `script.user_layout.primary_tablename`
- Once you are familiar with the concepts in this section, see Customizing with User Scripts.

When using the Administrative Console or **Customize** command you can simply uncheck the checkbox for the `active` column of the data sets you want to disable. The Client will remember the changes unless you drop the data source and start from scratch.

3.4 Decoding DMSII Dates, Times, and Date/Times

This section explains the following:

- How to decode DMSII dates, times, and date/time formats into appropriate relational database types by modifying the DMS_ITEMS Client control table via the `script.user_layout.primary_tablename` user script
- How to change the SQL data type of the resulting relational database column

> ✏️ **Note**
>
> You can make the same types of customizations to the Client control tables using the **Customize** command as you can by writing user scripts. You can find a complete description of the additional requirements for user scripts that are compatible with the Client Customizer in Appendix D: Customization Scripts. For information about the **Customize** command, see the Databridge Administrative Console Help.

After you are familiar with the concepts in this section, see Appendix D: Customization Scripts.

3.4.1 DMSII Dates

Even though DMSII did not have a date data type until the advent of DMSII 57.1, most DMSII sites use several common methods to store dates. This section includes ways to decode these types of date representations into a relational database date data type. A DMSII 57.1 date is stored as a REAL in DMSII and represents the number of days since 12/31/1600. The Client automatically converts DMSII 57.1 dates to relational database dates making it unnecessary to do any customization

The Databridge Client supports the following DMSII date encoding methods:

Script/ Filename	Description	See
DMSII GROUPS	Three numbers for year, month, and day	DMSII Dates Represented as a GROUP of Numbers- - approach #1 and DMSII Dates Represented as a GROUP of Numbers - approach #2
DMSII NUMBER values	Any of the following: • MISER database dates, usually NUMBER(5) • LINC database dates • Month, day, and year represented by a 6- or 8-character alpha string containing only digits • Delimited dates such as (03/10/10) • Dates with three-character month names (MAR102005) • Julian dates represented as a five-digit number (06905) or seven-digit number (0692005)	Decoding DMSII Dates Represented as ALPHA or NUMBER
DMSII ALPHA values	Any of the following: • LINC database dates • LINC database dates • Month, day, and year represented by a 6- or 8-character alpha string containing only digits • Delimited dates such as (03/10/10) • Dates with three-character month names (MAR102005)	
DMSII Times		DMSII Times Represented as ALPHA, NUMBER, or REAL
Custom DMSII dates	Any of the following: • Month/year without day or other unique variations • Non-Standard dates • Month/year without day or other unique variations	Custom DMSII Date/Time Represented as ALPHA or NUMBER

Choosing the SQL Data Type of the Relational Database Column {#b1jbb5sf}

Regardless of the original DMSII date structure, the resulting relational database column has a default `sql_type` of 12 (**smalldatetime**) in the case of SQL Server and a `sql_type` of 10 (**date**) in the case of Oracle.

To make the Client map a DMS item to a column that is a date data type, you must set the bit DIOPT_Clone_as_Date (2) in the `di_options` column of the corresponding DMS_ITEMS entry using the user script `script.user_layout.dataset`. Setting the configuration parameter `use_date` to true makes the Client use the `date` data type for all dates that have no time part, regardless of whether this bit is set or not.

SQL Server supports multiple date data types. You can make the Client generate different types of dates by using the `script.user_layout.dataset` user script to set the following bits in the `di_options` column of the corresponding DMS_ITEMS table entry:

- DIOPT_UseLongDate (128) causes the Client to a use a data type of 10 (**datetime**) instead of **smalldatetime**.

- DIOPT_UseLongDate2 (65536) causes the Client to use the **datetime2** data type. If both this bit and the DIOPT_UseLongDate bit are set, **datetime2** is used. Setting the configuration parameter `use_datetime2` to true makes the Client use the `datatime2` data type regardless of whether this bit is set or not when the DIOPT_UseLongDate bit is set.

- DIOPT_Clone_as_DateOnly (32768) causes the Client to use the **date** data type which is 3-bytes long and contains no time.

Relational Database Date Data Type	Value for sql_type Column
Microsoft SQL Server: **datetime** (8 bytes)	10
Microsoft SQL Server: **smalldatetime** (4 bytes)	12
Oracle: **date** (7 bytes)	10
Microsoft SQL Server: **int** Oracle: **number(10)** NOTE: The date is formatted according to the numeric_date_format configuration parameter, whose default value is 23 (mmddyyyy).	13
Microsoft SQL Server: **datetime2** (8 bytes)	19
Microsoft SQL Server: **date** (3 bytes)	20

For an example script, see Changing SQL Data Types.

DMSII DATES REPRESENTED AS A GROUP OF NUMBERS- - APPROACH #1

The DMSII GROUP must always contain a year and a month; the day can be omitted, in which case it defaults to 1.

To clone a DMSII date (represented as a group of numbers) as a relational database date

Write a user script (`script.user_layout`.*primary_tablename*) that does the following:

1. Sets the DIOPT_Clone_as_Date (2) bit in the `di_options` column for the GROUP

2. Sets the `dms_subtype` column of the group members in DMS_ITEMS to indicate which part of the date they represent, as follows:

Part of Date in GROUP	Value for dms_subtype Column
Year (assumes a 1900 base)	1
Month	2
Day	3
Year By default, yy values < 50 are 21st century years (20yy) and yy values > 50 are 20th century years (19yy).*	4
Absolute year	5
This is a 4-digit year specification (for example, 2010).	

The following SQL statements cause the Databridge Client to clone the DMSII group INV_DATE as a relational database date type.

Filename: `script.user_layout.inv`

update DMS_ITEMS set di_options=2
where dataset_name='INV' and dms_item_name='INV_DATE'
/***/
update DMS_ITEMS set dms_subtype=1
where dataset_name='INV' and dms_item_name='INV_DATE_YEAR'
/***/
update DMS_ITEMS set dms_subtype=2
where dataset_name='INV' and dms_item_name='INV_DATE_MONTH'
/***/
update DMS_ITEMS set dms_subtype=3
where dataset_name='INV' and dms_item_name='INV_DATE_DAY'

The **Customize** command does not support this method of handling GROUP dates. However, it does support the equivalent method described in the next section. When converting old scripts to a format compatible with the **Customize** command, the `dbscriptfixup` utility converts the changes made by this type of script to the format described below.

DMSII DATES REPRESENTED AS A GROUP OF NUMBERS - APPROACH #2

This version of Databridge Client now supports a new method of handling DMSII dates represented as a GROUP (or a nested GROUP). The Client redefines a group of like items, that can either be unsigned numbers or alpha items, as a single item having the common type and encompassing the entire GROUP. This operation is referred to as collapsing (or redefining) a GROUP). By collapsing a GROUP of numbers that represent a date, we effectively make the operation of cloning it as a relational database date equivalent to that of cloning a number that represents a date.

For example, this technique can collapse the year, month, and day in the following DMSII GROUP in the data set named EMPLOYEE into a single item that acts as a NUMBER(8) :

```
EMP-HIRE-DATE-YMD GROUP
(
    EMP-HIRE-YEAR NUMBER(4);
    EMP-HIRE-MONTH NUMBER(2);
    EMP-HIRE_DAY NUMBER(2);
)
```

The method described in the next section can then customize this column as needed. This technique also applies to date/time quantities represented as a group of like items.

To clone a DMSII date (represented as a group of numbers) as a relational database date

1. Sets the DIOPT_CollapseGroup (67,108,864) and the DIOPT_Clone_as_Date (2) bits in the `di_options` column.

2. Sets the `dms_subtype` column of the GROUP item in DMS_ITEMS to indicate the format in which the resulting date is encoded. See the section below for a list of date formats (the above date group is represented by a `dms_subtype` of 21).

The script to perform this action is:

Filename: `script.user_layout.employee`

update DMS_ITEMS set di_options=67108866, dms_subtype=21
where dataset_name='EMPLOYEE' and dms_item_name='EMP-HIRE-DATE-YMD'

Decoding DMSII Dates Represented as ALPHA or NUMBER

Use the following procedure to decode DMSII dates represented as NUMBER or ALPHA items to relational database data types.

To decode dates represented as NUMBER or ALPHA items

1. Write a script (`script.user_layout.` *primary_tablename*) that does the following:

2. Sets the DIOPT_Clone_as_Date (2) bit in `di_options` .

3. Sets the `dms_subtype` column in DMS_ITEMS to indicate the type of date encoding method used on the host, as follows:

Date Encoding Scheme	Value for dms_subtype Column
NUMBER(n) for MISER dates—days since 12/31/1899	1
NUMBER(n) for LINC dates—days since 1/1/*baseyear* (default 1957)	3
ALPHA(6) or NUMBER(6) with two-digit year yy (1900–1999)	
yymmdd	11
yyddmm	12
mmddyy	13
mmyydd	14
ddmmyy	15
ddyymm	16
ALPHA(5) or NUMBER(5) with two-digit year yy (1900–1999) and with days DDD where DDD is a number between 1–366 for Julian dates	
DDDyy	17
yyDDD	18
ALPHA(8) or NUMBER(8) with four-digit year yyyy	
yyyymmdd	
yyyyddmm	21
mmddyyyy	22
mmyyyydd	23
ddmmyyyy	24
ddyyyymm	25
	26
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	
DDDyyyy	27
yyyyDDD	28
ALPHA(6) or NUMBER(6) with two-digit year yy (1950–2049) where yy values < 50 are 21st century years (20*yy*) and yy values > 50 are 20th century years (19*yy*)	
	31
yymmdd_2000	32
yyddmm_2000	33
mmddyy_2000	34
mmyydd_2000	35
ddmmyy_2000	36
ddyymm_2000	
ALPHA(5) or NUMBER(5) with two-digit year yy (1950–2049) where yy values < 50 are 21st century years (20yy) and yy values > 50 are 20th century years (19yy) and with days DDD where DDD is a number between 1–366 for Julian dates.	
DDDyy_2000 yy*	37
DDD_2000	38

Date Encoding Scheme	Value for dms_subtype Column
ALPHA(8) with two-digit year yy (1900–1999) and with delimiter characters where / represents forward slash (/), hyphen (-), or period (.).	
yy/mm/dd	41
yy/dd/mm	42
mm/dd/yy	43
mm/yy/dd	44
dd/mm/yy	45
dd/yy/mm	46
ALPHA(10) with four-digit year yyyy and with delimiter characters where / represents forward slash (/), hyphen (-), or period (.).	
yyyy/mm/dd	51
yyyy/dd/mm	52
mm/dd/yyyy	53
mm/yyyy/dd	54
dd/mm/yyyy	55
dd/yyyy/mm	56
ALPHA(8) with two-digit year yy (1950–2049) where yy values < 50 are 21st century years (20yy) and yy values > 50 are 20th century years (19yy) and with delimiter characters where / represents forward slash (/), hyphen (-), or period (.).*	
yy/mm/dd_2000	61
yy/dd/mm_2000	62
mm/dd/yy_2000	63
mm/yy/dd_2000	64
dd/mm/yy_2000	65
dd/yy/mm_2000	66
ALPHA(7) with two-digit year yy (1900–1999) and three-character month abbreviation (mon). Month abbreviations are JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC unless specified otherwise by the months parameter in the Databridge Client configuration file.	
yymondd	
yyddmon	71
monddyy	72
monyydd	73
ddmmyy	74
ddyymon	75
	76
ALPHA(9) with four-digit year (yyyy) and three-character month abbreviation (mon). Month abbreviations are JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC unless specified otherwise by the months parameter in the Databridge Client configuration file.	
yyyymondd	81
yyyyddmon	82
monddyyyy	83
monyyyydd	84
ddmonyyyy	85
ddyyyymon	86

Date Encoding Scheme	Value for dms_subtype Column
ALPHA(7) with two-digit year yy (1950–2049) where yy values < 50 are 21st century years (20yy) and yy values > 50 are 20th century years (19yy) and with three-character month abbreviations (mon). Month abbreviations are JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, and DEC unless specified otherwise by the months parameter in the Databridge Client configuration file.	
yymondd_2000	
yyddmon_2000	91
monddyy_2000	92
monyydd_2000	93
ddmonyy_2000	94
ddyymon_2000	95
	96

The configuration parameter `century_break` allows you to adjust the range for the year. The default value for `century_break` is 50. A value of -1 causes the Client to automatically set the century break based on the year in the audit timestamp.

For example scripts, see Cloning a Numeric Field as a Date and Cloning an Alpha Field as a Date.

> ✏️ **Note**
>
> If your DMSII date format includes *mmyy* or *yymm* without a position for days, see Custom DMSII Date/Time Represented as ALPHA or NUMBER.

3.4.2 DMSII Times

The Databridge Client supports several DMSII ALPHA, NUMBER, or TIME encoding methods for time of day and elapsed time.

Choosing the SQL Data Type of the Relational Database Column

The relational database column---regardless of the original DMSII time structure---has a default `sql_type` of 17, which is a Microsoft SQL Server **int** or Oracle **number(6)**, except for TIME(12) and TIME(14), which are stored as a number (10). TIME(12) and TIME(14) are formatted as *ddddhhmnss*, where *dddd* is the number of days.

All other TIME types are formatted as *hhmnss*. To make the Client map a DMS item to a column that is a numeric time, you need to set the bit DIOPT_Clone_as_Time (256) in the `di_options` column of the corresponding DMS_ITEMS entry using the user script `script.user_layout.dataset`.

In the case of SQL Server, which has a **time** data type, the Client can store these values using the **time** data type. You can do this by setting the `di_options` bit DIOPT_Use_Time (131072) in the corresponding entry in the DMSII_ITEMS table using the `script.user_layout.dataset` user script. If you set both the DIOPT_Clone_as_Time bit and the DIOPT_Use_Time bit, the latter takes precedence.

DMSII Times Represented as ALPHA, NUMBER, or REAL

You can decode DMSII times represented as ALPHA, NUMBER, or REAL items to relational database data types using the Databridge host or the Databridge Client. To do this on the host (versus the Databridge Client), you must redefine the DMSII item using an ALTER REDEFINE. For more information, see Chapter 5 of the Databridge Programmer's Reference.

To decode those data types using the Databridge Client

Write a script (`script.user_layout` .*primary_tablename*) that does the following:

- Sets the DIOPT_Clone_as_Time (256) bit in `di_options` .

- Sets the `dms_subtype` column in DMS_ITEMS to indicate the type of time encoding method used on the host, as follows:

Time Encoding Scheme	Value for dms_subtype Column
ALPHA(6) or NUMBER(6) time of day in *hhmnss* format	1
REAL containing a TIME(1) value, which represents the time of day in 1/60th of a second	2
REAL containing a TIME(11) value, which represents the time of day in ticks (2.4 microseconds)	3
REAL containing a TIME(12) or TIME(14) value, which represents the elapsed time in ticks	4
REAL containing a DMSII 57.1 TIME, which represents the number of 100'th of seconds since midnight. These are automatically converted to TIME data types if the database supports it. Otherwise, they are stored as integer values of the form "hhmmss".	5
NUMBER(12) containing the time of day in hhmnssmmmmmm format where mmmmmm represents fractions of seconds.	6

For an example script, see Cloning an Alpha or Number Field as a Time.

3.4.3 Decoding DMSII Date/Times

The Databridge Client implements a set of `dms_subtype` values to decode DMSII items that include the date and time in a single item. Specifically, the Databridge Client contains values for DMSII ALPHA or NUMBER values that represent the date/time in a variety of ways, such as:

- Month, day, year, and time of day combined into a twelve-digit (031005112501) or fourteen-digit (03102005112501) number

- Julian dates and time of day represented as an eleven-digit number (06905112501) or a thirteen-digit number (0692005112501)

- DMSII item of type REAL that are 48-bits long and represent TIME(6), TIME(7), or TIME(60) type data which encode a date and a time. A new data type in DMSII 57.1, named TIMESTAMP, represents TIME(6) values. The Databridge Client automatically converts these items to the appropriate relational database date/time data type, thus eliminating the need to do any special customization.

To decode these types of date/time representations into a relational database date/time data type, see Decoding DMSII Date/Time Represented as ALPHA or NUMBER. When using these with SQL Server, you should set the `di_options` bit DIOPT_Use_LongDate to force the Client to use a data type of **datetime** rather than **smalldatetime**. When a data type of **smalldatetime** is used, the Client sets the values of seconds to zero (0), as SQL Server rounds the value to increments of .000, .003, or .007 seconds. You can aslo use a data type of **datetime2** instead of **datetime** by setting the `di_options` bitDIOPT_UseLongDate2 (65536). Setting the configuration parameter `use_datetime2` to true makes the Client use the `datatime2` data type regardless of whether this bit is set or not when the DIOPT_UseLongDate bit is set.

Decoding DMSII Date/Time Represented as ALPHA or NUMBER

You can decode DMSII date/time formats represented as NUMBER or ALPHA items to relational database date/time data types using the Databridge host or the Databridge Client. To do this on the host, you must redefine the DMSII item using an ALTER REDEFINE. For more information, see Chapter 5, "Alter Data Sets" in the Databridge Programmer's Reference.

To decode DMSII date/time formats represented as NUMBER or ALPHA items, write a script (`script.user_layout.primary_tablename`) that does the following:

- Sets the DIOPT_Use_Long_Date (128) bit in `di_options` .

- Sets the `dms_subtype` column in DMS_ITEMS to indicate the type of date/time encoding method used on the host, as follows:

> **Note**
>
> If your DMSII date/time encoding scheme is not listed in the following table, see the next section.

Date/Time Encoding Scheme	Value for dms_subtype Column
ALPHA(14) or NUMBER(14) with four-digit year followed by a six-digit time	
	121
yyyymmddhhmnss	122
yyyyddmmhhmnss	123
mmddyyyyhhmnss	124
mmyyyyddhhmnss	125
ddmmyyyyhhmnss	126
ddyyyymmhhmnss	
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	
	127
DDDyyyyhhmnss	128
yyyyDDDhhmnss	
ALPHA(12) or NUMBER(12) with two-digit year representing dates in both the 20th and 21st centuries followed by a six-digit time	
yymmddhhmnss	131
yyddmmhhmnss*	132
mmddyyhhmnss	133
mmyyddhhmnss	134
ddmmyyhhmnss	135
ddyymmhhmnss	136
ALPHA(11) or NUMBER(11) with two-digit year representing dates in both the 20th and 21st centuries where days DDD is a number between 1-366 for Julian dates followed by a six-digit time	
DDDyyhhmnss	137
yyDDDhhmnss	138
ALPHA(12) or NUMBER(12) with two-digit year yy (1900-1999) preceded by a six-digit time	
	211
hhmnssyymmdd	212
hhmnssyyddmm	213
hhmnssmmddyy	214
hhmnssmmyydd	215
hhmnssddmmyy	216
hhmnssddyymm	
ALPHA(11) or NUMBER(11) with two-digit year yy (1900-1999) and with days DDD where DDD is a number between 1-366 for Julian dates preceded by a six-digit time	
	217
hhmnssDDDyy	128
hhmnssyyDDD	

Date/Time Encoding Scheme	Value for dms_subtype Column
ALPHA(14) or NUMBER(14) with four-digit year preceded by a six-digit time hhmnssyyyymmdd hhmnssyyyyddmm hhmnssmmddyyyy hhmnssmmyyyydd hhmnssddmmyyyy hhmnssddyyyymm	221 222 223 224 225 226
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 hhmnssDDDyyyy hhmnssyyyyDDD	227 228
ALPHA(12) or NUMBER(12) with two-digit year representing dates in both the 20th and 21st centuries preceded by a six-digit time hhmnssyymmdd hhmnssyyddmm hhmnssmmddyy hhmnssmmyydd hhmnssddmmyy hhmnssddyymm	231 232 233 234 235 236
ALPHA(11) or NUMBER(11) with two-digit year representing dates in both the 20th and 21st centuries where days DDD is a number between 1-366 for Julian dates preceded by a six-digit time hhmnssDDDyy hhmnssyyDDD	237 238

The configuration parameter century_break allows you to adjust the range for the year.

For example scripts, see Cloning an Alpha or Number Field as a Date/Time.

Custom DMSII Date/Time Represented as ALPHA or NUMBER

You may be able to decode DMSII date/time formats represented as NUMBER or ALPHA items, and convert them to relational database date/time format even if you could not find the correct encoding scheme in the previous sections. For instance, if the DMSII date item has no day (*mmyy* or *yymm*), dms_subtype of 0x32 or 0x23 converts this to relational database date/time with a day as "1" and the time as all zeros. For this to work, the DMSII item cannot include any ALPHA data (such as slashes, dashes, or month names). Therefore, 01-FEB-14 would not convert, but 0214 would.

To decode these custom date or date/time layouts using the Databridge Client, write a script
(`script.user_layout.primary_tablename`) that does the following:

1. Sets the DIOPT_Clone_as_Date (2) and the DIOPT_VarFormat_Date (2048) bits in `di_options` .

2. Sets the `dms_subtype` column in DMS_ITEMS to indicate the hexadecimal string, in the same order as the host item layout, as
   follows:

Date/Time Encoding Scheme	Description	Hexadecimal Value for dms_subtype Column
yyyy	Four-digit year	1
yy	Two-digit year within 1950-2049 To adjust this range, use the century_break configuration parameter. See [century_break]().	2
mm	Two-digit month	3
dd	Two-digit day	4
hh	Two-digit hour	5
mn	Two-digit minutes	6
ss	Two-digit seconds **NOTE**: The Databridge SQL Server Client stores all host values for seconds (ss) as zero unless you add the DIOPT_Use_LongDate (128) bit to `di_options` in step one of the layout script. See "di_options" in DMS_ITEMS.	7
mmm	Three-digit fractions of seconds (milliseconds)	8
mmmmmm	Six-digit fractions of seconds (nanoseconds)	9
mm	Two-digit fractions of seconds (centiseconds)	10

As stated previously, the format can be as short as *yymm* (`dms_subtype` 0x23 or 35 decimal). Formats like *mmhhyy* are supported
(`dms_subtype` of 0x253 or 850 decimal) as well as longer ones. For example, a mainframe date/time layout of *mmsshhmnddyy*
uses the `dms_subtype` value of 0x375642 or 3626562 decimal.

Numeric Date and Time in Non-Contiguous Columns

When a DMSII date and time are in contiguous column, you can easily make the Client handle the combined columns as a single
date/time quantity by merging the two columns. You can do this by setting the bit 16777216 in `di_options` of the first item to make
the `define` command merge the two items when it maps them to the relational database table. You can then mark the item to be
cloned as a date and set the appropriate value for its `dms_subtype` column. For example, if you have an item that is a NUMBER(8)
representing a date which is immediately followed by an item that is NUMBER(6) representing a time, you can make the Client treat
the first item as if it were a NUMBER(14) ignore the second one. This can also be done by using an ALTER REDEFINE in
DBGenFormat.

When the two columns are not contiguous, use the `dms_concat_num` column to append the time part of the combined item to the
date part. This column must be set to the item number of the item containing the time value. The Client will effectively treat these
two items as if the second one were concatenated to the first one. You must also set the `di_options` bit 524288 (0x80000) to make
the Client include the second item in DATAITEMS with its `active` column set to 0. This is a lot more efficient than using
DBGenFormat to perform this operation.

See a sample script and its explanation here, Concatenating Two Items and Cloning the Result as a Date/Time

3.5 Creating Indexes for Tables

This section explains how the Databridge Client creates indexes for tables mapped from a DMSII data set.

Ideally, the Databridge Client uses the optimum SET among the various sets defined for the data set in the DASDL. Only SETs that have the NO DUPLICATES ALLOWED attribute (SETs with unique keys) qualify for this selection.

3.5.1 Keys Derived from the DMSII Database

First, the Databridge Engine decides whether any SETs meet this requirement. If more than one SET does, the Databridge Engine uses the SET with the least number of keys. In case of a tie, it uses the SET with the smallest-sized keys.

In addition, the DBGenFormat utility allows you to declare a primary key without modifying the DASDL. The Databridge Engine is responsible for passing information about DBGenFormat primary keys to the Databridge Client. The Databridge Client sometimes uses these keys for VIRTUAL data sets or any other types of data sets that do not have a SET that meets the requirements mentioned above. If you have both a qualified SET and a PRIMARY KEY defined in the GenFormat file, the Client uses the PRIMARY KEY.

> ✎ **Note**
>
> If a DMSII SET with the NO DUPLICATES ALLOWED attribute exists, we recommend that you use it as the source of the index rather than declaring a DBGenFormat primary key.

When the Databridge Engine uses a DMSII SET as the index for tables derived from the data set, the name of the DMSII SET is stored in the `set_name` column of the DATASETS Client control table. Alternatively, when the Databridge Engine uses a DBGenFormat primary key as the index for tables derived from the data set, the name "pk_set" is stored in the `set_name` column.

3.5.2 Using Sets with the KEYCHANGEOK Attribute

Some DMSII SETs have the KEYCHANGEOK attribute, which indicates that it is legal for the value of items that are members of the SET (that is, keys) to change. When the SET being used as the index has the KEYCHANGEOK attribute, this is reflected by bit 4096 (0x1000) in the `ds_options` columns of the corresponding row in the DATASETS control table. This causes the Client to register the keys it is using with the Databridge Engine, which then compares the keys in the before and after images of an update to determine if the update should be sent to the Client as a MODIFY when the keys are unchanged or as a MODIFY BI/AI pair when a key change occurs. This allows the Client perform the update by deleting the old record and inserting the new one when a key change occurs.

If the Client used a MODIFY when a key change occurred, the update statement would fail and the Client would then recover by doing an insert instead. This would result in the old record and the new record both being present in the database resulting in an incorrect replication.

3.5.3 RSNs and AA Values as Keys

If the Databridge Engine does not find a suitable index, the Client tries to use the RSN (record sequence number) or the AA Value (absolute address) of the records as the key. Both of these items are A-Series words (48-bit quantities). They are passed to the Client as part of the record header. Both use the same entry in the header, and the Databridge Engine informs the Client about what this item represents, as explained below. If the Client decides to use one of these quantities as the key, the `set_name` column is set to "aa_set" in the DATASETS Client control table. Otherwise, this column is left blank, indicating that there is no set usable as an index.

The Databridge Client can represent AA Values (or RSNs) the following ways:

- CHAR(12), where each character is the hexadecimal representation of the correspond digit (half-byte) in the A-Series word. This is the default.

- BINARY(6), a binary quantity that uses 48-bits where each byte in the A-Series word is represented by a byte in the relational database. See use_binary_aa.

- Using numeric fields to hold the AA Values (or RSNs). In this case the Databridge Client uses an appropriate numeric data type to hold the AA Values (or RSN), mainly, BIGINT for SQL Server and NUMBER(15) for Oracle. See use_decimal_aa.

> ✎ **Note**
>
> If a DMSII SET with the NO DUPLICATES ALLOWED attribute exists or the data set has an RSN, we recommend that you use one of these keys rather than declaring a DBGenFormat primary key.

RSNs are unique serial numbers that get assigned to records when they get created and remain associated with the record for the life of the record. You must have DMSII XE to be able to use RSNs. Furthermore, you must explicitly enable RSNs in the DASDL by adding the EXTENDED attribute to the data set. If you explicitly add a column to a data set whose value is the RSN, the Databridge Client will allow you to use this column as an RSN rather than a REAL. In such cases, the Databridge Engine automatically sets the `di_options` bit DIOPT_Clone_as_RSN in the corresponding DMS_ITEMS table entry to make the Client treat this item (which will be a REAL) as an RSN. See DMS_ITEMS.

AA Values are the absolute address (that is, the file address --- offset within the file --- of the records in the data set). They do not remain constant over time; however, in the following cases, AA_values are required to implement foreign keys to link records in related data sets:

- Any data set that contains one or more embedded data sets must always use AA Values as the key. Embedded data sets use Parent_AA Values to implement the link to their parent structures.

- When an active data set has links to another data set, the latter must use AA Values as the key.

In both of these cases, the Databridge Engine will use AA Values for the data set in question regardless of whether there is a SET that qualifies for being used as an index, or whether an RSN exists.

Not all data sets have valid AA Values; for example, ORDERED and COMPACT data sets do not have valid AA Values. When AA Values are used as the key, the `set_name` column of the DATASETS Client control table is set to the name "aa_set". The name "aa_set" causes the RSN or the AA Value to be used as part of the index using a column named my_rsn or my_aa depending on whether this is an RSN or an AA Value.

To find out if a data set has an RSN or a valid AA Value, you need to look at the `misc_flags` column of the entry for the data set in the DATASETS Client control table. The bit DSFLG_Static_AA (bit mask 64) is used to indicate whether the Client is using an RSN or an AA Value (1 indicates RSN and 0 indicates AA Value). The bit DSFLG_Valid_AA (bit mask 128) is used to indicate whether or not the data set has a valid AA Value (1 indicates a valid AA Value). The Client has no control over the selection of RSNs versus AA Values. This decision is made by the Databridge Engine.

The advantage of using the AA Value to generate a unique key is that it makes updates possible for data sets that could not otherwise be updated; however, this value is not an absolute constant. Any DMSII reorganization (record conversion, file format, or garbage collection) changes these values. You must re-clone a data set that uses AA Values as keys whenever the AA Values change. Therefore, we recommend that you consider creating a unique composite key rather than using AA Values.

The Databridge Client recognizes the names "aa_set", "user_set", and "pk_set" as special names (the use of the underscore is not allowed in DMSII names).

Forcing the Client to Use RSN or AA Values as Keys

You can force the Client to use the RSN or AA Value as the key for a specific data set by setting the `ds_options` bit, `DSOPT_Use_AA_Only` (bit mask 16384) in the DATASETS table entry for the data set in question.

> ✏️ **Note**
>
> You can also do this from the **Customize** command by using the checkbox **Use AA Values (or RSNs) As Keys** in the **Options** section of properties of the data set.

To perform this action globally, use the parameter `force_aa_value_only` with one of the following values. (For more details about this parameter, see force_aa_value_only. Note that this does not have any effect until you run a `redefine` command (with the -R option) to get the global setting applied to all the data sets.

Value	Description
0	Globally disables the parameter
1	Globally enables the parameter
2	Only applies to data sets that have an RSN; using a SET as the source for the index is always preferable to using AA Values that are volatile.

3.5.4 User Defined Keys in GenFormat

You can create a user-defined SET for a data set by using the PRIMARY KEY construct in GenFormat. When a PRIMARY KEY exists, it is used instead of a SET that would otherwise qualify as the source for the index on the table. To properly identify the source of such an index, the Databridge Client sets the `set_name` to "pk_set" when it originates from a PRIMARY KEY construct. The Databridge Client recognizes "pk_set" as a special name, just like "aa_set" and "user_set". The only difference between "user_set" and "pk_set" is their origin.

3.5.5 Composite Keys

Composite keys use several columns in a relational data table to form a unique index. The entries you make (via a user script) in the `item_key` column of the DMS_ITEMS Client control table determine the order in which the columns are used in the key.

> ✏️ **Note**
>
> If you specify a member of a DMSII GROUP as part of a composite key, you must also set the corresponding `item_key` column for the GROUP to a value of 1 so that the `define` (or `redefine`) command picks it up.

To avoid this step, define the composite key in the DBGenFormat parameter file on the host.

When to Use Composite Keys

We recommend that you create a composite key for data sets that do not have a unique key. Creating a composite key is required for the following data sets:

- Data sets that do not have valid RSNs or AA Values, such as COMPACT, ORDERED, and VIRTUAL data sets
- Data sets that use AA Values and for which garbage collection reorganizations are frequently performed.

> ⚠️ **Caution**
>
> If the composite key that you create is not unique, the following can occur:
> - If a duplicate record is encountered after you clone the data set, the index creation for the resulting table fails. The SQL query we use to eliminate duplicate records will get rid of all copies of the duplicate record.
> - If a duplicate record is encountered while attempting to insert a record during an update, the original record is deleted and replaced with the new copy of the record.

When you create a composite key, make sure that you enter the value "user_set" into the `set_name` column. If you do not, one of two things happens, as follows:

- If the `set_name` value is "aa_set", a column named my_aa, which contains the AA Value of the record is automatically included in the table.

- If the `set_name` value is blank, the program does not create an index, regardless of the values of the `item_key` column of the various DMS_ITEMS Client control table entries.

Once you are familiar with the concepts in this section, and you determine which data sets require composite keys, you must include the SQL statements in the data set mapping customization script for the data set (`script.user_layout.primary_tablename`).

Composite Keys Defined by the User

If the Databridge Engine does not find a suitable SET or DBGenFormat primary key, the Databridge Client allows you to create a composite key. You can also create a composite key when the Databridge Client decides to use AA Values as the primary key.

> ✏️ **Note**
>
> - If the added column is named "my_rsn," this indicates that it is an RSN, which makes an excellent key. Do not use composite keys when this is the case.
> - You must not create a composite key for a data set that contains embedded data sets or for a data set that has other active data sets linking to it when the handling of DMSII links is enabled.

If a data set does not have a DBGenFormat primary key or a DMSII set that qualifies for use as an index, and the AA Values are not valid, the `set_name` column in the DATASETS Client control table is left blank. In this case, you can clone the data set, but you cannot track updates.

When the DMSII data set does not have a key, we recommend that you create a composite key using the data set mapping customization script (`script.user_layout.primary_tablename`). See When to Use Composite Keys for more details about when to use a composite key.

Creating a Composite Key

- Modify `script.user_layout.primary_tablename` to do the following:

  - If you don't use the **Customize** command, set the `set_name` column of the DATASETS Client control table entry for the data set in question to "user_set". If you use the Client Customizer, this is done automatically.

  - Specify which items should be part of the composite key by assigning the appropriate values to the corresponding entries for the `item_key` column of the DMS_ITEMS Client control table. Such entries are identified by the values of the `dms_item_name` and the `dataset_name` columns.

- After you create the composite key, do one of the following:

  - If you have not cloned any tables, run the `define` command again.

  - If you have cloned tables, set the `status_bits` column for the corresponding entry in the DATASETS Client control table to 8, and run a `redefine` command.

- If you ran a `define` command (or if the `redefine` command prompts you to run a `generate` command) run the `generate` command from the working directory that for the data source. Otherwise, you'll be prompted to run the `reorg` command, which fixes the index for the table.

- From the data source's working directory, run a `process` command. This clones or re-clones the data set, if needed, and resumes tracking.

3.6 Adding a Non DMSII Column

Non DMSII columns (also called user columns) are generally used to store the audit file timestamp so that you can keep track of when the data was last updated. You can add non DMSII columns to your relational tables in any of the following ways:

- To add a non DMSII column to *every* data set, set the corresponding bit in the configuration file parameter `default_user_columns`; this parameter then assigns the appropriate value to the `external_columns` column of the DATASETS Client control table. The bits in this column determine which non DMSII columns are added to your data table.

- To prevent the Client from adding some of the non DMSII columns to secondary tables (for example, DMSII items that have an occurs clause), set the corresponding bit in the configuration file parameter `sec_tab_column_mask`. This parameter is used in conjunction with the `external_columns` column in the DATASETS table entry.

- To add a non DMSII column to most, but not all, of your data sets, use the script `script.user_layout.primary_tablename` to set the `external_columns` column of the DATASETS Client control table back to 0 for the data sets that you want to keep unchanged.

- To add a non DMSII column to only a few data sets, do not set the `default_user_columns` parameter. Instead, use the script `script.user_layout.primary_tablename` to modify the `external_columns` column of the DATASETS Client control table for the data sets you want to change.

3.6.1 Types of Non DMSII Columns

The Databridge Client offers several default non DMSII columns (user columns). You can add user columns to the relational database tables either by using user scripts, as described in this section, or by using the **Customize** command. For more information about the Client Customizer, see the Databridge Administrative Console Help.

> ✏️ **Note**
>
> The value for the Bit column in this table is equal to the value in the `dms_subtype` column of the DATAITEMS Client control table. The exception is bit 14, which results in a dms_subtype of 0. Bits are numbered from right to left; the right-most bit is 1.

Bit	Value	User Column Name	Description
1	1	update_type	Database update type, as follows: 0 for extract1 for create 2 for delete (bit 10 must also be enabled) 3 for modify **NOTE**: This value cannot be used at the same time as bit 11.
2	2	update_time	Time the update was applied to the relational database (PC time)
3	4	update_ts	(SQL Server Clients only) SQL Server timestamp data type. (The timestamp is a data type that exposes automatically-generated unique binary numbers within a database. It is not a true timestamp that contains a date and time value.)
4	8	audit_ts	DMSII audit file timestamp. This column is set to NULL during the initial clone. **NOTE**: This bit cannot be used at the same time as bit 13.
5	16	audit_filenum	Audit file number **NOTE**: If you use a decimal number, its precision must be at least 4. Otherwise, the value may be too large and result in a SQL error.
6	32	audit_block	Audit block serial number (ABSN) **NOTE**: If you use a decimal number, its precision must be at least 10. Do not use a data type of int, as the ABSN is a 32-bit unsigned number. Otherwise, the value may be too large and result in an overflow, which will result in a SQL error.
7	64	source_name	Data source name
8	128	source_id	Data source identifier as defined in the DATASOURCES Client control table
9	256	my_id	SQL SERVER IDENTITY column. Updates have no effect on this number. **NOTE**: For Windows Clients only: This column won't appear on Clients other than SQL Server, even if requested. The Oracle database provides the equivalent functionality with the ROWID pseudo-column, which is always present.

Bit	Value	User Column Name	Description
10	512	deleted_record	Delete indicator (key item). A nonzero value indicates that the record is deleted. This is actually the value of the Client machine's clock at the time of the deletion. Making this column part of the index allows multiple instances of a deleted record to coexist without being considered duplicate records. • This bit cannot be used at the same time as bit 11. These types are compared in Preserving Deleted Records. • The granularity of this column is in seconds. If you have applications that perform many delete/insert operations, you may want to add a delete_seqno column to prevent the Client from getting duplicate deleted records. The Client recovers from this by waiting one second and retrying the operation, which can significantly slow the Client's performance.

11	1024	update_type	Expanded database update type as follows:
			
			`0 for extract`
			`1 for create`
			`2 for delete`
			`3 for modify`
			
			• If the key for this record is reused, the key is removed when the new, duplicate record is inserted.
			
			• This value cannot be used at the same time as bit 1 or bit 10. Bits 10 and 11 are compared in Preserving Deleted Records.
			
			• This bit and bit 1 work in the same way, except that this bit preserves the deleted image.

12	2048	source_id	Data source identifier as defined in the DATASOURCES Client control table (key item)
13	4096	audit_ts	Expanded audit file time. This column contains the DMSII audit file timestamp during updates and the starting time of the data extraction during extraction. **NOTE**: This bit cannot be used at the same time as bit 4.
14	8192	user_column1	Generic user column whose entry is left as NULL
15	16384	sequence_no	A sequence number used in history tables to determine the order of updates when they have the same update_time values
16	32768	delete_seqno	Augments the `deleted_record` column with a sequence number to provide higher granularity and avoid creating duplicate deleted records.
17	65536	create_time	Time when the record was created in the relational database (PC time).
18	131072	user_column2	Generic user column whose entry is left as NULL.
19	262144	user_column3	Generic user column whose entry is left as NULL.
20	524288	user_column4	Generic user column whose entry is left as NULL.

3.6.2 Values for Non DMSII Columns

The bit numbers, decimal values, and hexadecimal values for the user column names are shown in the following table.

Default Name	Bit Number	Decimal Calue	Hex Value
update_type	1	1	0x00000001
update_time	2	2	0x00000002
update_ts	3	4	0x00000004
audit_ts	4	8	0x00000008
audit_filenum	5	16	0x00000010
audit_block	6	32	0x00000020
source_name	7	64	0x00000040
source_id	8	128	0x00000080
my_id	9	256	0x00000100
deleted_record	10	512	0x00000200
update_type	11	1024	0x00000400
source_id_key	12	2048	0x00000800
audit_ts	13	4096	0x00001000
user_column1	14	8192	0x00002000
update_seqno	15	16384	0x00004000
delete_seqno	16	32768	0x00008000
create_time	17	65536	0x00010000
delete_seqno	18	131072	0x00020000
delete_seqno	19	262144	0x00040000
delete_seqno	20	524288	0x00080000

3.6.3 Setting Up History Tables

The primary data tables use the CREATE, MODIFY, and DELETE records from the mainframe to build an exact duplicate of DMSII data sets.

A history table, on the other hand, treats these records as new records to insert, even though a history table is structured similarly to a primary data table. In effect, the history table becomes a log or record of mainframe changes. History tables are usually enabled as a device to feed data warehouse applications. History tables will continue to grow as Databridge replicates data, so you should purge them regularly after successful updates to the data warehouse.

To enable history tables, set DSOPT_Save_Updates (bit mask 8 of `ds_options` in the DATASETS Client control table). You must enable history tables before you generate Databridge Client scripts, as explained in the next section. If you want to set this bit for all data sets, you can set the configuration parameter `history_tables` to 1.

Each history table has the same name as the corresponding primary data table with a "_h" suffix.

It is also possible to create only history tables for a data set or for all data sets. To do this for all data sets, simple set the `history_tables` parameter to 2 in the configuration file. This will cause the `ds_options` bit DSOPT_History_Only (8192) to be set for all data sets. If you only want to do this for a few data sets, then you can use the user script `script.user_layout.dataset` to do this.

> ⚠️ **Caution**
>
> When setting bits in `ds_options` , beware that some bits may already be set. You should use the "|" operator for SQL Server and the BITOR function for Oracle to set a bit rather than setting the column to that value.

3.6.4 Modifying Non DMSII Column Names

The configuration file parameter `external_column[n]` allows you to tailor attributes, such as the column name, of individual non DMSII columns. For details and a list of allowable sql_type values, see external_column[n].

3.6.5 Preserving Deleted Records

Both the `deleted_record` column (bit 10) and the `update_type` column (bit 11 only) may be used to preserve deleted records, which is useful when trying to recreate updates to the database.

Be aware of the following when using these bits:

- Bit 11 preserves only the *last instance* of the deleted record. For example, if the key value of the deleted record is reused, the deleted record is replaced when the duplicate (new) record is inserted.
- Bit 10 results in the `deleted_record` column being included in the index. The value in this column is a time value, which makes the values in the index unique; therefore, you can keep multiple instances of the deleted record. The granularity of this column is in seconds, if you need coarser granularity you should add the `delete_seqno` column described in Values for Non DMSII Columns.

In addition, you must clean up deleted images when they are no longer needed.

> ✏️ **Note**
>
> If you use the first method (bit 11) to preserve deleted records, the deleted records will only survive during a re-clone if you set the `preserve_deletes` parameter to True. If you use the second method (bit 10), the deleted records will always be preserved during a re-clone.

3.7 Generating Databridge Client Scripts

In this phase, the Databridge Client generates script files that are used to create the Databridge data tables in the relational database and run the database bulk loader utility to populate those tables during the data extraction phase.

The `generate` command creates scripts only for those data sets that have an `active` column set to 1 in the corresponding entry in the DATASETS Client control table. The Databridge Client keeps track of the data sets that have been generated. These scripts will only be generated again if a `define` command is executed or if a `redefine` command determines that the layout of a table has changed. If you need to force the Databridge Client to generate the scripts for all data sets that have a corresponding `active` column value of 1 in the DATASETS Client control table, you can specify the `-u` option on the command line for the `generate` command.

To view a list of the scripts that are generated, see Summary of Script Files.

You can also perform this action from the Administrative Console by clicking **Actions > Generate Scripts**. If you use the **Customize** command and have a new data source, you will need to perform this step after you exit from the **Customize** command.

1. If you plan to use the dbridge.cfg file for signon parameters, set them before you continue. (See the *Databridge Installation Guide*.)

2. Make sure that the following parameters, which affect the `generate` command, are set correctly in the appropriate section of the Client configuration file:

   [params]
   global_table_suffix
   create_table_suffix
   create_index_suffix

   [bulk_loader]
   bcp_batch_size
   bcp_packet_size
   bcp_code_page
   bcp_copied_message
   sqlld_rows
   sqlld_bindsize
   inhibit_direct_mode
   enable_parallel_mode
   max_errors

   > ✏️ **Note**
   >
   > For your changes to take effect, you must run the `generate` command again and specify the -u option to force the program to regenerate the scripts.

3. Enter the following command:

   dbutility [*signon_options misc_options*] generate *datasource*

Where	Is
signon_options	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** [-U *userid*] [-P *password*] [-D *database*] **SQL Server:** [-U *userid*] [-P *password*] [-W] [-O *ODBCdatasource*]
misc_options	Any of the following miscellaneous command-line options: -T forces the Client to use a new trace file for this run, if tracing is enabled. -f *filename* to specify a configuration file other than the default dbridge.cfg file in the working directory. -L forces the Client to use a new log file for this run. -u unconditionally generates scripts for all tables mapped from data sets that have a corresponding `active` column value of 1 in the DATASETS Client control table. See dbutility Command-Line Options.
datasource	The name that matches the entry in the DATASOURCES Client control table. You can enter the data source name in uppercase or lowercase.

   Status messages indicate the progress of the command.

4. To check on the results of the `generate` command, see Summary of Script Files. For information on when to run `generate` next, see When to Run dbutility generate.

At this point, you are ready to run a `process` or `clone` command to create and populate the Databridge tables in the relational database with DMSII data. See Populating the Databridge Data Tables.

3.7.1 Example of Script Files

In this example, scripts are generated for the CUSTOMER data set and the PRODUCTS data set, as follows:

Windows Script Files

```
> dir /on dbscripts
bcp.customer.fmt(Microsoft SQL Server only)
bcp.products.fmt(Microsoft SQL Server only)
load.customer.cmd
load.products.cmd
script.clrduprecs.customer
script.clrduprecs.products
script.create.customer
script.create.products
script.drop.customer
script.drop.products
script.index.customer
script.index.products
sqlld.customer.ctl(Oracle only)
sqlld.products.ctl(Oracle only)
```

UNIX Script Files

```
> ls dbscripts
load.customer.sh
load.products.sh
script.clrduprecs.customer
script.clrduprecs.products
script.create.customer
script.create.products
script.drop.customer
script.drop.products
script.index.customer
script.index.product
sqlld.customer.ctl
sqlld.products.ctl
```

The script files are stored in the **dbscripts** subdirectory of the working directory, which is the directory from which you run the **dbutility** `generate` command.

3.7.2 Summary of Script Files

The `generate` command produces the following script files:

- SQL script files that create data tables and stored procedures to update them in the target relational database (script.create.*tablename*)

- SQL script files that remove selected records from a data table in the SQL*Loader (script.cleanup.*tablename*). See the table that follows for details about the conditions under which these scripts are generated

- SQL script files that remove false duplicate records that can occur during a long clone process of an active DMSII database, if `clr_dup_extr_recs` is set to True (script.clrduprecs.*tablename*)

- SQL script files that drop data tables from the target relational database (script.drop.*tablename*)

- SQL script files that create indexes for data tables in the target relational database (script.index.*tablename*)

- Windows command (or UNIX shell script) files to run the utility (load.*tablename*.cmd or load.*tablename*.sh). The bulk loader is used during the data extraction phase of a cloning operation of a data set.

- SQL*Loader control files for Oracle (sqlld.*tablename*.ctl) and bcp format files for Microsoft SQL Server (bcp.*tablename*.fmt).

The following table summarizes the scripts that are created for each Oracle table. Each DMSII data set that is cloned is mapped to one or more tables. The Databridge Client creates one set of files for each of these tables that have a corresponding `active` column value of 1 in the DATATABLES Client control table.

File	Description
SQL Server: `bcp.table.fmt`	This is a control file that contains the bcp parameters that describe the format of the data.
Oracle: `sqlld.table.ctl`	This is a control file that contains the SQL*Loader parameters that describe the format of the data.
Windows: `load.table.cmd` **UNIX:** `load.table.sh`	This is a Windows command file used to run the relational database bulk loader (bcp for Microsoft SQL Server and SQL*Loader for Oracle). This is a UNIX shell script used to run SQL*Loader.
`script.create.table`	This is a script that contains SQL statements to create the relational database table named table. It also contains the SQL statements to create the associated stored procedures for updating this table. Before starting the data extraction phase of a `process` or `clone` command, this script is executed to create the table and its associated stored procedures. The following stored procedures are used during the `process` and `clone` commands for updating the table (specified by table): i_table stored procedure for inserting a record d_table stored procedure for deleting a record u_table stored procedure for updating a record z_table stored procedure for deleting all rows for all occurrences of key in secondary tables using a single SQL statement
`script.drop.table`	This is a script that contains SQL statements to drop the relational database table named table and to drop the stored procedures associated with this table. `script.drop.tablescripts` are used by the process, `clone`, `drop`, and `dropall` commands to drop a specified table and its associated stored procedures. If a table to be cloned (ds_mode=0) already exists during a `process` or `clone` command, this script is executed to drop both the table and its stored procedures before recreating them. During a `process` or `clone` command, if the Databridge Client receives a message from the Databridge Engine indicating that a DMSII data set has been purged, this script is executed to drop the table. Immediately after the table is dropped, the script to recreate the table is executed.
`script.cleanup[2].table`	This script contains SQL statements to delete selected records from the relational database table. This script is typically called script.cleanup.table, except when both of the conditions below are true. In that case, an additional cleanup script named script.cleanup2.tableis also created to remove all records except the deleted records from the table. NOTE: This script is generated under rare conditions where tables are not fully re-cloned, as in the following cases: • The data set is set up to preserve deleted records. • The data set is a virtual data set that gets its input from more than one DMSII data set.

File	Description
`script.index.table`	This is a script that contains SQL statements to create an index for the given table. **NOTE**: This script is created only when the table has an index.
`script.clrduprecs.table`	This script removes records with false duplicate key values when the bit DSOPT_Clrdup_Recs (32768) is set in the `ds_options` column of the DATASETS table entry for the data set.

3.7.3 When to Run dbutility generate

Run **dbutility** `generate` when you need to create a new set of scripts for a data source. For example, you would run **dbutility** again in the following circumstances:

- If you accidentally delete one or more script files, repeat the **dbutility** `generate` command with the `-u` option. Make sure that the current directory is the working directory for the data source where you want **dbutility** `generate` to write the script files.

- If you disable cloning (set the `active` column to 0 in the DATASETS Client control table) for one or more data sets prior to running the **dbutility** `generate` command, no scripts are created for these data sets. If you later decide that you want one or more of these data sets to be cloned, set the `active` column back to 1, run the `redefine` command, and then run the `generate` command. The missing scripts are created and you can then run the `clone` command to clone the data set.

4. Cloning a DMSII Database

This chapter covers the steps to clone a DMSII database.

4.1 Cloning Issues for All Relational Databases

We recommend that you read this section before you use the `process` or `clone` commands.

Disk Space	You need to consider two types of disk space for the Databridge Client, as follows: • Database storage is required by both the relational database and the DMSII data. • Temporary file storage is required for Windows Clients during the cloning process. These temporary disk files hold the data used by the bulk loader utilities. For information on how to handle temporary file storage, see Controlling Temporary File Storage for the Windows Clients.
Column Order	The columns in the Client database are built in a different order than the order in the DMSII database. Specifically, the key items are placed first, followed by the non-key items in DMSII column order.
Databridge Client Log File	Logging and tracing are separate activities in the Databridge Client. Logging cannot be disabled. Log files are written to the **logs** subdirectory of the working directory. Trace files are only created when the `-t` or `-d` options are used and they are placed in the working directory. Using `-t1` is not allowed because this would create a second copy of the log file. You must specify at least one more bit in the trace mask for the option to be accepted.

4.2 Bulk Loader Parameters

Both dbutility `process` and dbutility `clone` use a bulk loader utility to populate the Databridge tables in the relational database during the data extraction phase (not during change tracking). Using the relational database bulk loader utility greatly increases the speed with which the Databridge data tables are populated.

This section lists the configuration parameters that affect the Databridge Client operations when using the relational database bulk loader utility. You can use these parameters to do the following:

• Control temporary file storage (`max_temp_storage` parameter, Windows Client only)
• Control the bulk loader utility maximum error count (`max_errors` parameter)

You can set the bulk loader parameters from the Administrative Console. These and other configuration parameters are available in the **Configure** item of the **Settings** the menu for the data source in question. For information, see the Administrative Console help **Help**.

Parameters that are specific to the SQL*Loader and BCP API are discussed in the next sections.

4.2.1 Controlling Temporary File Storage for Windows Clients

During cloning on Windows platforms, the Oracle Client and the SQLServer Client (unless directed to use the BCP API, which does not involve the use of temporary files) writes bulk loader data to multiple temporary text files for each table being loaded.

These temporary text files are used as holding areas for the bulk loader data. The Windows Client uses overlapped operations to write data to one set of text files while the bulk loader is loading tables from another set of files. The configuration file parameter `max_temp_storage` determines the maximum amount of storage to be used by all of the temporary files.

The Databridge Client writes data to as many temporary files as it needs, while keeping track of the total amount of storage used. When the amount of storage used exceeds half of the configured value of the configuration file parameter `max_temp_storage`, the Databridge Client closes all the temporary files and queue the tables on the bulk loader thread's work queue. (The default setting for `max_temp_storage` is 400 MB.) While the bulk loader thread is sequentially launching the loads for for these tables (which run as separate processes), the Databridge Client starts filling a new set of temporary files for the next group of loads. This mode of operation significantly enhances performance on systems that have more than one CPU.

4.2.2 Bulk Loader Operations for UNIX Clients

UNIX Clients do not use temporary text files; instead, they use pipes (such as lpipe_*nnn*.dat) to communicate data between processes. This introduces a lot more overlap between the Client and the bulk loader, resulting in a much smoother flow of data.

4.2.3 Controlling the Bulk Loader Maximum Error Count

The `max_errors` parameter controls the number of data errors allowed before the bulk loader's operations are canceled. The default value for `max_errors` is 10, which means that the bulk loader aborts after encountering 10 bad records. These bad records are written to the discard file for the table in the **discards** directory and information about the error is written into the bulk loader log file.

When several bulk loader errors occur, increasing the maximum error count allows you to gather all the errors in one run rather than finding 10 errors and then having to start over again. For more details, see the `max_errors` parameter description in [Bulk_Loader].

4.3 Oracle SQL*Loader Bulk Loader

This section lists the configuration file parameters that affect cloning with Oracle.

The `enable_parallel_mode` parameter, which is only meaningful when direct mode is enabled, causes the program to include the PARALLEL option in the SQL*Loader command line. In direct mode, the loader runs faster at the expense of system resources; however, enabling this option has a much more noticeable impact on Windows Clients than on UNIX Clients.

For non-US sites where the period (.) and comma (,) decimal characters are swapped, the Databridge Client automatically reads the database's NLS parameters and makes the necessary adjustments so that the SQL*Loader input records are formatted using the numeric characters that SQL*Loader expects.

The `inhibit_direct_mode` parameter applies when you run dbutility for a remote Oracle database using SQL*Net®.

The following parameters are meaningful only when `inhibit_direct_mode` is enabled.

- The `sqlld_rows` parameter defines the value to be used for the ROWS specification for SQL*Loader operations.
- The `sqlld_bindsize` parameter defines the value to be used for the BINDSIZE parameter for SQL*Loader operations. Increasing this value can speed up SQL*Loader operations when not using DIRECT mode (for example, running remote to a database on a UNIX system).

For more information about the bulk loader parameters mentioned here, see [Bulk_Loader].

4.3.1 Files Related to SQL*Loader

Each execution of SQL*Loader uses a control file (`load_` *nnn* `.ctl`, which is a copy of the file `sqlld` .*tablename*. `ctl` created by the `generate` command) and a data file (`lpipe_` *nnn* `.dat`) as input.

As a result of the bulk loading process, SQL*Loader produces a log file (`load_` *nnn* `.log`) and, if there are any records that cannot be loaded due to data errors, a discard file (`lpipe_` *nnn* `.bad`) for each table. Discard files are placed into the subdirectory named discards.

Windows Log Files

In Windows, to prevent log files and discard files from being overwritten as a result of successive executions of SQL*Loader during segmented bulk load operations, the Databridge Client uses the SQL*Loader log and discard files as temporary files and does the following:

- At the end of the first load segment, the Databridge Client copies the temporary log file to the permanent log file (`sqlld` *.tablename.* `log`). If a discard file was produced, the Databridge Client also copies the temporary discard file to the permanent discard file (`sqlld` *.tablename.* `bad`) in the **discards** folder.

- At the end of every subsequent load segment, the Databridge Client appends the temporary log files to the end of the permanent log file (`sqlld` *.tablename.* `log`). If a temporary discard file was produced, the Databridge Client either copies it or appends it to the permanent discard file (`sqlld` *.tablename* `.bad`), depending on whether this file exists or not.

- The Databridge Client deletes the temporary log and discard files as soon as they are appended to Databridge Client permanent log and discard files.

UNIX Log Files

In order to maintain compatibility with the Windows Clients, the UNIX Client renames the log and discard files at the end of a SQL*Loader operation. Therefore, the log file `load` *nnn.* *log* is renamed *sqlld* *tablename.* `log` and the discard file `lpipe_` *nnn* `.bad` is moved to the **discards** folder as `sqlld_` *tablename.* `bad` in the SQL*Loader shell scripts.

List of Files Related to SQL*Loader

The table below lists files related to SQL*Loader and Databridge Client operations. In some of the filenames below, *nnn* is the value in the `table_number` column of the DATATABLES Client control table. It is unique within each data source.

File	Description
`sqlld.tablename.ctl`	The SQL*Loader control file created by the `generate` command. It describes the format of the data in the data file (`lpipe_nnn.dat`).
`lpipe_nnn.dat`	**For Windows:** This is a temporary file that the Databridge Client creates. It contains the data to be loaded into an Oracle table. Since the Client uses two copies of this file simultaneously when doing overlapped operations, it appends a suffix to the file name to make it unique every time a new file is created. The suffix of the form "_nnn", where nnn is a number that starts at 1 and gets incremented by 1 each time a new file is created for the table in question. Thus the name `"lpipe_12.dat"` will be changed to `lpipe_12_1.dat` for the second file and so on. This file is automatically deleted after a successful load of a table. If the table is not loaded successfully, the file is not deleted. This gives you the opportunity to manually run SQL*Loader *to determine why it is failing.* *For UNIX:This is a UNIX pipe that the SQL*Loader shell script creates and uses to pass data to the SQL*Loader program. This pipe is automatically removed after a successful load of a table. If the Databridge Client or SQL*Loader abends, the pipe is not immediately deleted. If you run the Databridge Client again, you receive a warning message as the pipe is being deleted. You can safely ignore this warning, as this is not a fatal error.
`sqlld.tablename.log`	**For Windows:** This file is a concatenation of all of the load_nnn.log files created during the cloning process. **For UNIX:** This is the log file generated by SQL*Lodare. **CAUTION:** Do not delete the sqlld.*tablename*.log file until you have looked at it. It can contain valuable information such as error messages about rows that were not loaded.
`sqlld.tablename.bad`	**For Windows:** This file is a concatenation of all of the load_nnn.bad files created during the cloning process. It is created in the discards subdirectory only if discard records exist. **For UNIX:** This is the discard file generated by SQL*Loader that has been moved to the **discards** folder. **CAUTION:** Do not delete the sqlld.*tablename*.bad file until you have looked at it to determine which records were rejected by SQL*Loader. Correct the bad data. Then use SQL*Loader to load these records into the appropriate table.

The following temporary files are created while the bulk loader is being launched, but they are deleted before the run is completed:

- `load_nnn.ctl`
- `load_nnn.log` (renamed in UNIX to `sqlld_tablename.log`)
- `lpipe_nnn.bad` (renamed in UNIX to `sqlld.tablename.bad`)

You see these files only if the bulk loader operation abends.

4.4 Microsoft SQL Server BCP API and bcp utility

This section lists the configuration file parameters that affect cloning with BCP API and **bcp** utility. SQL Server Clients on Windows use the **bcp** utility by default.

You can make the SQL Server Client use the BCP API, which allows the program to perform bulk loader operations by making BCP API calls, which operate like SQL statements. This produces more overlap between the Client and the load operations, resulting in a much smoother flow of data. We have had some problems with the BCP API, which lead us to change the default mode of operations to use the **bcp** utility which is more reliable.

It is recommended that you use multi-threaded updates, as this allows multiple tables to be loaded simultaneously by different threads which increases the resource utilization and offers all the advantages of multi-threaded updates when doing data extraction.

> ✏ **Note**
>
> Bulk loader operations will run efficiently if the database recovery model is set to "Simple" or "Bulk-logged". If you are running a database with a recovery model of "Full", we recommend that you switch to "Bulk Logged" for the duration of the bulk-load and then switch back to "Full" recovery.

The following Client configuration file parameters affect the **bcp** utility or BCP API calls made by the Client. For more details, see [EbcdictoAscii].

Parameter	Description
bcp_batch_size	The Databridge Client using the BCP API or the **bcp** utility can load a table in several batches instead of loading the entire table in a single operation. You can control the batch size using this parameter.
bcp_code_page	Adds the -C code_page to the **bcp** command line, which specifies the code page of the data in the file. For example, because the Japanese code page is 932, setting this parameter to 932 adds -C 932 to the **bcp** command line. This parameter is only applicable when using the **bcp** utility.
bcp_packet_size	Defines the network packet size value for the **bcp** utility (applies to remote servers only). If you have wide tables, setting this parameter to a packet size larger than the default (4096) can speed up loading the data into the table at the expense of system resources. This parameter is only applicable when using the **bcp** utility.
bcp_copied_msg	Enables the **bcp_auditor** program to determine whether or not a **bcp** was successful in cases where the database language is not English. This parameter is only applicable when using the **bcp** utility.
bcp_delim	Defines the delimiter character **bcp** uses (the TAB character, by default). If you want to preserve TAB characters in your data, set this parameter to a value that allows multiple characters. This parameter is only applicable when using the **bcp** utility.
max_errors	Controls the bulk loader's tolerance to records that are discarded due to data errors.
max_temp_storage	Activates the segmented bulk load feature, which allows you to specify the maximum amount of storage that dbutility should use for temporary files. This parameter is only applicable when using the **bcp** utility.

4.4.1 bcp_auditor Utility

The **bcp** command files capture **bcp** execution output by redirecting the output to a temporary file. These command files then invoke the `bcp_auditor` utility to examine this file to determine if the **bcp** operation was successful. The `bcp_auditor` utility sets the exit code such that the Databridge Client can determine if the table load was successful.

4.4.2 Files Related to BCP

> ✏️ **Note**
>
> The SQL Server Client will only use the BCP API when specifically directed to do so. You can globally control whether the Client is to use the **bcp** utility or the BCP API, by setting the `use_bcp` parameter accordingly in the [bulk_loader] section of the Client configuration file. You must run a `redefine` command with the `-R` option to make this change take effect. You can force the Client to always use the **bcp** utility by using the `/l` command line option. Alternatively, you can control the loading of tables at the data set level by setting the DSOPT_Use_bcp (0x1000000) bit in the `ds_options` column for the corresponding entries in the DATASETS Client control table.

Each execution of **bcp** uses a format file (`bcp` .*tablename*. `fmt`) and a data file (`bcppipe` .*tablename*) as input.

As a result of the bulk loading process, **bcp** produces a log file (`load_*nnn*.log`) for each table. If there are any records that cannot be loaded due to data errors, **bcp** also produces a discard file (`load` _*nnn*. `bad`).

To prevent log files and discard files from being overwritten during segmented bulk load operations, Databridge Client treats **bcp** log and discard files as temporary files:

- At the end of the first load segment, the Databridge Client copies the temporary log file to the permanent log file (`bcp` .*tablename*. `log`). If a discard file was produced, the Databridge Client also copies the temporary discard file to the permanent discard file (`bcp` .*tablename*. `bad`).

- At the end of every subsequent load segment, the Databridge Client appends the temporary log files to the end of the permanent log file (`bcp` .*tablename*. `log`). If a temporary discard file was produced, the Databridge Client either copies it or appends it to the permanent discard file (`bcp` .*tablename*. `bad`), depending on whether this file exists or not.

- The Databridge Client deletes the temporary log and discard files as soon as they are appended to Databridge Client permanent log and discard files.

Files related to **bcp** and Databridge Client operations are listed in the following table. In some of the filenames below, *nnn* is the value for the `table_number` column in the DATATABLES Client control table. The table number is unique within each data source.

File	Description
`bcp.tablename.fmt`	The bcp format file that is created by the `generate` command. It describes the format of the data in the data file (`bcppipe`.*tablename*).
`bcppipe.tablename`	A temporary file created by the Databridge Client. It contains the data to be loaded into a Microsoft SQL Server table. Since the Client uses two copies of this file simultaneously when doing overlapped operations, it appends a suffix to the file name to make it unique every time a new file is created. The suffix is of the form "*_nnn*" where *nnn* is a number that starts at 1 and gets incremented by 1 each time a new file is created for the table in question. Thus the name " `bcppipe.customer` " will be changed to " `bcppipe.customer_1` " for the second file and so on. This file is automatically deleted after a successful load for a table. If the table is not loaded successfully, the file is not deleted. This gives you the opportunity to manually run bcp to determine why it is failing. **Important:** The `bcppipe`.*tablename* files can be quite large. When these files are no longer needed, make sure you delete them to prevent errors from occurring.
`bcp.tablename.log`	A concatenation of bcp screen output created during the cloning process. The files are created in the working directory for the data source. **CAUTION:** Do not delete the `bcp`.*tablename*.`log` file until you have looked at it. It can contain valuable information such as error messages about rows that were not loaded.
`bcp.tablename.bad`	A concatenation of all of the `load_nnn.bad` files created during the cloning process. These files are created in the discards subdirectory. **CAUTION:** Do not delete the bcp.tablename.bad file until you have looked at it. It can contain valuable information such as which rows were not loaded. Correct the bad data and use bcp to load these records into the appropriate table.

The following temporary files are created while the bulk loader is being launched, but they are deleted before the run is completed:

- `load_nnn.log`
- `load_nnn.bad`

You see these files only if the bulk loader operation abends.

4.4.3 Files related to the BCP API

When using the BCP API all errors are logged to the Client log file. If there are discarded records, they are written to the *tablename*.`bad` file located in the discards folder, this is similar to discards during the tracking phase.

4.5 Configuring Host Parameters

TCP/IP throughput is greatly affected by the `BLOCKTIMEOUT` parameter on the host. Typically, the default is 100, which is acceptable for character-oriented communications (for example, Telnet VT™ 100 emulation), but not good for record and block-oriented communications, as with Databridge or FTP (file transfer protocol). For Databridge communications, you can increase throughput by reducing the `BLOCKTIMEOUT` parameter to a value of 2.

If the Databridge Client system is on a different subnet from the mainframe, put it on the same subnet so that Ethernet packets can be larger. If you cannot put the Databridge Client on the same subnet as the mainframe, you can improve throughput by adjusting BLOCKSIZE on the host and TCP/IP Window Size on the Windows Server PC.

4.5.1 Running tcptest

During the initial setup, use the `tcptest` command to determine if the TCP/IP interface is operating properly. Before you run the `tcptest` command, you must define a data source. For more information, see the table in dbutility Commands. An example of the test is shown below:

```
E:\>dbutility tcptest demodb 111.222.33.444 5555 100 1000
11:49:10 Databridge Client version 7.0.0.000 [OCI/Oracle]
11:49:10 (C) Copyright 2021 Micro Focus or one of its affiliates.
11:49:14 Connecting to 111.222.33.444, port 5555
11:49:16 TCP_Test: len=100, count=1000
11:49:17 Bytes Processed 100.00 KB of DMSII data in 1.000 secs, throughput = 100.00 KB/sec
11:49:17 Bytes Received 112.00 KB in 1.000 secs, total throughput = 112.00 KB/sec
11:49:17 TCP/IP_time = 0.841 secs, (84.10% of total time)
11:49:17 TCP Test completed successfully
11:49:17 Client exit code: 0 - Successful
```

4.6 Populating the Databridge Data Tables

Before you populate the Databridge data tables, determine if you need to customize the character translation tables or not. If you do, modify the [EbcdictoAscii] section of the Client configuration file before you run either the `process` or `clone` command. For more information on character translation tables and modifying the configuration file, see [EbcdictoAscii] and Export or Import a Configuration File.

You can populate the Databridge data tables in the relational database using either of the following methods:

- dbutility `process`
- dbutility `clone`

The `process` and `clone` commands use the relational database bulk loader utility to populate the Databridge tables. *We recommend that you read one of the previous sections,* Oracle SQL*Loader Bulk Loader *or* Microsoft SQL Server BCP API and bcp utility *before you use the dbutility* `clone` *or dbutility* `process` *command.*

The `process` command is typically used to populate the data tables. The `clone` command is a special case of the `process` command that allows you to clone a small number of data sets without changing the values of the corresponding entries in the `active` column of the DATASETS Client control table.

4.7 Data Validation and Discard Files

While processing DMSII extract and update records, Databridge validates all numeric and alpha fields. Fields that contain NULL values (data with all high-bits set) usually are recognized as DMSII NULLS. In this section, the following types of data validation and discard files are described:

- Numeric data validation
- Alpha data validation
- Date validation
- Special handling of key items in discard files
- The handling of blank character data for key items in the Databridge Client for Oracle

4.7.1 Numeric Data Validation

Numeric data that contains illegal digits (for example, values other than 0 through 9, excluding the sign field for signed numbers) are flagged as bad. If the `da_options` column of the corresponding DATAITEMS control table entry has the DAOPT_Allow_Nulls bit (1) set, Databridge treats numeric items that have bad digits as NULL.

The configuration parameter `allow_nulls` defines the default value for this bit, which can be altered by user scripts. If the bit is zero, the NULL or bad numeric data is stored as either all 9s or all 0s based on the value of the configuration parameter, `null_digit_value` (default value is 9). For more information, see bracket_tabnames and null_digit_value.

4.7.2 Alpha Data Validation

With alpha data, bad characters are usually replaced with a question mark (?) instead of the whole field being set to NULL. The Client configuration file parameter `inhibit_ctrl_chars` determines whether or not control characters are to be treated as bad characters (the program treats a few control characters such as NUL, CR and LF as bad regardless of the value of this parameter). The Client configuration file parameter `inhibit_8_bit_data` determines whether or not 8-bit characters are to be treated as bad characters. The Client configuration parameter `convert_ctrl_char` (which is incompatible with `inhibit_ctrl_chars`) replaces control characters by spaces instead or question marks. For more information, see [Bulk_Loader].

The Client configuration file parameter `alpha_error_cutoff` determines the percentage of bad characters in an ALPHA field that are tolerated before the entire field is declared bad and treated as NULL.

If ALPHA data is stored as binary data, no alpha data validation is performed because no invalid values exist in binary data. See the DIOPT_Clone_as_Binary option in the `di_options` column of DMS_ITEMS.

NULL data is treated as NULL if the `da_options` column of the corresponding DATAITEMS control table entry has the DAOPT_Allow_Nulls bit (1) set. Otherwise, the NULL data is stored as blanks.

4.7.3 Date Validation

Whenever Databridge processes numeric or alpha items that are cloned as relational database date data types, it checks the validity of the data. Invalid dates are usually treated as NULL. The Databridge Client for Microsoft SQL Server stores bad or NULL dates as 1/1/1900, when the DAOPT_Allow_Nulls bit (1) in the `da_options` column of the corresponding DATAITEMS control table entry has not been set. The Databridge Client for Oracle uses the date 1/1/0001 instead. A numeric date of all 0s or all 9s is treated as NULL rather than an error. Similarly, an ALPHA date that is all blanks is treated as a NULL date.

4.7.4 Special Handling of Key Items in Discard Files

Because the stored procedures used during update processing use equality tests in the *where* clauses, key items (items that are used in the index for a table) can never be NULL. In relational databases, you cannot use equality tests for items that are NULL.

If a key item has a data error or it is NULL, Databridge places the entire record in a discard file named *tablename*.`bad` in the discards subdirectory. The syntax for discard file data is the calling sequence that would typically be used for the stored procedure that performs the update. Therefore, discarded records from both the data extraction and update phases are identical. Databridge preserves bad numeric digits and characters to help you better troubleshoot the problem.

> ✏️ **Note**
>
> During data extraction records discarded by the Client and those discarded by the bulk loader end up in different files in the **discards** folder (in the case of the SQL Server Client *tablename*.`bad` and `bcp`.*tablename*.`bad` respectively). Bulk loader discards are extremely rare as the Client catches all the data errors.

4.7.5 Handling Blank Character Data for Key Items (Databridge Client for Oracle)

The Databridge Client strips all trailing blanks when constructing SQL statements using **varchar** data. When an application reads the records back from the database, the access routines put back the trailing blanks, greatly reducing the storage requirements for the SQL statements and bulk loader data files.

In Oracle, **char** or **varchar** items that have a length of 0 are treated as NULL. If any of the key items used in `where` clauses are NULL, the corresponding update or delete SQL statements fail as mentioned above. To prevent the key item from becoming NULL, the Databridge Client for Oracle keeps the last blank of the item.

4.8 The Process Command

The `process` command is the main command of the Databridge Client. It populates and updates the tables for all data sets whose `active` column is 1 in the corresponding entries of the DATASETS Client control table. Since the `define` command initializes the `ds_mode` column, all the selected data sets are cloned the first time you run a `process` command.

> ✏️ **Note**
>
> If you do not select specific data sets in the data set global mapping customization script, the Databridge Client automatically clones all data sets except for remaps, the restart data set, and the global data set. This operation may take a very long time and require a lot of disk space.

You can schedule the `process` command to update the Databridge data tables. The schedule becomes effective after you run the `process` command for the first time. For more information, see Scheduling dbutility Updates.

To populate the Databridge data tables in the relational database via the **dbutility** `process` command, you must first make sure that the current directory is set to the working directory you created for this data source. This must be the same working directory you used when you executed a `generate` command for this data source; otherwise, the Databridge Client cannot locate the scripts to create and populate the Databridge data tables.

4.8.1 Cloning a DMSII Database

Use the following procedure to clone a DMSII database via the `process` command.

To run the `process` **command**

1. Make sure that Databridge Server is running. If it is not, the Databridge Client will try to connect to the host and eventually time out.

2. Make sure that your signon parameters are configured appropriately.

3. If you plan to use the [EbcdictoAscii] section to customize character translation or any other parameters in the `dbridge.cfg` file, set them before you continue. In particular, make sure you have appropriate settings for the following parameters. (For information on setting these parameters, see Appendix C: Client Configuration.)

4. The following parameters affect the way a **process** or **clone** command operates. You can change these parameters before running these command without running a **redefine** command as they do not change anything in the control tables:

```
alpha_error_cutoff
aux_stmts
batch_job_period
century_break
commit_absn_inc
commit_idle_database
commit_longtrans
commit_time_inc
commit_txn_inc
commit_update_inc
controlled_execution (dbutility only)
convert_reversals
correct_bad_days
dbe_dflt_origin
defer_fixup_phase
discard_data_errors
display_bad_data
eatran_dll_name
enable_af_stats
enable_doc_records
enable_encryption
enable_minimized_col
enable_optimized_sql
engine_workers
error_display_limits
inhibit_8_bit_data
inhibit_console
inhibit_ctrl_chars
inhibit_drop_history
inhibit_init_values
keep_undigits
linc_century_base
masking_parameter (SQL Server only -- run a generate command when changed)
max_clone_count (Only meaning when using the -s option)
max_discards
max_retry_secs
max_srv_idle_time
max_temp_storage (Windows only)
max_wait_secs
n_dmsii_buffers
n_update_threads
null_datetime_value (SQL Server only)
null_datetime2_value  (SQL Server only)
null_digit_value
numeric_date_format
preserve_deletes
set_blanks_to_null
set_lincday0_to_null
show_perf_stats
show_statistics
show_table_stat
sql_exec_timeout
sql_heart_beat
statistics_increment
stop_after_fixups
stop_after_gc_reorg
stop_after_given_afn (dbutility only)
stop_on_dbe_mode_chg
track_vfds_nolinks
use_ctrl_tab_sp
use_dbwait
use_latest_si
```

5. Enter the following command:

```
dbutility [signon_options misc_options] process datasource
```

Where	Is
signon_options	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** [-U *userid*] [-P *password*] [-D *database*] **SQL Server:** [-U *userid*] [-P *password*] [-W] [-O *ODBCdatasource*]
misc_options	See table in the next section titled "Process Command Options".
datasource	The name of the data source specified in the DBServer control file (DATA/SERVER/CONTROL) or via Enterprise Server.

If the Databridge Client connects to DBServer, it selects all the data sets whose corresponding `active` columns have a value of 1 in the DATASETS table. Next, the Databridge Client requests that DBServer clone all the selected data sets. At the end of the data extraction phase, the Databridge Client issues another request to start sending the fixup records followed by updates. The processing of audit files continues until there are no more audit files available.

If the Databridge Client connects to DBEnterprise, DBEnterprise supplies the data, either by reading the DMSII data set directly (direct disk) or by issuing a request to DBServer to have Databridge Engine read a block of data from a specific region of the disk (remote regions). DBEnterprise then processes this block of data. Since Databridge Engine is only reading raw data in remote regions mode and does not do any processing of this data, this mode of operations is less expensive than having the Client connect directly to DBServer in term mainframe resource utilization. Direct disk mode offers the biggest mainframe resource savings, as DBServer is not involved in reading data sets.

In the case of audit file data, DBEnterprise either reads the data from its caches (if configured), or it reads the audit file directly by issuing a request to DBServer to have Databridge Engine read a block of data from a specific region of the disk.

After the cloning of the DMSII database completes, the tables in the relational database will contain the same data as DMSII. At this point you can execute SQL queries to view the data and make sure that all the tables have been populated. When you are ready to update the relational database with changes made to the DMSII database, see Updating the Databridge Data Tables.

5. Process Command Options

Command-line options related to the `process` command are as follows:

Option	Description
`-c`	Toggles the `defer_fixup_phase` configuration file parameter.
`-d`	Enables default tracing.
`-f` filename	Specifies a configuration file other than the default dbridge.cfg file in the working directory.
`-l`	(SQL Server only) forces the Client to use the **bcp** utility instead of the BCP API.
`-s`	Tells the Client not to use the bulk loader.
`-t`	Enables selective tracing.
`-w`	Toggles the `use_dbwait` parameter in dbridge.cfg
`-K`	Inhibits the audit file removal WFL from being run on the host.
`-L`	Forces the Client to use a new log file for this run.
`-T`	Forces the Client to use a new trace file for this run, if tracing is enabled.

For information on the command-line options, see dbutility Command-Line Options.

5.0.1 Terminate Cloning

Use the following procedures to stop the cloning process before it is complete.

To terminate cloning

- When using the service, from the Administrative Console, use the `Abort` command in the **Run** menu for the data source.
- When using **dbutility**, use the `QUIT NOW` command.

To terminate processing during the fixup and tracking phases

- When using the service, from the Administrative Console, use the `Stop` command in the **Run** menu for the data source. The Databridge Client will stop at the next quiet point.
- When using **dbutility**, use the `QUIT` command (or the SIGTERM (15) signal on UNIX).

> ✏ **Note**
>
> If you issue a `QUIT` command or send a SIGTERM signal to the program during the data extraction phase, the Databridge Client stops only when the fixup phase begins.

When you terminate the Client during the fixup phase or during updates, the `process` command restarts from the last commit point. If you terminate the Client during the data extraction phase, only the data sets that have successfully completed the data extraction phase (ds_mode = 1) are recoverable. You can resume the process by running another `process` command.

In the unlikely event that all of these commands fail to terminate the Client, press Ctrl+C or kill the run.

5.0.2 Tracking the State of Data Sets

The DATASETS Client control table keeps track of the state of data sets. State information consists of the `ds_mode` value and the DMSII audit file location from which subsequent updates should be processed. The audit file location includes the AFN, the ABSN, the segment and index in the audit files, and the audit file time stamp. These values, which are collectively referred to as the stateinfo, are stored in the `audit_filenum`, `audit_block`, `audit_seg`, `audit_inx`, and `audit_time6` columns of the DATASETS Client control table. The column `audit_ts` contains a date/time value, which corresponds to the `audit_time6` data, which is binary and represents a DMSII TIME(6) value. This last column is not part of the stateinfo; it is there because knowing the audit time stamp value can sometimes be very useful.

Each subsequent time you run a `process` command, the Databridge Client passes the stateinfo and the mode of each data set to the Databridge Engine. The Engine uses this information to determine whether data sets should be cloned and the starting location in the audit trail. From that starting location, the Databridge Engine begins processing updates to the DMSII database. Every time a transaction group ends, the Databridge Client updates the stateinfo for the data sets in the DATASETS Client control table. At the end of the `process` command, the location of the last quiet point in the audit trail is saved in the DATASETS Client control table. This is the starting point for the next Client run (`process` command).

If the `in_sync` column of a data set has a value of 1, its stateinfo columns may be out-of-date. You can determine if it is current by checking the Global_Dataset entry in the DATASETS control table. For more information, see Optimizing State Information Passing.

5.0.3 ds_mode values

The following values are defined for the `ds_mode` column of the DATASETS Client control table:

Value	Name	Description
0	CLONE	Initial state of `ds_mode` before the data set is cloned.
1	FIXUP	Data extraction completed, fixup processing not completed.
2	NORMAL	Normal update tracking mode.
10	BCP-FAILURE	The bulk loading of the table failed. Further processing is not possible until the problem is resolved.
11	PRE-FIXUP	Data extraction completed, fixup processing cannot be done due to index creation errors or lack of an index.
12	INVALID-AA	AA Values invalidated by a DMSII garbage collection reorganization.
31	NEEDREORG	The data set needs to be reorganized and the `redefine` command has created scripts to make the relational database table match the new layout that resulted from the reorganization of the DMSII data set. You must run the `reorganize` command in order to run the reorganization scripts created by the `redefine` command.
33	REORGFAILED	The data set needs to be reorganized and the scripts created by the `redefine` command for this data set failed when the `reorganize` command was run. In this case, you must manually alter the table or re-clone it.

In the case of DMSII reorganizations, the `status_bits` column in the DATASETS table is used instead. The Databridge Client leaves the `ds_mode` column unchanged and sets the DS_Needs_Redefining bit (8) of the `status_bits` column of the DATASETS Client control table.

Following the initialization (purge) of a data set, the Client is notified of the purge. The Client drops the tables for the data set and recreates them. The `ds_mode` of the data set is set to 2 and the index for the empty tables are created. This enables the normal update processing to repopulate the tables. .

5.1 The Clone Command

From a command line, use the `clone` command to select the data sets you want to clone. You can use this command for cloning or recloning. To update the resulting Databridge data tables, you must use the `process` command. The `process` command is generally recommended instead the `clone` command, unless you want to deal only with a specific data set without processing updates at the same time.

The `clone` command is basically a `process` command, except that it forces the data sets specified on the command line to have their `ds_mode` set to 0 and it treats all data sets not specified on the command line as if their `active` column is 0.

To populate the Databridge data tables in the relational database via the `clone` command, first make sure that the working directory is set to the directory you created for this data source. This must be the same directory as the working directory used when you executed a `generate` command for this data source; otherwise, the Databridge Client *cannot* locate the scripts to load the Databridge data tables.

5.1.1 Cloning Specific Data Sets

Using the **clone** command follow the exact same procedure described in the section on "Cloning a DMSII Database". type the following command line:

```
dbutility [signon_opts misc_opts] clone source dataset1 [... datasetn]
```

Where	Is
signon_opts	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** [-U *userid*] [-P *password*] [-D *database*] **SQL Server:** [-U *userid*] [-P *password*] [-W] [-O *ODBCdatasource*]
misc_options	See table in the next section titled "Clone Command Options".
source	The name of the source specified in the DBServer control file or by Enterprise Server.
`dataset1 [... datasetn]`	The names of the data sets you want to clone. You must specify at least one data set name. If you specify more than one data set name, separate the names with spaces. Note the following: • The data set names you enter must match the names of the data sets as they are defined in the DASDL for the *DMSII database*. Databridge Client automatically converts them to uppercase for you. For example, if the data set you want to clone is named ORD-DETAIL, you must type ORD-DETAIL or `ord-detail` . You must use a hyphen (-), not an underscore (_). • The exact data set names are listed in the DATASETS Client control table. • If a DMSII data set is a relational database reserved word, enter it normally without quotes or any other delimiter. • The `active` column of the selected data sets must be set to 1. Otherwise, an error appears when you specify the data set on the command line.

> ⚠️ **Warning**
>
> If for some reason the `clone` command abends, do not rerun it before you determine whether or not some of the data sets completed the data extraction phase and are recoverable. *Rerunning the* `clone` *command starts the cloning operations from scratch.*

If you need to re-clone the entire database you can use the `process` command with the `-Y` option. When using **dbutility** you need to add the text "all_datasets" as an argument to the `-Y` option.

6. Clone Command options

Command-line options related to the `clone` command are as follows:

Option	Description
`-c`	Toggles the `defer_fixup_phase` configuration file parameter . When you use this option, the dbutility `clone` does not enter the fixup phase at the end of data extraction. Instead of issuing request to the Databridge Server to initiate the fixup phase, the Databridge Client terminates. The `ds_mode` values of all cloned data sets remain set to 1 with all of the necessary stateinfo stored in the Client control tables (for example, `audit_filenum`, `audit_block`, and `host_info`). The next `process` command then picks up where the `clone` command left off.
`-d`	Enables default tracing.
`-f` filename	Specifies a configuration file other than the default dbridge.cfg file in the working directory.
`-l`	(SQL Server only) forces the Client to use the **bcp** utility instead of the BCP API.
`-s`	Tells the Client not to use the bulk loader.
`-t`	Enables selective tracing.
`-x`	Reverses the meaning of the data set list for the clone command, as follows: Without the -x option, the Databridge Client clones the data sets listed on the command line. With the -x option, the Databridge Client clones all active data sets except those listed on the command line.
`-L`	Forces the Client to use a new log file for this run.
`-T`	Forces the Client to use a new trace file for this run, if tracing is enabled.

For information on the command-line options, see dbutility Command-Line Options.

6.1 Configuring for Optimal Performance

Several configuration file parameters have a very visible effect on the performance of the Databridge Client. Databridge Clients operate efficiently with the following *default* configuration file parameters:

- `max_temp_storage` (Windows only)
- `aux_stmts`
- `optimize_updates`

This section discusses these parameters and other factors that can make Databridge Client run more efficiently.

6.1.1 Overlapped Bulk Loader Operations for Windows

> ✏️ **Note**
>
> The SQL Server Client uses the **bcp** utility by default in this version, as using the BCP API did not prove to be very reliable.

The Windows Clients use a separate thread to bulk load tables using SQL*Loader or **bcp** during the data extraction phase of data set cloning. The Client creates a set temporary files for the tables that hold the data for the records received from the Databridge Engine. When the total temporary file size reaches half of the value specified by the parameter `max_temp_storage`, all the tables that have temporary files get placed on the bcp thread's work queue. The bcp thread processes the list of tables and does the loads sequentially. While this is happening the Client creates a new set of temporary files for tables that that get additional records and continue processing DMSII data until the total temporary file size reaches the above mentioned threshold. If the bcp thread is done loading the table involved, processing continues. However, if the bcp is not finished loading the data for the table Client stops processing DMSII data until it is signalled that the loading of the table completed.

Starting with the 6.6 Client multi-threaded updates were extended to also include extracts, which improves performance as the load is distributed among the update threads. This leads to better resource utilization, as multiple CPUs are working concurrently on processing the data for the records that are received from the Databridge Engine.

> ✏️ **Note**
>
> Configuration file parameters for increasing bulk loader speed are listed with the related bulk loader utility in the sections Oracle SQL*Loader Bulk Loader and Microsoft SQL Server BCP API and bcp utility. See a complete list in Appendix C's section on [Bulk_Loader]

Adjust the `max_temp_storage` value to determine what works best for your site. Setting this parameter too high tends to reduce the benefits of using multiple threads to launch bulk loader operations. Conversely, setting this parameter too low tends to increase overhead, particularly when the record sizes are large, by firing off too many bulk loader operations. A value in the range 400M (default) and 1G seems to work best.

> ⚠️ **Caution**
>
> Do not set this parameter to 0, or *the program will bulk load tables in one step, use a huge amount of temporary storage, and eliminate all overlapped processing*.

For details about `max_temp_storage` parameter, see [Bulk_Loader].

Using the BCP API eliminates the use of temporary files and allows for overlapped data extraction operations. Combined with optimized Client code for data extraction, clone speeds are much faster than before (especially in cases where the bottleneck is the CPU). The benefits of this mode of operation are much more dramatic when using a system with multiple CPUs.

6.1.2 Overlapped Index Creation

The Databridge Clients use a separate thread to execute the index creation scripts for tables after the data extraction for the table is completed. This allows lengthy index creation operations to be overlapped with the bulk loading of tables and has a noticeable impact on speeding up the data extraction process when many data sets are involved.

6.1.3 Optimizing State Information Passing

The Databridge Client optimizes the process of updating the stateinfo in the DATASETS Client control table, which is identical for all data sets that are in update tracking mode (ds_mode=2). Instead of updating every data set each time the stateinfo is updated by Databridge Engine prior to a COMMIT, it stores the common information in a dummy data set in the DATASETS table named `Global_DataSet`. When the Databridge Client is finished processing updates, the stateinfo in the `Global_DataSet` entry in the DATASETS Client control table is copied to all data sets that need to be updated with the stateinfo. The `Global_DataSet` row is

stored in the DATASETS Client control table along with the `in_sync` column that keeps track of stateinfo synchronization between updated data sets and the `Global_DataSet`. This significantly reduces the number of SQL update statements for the DATASETS Client control table when you have a large number of data sets.

To keep the DATASETS table current, particularly when READ ACTIVE AUDIT is set to TRUE in the Databridge Engine Control File, the Databridge Client copies the stateinfo in the `Global_DataSet` entry to all data sets whose `in_sync` column is 1 after an audit file switch.

When the Client is restarted after it abends, it detects the fact that the data sets are out of sync. The Client corrects this situation by copying the global stateinfo to all data sets whose `in_sync column` is 1 and setting all of the `in_sync` columns to 0.

6.1.4 Multiple Statements and Pre-parsed SQL Statements

The `aux_stmts` parameter applies only to the Databridge Client during update processing (not cloning).

The `aux_stmts` parameter defines the maximum number of ODBC or OCI auxiliary statements that can be assigned to SQL statements. Using auxiliary statements allows SQL statements to be parsed once and executed multiple times, as long as the auxiliary statement is not reassigned to hold another SQL statement.

In general, higher values for `aux_stmts` result in faster update processing times at the expense of more memory usage. If you have the memory to spare, increase this parameter as needed. The optimum setting for this parameter will depend on the nature of your data. We recommended using a value of 100 or higher, which allows the SQL statements to be re-executed without having to reparse them and re-bind their host variables.

For details, see aux_stmts.

6.1.5 Reducing the Number of Updates to Data Tables

If your most frequently updated data sets have a significant number of items with OCCURS clauses that are not flattened, you may want to set the DSOPT_Use_bi_ai bit in the `ds_options` column of the corresponding DATASETS entries. The configuration file parameter `optimize_updates` causes the `define` command to set this bit for all data sets that have active items with unflattened OCCURS clauses.

> ✏️ **Note**
>
> To set this parameter from the Client Configurator by checking the checkbox "Optimize SQL updates" in the **Customizing > Advanced** page of the Client Configuration dialog.

If the ratio of SQL rows to DMSII records is five or more during update processing, setting this parameter to True will likely improve performance. Note that this increases the TCP/IP and CPU overhead. If the ratio of SQL rows to DMSII records is low, you won't see any significant reduction in SQL overhead, which can hinder performance.

For best results, set the DSOPT_Use_bi_ai bit only for data sets that have a high ratio of SQL rows to DMSII records. For example, a data set that has only one item with an OCCURS 2 TIMES clause is a poor candidate for the DSOPT_Use_bi_ai bit (SQL rows/DMSII records = 3). Conversely, a data set that has 3 items with OCCURS 12 TIMES clauses is a good candidate to use the DSOPT_Use_bi_ai bit (SQL row/DMSII records = 37).

For details, see optimize_updates.

6.1.6 Commit Frequency

The size of transactions in the relational database can have a noticeable impact on performance during update processing. Large transaction sizes will result in the Client getting locked out during the commit when it tries to update a table. Conversely small transactions can add overhead by constantly committing a small number of updates. Some site use very low commit frequency parameters to get lower lag times during on-line periods (lag time is defined as the elapsed time between the time when an update is applied to the relational database and the time it was applied to DMSII). This type of commit frequency might not work well

during the processing of audit files created by batch jobs. To solve this problem we implemented a second set of commit frequency parameters that are enabled by setting the `batch_job_period` parameter to define the time period during which batch jobs run, for example `batch_job_period` = 22:00, 01:00. This example specifies that batch period spans from 11:00 pm to 1 am. When the Client detects that the audit timestamp crosses these boundaries it switches to using the appropriate values for the commit frequency parameters. On-line periods use the first set of values, while batch period use the second set of values. You can override these settings from the Administrative Console, however these settings will be overridden by the configured values at the next period switch.

6.1.7 Other Considerations

A few other factors that can significantly affect performance include:

- The number of CPUs (at least four are recommended)

- The type of CPU

- The amount of memory on your Client machine

- The type of disks you use. Redundant array of independent disks (RAID) or striped disks are recommended. During data extraction, do not use the same physical disks for temporary files and the database files. Ideally, use RAID for the database files and a separate disk for the Databridge Client files (bulk loader temporary files, scripts files, log files, and so on).

- The condition of your database

6.2 Tips for Efficient Cloning

When you first run the `define` command to populate the Client control tables, you will notice that most DMSII data sets are set to be cloned. Although you can accept the default data sets and their items for cloning, note the following:

- Cloning an entire DMSII database can take *several hours or more*. Most sites do not clone the entire DMSII database.

- The cloning of the DMSII restart data set is automatically disabled. The restart data set is related to restarting the DMSII database only.

- If you clone virtual data sets, do not disable the cloning of the data sets from which the virtual data set is derived. (Virtual data sets have a value of 17 in the `subtype` column of the DATASETS table.) Virtual data sets are created on the host and are explained in the Databridge Programmer's Reference.

- Make sure that you have enough disk space on the relational database server for the DMSII data. If there is not enough room, the cloning process stops. In addition to the space required for DMSII data, you must have some additional file space for the temporary files used by the bulk loader utilities (bcp for Microsoft SQL Server; SQL*Loader for Oracle).

> ✏️ **Note**
>
> The bulk loader temporary files should not be on the same disk as the relational database.

- You do not have to clone all of the data sets at one time. One approach is to clone the most essential data sets and then determine how much space is still available.

- If you do not want to clone secondary tables (those tables generated from a data set), you have two choices. In either case, the primary table is still generated, while the resulting secondary tables are not.
  - Set the value of the `active` column (for that table) in the corresponding DATATABLES Client control table entry to 0 (`script.user_define` .*primary_tablename*).
  - Set the value of the `active` column in the corresponding DMS_ITEMS Client control table entry to 0 for an item with an OCCURS clause (`script.user_layout` .*primary_tablename*).
  - Flatten the OCCURS in either the primary or the secondary table.

- If the `active` columns for all tables related to a data set are 0, Databridge sets the `active` column of the corresponding entry in the DATASETS table to 0.

- The cloning of DMSII remaps is automatically disabled because the remaps are just different views of the base data sets; the Databridge Client assumes that the base data set will be cloned. We recommend, therefore, that you clone the base data set and then set up a view in the relational database to achieve the same result as the REMAP.

  If you do want to clone a remap of a data set instead of the base data set, you can do so by changing the values of the `active` columns of the data sets in the DATASETS Client control table. You can identify remaps by their base structure number (base_strnum). For example, if structure numbers (strnum) 11, 121, and 227 are remaps of structure number 10, the base structure number for structures 11, 121, and 227 is 10.

  For data sets that are not remaps, the `strnum` and `base_strnum` columns are equal. If you do not want to clone anything related to a particular data set, set the value of the `active` column (for that data set) in the corresponding DATASETS Client control table entry to 0. No tables (primary or secondary) are generated from this data set.

  For more information about REMAPS, see the next section.

6.3 REMAPS

If the base structure of a REMAP is not visible to the Client (due to GenFormat filtering or the use of a logical DMSII database) the Client will set the `active` column of the first REMAP to 1, and clone it in place of the base structure. If the base structure of an embedded data set is not visible to the Client and it has a REMAP, the Client will set the `active` column of the REMAP to 1 and use it as the parent instead.

> ✏️ **Note**
>
> If you are using the Administrative Console or Client Configurator and you want to change the `active` column of a data set that is 0, you should be aware that the data sets view has a filter.
>
> The picture below shows this for the Administrative Console. To access it, click on the Filters button above the data sets. To see the data sets that have their `active` column set to 0, you need to click on **Inactive** and then on **Apply**. This causes data sets with their `active` column set 0 to be included in the data set view. The items in question will have "Inactive" in their Status column to indicate that their `active` columns are 0. You can change the `active` column by clicking on the gear icon to the left of the data set and moving the slider labeled **Active** to the on position (dark blue).

7. Updating the Relational Database

This chapter covers updating the Databridge Client data tables in the relational database with changes made to the DMSII database. In addition, it explains DMSII reorganizations and how they update the Databridge Client data tables.

7.1 Updating the Databridge Data Tables

Updating is the process of applying the DMSII database changes to the Databridge data tables in the relational database by sending only the changes, not all of the data, to the Databridge data tables.

You can update the Databridge data tables after they have been cloned as long as they meet the following requirements:

- Each Databridge data table you want to update has a unique index. If a table you want to update does not have a unique index, see Creating Indexes for Tables.
- The DMSII database has not been reorganized or rolled back. If the DMSII database has been reorganized, see DMSII Reorganizations and Rollbacks.

You can update the Databridge Client data tables by running a `process` command each time you want the update to occur, or you can schedule a `process` command to run at fixed times or a fixed amount of time after the run finishes. How often you update the Databridge tables depends on the following:

- How current you want the data to be. For time-critical information, you may want to update the Databridge data tables several times a day.

> ✏️ **Note**
>
> If you do *not* update the relational database often, it might be more effective for you to re-clone the DMSII data sets rather than update them. For example, if you are interested in weekend information only, and *several* audit files have been closed (and possibly moved to tape) during that time, recloning may be faster than updating.

- How often audit files are available on the host. When an audit file is not available, the Databridge Engine temporarily stops processing until the audit file is available. The Databridge Engine can access the active DMSII audit file when the DBEngine control file parameter READ ACTIVE AUDIT is set to true.
- How often closed audit files are available on the host. In some cases, a closed audit file is not available because it has been copied to tape. In this instance, a host operator must mount the tape before the closed audit file can be made available to Databridge.

7.2 Performing Updates Without Using Stored Procedures

Version 6.6 of the Databridge software introduced a new way of updating tables that does not use stored procedures and is more efficient. This feature can be controlled globally by setting the configuration parameter `use_stored_procs` to False, see use_stored_procs.

This parameter makes the `process` and `clone` commands generate the actual SQL command instead of using a stored procedure call to perform an update. The Client still uses host variables, as was the case with stored procedures calls. Executing the SQL directly eliminates some overhead and makes processing the update faster. You can also control this on a data set by data set basis as described in use_stored_procs.

> ✏️ **Note**
>
> Setting configuration parameter `use_stored_procs` to False, does not have any effect, unless you run a `redefine` command with -R option (redefine all) to propagate this setting to the `ds_options` columns of the data sets. The `redefine` command with return an exit code of 2033 (Reorg command required). You need run the `reorganize` command to complete task. This command will generate new scripts for the tables and refresh all the data sets, which results in the dropping of all the stored procedures, which will no longer be used. If you revert to using stored procedures, the procedure is the same, in this case the `reorganize` command will recreate the stored procedures.

7.3 Scheduling Updates

The **dbutility** `process` command has a built-in scheduling mechanism that allows the run to hibernate and resume at the next scheduled time. When the Databridge Server sends the Client an end-of-audit-reached status, dbutility normally terminates. However, if you enable scheduling, the Databridge Client disconnects from the server and the database and hibernates until the next scheduled `process` command, when it reconnects to the server and the database. This scheduling mechanism only works after you run dbutility for the initial clone of the DMSII database. If the Client crashes or the power fails, scheduling will fail. Service-based scheduling has none of these shortcomings, as the scheduling task is taken over by the Client Manager service.

If you use the Client Manager service, it takes over this functionality. When a **DBClient** run terminates, the service determines when the next `process` command should be run and starts it when that time arrives. The advantage of service-based scheduling is that it is immune from system failures, as the service automatically gets restarted when the system is rebooted. For details about service-based scheduling, see the Databridge Client Console **Help**.

To schedule updates

1. Uncomment the scheduling parameters in the Databridge Client configuration file. Scheduling parameters are listed under the [Scheduling] header in the configuration file.

2. Select one of the following scheduling methods:

    - Daily scheduling

    - Fixed-delay scheduling

3. Enter appropriate values for the following scheduling parameters, depending on whether you are using daily scheduling or fixed-delay scheduling. Each parameter is explained in [Scheduling].

```
[Scheduling]
;
;dbutility process command only
;
;daily                = 08:00, 12:00, 17:00, 24:00
;exit_on_error        = false
;sched_delay_secs     = 600
;sched_minwait_secs   = 3600
;sched_retry_secs     = 3600
;blackout_period      = 00:00, 02:00
```

As long as the `process` command completes successfully, dbutility becomes inactive (sleep) until the next scheduled time. If the scheduled dbutility `process` command is successful, the following message appears:

```
Next update for DataSource datasourcename will run at hh:mm (delay = nn secs)
```

Scheduling of updates will continue until any of the following occurs:

- You reboot the Databridge Client machine or end the Databridge Client session

- You enter a `SCHED OFF` console command when dbutility is processing updates

- A DMSII reorganization (other than a garbage collection)

> ✏️ **Note**
>
> If you must stop the dbutility program, we recommend that you use the `QUIT` command to exit at the next quiet point. If the Client is waiting for the server to send it updates when none are available and the `use_dbwait` configuration file parameter is set to True, you can use the `QUIT NOW` command, which resets the connection to the server and terminates the Client run. If needed, you can also press Ctrl+C to terminate a session while dbutility is processing updates; however, we do not recommend this approach.

7.3.1 Scheduling Examples

Daily Schedule Example

The following example uses the daily scheduling method. In this example, the Databridge Client runs only twice a day -- once midway through the business day and once at the end of the business day. If the `process` command fails, the Databridge Client waits 10 minutes before retrying.

```
[scheduling]
daily           = 12:00, 17:00   ; run the process at noon and 5PM
sched_retry_secs = 600           ; retry in 10 minutes after a failure
```

Fixed-Delay Example

The following example uses the fixed-delay scheduling method. In this example, the Databridge Client runs the `process` command 4 hours (240 minutes) after the run finishes. If the `process` command fails, the Databridge Client retries every 30 minutes.

```
[scheduling]
sched_delays_secs = 14400
sched_retry_secs  = 1800
```

7.4 Scheduling Blackout Periods

You can schedule blackout periods during which the Client suspends all processing and updates to allow for routine maintenance. To use this feature with the service-controlled Client, you can set the Blackout Period value from the Client Console by clicking on the **Configure** item in the data source **Settings** menu.

7.5 Unscheduled Updating

Use this procedure when you want to run **dbutility** `process` independent of scheduling.

- Make sure that the Databridge Server is running. If it is not, the Databridge Client will try to connect to the server and eventually time out.

- Make sure that your signon parameters are set appropriately.

- If the [EbcdictoAscii] section of the configuration file (to customize character translation) has changed since the initial clone, your data may not be consistent. You might need to re-clone.

- Make sure that the current directory is the one you created for this data source. This ensures that Databridge Client can locate the scripts. (Scripts are only required during an update if there's a purged data set.)

- Enter the following:

  dbutility [signon_options misc_options] process datasource

Option	Description
signon_options userid	For each Databridge Client type, the following command-line options specify the relational database signon parameters: **Oracle:** [-U] [-P *password*] [-D *database*] **SQL Server:** [-U *userid*] [-P *password*] [-W] [-O *ODBCdatasource*]
misc_options	See table in the next section titled "Process Command Options".
datasource	The name of the data source specified in the DBServer control file or by Enterprise Server.

When you run a `process` command to update the Databridge tables in the relational database, the following occurs:

- All modified records are overwritten with their new values. If the target record is not found in the table, the Databridge Client adds the record to the table instead.

- All deleted records are deleted.

- All added records are inserted into to the data tables. If the target record is already in the table, the Databridge Client modifies the record in the table instead.

8. Process Command Options

Command-line options related to the `process` command are as follows:

Option	Description
`-d`	Enables default tracing.
`-f filename`	Specifies a configuration file other than the default dbridge.cfg file in the working directory.
`-t`	Enables selective tracing.
`-w`	Toggles the `use_dbwait` parameter in dbridge.cfg
`-K`	Inhibits the audit file removal WFL from being run on the host.
`-L`	Forces the Client to use a new log file for this run.
`-T`	Forces the Client to use a new trace file for this run, if tracing is enabled.

For information on the command-line options, see dbutility Command-Line Options.

8.1 Anomalies That Can Occur In Updates

When the Databridge Client updates the relational database, the following anomalies can occur:

Last quiet point (QPT) in an audit file	When processing an update transaction group since the last quiet point, Databridge Engine does the following when it reaches the end of the last available audit file: • Aborts the current transaction group so that the updates are rolled back. These are not duplicate updates, but updates that could not be committed. These updates will be reapplied the next time you run a `process` command. • Sends the Databridge Client a status indicating that the transaction group was rolled back. Upon receiving this status, the Databridge Client does not display any messages.
Host application rolls back changes (This is a partial DMSII rollback, not to be confused with a total DMSII rollback.)	If a host application encounters an error condition while updating the DMSII database, it rolls back all of the changes it made. In this case, Databridge Engine aborts the updates when it finds the aborted transaction indication in the audit file. Databridge Engine handles the situation in one of two ways based on the setting of the DBEngine control file parameter CONVERT REVERSALS TO UPDATES: 1. If `CONVERT REVERSALS TO UPDATES` is FALSE (the default setting), Databridge Engine sends an abort transaction status to the Databridge Client and then reprocesses the transaction group, excluding any updates by the program(s) that rolled back its updates. In this case, none of the updates in the aborted transaction are applied to the data tables. 2. If `CONVERT REVERSALS TO UPDATES` is TRUE, Databridge Engine will continue to process the audit file, converting the items marked as reversals to normal updates, in a manner similar to the method employed by DMSII. (DMSII aborts transactions by reversing the updates previously done. Thus a CREATE will be reversed to a DELETE, a DELETE reversed to CREATE, MODIFY to a MODIFY using the Before Image). All updates, including those that were aborted and their reversals, are applied to the data tables.

9. DMSII Reorganization and Rollbacks

DMSII Reorganization and Rollbacks

This section lists changes that can occur to the DMSII database, how those changes affect the Databridge Client control and data tables, and how to handle them on the Client database. For instructions on handling a DMSII reorganization on the host, see Prepare for a *DMSII Reorganization* in Chapter 10 of the *Databridge Host Administrator's Guide*.

9.1 Initializations

A DMSII initialization occurs when a DMSII data set is purged of its records. When a data set is initialized, Databridge Engine sends the Databridge Client a stateinfo record with a mode value of 4. The Databridge Client performs the actions described below after displaying the following message:

```
DataSet name[/rectype] has been purged
```

The Client drops all of the tables belonging to this data set and re-creates them, effectively purging the tables of all records. When Databridge Engine is done sending stateinfo records, it sends a status of DBM_PURGE(21), causing the Client to display the following message:

```
DataSets purged by Databridge Engine
```

The normal update processing will repopulate them.

> ✏️ **Note**
>
> An initialization does not change the data set format level in the DMSII database.

9.2 Reorganizations

Although there are three types of DMSII database reorganizations (record format conversion, file format conversions, and garbage collection reorganizations), the *types* of reorganizations are not as important as whether the reorganization changes record *layouts* or record *locations*, as follows:

- DMSII record format conversions change record *layouts*. When a data set is affected by a record format conversion, parallel changes must be applied to the Client database. See Managing DMSII Changes to Record Layout.
- DMSII file format conversions and garbage collection reorganizations change record *locations*. Only certain data sets require recloning in this case. See DMSII Changes to Record Locations.

> ✏️ **Note**
>
> Filler substitutions are handled the same as a record format reorganization. In a filler substitution, there is a change to the `item count` column in record for the data set in the DATASETS table.

9.2.1 Managing DMSII Changes to Record Layout

Use this procedure if a DMSII reorganization changes the layout of records. DMSII record layouts are changed in the following circumstances:

- Record format conversion (also called structural reorganization in this section)
- Filler substitutions

When Databridge Engine notifies the Databridge Client that the layout of records have changed, the Databridge Client returns a message for each reorganized data set and then prompts you to run a `redefine` command followed by a `reorganize` command. It then returns an `exit_status` value of 2 (DMSII reorganization). The `redefine` command can determine whether the layout for the data tables have been affected by the DMSII layout change and if the affected data sets need to be re-cloned. (For more information about the `redefine` command does, see About the redefine Command.)

To run the redefine command

1. If the DMSII changes are extensive or complex, we recommend that you back up the relational database before proceeding.

2. If you use the Administrative Console's **Customize** command to customize the table layouts, skip steps 3 through 5 and run the **Customize** command instead. It will perform the same actions as the `redefine` command, but will also allow you to make customizations for the data sets affected by the reorganization.

3. Modify user scripts as required by the DMSII layout changes.

4. Run the `redefine` command as follows:

   ```
   dbutility redefine datasource
   ```

   > 🔥 **Important**
   >
   > If any changes caused by the reorganization are not supported, the `redefine` command does not create the reorganization scripts. Instead, it sets the `ds_mode` column of the corresponding data set to 0, which forces the data set to be re-cloned. If the changes caused by the reorganization are allowed, the `redefine` command sets `ds_mode` to 31.

5. If the `redefine` command results in errors because the user scripts were improperly updated, run a `reload` command using the unload file automatically created by the redefine command. This file is named "*datasource*.reorg_*nnn*.cct", where *datasource* is the data source name (in uppercase) and *nnn* is the old update level of the database. This restores the control tables to the state they were in before the `redefine` command was run. See The Reload Command. Correct the user scripts and rerun the `redefine` command until no errors result.

6. Examine the reorg scripts created by the `redefine` command (or the **Customize** command) to make sure they are reasonable before proceeding any further. These scripts are created in the working directory and have names of the form "script.reorg_*nnn.tablename*", where *nnn* is the old update level of the DMSII database. If you see SQL statements that are likely to take a very long time to execute consider restoring the control tables and setting the `use_internal_clone` option or the corresponding `ds_options` bit for the data set. This will use a "select into" (CTAS in Oracle) to copy the old table (that is first renamed) to recreate the table with the needed changes while preserving the original data. The major time consuming operations are setting the initial values for added columns and ALTER commands that change the data type of columns.
   Doing this for a table with several million records can take a long time. The internal clone operate at bulk loader speed and is subject to the same rules as the bulk loader.

7. (This step automatically executes a `generate` command.) Run the `reorganize` command as follows:

   ```
   dbutility reorganize datasource
   ```

The `reorganize` command does the following:

- It generates new Client scripts for all data tables whose layouts have changed by running the same code that a `generate` command would.

- For each data set affected by the reorganization, it runs the scripts created by the `redefine` command to reorganize the tables associated with the data set. If these scripts run successfully, it restores `ds_mode` to its value before the reorganization. Conversely if the script fails, it sets `ds_mode` to 33 for the data set in question to indicate that the attempt to reorganize the table has failed, and it stops prematurely.

  If the command completes successfully, proceed to step 8. Otherwise, determine why the command failed and decide what to about it.

  Your options include:

  - Give up and re-clone the data set that could not be reorganized by setting its `ds_mode` to 0. Rerun the `reorganize` when you do this, as the command stops when it encounters an error. Restarting it after taking care of the problem data set will complete the task for any data set that still have a `ds_mode` of 31. If you try to run a `process` command while some data sets still have their `ds_mode` columns set to 31, this will result in an error telling you that you need to run a `reorganize` command.

  - Correct the script that failed, set its mode back to 31, and rerun the `reorganize` command.

  - If you are proficient in SQL, you can reorganize the table using external means to the Client to perform the action that the reorg scripts were attempting to do. If you succeed you can then set `ds_mode` back to its original value (which will most likely be 2). You also will need to run a `refresh` command for the problem data set to replace the old stored procedures which are out-of-date.

  The `reorganize` command is restartable after a failure. The data sets that were already processed successfully will not be affected by rerunning the command, and the data set that caused the command to fail will be skipped unless its `ds_mode` column is set to 31.

8. Run a `process` command to resume change tracking:

```
dbutility process datasource
```

About the redefine Command

You will be prompted to run the `redefine` command when a data set is reorganized or when the Support Library is recompiled. (A Support Library recompile indicates that either the layout has changed, such as changes to ALTER or FILTER, or the SUPPORT option in the SOURCE declaration changed.)

In all of the aforementioned cases, Databridge Engine treats the situation like a reorganization and requires that you run a `redefine` command.

When you run the `redefine` command, it does the following:

- Creates a backup of the Client control tables for the data source by silently performing an `unload` command. The unload file is created in the data source's working directory when the Client first detects the reorganization. The unload file is named "*datasourcereorgnnn*.cct" where *nnn* is the value of the update level prior to running the `redefine` command (and is saved to the `old_update_level` column of the DATASOURCES entry).

- Re-creates the relational database layout for all data sets that are marked as needing to be redefined.

- Runs user scripts (if you use them) to preserve changes to the Client control tables. If you are using the Administrative Console's **Customize** command, all changes are restored from the old controls tables.

- Determines which data sets have tables whose layouts have changed, updates the `ds_mode` column in DATASETS accordingly, and creates reorganization scripts that will alter the relational database tables to match the changes in the reorganized DMSII data sets.

Reorganized Data Sets

When a data set has been reorganized (`status_bits` = 8), the `redefine` command compares the layouts of tables mapped from the data set in the existing Client control tables with the new layouts and does the following:

- If no changes occur to the layouts of tables mapped from the data set, the `redefine` command sets the data set `ds_mode` column its value before the reorganization (1 or 2), indicating that the data set is ready to be updated.

- For tables for which the layout has changed, the `redefine` command creates reorganization scripts that will modify the relational database tables to match the changes in the reorganized DMSII data sets.

- If the DMSII reorganization introduces one or more new columns, one of the following occurs based on the value of the parameter `suprress_new_columns` .

Is	Result
True	The `active` column is set to 0 for new items in the DATAITEMS Client control table and for new tables in the DATATABLES Client control table. The next `process` command does not re-clone the data set.
False	The new columns are added to the tables in the Client database. These columns will be set to the appropriate values based on their INITIALVALUE defined in the DASDL. The next `process` command will continue to populate the table including the new column. If new tables appear, the data set will be re-cloned.

- If the reorganization introduces one or more new data sets, one of the following occurs base on the value of the parameter `suppress_new_datasets` .

Is	Result
True	Databridge Client sets the `active` column in the corresponding entry in the DATASETS Client control table to 0, and the data set is not mapped.
False	Databridge Client sets the `active` column in the corresponding entry in the DATASETS Client control table to 1 (unless the data set is a REMAP), and the layout of the corresponding relational database tables is defined in the DATATABLES and DATAITEMS Client control tables. You must run a `reorganize` or `generate` command to create the scripts for these new tables. These data sets are automatically cloned the next time you run a `process` command.

- For any reorganized data set whose `active` column is 0, the `redefine` command updates the corresponding Client control table entries, leaving the `active` column set to 0. This ensures that if you later decide to clone that data set, you only need to set the `active` column to 1 and execute a `redefine` and a `generate` command.

9.2.2 Performing Reorganizations Using an Internal Clone

Version 6.6 of the Databridge software introduced a new way of reorganizing tables that does not use `alter` commands. In some cases, the process of reorganizing a table by using `alter` command can be very expensive. For example, if you try to change a column that is an **int** to a **dec(10)** when using SQL Server, the `alter` command will cause every single change to be logged, which can have rather disastrous effects if the table is large. If you run out space for the log, the `alter` command abends, leading to a massive rollback.

The `use_internal_clone` parameter allows you to select the default method of doing reorganizations. See use_internal_clone for more information. You can then override it (on a data set by data set basis) by using the Client Configurator to change the setting of the `ds_options` bit DSOPT_Internal_Clone (see DSOPT_Internal_Clone in the section DATASETS Client Control Table for a description of this bit).

The internal clone is comparable (in terms of speed) to using the bulk loader to copy the data from the old table to the new table. In the case of SQL Server, to make it run fast you must make sure that database's recovery model is not set to "Full", as was the case of for the bulk loader (temporarily change the database model to ""Simple" or "Bulk-logged" when you run an internal clone).

9.2.3 DMSII Changes to Record Locations

DMSII record locations are changed in the following circumstances:

- Garbage collections reorganizations
- File format conversions
- Record format conversions

Garbage collection and file format conversion reorganizations only affect data sets that use AA Values as keys. Therefore, unless the data sets using AA Values as keys are small and garbage collection reorganizations at your site are infrequent, we recommend that you use RSNs. (If you're unable to use RSNs, composite keys are a viable alternative to AA Values. However, they are error prone and can result in false duplicate records.)

When a data set is affected by a garbage collection reorganization or a file format conversion, the Databridge Client sets the `ds_mode` column to 12 in the DATASETS Client control table and displays the message:

```
WARNING: DMSII reorganization has occurred; AA Values for DataSet name [/rectype] are no longer valid
```

When a record format conversion affects a data set that uses AA Values, the `redefine` command forces that data set to be re-cloned, even if the tables derived from the data set are not affected by the reorganization.

9.3 DMSII Reorganization When Using Merged Tables

The merged tables feature combined with multi-source databases allows a user to store data from multiple separate DMSII databases into a single relational database. The requirement is that all the DMSII databases have the same DASDL and always be kept in sync, as far as reorganizations are concerned. This section documents how to go about handling such reorganizations. Everything we said about the single data source case still applies here. The first thing you need to do is to let the Clients catch up with all the updates until it gets to the point in the audit trail where the reorganization occurred. Make sure that you let all the Clients catch up before doing anything else.

Once all the data sources are caught up, you will need to run `redefine` commands for all of the data sources. Once this is completed you will then need to run a `reorganize` command for one of the data sources. **Do not do this for more than one data source**, as there is only one set of tables in the relational database and if any of the scripts alter a table running the scripts a second time will usually result in SQL errors, as the ALTER commands will most likely not be valid. For example if the ALTER command adds a column, an attempt to add it again will fail. We added the `-n` option to the `reorganize` command to make it work for the second and any subsequent data sources in a multi-source environment. This allows you to get all the data sources ready for processing updates by generating scripts for reorganized data sets and refreshing the stored procedures for the tables associated with such data sets. Finally the command updates the `ds_mode` column in DATASETS, restoring it to the value it had before the redefine command was run. This command appears near the bottom of the **Advanced** menu for the data source in the Administrative Console's **Customize** command. If the reorganization requires that a data set be re-cloned, you should add the `-k` option to the first process or clone command you use so the table gets dropped. In the absence of the `-k` option, the Client will run the cleanup script, which removes all the records associated with the current data source. The alternative is to manually drop the table for the first data source. Once the tables to be re-cloned have been dropped, the remaining data sources can operate normally.

9.4 Rollbacks

A DMSII "rollback" restores the DMSII database to an earlier point in time in order to correct a problem. While the DMSII database is being restored, replication stops. The Client must then be restarted for replication to resume.

If the Client has processed updates after the DMSII restore point, this replicated data will be wrong. Upon finding bad information in the stateinfo, the Databridge Engine typically returns a message stating that a rollback has occurred. To resolve this problem, the relational database must also be rolled back (to the DMSII restore point or earlier).

If the Client hasn't processed updates after the DMSII restore point, no action is required. This can often be the case as the Client tends to lag behind the DMSII database by several audit files during the processing of audit files generated by batch jobs.

9.4.1 Recovering from DMSII Rollbacks

You'll need to recover the relational database after a DMSII rollback In situations where the Client database is caught up with the DMSII database (that is, there is no lag time between the two). There are two preferred ways to do this:

Programmatic rollback	Undoes all transactions that occurred after the specified rollback point (typically a time prior to the DMSII restore point). This is only possible if the relational database is audited, which is rarely the case.
Reload the database	Entails reloading the database from a backed-up copy. This requires that all of the audit files—from the point when the relational database was backed up forward—to be available. If the audit files aren't available, recloning is the only option.

Recloning the database is usually very time-consuming and is only recommended as a last resort or in cases where the relational database contains little data or if the required audit files are not available. For information about recloning, see Recloning.

> ⚠️ **Caution**
>
> Using shortcuts to recover a relational database after a DMSII rollback, such as updating the tables using scripts or resetting the State Info, is not only ineffective but problematic. These methods leave obsolete updates in the Client database and may cause valid updates to be skipped after the Databridge Client resumes tracking.

9.5 Recloning

Reasons for recloning include the following:

- DMSII reorganization

- DMSII rollback

- An update is not possible (for example, because a table does not have a unique key)

- One or more of the Databridge data tables in the relational database were removed

You can use either the `process` or `clone` command to re-clone data sets. The `clone` command lets you specify individual data sets on the command. The `process` command automatically re-clones all data sets whose `active` column is 1 and whose `ds_mode` column is 0. Both commands perform fixups, tracking and processing updates as needed (unless the `defer_fixup_phase` or the `stop_after fixups` parameter is set to True). See Recloning Individual Data Sets.

If you're recloning the entire database, the process is more involved. See Recloning a Database.

9.5.1 Recloning Individual Data Sets

Use one of the following procedures to re-clone data sets.

1. Set the current directory to the one you created for the data source (the directory from which you ran a `generate` command for the data source). Make sure that the directory contains the scripts for this data source.

2. Set the `ds_mode` column (in the DATASETS Client control table) to 0 for the data sets you want to clone by running a SQL command. If you are recloning all data sets, using the " `-Y reclone_all` " option eliminates the need to do this, as the Client will update the DATASETS table automatically when this option is used.

3. Run the `process` command with the -y option, as follows:

```
dbutility process -y datasource
```

The `-y` option forces any data sets whose `ds_mode` is set to 11 or 12 to be recloned, in addition to the recloning data sets whose `ds_mode` is set to 0. After the data extraction process is complete for the data sets being recloned, Databridge data tables whose `active` columns are set to 1 in their corresponding Client control table (and whose `ds_mode` is set to 2) are updated.

To reclone with a `clone` command

1. Set the current directory to the one you created for the data source (the directory from which you ran a `generate` command for the data source). Make sure that the directory contains the scripts for this data source.

2. Set the parameter `defer_fixup_phase` to True to suspend audit file processing. If you don't do this, audit files will be processed twice, once for the data set you clone and once for all of the other data sets.

3. Synchronize the tables by running a `process` command. Synchronization occurs when all data sets reach the same point in the audit trail.

For clone command syntax, see dbutility Commands.

9.5.2 Recloning a Database

Recloning the relational database can be an efficient means of recovering it if it doesn't contain a lot of data. Otherwise, it can be time-consuming and costly, as recloning uses host resources. These reasons alone often make recloning a last resort when no backup is available. (These issues are one of the reason why we developed Enterprise Server. It makes processes like this one more efficient.)

We recommend that you use the following procedure instead of setting `ds_mode` to 0 for all data sets using a SQL query and running a `process` command, because it ensures that you have the latest copy of the DMSII layout.

• Make sure that you have the latest copy of the DMSII layout.

• Run a `drop` command to drop the data source.

• Run a `define` command.

• Run a `generate` command.

• Run a `process` command.

9.5.3 Adding a Data Set

Use this procedure to add a data set after you clone the DMSII database. You don't need to reclone the entire database.

To add a data set

1. Run a relational database query tool and list the contents of the DATASETS Client control table with the following SQL command:

```
select dataset_name, active, data_source from DATASETS
```

2. Set the `active` column for the data set you want to add to the Databridge data tables to 1 (on), as follows:

```
update DATASETS set active=1 where dataset_name='datasetname'
```

3. Run a `redefine` command.

4. Run a `generate` command to create new scripts that populate the resulting table.

5. Run one of the following commands to populate the new tables that correspond to the new data set:

```
dbutility process datasource

--or--

dbutility clone datasource datasetname
```

> ✏️ **Note**
>
> If you run the a `process` command, the Databridge data tables whose `active` columns are set to 1 in their corresponding Client control table are also updated at this time.

After you complete this procedure, update your data set selection script (`script.user_datasets` .*datasource*) so that you do not lose this change the next time you run a `define` command.

9.5.4 Dropping a Table

Use this procedure when the Client no longer uses a Databridge data table in the relational database.

To drop a table from the Administrative Console, see the Databridge Client Console Help.

To drop a table

1. Update your data set global mapping customization and global data table customization scripts, depending on whether you are dropping a primary or secondary table, to reflect this change. See Customizing with User Scripts.

2. If you are dropping all of the tables derived from a data set, set the `active` column corresponding to the data set to 0 (in the DATASETS Client control table) and then run the data set selection script (`script.user_datasets` .*datasource*) using the dbutility `redefine` command.

3. If you are dropping a secondary table, set the `active` column corresponding to the table to 0 (in the DATATABLES Client control table) and then run the data table customization script (`script.user_define` .*primary_tablename*) for the primary table using the `redefine` command.

4. From a command line, set the current directory to the working directory for the data source, and then run a script, such as the following (Windows)

   ```
   dbutility -n runscript dbscripts\script.drop.tablename
   ```

9.6 Backing Up and Maintaining Client Control Tables

To help you maintain your Client control tables, Databridge provides three commands that allow you to backup, restore, and recreate copies of your Client control tables. In this section, each of these commands is described.

9.6.1 The Unload Command

The `unload` command creates a text file that contains a record for each of the entries in the various Client control tables. For best results, run an `unload` command before running a `redefine` command.

Format	The format of the `unload` command is as follows: `dbutility [options] unload` *datasource filename*
Options	The list of options is the same as those for signon_options. Additional options include `-t` , `-T` , and `-f` .
Data Source	If a *datasource* of "_ALL" is specified, the Databridge Client writes all data sources to the backup file (*filename*). If a specific data source is specified, the Databridge Client writes only the entries for that data source to the backup file.

Sample Run

15:05:25 dbutility unload demodb demodb.cct

15:05:25 Databridge Client version 7.0.0.000 [OCI/Oracle]

15:05:25 Copyright (C) 2019 Micro Focus or one of its affiliates.

15:05:30 Loading control tables for DEMODB

15:05:32 Unloading control tables for DEMODB

15:05:32 Control tables for DataSource DEMODB written to file "demodb.cct"

```
 15:05:32 Client exit code: 0 - Successful
```

9.6.2 The Reload Command

The `reload` command enables you to restore the Client control tables from a file that was created using the `unload` command.

Format	The format of the `reload` command is as follows: `dbutility [signon options]reload` *datasource filename [dataset1, dataset2, …]* **NOTE**: Client control table changes made since the tables were unloaded will be lost. Depending on what has changed, data table record could also be affected, requiring recloning.
Options	The list of options include `-t` , `-T` , `-f` , and `-k` . The -k option forces Databridge to keep the stateinfo in the control tables for data sets that are in normal mode (ds_mode = 2) and that have `client_fmt_level` and `item_count` columns that remain unchanged (there is no reorganization involved).
Data Source	If a *datasource* of "_ALL" is specified, the Databridge Client restores all data sources contained in the backup file. If a specific data source is specified, the Databridge Client restores only the entries for that data source from the file. If this is further qualified by a data set list, the Databridge Client restores only the entries for the data sets specified. Note that all the data sets specified in the list must already exist.

Sample Run

```
17:16:26 dbutility reload demodb demodb.cct

 17:16:27 Databridge Client version 7.0.0.000 [OCI/Oracle]

 17:16:27 Copyright (C) 2021 Micro Focus or one of its affiliates.

 17:16:35 Reloading Control table entries for DataSource DEMODB from file "demodb.cct"

 17:16:45 Control tables for DataSource DEMODB reloaded from file "demodb.cct"

 17:16:45 Client exit code: 0 - Successful
```

9.6.3 The Refresh Command

The `refresh` command enables you to drop and recreate all of the stored procedures for the tables associated with the given data set in the specified data source. It is a variation of the `runscripts` command that is designed to run portions of the Databridge Client scripts (`script.drop` .*tablename* and `script.create` .*tablename*). This command is useful when you want to manually handle a data set that would otherwise be recloned after a DMSII reorganization.

> ✏️ **Note**
>
> In case of variable-format data sets, the tables for all the record types that have their active column set to 1 in the DATASETS Client control table, are refreshed.

Format	The format of the `refresh` command is as follows: `dbutility [options] refresh` *datasource dataset*
Options	The list of options is the same as those for signon_options. If "_ALL" is specified for dataset, Databridge Client refreshes the stored procedures for all active tables that correspond to data sets whose `active` columns are 1. If a specific data set is specified, the Databridge Client refreshes only the stored procedures for the tables mapped from that data set. All tables for the specified dataset must have been created.

Sample Run

```
12:39:45 dbutility refresh DEMODB CUSTOMER

 12:39:45 Databridge Client, Version 7.0.0.000 (64-bit) [OCI/Oracle]

 12:39:45 Copyright 2021 Micro Focus or one of its affiliates.

 12:39:45 Loading control tables for DEMODB

 12:39:45 Stored procedures for all tables of DataSet CUSTOMER successfully refreshed

 12:39:45 Client exit code: 0 - Successful
```

In this case, the data set CUSTOMER is mapped to a single table named customer. The `refresh` command executes the following SQL statements.

```
begin drop_proc('u_customer');end;

 begin drop_proc('i_custmer'); end;

 begin drop_proc('d_customer'); end;

 create procedure u_customer (...) update customer set ... where ... ; end;

 create procedure i_customer (...) insert into customer (...) values (...); end;

 create procedure d_customer (...) delete from customer where ... ; end;
```

This effectively replaces all of the stored procedures with a fresh copy, while leaving the tables unchanged. This command is particularly useful when the index of the tables has changed. For example, if the data set CUSTOMER initially uses AA Values as keys, and a DMSII garbage collection occurs, you can avoid recloning this data set if it is mapped to a single table by creating a composite key.

10. Data Mapping

This chapter shows you how the Databridge Client maps DMSII data structures to relational database structures.

10.1 DMSII and Relational Database Terms

The following table shows the equivalent terms for DMSII structures and relational database structures:

DMSII	Relational
Data set	Table
DMS item (data item)	Column
Record	Row (record)
Set	Index **NOTE**: A relational database index is a set of column names that is used to efficiently access a row (of a table).
Key	Key

10.2 DMSII and Relational Database Data Types

The Databridge Engine retrieves the requested DMSII data, and DBServer passes the data to the Databridge Client, where it is assigned to standard relational database data types.

The following table lists equivalent data types for DMSII, Microsoft SQL Server, and Oracle.

DMSII	SQL Server	Oracle
ALPHA (<= char_limit bytes)	CHAR *	CHAR +
ALPHA (>char_limit bytes)	TEXT	VARCHAR2
ALPHA (>varchar2_limit_bytes)	TEXT	CLOB
BOOLEAN	BIT	NUMBER(1)
FIELD: FIELD(n) where n is < 16 FIELD(n) where n is < 32 FIELD(n) where n is >= 32	 SMALLINT INT BIGINT ‡	 NUMBER(5) NUMBER(10) NUMBER(10-15)
NUMERIC: NUMBER(n) where n is a DMSII declared length <= 2 NUMBER(n) where n is a DMSII declared length <= 4 NUMBER(n) where n is a DMSII declared length <= 9 NUMBER(n) where n is a DMSII declared length <= 15 NUMBER(n) where n is a DMSII declared length >15 NUMBER(n,m) where n is a DMSII declared length and m is the number of places after the decimal point.	 TINYINT § SMALLINT INT BIGINT ‡ DEC(n) DEC(n,m)	 NUMBER(n) NUMBER(n) NUMBER(n) NUMBER(n) NUMBER(n) NUMBER(n,m)
REAL: REAL(n) where n is a DMSII declared length <= 2 REAL(n) where n is a DMSII declared length <= 4 REAL(n) where n is a DMSII declared length <= 9 REAL(n) where n is a DMSII declared length > 9 REAL(n,m) where n is the DMSII declared length and m is the number of places after the decimal point. REAL, with no precision or scale	 TINYINT § SMALLINT INT DEC DEC(n,m) FLOAT	 NUMBER(n) NUMBER(n) NUMBER(n) NUMBER(n) NUMBER(n,m) FLOAT

* VARCHAR if the configuration parameter `use_varchar` is set to True.

† VARCHAR2 if the configuration parameter `use_varchar` is set to True.

‡ If the configuration parameter `use_bigint` is set to False, DEC(n) will be used instead.

§ Note that if the number is signed, SMALLINT is used instead. TINYINT is an unsigned quantity in SQL Server.

10.2.1 Databridge Data Types

IMAGE is a Databridge type that allows you to store an ALPHA item as binary by using the REDEFINE clause of the ALTER command in GenFormat on the host.

When the Databridge Client encounters an item of type IMAGE, it automatically sets the `ds_options` DIOPT_Clone_as_Binary bit in the DMS_ITEMS Client control table.

10.3 Supported DMSII Structures

This section lists DMSII structures that are supported by the Databridge Client. If you are the relational database administrator and have no experience with DMSII databases, this section will be more useful to you if you are working with a DMSII database administrator.

In addition to fixed-format data sets and variable-format data sets, the Databridge Client supports the following DMSII structures:

- Embedded data sets (see exceptions in the following section)

- Remaps

- Logical database

- GROUP

- FIELD items for GROUPs of BOOLEANS

- OCCURS

- GROUP OCCURS

- Data sets with more items than the maximum number of columns supported by the relational database

- Data sets that generate relational tables whose record sizes exceed the Microsoft SQL Server maximum record size.

- DMSII links in DIRECT, UNORDERED, and STANDARD data sets. (Links in variable-format data sets are cloned but not tracked.)

Some of these structures may not be supported by your relational database. The DMSII structures that are not supported by relational databases are mapped into a form that the relational database can use. Each of these structures and the way they are mapped in the relational database are explained in the remainder of this chapter.

10.4 Unsupported DMSII Structures

When the Databridge host replication software does not support a particular DMSII structure, the Databridge Client may or may not issue a warning message, depending on the DMSII structure. For example, a message is generated when the data set has no keys.

The Databridge Client does not support the following DMSII structures:

- Embedded data sets within an ORDERED or COMPACT data set

- Embedded data sets if the INDEPENDENTTRANS option is reset

- POPULATION items

- COUNT data items

- FILLER data items

- AGGREGATE data items

10.4.1 Embedded Data Sets

An embedded data set is a DMSII representation of a hierarchical relationship or tree structure. When a DMSII data set contains another data set as an item, that data set is called an embedded data set. The data set in which it is declared is called the parent of the embedded structure. You can think of the embedded data set as the "child" of the "parent" data set.

To represent this parent-child relationship in a relational database, the Databridge Client uses a foreign key that points to the parent data set. This foreign key is represented by the value in the `parent_aa` column in the table that corresponds to the embedded data set. The `parent_aa` column holds the parent record's key.

DMSII DASDL Showing an Embedded Data Set

The following is an excerpt from a DMSII DASDL that shows how an embedded data set is defined.

```
GENEALOGY DATA SET
  (
   PARENT-FAT-NAME        ALPHA (30);
   PARENT-MOT-NAME        ALPHA (30);
   PARENT-MOT-MAIDEN      ALPHA (30);
   PARENT-FAT-BDATE       NUMBER (06);
   PARENT-MOT-BDATE       NUMBER (06);
   FILLER                 SIZE (06);
%
   CHILD                  DATA SET;
     (
      CHILD-NAME          ALPHA (30);
      CHILD-STATUS        ALPHA (11);
```

```
      CHILD-BDATE          NUMBER (06);
      CHILD-GENDER         FIELD
        (
        CHILD-MALE         BOOLEAN;
        CHILD-FEMALE       BOOLEAN;
        );
      CHILD-FILLER         FIELD (01);
%
      );
  );
```

Resulting Tables

The following examples are for Microsoft SQL Server.

Ignoring any set definition, the resulting relational database tables are as follows:

- Genealogy (the parent data set is cloned to its own primary table)
- Child (the embedded data set is cloned to its own secondary table with a pointer to its parent table)

Parent Table: genealogy (table name). Where the `my_aa` column is a unique key for the record derived from the DMSII AA Value of this record.

```
  (
  my_aa             char(12),
  parent_fat_name   char(30),
  parent_mot_name   char(30),
  parent_mot_maiden char(30),
  parent_fat_bdate  int,
  parent_mot_bdate  int
  )
```

Child Table: child (table name)

```
(
    my_aa            char(12), - child table's key
    parent_aa        char(12), - foreign key of parent table
    child_name       char(30),
    child_status     char(11),
    child_bdate      int,
    child_male       bit,
    child_female     bit,
    child_filler     smallint
  )
```

10.4.2 Selecting Embedded Data Sets for Cloning

When you run a `clone` command, by default, the Databridge Client selects embedded data sets along with the parent structures. If you have altered the values for the active column in the Databridge Client control tables, however, check the values for the embedded data set and its parent data set.

> ⚠️ **Caution**
>
> If you want to clone an embedded data set, you must also clone the parent structure. Failure to do this results in the following error message from the Databridge Engine on the host:
>
> ```
> 0043 Parent of embeddeddataset must be selected.
> ```

10.4.3 Record Serial Numbers

A record serial number (RSN) is a unique number (48-bits long) that is associated with a record in a data set. The RSN is guaranteed to be unique, and it stays with a record for the life of the record. Updates do not affect the RSN; therefore, RSNs are ideal keys for Databridge tables. However, RSNs are available only when using DMSII XE.

DMSII XE adds the RSN to every record of a data set that has the EXTENDED attribute set. As long as the EXTENDED attribute is set, Databridge can access the RSN, unlike application programs that can access the RSN only if it has been explicitly declared in the

DASDL source. The Databridge Engine is designed to use RSNs instead of AA Values whenever possible. Regardless of whether RSNs are present, AA Values are used for data sets that contain embedded data sets or DMSII links.

Since RSNs and AA Values are the same length, they are interchangeable, unless the data sets are embedded data sets or contain DMSII links. If the Databridge Engine is not handling either of these types of data sets *and* an RSN is present, Databridge Engine uses the RSN rather than the AA Value. In this case, the resulting column of the DATAITEMS Client control table is named `my_rsn` instead of `my_aa` to differentiate it from an AA Value. In both cases, the `set_name` column of the DATASETS Client control table will have a value of "aa_set".

10.4.4 AA Values

AA is a DMSII term that stands for absolute address. An absolute address value is an A Series WORD (48-bits in length). In the Databridge Client, AA is the hexadecimal representation (12 character strings containing the characters 0--9 and A--F) of the AA Value on the host. Databridge Client uses the AA Values to implement unique keys for the parent structures of embedded data set records. It also uses AA Values to reference the records of data sets that do not have DMSII SETS with the NO DUPLICATES ALLOWED attribute.

AA Values are not constant. Any DMSII reorganization (record conversion, file format, or garbage collection) changes these values.

> ✏ **Note**
>
> Databridge Client supports numeric AA Values that are stored as NUMBER(15) in Oracle and BIGINT in SQL Server. It also supports binary AA Values that are stored as RAW(6) in Oracle and BINARY(6) in SQL Server.

10.4.5 DMSII Links

The Databridge Client implements DMSII link items, such as MEMO items in LINC databases, using an AA Value. You can use the AA Value as a foreign key when you want data from the tables associated with the link item. To enable support for DMSII links, you must do the following:

- Enable DMSII link support in the Databridge Engine control file.
- Set the Client configuration file parameter `enable_dms_links` to True.

10.4.6 Variable-Format Data Sets

DMSII variable-format data sets consist of a collection of dissimilar records containing a fixed part that is common to all records, and a variable part that depends on the record type. The individual records are identified by the record type, which is a special data item that has a value between 0 and 254.

- A record type of 0 indicates that the record does not contain a variable part.
- A record type of 1--254 indicates that the record contains the corresponding variable part in addition to the fixed part that is always present.

The concept of variable-format tables does not exist in relational databases. Therefore, the Databridge host software handles the various types of variable-format records as different structures. Databridge references these structures by a data set name and a record type (all other data set types have a record type value of 0). The Databridge Client uses the notation *datasetname/rectype* when referring to variable-format data sets in all messages. The Databridge Client handles these structures as if they were logical data sets; thus, each individual record type of a variable-format data set is mapped to a different relational database table. Variable-

format data sets are tracked and updated like fixed-format data sets. Links in variable-format data sets, however, are not tracked, but retain their values from the initial clone. The link values for any records created after the clone will be null. (See track_vfds_nolinks.)

Fixed Part Only Records	Even though type 0 records are not explicitly declared in the DMSII DASDL, applications can create such records by simply setting the record type to 0. The Databridge software always defines a data set with a record type of 0 (`rectype` column in the DATASETS Client control table) for type 0 records of variable-format data sets. This data set is mapped to a table whose name is derived from the data set name (that is, name converted to lowercase and all dashes replaced by underscores). Note that unless the DMSII applications explicitly store type 0 records in the data set, this table may be empty. If you know this is the case, you may want to disable the cloning of the type 0 records for the data set by setting the `active` column of the corresponding entry in the DATASETS Client control table to 0.
Variable Format Records	All other record types are treated as if they were contained in a separate structure. The primary tables for these structures are named by appending the suffix "_type#* to the name mapped from the data set name, where # is the decimal value of the record type (that is, a value between 1 and 254). Note that the fixed part of the variable-format records and the record type are also stored in the resulting relational database table.

DMSII DASDL Showing Fixed- and Variable-Length Records

```
MAIN            DATA SET
(
    VAR-REC     RECORD TYPE (3);
    CUST-NO     NUMBER (08);
    CPU         ALPHA (06);
    FILLER      SIZE (05);
),

%

1:
(
    SMSA        ALPHA (04);
    SALES       ALPHA (06);
)

%

2:
(
    STATUS      ALPHA (01);
    RECEIVED    ALPHA (06);
    ORDER-DATE  NUMBER (08);
)

%

3:
(
    SITE        ALPHA (07);
    SOURCE      ALPHA (07);
    CLASS       ALPHA (02);
);

%

MAINSET         SET OF MAIN
                KEY (CUST-NO)
                NO DUPLICATES;
```

10.4.7 Resulting Tables

The examples in this section assume that the Microsoft SQL Server database is being used and that the Client uses the SET MAINSET as the source for index for the various tables.

The following tables are derived from the variable-format data set MAIN:

- main (type 0 records)

- main_type1 (type 1 records)

- main_type2 (type 2 records)

- main_type3 (type 3 records)

> ✏️ **Note**
>
> All four tables contain the fixed part of the data set. The var_rec column is the record type; all records in the individual tables will have the same value in this field.

Record Type 0 Table

The table named main represents all type 0 records that do not have a variable part. The var_rec column of all records in this table will have a value of 0. Note that this table may be empty if your application does not use type 0 records. The SQL statement to create this table is shown as follows:

```
create table main
(
  cust_no int,
  var_rec smallint,
  cpu char(6)
)
```

Record Type 1 Table

The table named main_type1 represents all type 1 records. The var_rec column of all records in this table will have a value of 1. The SQL statement to create this table is shown as follows:

```
create table main_type1
(
    cust_no   int,
    var_rec   smallint,
    cpu       char(6),
    smsa      char(4),
    sales     char(6)
)
```

Record Type 2 Table

The table named main_type2 represents all type 2 records. The var_rec column of all records in this table will have a value of 2. The SQL statement to create this table is shown as follows:

```
create table main_type2
(
    cust_no   int,
    var_rec   smallint,
    cpu       char(6),
    status    char(1),
    received  char(6),
    order_date int
)
```

Record Type 3 Table

The table named main_type3 represents all type 3 records. The var_rec column of all records in this table will have a value of 3. The SQL statement to create this table is shown as follows:

```
create table main_type3
(
    cust_no   int,
    var_rec   smallint,
    cpu       char(6),
    site      char(7),
    source    char(7),
    class     char(2)
)
```

10.4.8 Split Variable Format Data Sets Option

When the `ds_options` bit DSOPT_Split_Varfmt_ds (bit value 65536) is set, variable format data sets are treated slightly differently. The record type 0 tables contains the fixed part of all records regardless of their record types. However, the table has exactly the same layout as above. The tables for all the other records only contain the variable part of the records and the keys from the fixed part.

The table named main_type1 in the above example will now contain the key cust_no and the variable part. The SQL statement to create this table is shown as follows:

create table main_type1 (cust_no int, smsa char(4), sales char(6))

10.5 Changing the Default Data Type

In most cases, the default data types are sufficient. If you want to change the data type, however, use a relational database query tool to edit the `sql_type` column in the DATAITEMS Client control table, or put the SQL statements in user scripts as explained in Customizing with User Scripts.

> ⚠ **Caution**
>
> When changing the default data type, make sure that you choose a correct data type or the data may not be correctly stored in the relational database.

Most of these relational database data types can be changed using data table customization user scripts or the Administrative Console's **Customize** command is the **Settings** menu for the data source.

Value for sql_type	Generic Data Type	Microsoft SQL Server Data Type	Oracle Data Type
0	bit	bit	number(1)
1	char	char	char
2	varchar	varchar	varchar2
3	byte	tinyint	number(3)
4	short int	smallint	number(5)
5	long int	int	number(10)
6	float	float	float
7	text	text	clob
8	binary	binary	raw
9	varbinary	varbinary	raw
10	datetime	datetime	date
11	packed BCD	dec	number
12	smalldatetime	smalldatetime	date
13	numeric date	int	number(10)
14	unsigned long	binary(4)	raw(4)
15	timestamp	timestamp	N/A
16	serial	{int - bigint - dec(n) - identity}	N/A
17	numeric_time	int	number(6)
ticks	int	number(6) or number(10)	N/A
18	int64	bigint	NA
19	date	date	NA
20	datetime2	datetime2	timestamp
21	time	time	N/A
22	uniqueidentifier	uniqueidentifier	N/A

10.6 Handling DMSII GROUPs

A GROUP is a DMSII construct that allows the data items that belong to the group to be referenced at one time (for example, as one item). The concept of GROUP does not exist in a relational database. Therefore, if the DMSII database you replicate has one or more GROUPs, the Databridge Client ignores the GROUP name and instead treats each item within the GROUP as a regular data item. All items in a DMSII GROUP share the same parent item number, which is the item number of the GROUP item.

Following is an example of the DMSII GROUP item in the data set called ADDRESS. This GROUP item consists of the data item CITY and the data item STATE.

DMSII DASDL Showing GROUP

The following is an excerpt from a DMSII DASDL that shows how a GROUP item is defined. With the GROUP item, you can access both city and state with one reference.

```
ADDRESS          DATA SET
(
    STREET       ALPHA (20);
    APARTMENT    ALPHA (5);
    CITY-STATE   GROUP
    (
        CITY     ALPHA (20);
        STATE    ALPHA (2);
    );
    COUNTRY      ALPHA (20);
    ZIPCODE      NUMBER (5);
    POSTFIX      NUMBER (4);
);
```

The next example shows how the same DMSII GROUP item is mapped to a relational database.

Relational Database Table

The following example is for Microsoft SQL Server.

The table name is the lowercase form of the DMSII data set name. The GROUP item CITY-STATE is ignored. The data items in that group are included in the relational database table as if they were ordinary DMSII data items.

```
address (table name)
street        apartment   city    state    country   zipcode  postfix

May St.       3           Paris   OH       USA       15010    2146
Elm Ln.                   River   SD       USA       24906    3381
```

If there are duplicate names among members of various groups within a data set, the Databridge Client resolves the conflict by appending a digit to the column name to make it unique.

10.7 Handling DMSII OCCURS

An OCCURS clause is a DMSII construct that describes the number of times an item is present or repeats within a data set. Because relational databases do not support the OCCURS construct, these clauses generate additional tables, which can degrade the performance of update processing.

You can control how items with an OCCURS clause are mapped on an item by item basis by flattening OCCURS. See Flattening OCCURS Clauses.

Default OCCURS Handling

If you don't flatten OCCURS, Databridge Client creates a new table for each data item that contains an OCCURS clause. The keys from the data item's parent data set are used as keys in the new table. In addition, a new key (named `index1`) is created to establish a unique composite key for each recurring data item.

For example, a DMSII data set has a data item with an OCCURS clause will result in two relational database tables:

- The first table (called the primary table) is named using the lowercase form of the DMSII data set name with all hyphens changed to underscores. It contains the key items as well as all data items that do not have OCCURS clauses.

- The second table (called the secondary table) is named by appending an underscore and the data item name to the primary table name. This table contains all of the OCCURS items; however, each table has a unique key created by `index1`. (Names that exceed the character limit are truncated. If the truncation results in a duplicate item names, the last characters of the name are changed to digits).

Handling OCCURS items this way can significantly degrade the performance of update processing if the number of occurrences is large. The storage required to hold the keys of the secondary table items can also be substantial. For example, an OCCURS 100 TIMES clause can turn a single DMSII update into 101 relational database updates. See DMSII DASDL with OCCURS for an example of a DMSII data set that has a data item with an OCCURS clause.

10.7.1 DMSII DASDL with OCCURS

The following excerpt from a DMSII DASDL shows how an OCCURS clause is defined.

```
ORDERS              DATA SET
(
    ORDER-ID        ALPHA (4);
    ORDER-DATE      ALPHA (5);
    ORDER-ITEM      OCCURS 10 TIMES NUMBER (8);
);

BY-ORDER-ID SET OF ORDERS
KEY IS
(
    ORDER-ID
)
NO DUPLICATES,
INDEX SEQUENTIAL;
```

The OCCURS clause allows access by subscripting (indexing) within an application program. Because relational databases do not allow subscripting (indexing), the Databridge Client maps the subscript into an additional key. The OCCURS items, then, are available by row.

When this ORDERS data set is cloned into the relational database, it is mapped into the following two tables. These tables show how the DMSII OCCURS clause appears in a relational database.

Table 1

This table is named the same as the ORDERS DMSII data set, and it contains the key item plus all non-OCCURS items. Assuming the ORDERS DMSII data set has 50 records, this table has 50 rows.

```
orders (table name)
order_id     order_date
--------     ----------
1201         jan12
  .            .
  .            .
  .            .
1250         feb12
```

Table 2

This table name combines the DMSII data set name and the name of the data item which has an OCCURS clause. It contains all the occurrences of the OCCURS data item ORDER-NUM.

Continuing with the example from Table 1 with 50 records (rows), this table has 500 total rows. For every order_id key (50 total), there are ten OCCURS items (as declared in the DASDL on the previous page).

```
orders_order_item (table name)
order_id     index1     order_item
--------     ------     ----------
1201         1          00007390
1201         2          00001293
1201         3          00007748
1201         4          00009856
1201         5          00003736
1201         6          00002278
1201         7          00004327
1201         8          00009463
1201         9          00008638
1201         10         00008954
1202         1          00001754
1202         .          00005309
1202         .          00004537
1202         10         00005940
1203         1          00005430
1203         .          00005309
1203         .          00004537
1203         10         00006587
  .          .            .
  .          .            .
  .          .            .
```

10.7.2 Flattening OCCURS Clauses

The `flatten_all_occurs` parameter makes the `define` and `redefine` commands set the value of the bit DIOPT_Flatten_Occurs (1) in the `di_options` column in the DMS_ITEMS table for all items that have OCCURS clauses. You can set this parameter from the Client

Configurator or by editing the configuration file to specify whether to globally flatten OCCURS clauses for a data source. By using user scripts, you can control this option for individual items.

The Databridge Client provides two options for handling OCCURS clauses.

Flatten OCCURS to the primary table	Each occurrence of the item is mapped into a separate column in the primary table. Use this method if the number of occurrences is not too large and applications access the occurring items by column name (versus numeric index). This is the default method for flattening OCCURS clauses and only requires that the above mentioned bit be set in the `di_options` column in the DMS_ITEM entry for the item with the OCCURS clause.
Flatten OCCURS to a new secondary table	In the secondary table, all of the occurring items are mapped to a single row that contains the keys and all of the occurrences of the item. Use this method to flatten OCCURS clauses that have a large number of occurrences. To make this happen you need to set the bit DIOPT_FlatSecondary(4096) in the `di_options` column in the DMS_ITEMS table for any items with an OCCURS clause that you want flattened in this manner. If both this bit and the DIOPT_Flatten_Occurs bit are set, this bit takes precedence.

When using the Administrative Console you can set the parameter `flatten_all_occurs` by using the **Configure** command in the **Settings** menu for the data source. To customize the handling of individual items with OCCURS clause use the **Customize** command in the **Settings** menu for the data source. Click on the data set involved and click on the wrench (properties) of the item in question in the DMS Items view and pick the entry you want from the drop down list for the "Flatten Occurs" property. The choices are "Don't Flatten", "Flatten Within Table", "Flatten into Secondary Table" and if applicable "Flatten to String".

10.7.3 Flattening OCCURS Clauses to a String

Single items of type NUMBER(n) or ALPHA(n) with an OCCURS clause can be flattened to a character string represented by a CHAR or VARCHAR data type. You can have fixed format strings or CSV format strings, where the delimiter character can be selected via the `dms_subtype` column in DMS_ITEMS. This feature is controlled by the DIOPT_Flatten2String bit in the `di_options` and the `dms_subtype` column. If the `dms_subtype` is 0, fixed format is used and if the `dms_subtype` is non-zero it specifies the delimiter character used in the CSV format. NULL data is represented by blanks in fixed format and empty fields in CSV format (i.e. two consecutive delimiters or a delimiter at the end of the data). For example a NUMBER(1) OCCURS 20 TIMES can be flattened to a column that is a CHAR(20) when using fixed format.

10.7.4 Flattening OCCURS Clause for Three-Bit Numeric Flags

MISER systems store certain flags as arrays of single-digit numbers, where each number is used to hold three Boolean values. The Databridge Client can be directed to map these items as a series of Booleans data items (bit in SQL Server). To do this, set the DIOPT_Flatten_Occurs bit (1) and the DIOPT_Clone_as_Tribit bit (16) in the `di_options` column of the corresponding DMS_ITEMS record.

An example for the item L-LOCK-FLAG in the data set LOAN follows:

Filename: `script.user_layout.loan`

```
update DMS_ITEMS set di_options=17
where dataset_name = 'LOAN' and rectype=0 and dms_item_name = 'L-LOCK-FLAG'
     and data_source = 'MISDB'
```

In the above example, if the L-LOCK_FLAG has an OCCURS 20 TIMES clause, 60 items of type bit named l_lock_flag_01 to l_lock_flag_60 are created.

These items can also be flattened to a secondary table by setting the bit DIOPT_FlatSecondary(4096) in the `di_options` column for the corresponding entry in the DMS_ITEMS table.

10.7.5 Flattening OCCURS Clause for Items Cloned as Dates

The following script directs the `define` and `redefine` commands to map an item with an OCCURS clause as a series of columns, whose data type is a relational database date type, in the corresponding table. Furthermore, it specifies that the DMSII item, which is of type NUMBER(8), contains a date in the *mm/dd/yyyy* format.

Filename: `script.user_layout.billing`

```
update DMS_ITEMS set di_options=3, dms_subtype=23
where dms_item_name = 'BILLING-DATES' and dataset_name = 'BILLING'
```

10.7.6 DMSII GROUP OCCURS

The following is an excerpt from a DMSII DASDL that shows a GROUP item that has an OCCURS clause.

```
SALES                    DATA SET
(
    PRODUCT-CODE         ALPHA (10);
    PRODUCT-NAME         ALPHA (20);
    SALES-HISTORY GROUP  OCCURS 5 TIMES  %FIVE YEAR HISTORY
    (
        TOTAL-UNITS-SOLD     NUMBER (10);    %FOR THE YEAR
        YEARLY-SALES-AMOUNT  NUMBER (S12,2); %BY MONTH
    );

);
SH-PRODUCT-CODE-SET SET OF SALES-HISTORY
    KEY IS
    (
        PRODUCT-CODE
    )
NO DUPLICATES,
INDEX SEQUENTIAL;
```

When this SALES data set is cloned into the relational database, it is mapped into the following tables:

Table 1 (primary table)

This table is named the same as the SALES DMSII data set, and it contains the key item and the data items that do not have OCCURS clauses. Because the GROUP item has an OCCURS clause, none of the GROUP items are included in this table. Assuming there are five records in the DMSII data set, there are also five rows in this relational database table.

```
sales  (table name)
product_code    product_name
------------    ------------
BC99992121      Widget
TR55553440      Mixer
HM44447322      Gadget
PP77778299      Twirler
DG22221163      SuperMix
```

Table 2 (secondary table)

This table is named: `datasetname` + GROUP_OCCURS_name

Assuming there are five records in the DMSII data set, there are 25 records in this relational database table. The main difference here is the addition of an index to denote the occurrence number of the item.

```
sales_sales_history (table name)

product_code    index1    total_units_sold    yearly_sales_amount
------------    ------    ----------------    -------------------
BC99992121      1         55543665            123456789.01
BC99992121      2         83746994            234567890.12
BC99992121      3         33847295            345678901.23
BC99992121      4         57483037            456789123.45
BC99992121      5         10947377            567891234.56
TR55553440      1         56722221            678912345.67
TR55553440      2         74838976            789123456.78
TR55553440      3         54793873            891234567.89
TR55553440      4         99048900            912345678.90
TR55553440      5         22308459            123456789.01
HM44447322      1         75032948            234567890.12
HM44447322      2         30750344            345678901.23
HM44447322      3         90570340            456789123.45
HM44447322      4         57948755            567891234.56
HM44447322      5         44874733            678912345.67
```

```
        .           .              .                .
        .           .              .                .
```

10.7.7 DMSII Nested OCCURS

The following is an excerpt from a DMSII DASDL showing a GROUP with an OCCURS clause that contains an item with an OCCURS clause.

This example helps to reinforce the previous examples of how DMSII GROUP and OCCURS are mapped to a relational database.

```
SALES                  DATA SET
 PRODUCT-CODE          ALPHA (10);
 PRODUCT-NAME          ALPHA (20);
 SALES-HISTORY GROUP   OCCURS 5 TIMES %FIVE YEAR HISTORY
   (
   TOTAL-UNITS-SOLD    NUMBER (10); %FOR THE YEAR
   MONTHLY-SALES-AMOUNT NUMBER (S12,2) OCCURS 12 TIMES;
   );
SH-PRODUCT-CODE-SET    SET OF SALES-HISTORY
   KEY IS
   (
   PRODUCT-CODE
   )
NO DUPLICATES,
INDEX SEQUENTIAL;
```

When this SALES data set is cloned into the relational database, it is mapped into the following three tables:

- `sales`

  `(primary table, table name derived from datasetname)`

- `sales_sales_history`

  `(secondary table, table name derived from datasetname + GROUPOCCURSname)`

- `sales_monthly_sales_amount`

  `(secondary table, table name derived from datasetname + OCCURSitemname)`

Table 1

This table is named the same as the SALES DMSII data set.

It contains the key item and all non-OCCURS data items. Because the GROUP has an OCCURS clause, none of the GROUP items are included in this table. Assuming there are five records in the DMSII data set, there are five rows in the resulting relational database table.

```
sales  (table name)
product_code       product_name
-----------        ------------
BC99992121         Widget
TR55553440         Mixer
HM44447322         Gadget
PP77778299         Twirler
DG22221163         SuperMix
```

Table 2

This table is named: `datasetname + GROUP_OCCURS_name`

Assuming there are five records in the DMSII data set, there are 25 rows in this table. Note the addition of the `index1` column to denote the occurrence number of the group.

```
sales_sales_history (table name)

product_code    index1    total_units_sold
-----------     ------    ----------------
BC99992121      1         55543665
BC99992121      2         98075300
BC99992121      3         77476478
BC99992121      4         76593939
BC99992121      5         33728282
TR55553440      1         87548974
TR55553440      2         56722221
TR55553440      3         11910078
TR55553440      4         47589474
TR55553440      5         57987999
```

```
HM44447322    1         75533785
HM44447322    2         33673391
HM44447322    3         74904532
HM44447322    4         98724498
HM44447322    5         39875992
    .         .             .
    .         .             .
```

Table 3

This table is named: `datasetname + OCCURSitemname`

Assuming there are five records in the DMSII data set, there are 300 rows in this table (12 occurrences of monthly_sales_amount for each of 5 occurrences of sales_history for each product code). In the table below, `index1` is the subscript of the GROUP OCCURS (1--5) and `index2` is the subscript of the monthly sales amount, with subscripts (1--12).

In this example, the OCCURS level of the items MONTHLY-SALES-AMOUNT is 2, while the OCCURS level of the item SALES-HISTORY is 1.

```
sales_monthly_sales_amount (table name)
product_code    index1    index2    monthly_sales_amount
------------    ------    ------    --------------------
BCS9992121      1         1         1075.36
BCS9992121      1         2         49397.90
BCS9992121      1         3         49375.93
BCS9992121      1         4         22840.97
BCS9992121      1         5         38984.02
BCS9992121      1         6         40039.84
BCS9992121      1         7         33875.93
BCS9992121      1         8         35000.22
BCS9992121      1         9         65876.52
BCS9992121      1         10        20402.55
BCS9992121      1         11        17575.00
BCS9992121      1         12        41938.74
BCS9992121      2         1         .
BCS9992121      2         2         .
BCS9992121      2         3         .
BCS9992121      2         4         .
BCS9992121      2         5         .
BCS9992121      2         6         .
BCS9992121      2         7         .
BCS9992121      2         8         .
BCS9992121      2         9         .
BCS9992121      2         10        .
BCS9992121      2         11        .
BCS9992121      2         12        .
BCS9992121      3         1         .
    .           .         .         .
    .           .         .         .
    .           .         .         .
    .           .         .         .
```

10.7.8 OCCURS DEPENDING ON

DMSII uses the DEPENDING ON clause (usually with COMPACT data sets) to conserve disk space. For COMPACT data sets, the DMSII work area always contains a fully expanded version of the record; however, the record is compacted when it is stored on disk. The exact syntax for OCCURS DEPENDING ON clause is as follows:

```
item_name OCCURS n TIMES DEPENDING ON depends_item_name;
```

The value *n* defines the maximum number of occurrences of the data item *item_name*, while the value of the depends item *depends_item_name* controls the number of occurrences of the item that are stored. This last number cannot exceed *n*. Information on an OCCURS DEPENDING ON clause is relayed to the Databridge Client, enabling the Databridge Client to suppress extraneous columns that do not actually exist. If the DEPENDS data item has a value of 3, and the OCCURS clause is OCCURS 10 TIMES, the last 7 columns are not included.

10.7.9 Handling Unflattened OCCURS DEPENDING ON Clauses

To handle a changing depends item, the Databridge Client uses before-image/after-image (BI/AI) pairs for data sets that have items with OCCURS DEPENDING ON clauses that are not flattened.

First, the Databridge Client checks the old and new values of the DEPENDS data item to determine how to execute the modify. The modify is handled in one of the following ways:

- If the value of the DEPENDS data item is unchanged, the Databridge Client updates the corresponding rows in the secondary tables as usual. (Redundant updates are suppressed if the `ds_options` bit DSOPT_Use_bi_ai is set.)

- If the value of the DEPENDS data item *increases* from *m* to *n*, the first *m* items are updated normally. The newly added items (*m*+1 through *n*) are inserted into the secondary table.

- If the value of the DEPENDS data item *decreases* from *m* to *n*, the first *n* items are updated normally. Items that are no longer present (*n*+1 through *m*) are deleted from the secondary table.

10.8 Relational Database Split Tables

A *split table* occurs when a DMSII data set record requires more than one table in the relational database to hold the data. Split tables occur in the following circumstances:

- When a table mapped from a DMSII data set has more than the maximum number of columns allowed by the relational database. The `maximum_columns` parameter in the configuration file allows you to reduce this value.
- When a relational database table's record size exceeds the Microsoft SQL Server maximum record size (approximately 8K -- the actual value depends on the number of columns in the table).

When the `define` (or `redefine`) command reaches the point where one of the above conditions is satisfied, it stops adding columns to the table (named the same as the DMSII data set). It then starts a new table that contains the same keys as in the original record of the primary table, followed by the remaining items in the data set at the point the split occurred. Note that there is always the possibility of having multiple splits for data sets that have a large number of columns. The flattening of OCCURS items can easily lead to split tables.

A split can occur in the middle of flattening an OCCURS clause, which can be rather awkward. In order to better control where a split occurs we added the `di_options2` bit DIOPT_Split_Here (4) that forces the table split to occur after the item in question is processed.

> ✎ **Note**
>
> When a DMSII data set is split into more than one relational database table, a WARNING message appears during a `define` or `redefine` command. In addition, each split table duplicates the keys in the original table.

10.8.1 Split Table Names

The new table is named using the original (parent) table name with a number (usually 1) appended to it to make it unique. All subsequent tables created from the same data set have the original table name with a numeric suffix that is incremented by 1 each time a new split table is created.

10.8.2 Keys for Split Tables

For a data set with keys, the keys of the original data set are duplicated in the split tables because you must access each of these tables individually. The process of splitting the data set into tables continues until there are no more data items left in the data set.

The following examples show the mapping of a data set that has 600 items (5 of which are keys) to a relational database that limits the number of columns in a table to 250. The result is tables that contain a total of 610 columns, where the 5 keys are duplicated across all 3 tables. If the original table is named savings, the remaining two tables are named savings1 and savings2, unless these names are already in use.

tablename	tablename1	tablename2
250 columns (first 5 are keys)	5 keys and 245 columns	5 keys and 105 columns

The five keys are duplicated in each table. To search these split tables, you must explicitly open each table. The tables are not automatically linked.

10.9 Relational Database Table and Column Names

When you clone a DMSII database, the Databridge Client names the relational database tables and columns the same as their equivalent DMSII data sets and data items. However, some differences exist. In this section, the differences between the names are explained.

10.9.1 Uppercase and Lowercase

All DMSII data set, data item, and set names are uppercase. These names are also stored in uppercase in the DATASETS and DMS_ITEMS Client control tables. Their equivalent relational database table, column, and index names are stored in lowercase in the DATATABLES and DATAITEMS Client control tables.

- All DMSII data set names are stored in the DATASETS Client control table in uppercase, just as they appear in the DMSII database. The equivalent relational database table name is converted to lowercase and is stored in the DATATABLES Client control table. Thus, a data set named CREDIT in the DMSII database is named credit in the relational database.

- All DMSII data item names are stored in the DMS_ITEMS Client control table in uppercase, just as they appear in the DMSII database. The equivalent relational database data item name is converted to lowercase and is stored in the DATAITEMS Client control table. Thus, a data item named LIMIT in the DMSII database is named limit in the relational database.

> ✏️ **Note**
>
> You must type these names in the correct case. If you are using the relational database table name as a character string value in a SQL statement (for example, `'tablename'`), you must use lowercase.

10.9.2 Hyphens and Underscores

The hyphen (-) in the DMSII name becomes an underscore (_) in the relational database name. The only exception is a data source name that is allowed to contain hyphens.

10.9.3 Name Length

The limit for a DMSII data set name is 17 characters, and DMSII item name is limited to 128 characters. Relational databases typically limit table names to 30 characters; however, the Databridge Client reserves two characters for the prefix of the stored procedure names for updating the table (i_ *tablename*, d_ *tablename*, u_ *tablename*). Thus, the table names are actually limited to 28 characters. Similarly, the Databridge Client adds a one or two character prefix to the item names to create a unique name for the parameters of the stored procedures. The Databridge Client for Microsoft SQL Server uses a prefix of @ while the Databridge Client for Oracle uses a prefix of p_. To avoid using names that are too long for the relational database, items names are limited to 29 characters for SQL Server or 28 characters for Oracle.

With this limit of 28 characters for a table name, typically all the DMSII names fit into the relational database table name or column name. In cases where data set, data item, or other structure names are concatenated and therefore become too long for a relational database, the Databridge Client truncates the name.

10.9.4 Duplicate Names

If two data sets have the same name in two different DMSII databases (or data sources, from the Client perspective), the Databridge Client appends the number 1 to the duplicate table name the first time it is encountered. If a table already exists with the duplicate name with "1" appended to it, the Databridge Client appends the number "2" instead and so on until a unique table name is created.

For example, if DMSII database A has a data set named PRODUCTS and DMSII database B also has a data set named PRODUCTS, the resulting Databridge table names would be products and products1.

If you combine this duplicate data set name convention with the convention for naming split tables (when one data set results in more than one table), you can have multiple suffixes for short names.

For example, if you have two data sources with a data set named CUSTOMER, which also generates split tables, the tables are renamed as follows:

- `customers` and `customers1` in the first data source
- `customers11` and `customers12` in the second data source (as the primary table was renamed customers1)

Duplicate item names may result in the following cases:

- When you use the same name for items in two different GROUPs. DMSII allows this, but the Databridge Client ignores GROUPs.
- When you truncate two long DMSII item names that are almost identical

The Databridge Client handles duplicate item names the same way that it handles duplicate table names.

10.9.5 Reserved Keywords

You cannot use reserved keywords for relational database object (table, column, index, etc.) names. For example, "order" is an SQL keyword; therefore, you cannot rename a relational database table or column as "order".

If an existing DMSII data set is named ORDER, the Databridge Client stores ORDER in the DATASETS Client control table and an equivalent relational database table called "order_x" in the DATATABLES Client control table. This same convention of adding "_x" to rename a table whose name is a reserved word applies to DMSII data items. For information on reserved words in your relational database, see the related database documentation.

The SQL Server Client allows you to use reserved words as object names, as long you enclose them in square brackets in SQL statement. The configuration file parameter `bracket_tabnames` allow you use reserved words like "order" as table names. Setting this parameter to True makes the Client use brackets around table names that are reserved words.

11. OCCURS Table Row Filtering

OCCURS tables are secondary tables generated by the Databridge Client when OCCURS clauses for items (or GROUPs) are not flattened. This is the default behavior of the Databridge Client. It involves creating a separate row in these tables for each occurrence of the item (or GROUP) with the keys of the primary table record duplicated and an additional column named `index1`, which contains the occurrence number (starting at 1), added to them. In the case of nested OCCURS clauses you end up with two tables, the first of which could be suppressed when you have nothing but keys in it (i.e. you have a GROUP within an OCCURS clause that contains only a GROUP, which also has an OCCURS clause). In the case of nested OCCURS clauses the second table has two columns named `index1` and `index2` added. These columns hold the occurrence numbers of the corresponding items (or GROUPS) within the OCCURS clauses.

Not all of the rows in such tables contain meaningful data, for this reason it is sometimes desirable to discard the ones with meaningless data. There are several advantages to doing this:

- It saves storage, as these secondary tables are quite expensive, particularly when the item with the OCCURS clause is a single item.

- The users of the database do not have to discard unwanted data when they fetch data from the secondary table.

- The number of updates is significantly reduced, resulting in better performance. This can further be improved by setting the `optimize_updates` parameter to true. This parameter only applies updates to rows that are actually changed. This avoids doing redundant updates, and can thus greatly improve performance. The process of discarding rows that do not contain meaningful data is done by defining a set of filters for such tables that describe the conditions under which the rows should be discarded. This requires having access the before and after images for updates, as a change in the data can affect whether the row is to be filtered or not. Since we already have the before and after images when doing filtering, enabling `optimize_updates` does not add any additional overhead, other than the comparison of the before image and after image data to determine if anything changed, which is a lot quicker than executing a redundant update (that is SQL that does not change anything).

11.1 Filter Source File

The implementation of row filtering for secondary tables derived from items with OCCURS clause does not involve changing any configuration file parameters. All you need to do is to create a text file that specifies the filtering conditions for all such tables that need to be filtered. We refer to this text file as the filter source file. This file normally resides in the **config** sub-directory of the data source's working directory.The filter source file, which is formatted in a somewhat similar manner to the row filtering sections of **GenFormat**, defines the filters for the various secondary tables using SQL-like statements. This file is then be compiled using a utility called **makefilter**, which is included in the Client files. The **makefilter** utility checks the syntax of the filter source file and validates all the specified table and column names. It then creates a binary file named "dbfilter.cfg" in the **config** sub-directory of the Client's working directory. This file then gets loaded and bound to the corresponding data tables and data items at the start of a Client `process` or `clone` command. The Client looks for the file "dbfilter.cfg" and loads it when it is present. The binding process replaces column numbers by pointers to the structures that hold the corresponding DATAITEMS control table entries. The Client uses a general purpose filtering procedure that interprets the filter pseudo code using the DMSII data buffer for the update and returns a result that indicates whether or not the row should be discarded. The Client can thus determine whether or not to insert (load in the case of data extraction) or update a row in the table. In the case of a delete we do not bother with filtering, we simply delete all rows that have the keys of the parent table record (i.e. for all values of `index1`). To make the Client run efficiently, we made it use host variables to do these sort of operations, which we refer to as DELETE_ALL operations (when using stored procedure we use the z_*tablename* stored procedure for this purpose). This means that besides INSERT, DELETE and UPDATE statements we also have compound DELETE statements for OCCURS tables (i.e. delete from *tabname* where *key1=val1* and ... *keyn=valn*; without specifying a value for `index1`).

11.2 The Filter File

The filter source file, which is modeled after the row filtering in **GenFormat**, uses a syntax that defines the conditions when a row is to be discarded, rather than when it is to be selected. The statements are free format and can extend over multiple lines, but they must be terminated by a semicolon. You can add comments using "// ...", which makes the scanner stop scanning the image before the slashes.

By using **delete** statements instead of **select** statements we make the "**where**" clause define the conditions under which a row is filtered out rather than selected. The reason for doing this, is that it is easier to follow (no need to use De Morgan's law). An example of a filter file source follows.

Sample Filter File
delete from customer_hold_information where hold_type = 0 or hold_type = 4;
delete from customer_account_abbr where account_abbr = " ";
delete from meter_readings where amount_read = NULL;

The **makefilter** program converts these filters into a list of tokens that contain all the required information for processing them using the general purpose filtering procedure that acts like a VM that executes the filter pseudo-code.

Any table that is not specified in the filter file will have no filter and will be treated normally. Filtering is limited to secondary tables derived from items with OCCURS clauses (a.k.a. OCCURS tables). We allow the testing for NULL by using "*column_name*= NULL" or "*column_name*!= NULL" (or "*column_name* <> NULL"), which is not proper SQL. If the item is ALPHA the fact that NULL is not in quotes is enough to distinguish it from the value "NULL". Unlike relational databases, NULL in DMSII is an actual value (typically all high values for numeric items and all low values for ALPHA items). All constants are stored in the data area of the filter using their corresponding DMSII representations. Character constants are automatically space padded. Numeric constants have leading zeroes added.

The 3 types of tokens involved in these expressions are **variables** (i.e. column names), **constants** and **operators**. Constants consist of a data type (that matches the first operand's type, which must be a column name), an offset into the filter's data area (the length is the same as that of the associated column name). We do not need the declared length, as all comparisons work at the character or digit level (we already do this when testing for NULL). Operators also include and end-of-statement indicator which corresponds to the semicolon in the pseudo-SQL statements in the filter source file. All comparisons must start with a column name and the second operand must be a constant or the word "**null**". Comparing two columns as a condition for filtering is not allowed. All object names are case sensitive and must be typed in lower-case, keywords and the word NULL are not case sensitive. String constants must be enclosed in double quotes (the use of single quotes is not currently supported).

> 🔥 **Important**
>
> String constants must be enclosed in double quotes (the use of single quotes is not currently supported).

In the case of a DMSII structural reorganization the filters must be recompiled if any of the data sets that have filters for secondary tables are affected by the reorganization. The Client automatically takes care of this by initiating a recompile of the filter at the end of `define` and `redefine` commands or an Administrative Console **Configure** command run, when there is filter file present in the **config** directory.

The changes to the Client itself are pretty straightforward and involve using the filter routine on the image to determine whether it gets discarded or not. The Client handles the situation where an item, that was not stored, needs to be stored after an update (in this case the Client does an INSERT). Similarly, it handles the situation where an item, that was being stored, needs to be discarded after an update (in this case the Client does a DELETE). The remaining cases are handled normally, if the item was discarded and still needs to be discarded, we do nothing. And if the item was stored and still needs to be stored we update it, unless `optimize_updates` is True, in which case we skip the update if the values of all the columns are unchanged.

The following table summarizes the supported operators and their relative precedence.

Level	Operators
1	=,>,<,>=,<=,!= (or <>)
2	AND
3	OR

The use of parentheses is allowed, but usually not usually necessary. There is no limit to the number of items that can be specified in the **where** clause, other than the actual number of data items that are not keys contained in the table.

The use of DMSII items whose data type is REAL are restricted to tests for NULL and 0 in filters. Items that are not nullable in DMSII cannot be tested for NULL. When using items whose data type is BOOLEAN you must use 0 or 1 in the constants (the use of TRUE and FALSE is currently not supported).

The **makefilter** program has two commands, `import` and `display`. The `import` command compiles the filter source file, which can be specified using the -f option, to create the binary filter file **dbfilter.cfg**. If no filter file is specified the command tries to use the file **dbfilter.txt** in the config subdirectory of the data source's working directory. The `display` command produces a report that describes the content of the binary filter file. All **makefilter** log output is written to the file "*prefix*_flt_*yyyymmdd*[_*hhmmss*].log" keeping it separate from the Client log files.

12. Databridge Client Control Tables

This chapter describes the six Client control tables and the properties in each table that can be customized. For best results, use the Administrative Console's **Customize** command and other commands to customize your Client control tables.

Control tables do not contain replicated DMSII data. To store replicated data, the relational database uses data tables, which are created using information from the control tables. The control tables hold the layout information of the DMSII database (from the DMSII DESCRIPTION file) and the layout of the corresponding relational database tables. Each relational database has one set of Client control tables that includes the following tables: DATASOURCES, DATASETS, DATATABLES, DMS_ITEMS, DATAITEMS and AF_STATS. The AF_STATS table, which was added in version 7.0, is used to optionally store the audit file statistics that allow users to look at the audit file processing statistics for the last 9999 audit files.

The Databridge Client uses several columns of the control tables to determine how DMSII database objects are represented in the relational database layout. While Databridge makes many of these decisions, some properties can be customized by using the Administrative Console's Customize command or user scripts. For example, you can rename columns, combine like items, and flatten OCCURS.

For instructions on backing up the Client control tables, see Backing Up and Maintaining Client Control Tables.

12.1 DATASOURCES Client Control Table

The DATASOURCES Client control table contains the data sources defined for the Databridge Client. Each data source represents both a connection to a Databridge Server on the MCP (or Enterprise Server) and a DMSII database. You can define more than one data source within the DATASOURCES Client control table as long as each data source name is unique. All of the data sources you define within the DATASOURCES table apply to one relational database.

> ✏️ **Note**
>
> Do not modify Client control tables directly. Instead, use the commands in the Administrative Console's **Settings** menu to customize data sources.

The following table contains descriptions of each column, in the order in which it appears in the DATASOURCES table.

Column	Description
data_source	This value is the name you give the data source when you use the Databridge Client `define` command. The name can be a maximum of 30 characters, and it must match one of the following: • The entry for SOURCE in the DBServer control file on the host. • A base or filtered source as defined for Enterprise Server.
hostname	This column specifies the host name or IP address of the Databridge Server.
hostport	This column specifies the TCP/IP port number used to connect to the Databridge Server.
hostprot	Reserved
stop_time	This column specifies the start of the Databridge Client blackout period expressed as an integer value representing 24-hour time (hhmm format). At a few key points during execution, the Databridge command-line Client (**dbutility**) tests this column to determine whether or not it should continue processing. The configuration file parameter `controlled_execution` enables this feature while the parameter `min_check_time` specifies the minimum delay time (for example, a typical time value may be 5 minutes) between checks of `stop_time`. The program checks at the start of a `process` or `clone` command and after commits, provided enough time has elapsed since the last check. **NOTE**: Service-based operations ignore the value of this column, as the service takes over this functionality.
end_stop_time	This column specifies the end of the blackout period for the Databridge command-line Client (**dbutility**). It is expressed as an integer value representing 24-hour time (hhmm format). For example, if `stop_time` is 2000 and `end_stop_time` is 200, the Databridge Client refrains from running between 8:00 p.m. and 2:00 a.m. **NOTE**: Service-based operations ignore the value of this column, as the service takes over this functionality.
update_level	This column contains the update level of the DMSII database at the time the last `define` or `redefine` command was run.

Column	Description
status_bits	This column contains a set of bits that the Databridge Client sets. Some of these bits contain state information that is useful. Modifying this column can disrupt the Client operations. The following list covers the columns in the DATASOURCES control table that can be modified in user scripts. **Bit and Description** • **1-256**: For internal use only. • **512**: SRC_NotBackedUp - When this bit is set, an unload file is created to ensure that the data source backup is not overwritten; the bit is then cleared. After the Client resumes audit file processing and a transaction group is successfully processed, this bit is set. • **1024**: SRC_FileXtract - This bit indicates that the data source is a FileXtract file rather than a DMSII database. • **2048**: SRC_ITRANS - This bit echoes the value of the DMSII database's INDEPENDENTTRANS flag. • **4096**: Reserved • **8192**: SRC_DBEnterprise - When this bit is set, it indicates that the data source is an Enterprise Server data source, versus a DBServer data source. • **16,384**: Reserved • **32,768**: Reserved • **65,536**: Reserved • **131,072**: SRC_Upgraded - This bit is set by the **dbfixup** utility, when there are OCCURS clauses that are not flattened, to indicate that we have just done an upgrade to the 6.6 (or newer) Client software. Upon seeing this bit set the Client automatically executes a `refresh` command to create the *z_tablename* stored procedures that are used to do deletes in OCCURS tables. The Client clears this bit if the `refresh` command is successful. This avoids getting SQL errors when the Client tries to use these procedures, which did not previously exist. • **266,144**: SRC_RequiredAware - This bit is set by the 6.6 (or newer) Clients when a `define` command is executed. It indicates that the Client should honor the REQUIRED property in the DASDL and set the corresponding items not to allow nulls (NOT NULL). The purpose of this bit is preserve backward compatibility by preventing data sources created by older Clients from having all the REQUIRED items changed to not allow nulls.
tab_name_prefix	This column holds an optional one to eight-character prefix which is added to all table names in the data source. This prefix, which you must supply, allows you to distinguish between identically named data sets in multiple data sources, without having the `define` and `redefine` commands rename tables to eliminate name conflicts. The configuration file parameter `use_column_prefixes` extends this prefix to all column names.
data_source_id	This column allows you to provide a numeric identifier to distinguish records that belong to a particular data source from other records in a multi-source environment using a user script, or a relational database query tool. In addition, you must set the `external_columns` column to 128 or 2048 for all the data sets belonging to this data source.
last_run_status	This column holds the exit code of the last Databridge Client `process` or `clone` command that was run. When the exit status is not available (such as when the Databridge Client is running or abended), the entry has a value of 9999.
stop_afn	This column specifies the AFN value when the configuration file parameter `stop_after_given_afn` is enabled. **NOTE**: Service-based operations ignore the value of this column, as the prefered way of doing this is use a stopper program combined with the `stop_task` parameter in the Client.

Column	Description
af_origin	This column specifies the origin of the current audit file being processed. The following values are defined for this column: • 0: Audit file processed by Databridge Engine • 1: Reserved • 2: Audit file processed by Enterprise Server using Databridge Engine to access regions. This is referred to as "remote regions" (or "indirect disk"). • 3: Audit file processed by Enterprise Server using direct disk I/O. This is referred to as "direct disk" and is the most economical way to process audit files in terms of host resource utilization. • 4: Cached audit file processed by Enterprise Server.
server_version	This column indicates the version of DBServer last used by the Client.
engine_version	This column indicates the version of Databridge Engine last used by the Client.
support_version	This column indicates the version of the Support Library last used by the Client.
dbe_version	This column indicates the version of Enterprise Server last used by the Databridge Client.
client_version	This column indicates the version of the last **dbutility** or **DBClient** that was run for this data source
cfgsrvr_version	This column indicates the version of **DBCIntCfgServer** that was last used by the service to access the data source.
service_version	This column indicates the version of the **DBCIntControl** service that launched a Client run for the data source.
old_update_level	This column holds the previous value of update level when running when a `redefine` command is run. This value is used to name the reorg scripts that contain the DMSII database's update level.
db_timestamp	This column contains the timestamp of the DMSII database, which is the time when the database was created. It is used by the Client to verify that the Client is using the same database as it originally was. If this column is set to zeroes, then this test is not performed. CAUTION: This column contains a DMSII TIME(6) value, which is binary and 6-bytes long. For SQL Server, set db_timestamp=0. For Oracle, set db_timestamp ='000000000000'.
reader_info	This column contains the name and version of the reader used to read audit files (or flat files in the case of a FileXtract data source).
dms_dbase_name	This column contains the name of the DMSII database, which is not always the same as the data source name.

> ✎ **Note**
>
> The data source CTLTAB_VERSION in the DATASOURCES table is a special entry created by the Databridge Client. It indicates the version of the Client control tables. Do not try to process this data source, and do not remove it from the table.

12.2 DATASETS Client Control Table

The DATASETS table contains information about each DMSII data set as permitted by the Databridge host support library filters. The DATASETS table contains state information for each data set visible to the Client, including the current replication phase of the data set. When the data has been successfully extracted, this table includes the location in the audit trail from which the last group of updates for the data set were read, including the audit file number, the audit block sequence number, the segment number, and the index that identify the physical location of the block in the audit file, and a timestamp.

> ✏️ **Note**
>
> The Service Pack 1 Oracle Client uses the data type TIMESTAMP, which supports fractions of seconds, for the `audit_ts` column. To use this data type instead of DATE, set the `sql_type` to 20 and specify the number of fractional digits desired using the `sql_length` column for the user column. You can use the DATETIME2 data type, which also has a `sql_type` of 20, for these columns in the SQL Server Client and use the `sql_length` to specify the number of fractional digits desired. TIMESTAMP supports 0 to 9 fractional digits, while DATETIME2 support 0 to 7.

The `active` column of the DATASETS table controls the selection of *all* tables mapped from a DMSII data set. (The SQL statements in your user scripts use the `active` column in this table to specify data sets you do not want to clone.) If you use the DATASETS table to disable cloning for a data set, you disable cloning for all tables related to that data set.

For example, one DMSII data set with a nested OCCURS item can generate multiple tables. If you do not want to clone any of these tables, use the `active` column in the DATASETS Client control table to turn off cloning. For more information on selectively cloning data sets, Tips for More Efficient Cloning.

The following table contains descriptions of each column in the DATASETS Client control table. Included is the abbreviated column name that the `display` command writes to the log file.

Column name	Display	Description
data_source		This column contains the name of the data source that identifies the DMSII database from which the data was taken. The name must match one of the following: • The entry for SOURCE in the DBServer control file on the host. • A base or filtered source as defined for Enterprise Server.
dataset_name	ds	This column contains the name of the DMSII data set.
rectype	/type	This column, which is zero for all fixed-format data sets, contains the record type of a DMSII variable-format data set as follows: **Record Type and Description** • 0: For a variable-format data set, this represents records that have no variable part. • 1−254: Represents the variable-format record type as defined in the DASDL.
set_name	set	This column contains the name of the DMSII set that Databridge Engine uses as an index source for tables mapped from the data set. The names "aa_set", "user_set", and "pk_set" are special set names that the Databridge Client uses when a DMSII set is not available. The name "aa_set" indicates that AA Values (or RSNs) will be used as the source for the index. The name "user_set" indicates that the set is user-defined. The name "pk_set" indicates that the set is defined in GenFormat using the PRIMARY KEY statement.
active	A	During a `define` or `redefine` command, this column determines whether or not a data set is mapped. During a `process` command, the value of this column determines if the data set is to be selected for cloning or updating. The default, 1, indicates that the data set will be mapped (cloned or updated). A value of 0 indicates that the data set will not be mapped (cloned or updated). The `define` and `redefine` commands change the value in the `active` column to 0 for the global data set, the restart data set, and remaps (unless the base structure is filtered out) in the DMSII database. **NOTE:** When you change the DATASETS `active` column value to 0 to disable cloning, all tables related to the data set are disabled. For example, if a DMSII data set is represented by three relational database tables, none of the three relational database tables will be cloned.
strnum	ST#	This column contains the DMSII data set's structure number.

Column name	Display	Description
audit_filenum	AFN	This column contains the current DMSII audit file number. DMSII audit files are created and stored on the host; they contain a record of all updates to the DMSII database and are named as follows: *databasename*/AUDIT*nnnn* where *databasename* is the name of the DMSII database, AUDIT is a literal, and *nnnn* is the AFN (audit file number) whose value is a number between 1 and 9999. Before you run a `process` command to clone a DMSII database, the audit file number (and all the other audit file location information) is zero; subsequent `process` commands fill these records with the ending audit file location.
audit_block	ABSN	This column contains the audit block serial number in the audit file. Because DMSII uses 32-bit audit block serial numbers, the data type for this column is binary (raw in Oracle). All displays in the log file show this value as a 10-digit unsigned number. If you access this column via a relational database query tool, the hexadecimal value appears instead, as the column's data type is **binary(6)** in SQL Server and **raw(6)** in Oracle).
audit_seg	SEG	This column contains the segment number within the audit file.
audit_inx	INX	This column contains the index within the audit file segment.
audit_ts	Time Stamp	This column contains the audit file timestamp represented as a relational database date/time (datetime2 for the SQL Server Client and timestamp for the Oracle and Kafka Clients).

Column name	Display	Description
ds_mode	M	There are a few instances where you may need to change the mode. The `ds_mode` value provides the following information about the data set: **Value and Description** • 0: The data set is ready to be cloned; all tables and stored procedures are dropped and recreated. • 1: The data set is in the fixup phase; data extraction is complete and the table is being updated with changes that occurred during the extraction. The integrity of the data in the tables mapped from the data set is not guaranteed to be correct until the fixup phase is complete. • 2: The data set is ready to be updated. This implies that it has already been cloned and the fixup has been completed. This is the most common mode. • 10: An error occurred during the dat extraction. • 11: An error occurred during index creation or the tables mapped from this data set do not have an index defined. • 12: The data set is using AA Values as keys, and the AA Values are no longer valid because the data set has been reorganized. • 31: The data set must be reorganized and the `redefine` or the Administrative Console's **Customize** command has created scripts to make the relational database table match the DMSII data set. You must run the `reorganize` command in order to run the reorganization scripts created by the `redefine` command. • 33: The `reorganize` command failed for this data set. In this case, you must manually update the table by trying to fix the failed script. Then, set `ds_mode` to 31 and repeat the `reorganize` command. If that fails, you must re-clone it.
host_fmtlevel		This column contains the format level as seen by the host. The value is the update level received from Databridge Engine in the last STATEINFO record.
client_fmtlevel	FMT	This column contains the format level as seen by the Databridge Client. The value is determined by the `define` and `redefine` commands. Typically, the host and Client format levels are the same until a DMSII reorganization is detected.
recsz_bytes	RSZ	This column contains the size of the record in bytes.
parent_strnum	P#	This column contains the parent structure number. This column is used for embedded data set information.
num_children	#C	This column contains the number of child structures for the parent structure. This column is used for embedded data set information.
base_strnum	B#	This column contains the base structure number. If the value in this column is not equal to the value in the strnum column, this data set is a remap of the data set whose structure number is base_strnum.

Column name	Display	Description
subtype	ST	This column contains the structure subtype of the DMSII data set: **Value and Description** • 0: Standard data set • 1: Random data set • 2: Ordered data set • 3: Unordered data set • 4: Global data set • 5: Direct data set • 6: Compact data set • 16: Restart data set • 17: Virtual data set
in_sync		The `in_sync` column tracks data sets whose stateinfo is synchronized with the stateinfo stored in the Global_DataSet row for the data source in the DATASETS Client control table. Global_DataSet is a dummy data set that holds the common stateinfo for data sets whose ds_mode is 2. When the Databridge Client is finished updating, the stateinfo in the Global_DataSet is copied to all data sets that need to be updated with the stateinfo. Values in this column indicate the following: **Value and Description** • 0: The data set stateinfo is current • 1: The data set stateinfo must be corrected at the end of update processing to reflect the stateinfo as it is stored in the Global_DataSet
item_count	ICNT	The value in this column represents the number of items in the DMSII data set and is used by Databridge Engine to detect filler substitutions and changes in DBGenFormat.

Column name	Display	Description
audit_time6		The value in the `audit_time6` column is the DMSII timestamp stored as 6 binary characters. The Client uses this value when it sends state information (stateinfo) to Databridge Engine at the beginning of a `process` command. The Client does not use the value in the `audit_ts` column, as it is not accurate enough to use in communications with the Databridge Engine. Instead, the original DMSII timestamp is used. It is much more accurate and has a much smaller granularity than relational database date/time values. **CAUTION:** When you enter values for the stateinfo, you must set the `audit_time6` column to 0 because the Databridge Engine uses this value to detect DMSII rollbacks. If the value of the timestamp is 0, Databridge Engine bypasses this test. For SQL Server, set audit_time6=0. For Oracle, set audit_time6='000000000000'.
host_info		The information in the `host_info` column is provided by the Databridge Engine during data extraction. It enables the Databridge Client to recover fixups if the command is aborted. This information is stored as 6 binary characters.

ds_options	OP	The following bits (which can be set through customization user scripts or by using the Administrative Console's **Customize** command) control how data sets are mapped:

Bit and Description

- **1:** DSOPT_Use_bi_ai - This bit is set by the `define` command for data sets that have OCCURS clauses that were not flattened when the configuration file parameter `optimize_updates` was set to True. This bit causes the program to request that the Databridge Engine send all updates the data set, involving a key change, as BI/AI pairs. You can set this bit to 0 via user scripts if you want to disable optimization of updates for this data set.

- **2:** DSOPT_No_Loader - This bit causes the Databridge Client not to use the bulk loader during the data extraction phase of this data set. It is effectively a localized form of the `/s` option (which applies to all data sets).

- **4:** DSOPT_No_StoredProcs – This bit causes the Databridge Client not to use stored procedures when doing updates. Updates still use host variables, but instead of generating a stored procedure call, the Client generates the actual SQL statement to do the update.

- **8:** DSOPT_Save_Updates - This bit causes the Databridge Client to generate history tables for all tables that are mapped from the data set.
  To determine whether the history tables are populated with clone data only or clone and update data, see history_tables.

- **16:** DSOPT_Include_AA - This bit is deprecated and should not be used to force the Client to use AA Values (RSNs) as the source for the index. Use the bit DSOPT_Use_AA_Only instead.

- **32:** DSOPT_Ignore_Dups - When set, this bit has exactly the same effect as the configuration parameter `suppress_dup_warnings`, except that it only applies to the individual data sets for which it is set.

- **64:** DSOPT_Select_Only - This bit inhibits the creation of tables and stored procedures for the data set. It is used for data sets that provide input to virtual data sets and are not otherwise mapped to any tables.

- **128:** DSOPT_Keep_Null_Alpha_Keys - This bit indicates that the program should treat NULL alpha keys as blanks instead of discarding such records.

- **256:** DSOPT_Supp_New_Columns - This bit, which is initially set to reflect the value of the `suppress_new_columns` parameter for the corresponding data set, can be modified via user scripts or the Administrative Console's **Customize** command. The `redefine` command uses this bit when determining how to handle new columns.

- **512:** DSOPT_MultiInput - When the `automate_virtuals` and `miser_database` parameters are enabled, this bit indicates that data for the virtual data set comes from more than one real data set.
  When this bit is set, the Client tests the `res_flag` column (identified by a `dms_subtype` value of 255) before executing the stored procedure i_*tablename*. If the flag is set, the insert is done normally; otherwise, the stored procedure r_*tablename* is called to update the `res_flag`. If the update fails, an insert is performed instead.

- **1024:** DSOPT_MultiSource - This bit indicates that the tables generated from the data set get their input from more than one data

Column name	Display	Description
changes		These bits are used by the `redefine` and the Administrative Console's **Customize** command to indicate the changes that the command detected.

Bit and Description

- **0:** Description
- **1:** CHG_new - New entry
- **2:** CHG_modified - Modified entry
- **4:** CHG_del_before - One or more entries before this one were removed.
- **8:** CHG_del_after - One or more entries after this one were removed.
- **16:** CHG_format_level - The data set's format level changed (that is, a DMSII structural reorganization that affects this data set has occurred).
- **32:** CHG_item_count - The data set's item count has changed (that is, a filler substitution reorganization has occurred).
- **64:** CHG_user_change - There were user changes to the DMS_ITEMS or the DATAITEMS tables (that is, the layout has changed) as a result of actions by the user rather than a DMSII reorganization.
- **128:** CHG_links_change - DMSII links changed for the data set.
- **256:** CHG_AA_values_changed - This bit indicates that the data sets AA Values are no longer valid. The bit is set by the `redefine` or the Administrative Console's **Customize** command but is otherwise not used by the Client.
- **1024:** CHG_deleted - The item was deleted.

Column name	Display	Description
status_bits	SB	The following bits are used by this column:

Bit and Description

- **1**: Description

- **1**: DS_Needs_Mapping - This bit indicates that the data set has not been mapped. All data sets that have their corresponding `active` column set to 0 in the data set selection script `script.user_datasets.datasource` also have this bit set. If you decide to clone such a data set, you must set the `active` column to 1 and run a `redefine` or the Administrative Console's **Customize** command to perform the mapping.

- **2**: DS_Needs_Generating - This bit indicates to the `generate` command that the scripts for the data set need to be generated. Note that the `generate` command only generates scripts for data sets that have this bit set. The `define`, `redefine` and the Administrative Console's **Customize** command automatically set this bit.

- **4**: DS_Needs_Remapping - This bit forces the `redefine` or the Administrative Console's **Customize** command to refresh the mapping. After you make changes to the data table user_define customization scripts, you may want to set this bit before you execute a `redefine` command.

- **8**: DS_Needs_Redefining - This bit is automatically set by the `process` and `clone` commands when Databridge Engine detects a structural reorganization or a filler substitution for the data set. You can set this bit to force the `redefine` command to refresh the DMSII layout.

- **16**: reserved

- **32**: reserved

- **64**: This bit indicates that the AA Values are invalid. Do not modify this value.

- **128**: This bit indicates that the index creation failed. Do not modify this value.

- **256**: This bit indicates that the data set is in fixup mode. Do not modify this value.

misc_flags	MISC	This column contains an integer that holds a series of flags set by Databridge to reflect some characteristics of the individual data sets.

NOTE: Do not change these bits.

Bit and Description

- **1-8:** Description
- **16:** DSFLG_Links - This flag, set by the Databridge Engine in response to a DB_DataSets remote procedure call (RPC), indicates that the data set has DMSII links to other data sets.
- **32:** DSFLG_Altered - This flag, set by the Databridge Engine in response to a DB_DataSets RPC, indicates that the data set was altered by the support library.
- **64:** DSFLG_Static_AA - This flag, set by the Databridge Engine in response to a DB_DataSets RPC, indicates that the Databridge Engine is using RSNs (record serial numbers) in place of AA Values. RSNs are only available in a DMSII XE system where each record in a data set is assigned a unique serial number. Using the RSN in place of AA Values eliminates the need to re-clone tables after a DMSII garbage collection reorganization.
- **128:** DSFLG_Valid_AA - This flag, set by the Databridge Engine in response to a DB_DataSets RPC, indicates that the data set has valid AA Values. Not all data sets have valid AA Values. For details, see Composite Keys.

  NOTE: This bit does not apply to RSNs, which are always valid; it applies to the AA Values.

- **256:** DSFLG_Has_Occurs - This flag indicates that the data set contains items with unflattened OCCURS clauses. The program uses this bit in conjunction with the `optimize_updates` parameter to determine whether the DSOPT_Use_bi_ai bit in the `ds_options` column should be set. The DSOPT_Use_bi_ai bit can be reset by the user to prevent the use of before/after images for data sets where this action offers no significant performance improvements (for example, an OCCURS 2 TIMES clause is probably not worth optimizing).
- **512:** DSFLG_Uses_AA_values - This flag indicates that the data set uses AA Values as keys. The program uses this flag to avoid having to look at the table's columns to determine whether AA Values are used.

  NOTE: This bit is not set when the Databridge Client uses RSNs instead of AA Values.

- **1024:** DSFLG_Has_Links - This flag indicates that the data set has active DMSII links. This bit can be zero if all the links have their `active` columns set to 0 in DMS_ITEMS.
- **2048:** DSFLG_Is_LinkedTo - This flag indicates that one or more data sets have active DMSII links that use AA Values as foreign keys to point to this data set. The program uses this information to force the AA Values to be used as the keys for the tables derived from this data set.
- **4096:** DSFLG_Occ_Depends - This flag indicates that the data set contains items with unflattened OCCURS DEPENDING ON clauses. The program uses this bit to request that the Databridge Engine send updates to this data set as before/after images, regardless of the value of DSOPT_Use_bi_ai bit in `ds_options` for this data set.

Column name	Display	Description
max_records	MAXRECS	This column contains an integer that holds the maximum row count of the data set as *estimated* by the Databridge Engine. This is the exact number that appears in DBLister reports. The Databridge Engine computes this estimate by dividing the file size by the record size. This value is very inaccurate in the case of variable-format data sets because it is impossible to determine how many records of a given type exist in the data set without doing a full scan of the data set.
virtual_ds_num	VDS	This column contains an integer value that holds the structure number of the virtual data set to which the DMSII data set is linked. This column is used by the parent data set to point to the associated virtual data set. When more than one virtual data set is derived from a real data set, these data sets are chained together using this column.
real_ds_num	RDS/type NOTE: This display name is combined with the real_ds_rectype value.	This column contains an integer that holds the structure number of the primary real data set from which the virtual data set is derived. When more than one virtual data set is derived from a real data set, these data sets all point back to the real data set through their real_ds_num column. These real data sets are chained together, starting with the primary data set, by using the otherwise unused real_ds_num columns of the actual data sets.
real_ds_rectype		The integer in this column represents the record type of the variable-format data set. This information serves to further identify a variable-format data set when it is cloned as a virtual. In addition, the variable-format data set is linked to the virtual data set through the virtual_ds_num and real_ds_num columns.
external_columns	EXTC	This column contains an integer value that determines which predefined non-DMSII columns are automatically added to this data set. For a description of these bits, see Numeric Date and Time in Non-Contiguous Columns.
ds_user_bmask		This column, which shadows the ds_options column, contains a bit mask that represents the columns in ds_options that were customized. This column is used by the redefine or the Administrative Console's **Customize** command to restore the portion of ds_options that has been customized while leaving the remaining bits intact.
links_sz_bytes		This column contains the size of the link data, in bytes. Link data is no longer stored in the actual record, instead the record is extended by the size of the link data where the link data is placed during data extraction. These areas are not necessarily contiguous in the DMSII record; the DMSII offsets have been adjusted to make them to look contiguous in the Client.

Column name	Display	Description
links_offset		This column is used by the Client to determine where the link area for the record starts.
vp_link_offset		Variable format data sets have links in both the fixed part and the variable part, causing the Client to receive two LINK_AI records. This offset value indicates where the second part of the links area starts. By comparing the offset received from the Engine, the Client can tell where the link data should be stored.
item_name_prefix		This column is used by the Client to automatically strip fixed size prefixes from data item names. One frequently finds DMSII databases where the data set names (or a shortened form of these names) is used as a prefix for every item name. The Client has the ability to get rid of these prefixes without requiring any complex actions other than putting the prefix to be stripped in this column, without the trailing dash.
rows_extracted		This column is used by the Client to save the number of DMSII records that were received during the data extraction phase.
client_discards		This column, which is currently only used by the Flat File Client, is used to record the number of records discarded during data extraction phase.
extract_priority		This column is used to affect the order in which data sets are extracted. The data extraction is now ordered by `extract_priority` (highest value first) and `strnum` (lowest value first). By setting this column to a positive number you can change the order in which data sets are extracted.
dflt_options		This column is used to keep track of the initial state of the `ds_options` bits before any customizations are performed. It allows the Client to clear the `ds_user_bmask` bits that were set by customizations that were later undone.

12.3 DATATABLES Client Control Table

The DATATABLES Client control table is used primarily to disable cloning for one or more of the secondary tables mapped from one DMSII data set. For example, a DMSII data set with several OCCURS items generates multiple relational database tables. If you do not want to clone particular secondary tables, use the active column in the DATATABLES Client control table to turn off cloning for those secondary tables.

The DATATABLES Client control table contains the entries for each of the relational database tables mapped from the DMSII data sets listed in the DATASETS table. These entries include relational database information rather than DMSII information. For example, the DMSII data set name (in the column named `dataset_name`) is listed along with the corresponding relational database table name (in the column named `table_name`). Since a data set can be mapped to several relational database tables (such as when a data set contains OCCURS items), the `prim_table` column is used to identify the primary table.

The following table contains descriptions of each column in the DATATABLES Client control table. Included is the abbreviated column name that the `display` command or the Administrative Console's **Log Control Tables** command that can be found in the data source's **Advanced** menu.

Column	Display	Description
data_source		This column contains the name of the SOURCE name that identifies the DMSII database from which this data was taken. The data source name is defined when you run a `define` command. It must match the data source name in the DBServer control file on the host.
dataset_name	`ds`	This column contains the name of the DMSII data set from which this table was mapped.
table_name	`table name`	This column contains the name of the table as it appears in the relational database. DMSII data sets correspond to relational tables.
index_name	`index`	This column contains the name of the relational database index that is created for fast access to this table. If the table has no index, this column is blank. This index is created via the Databridge Client script named `script.index.tablename`. The value in this column is the index name used in the CREATE INDEX SQL statement.
rectype	`/type`	This column, which is zero for all tables mapped from fixed-format data sets, contains the record type of a DMSII variable-format data set. **Record Type and Description** • **0**: For a variable-format data set, this represents records that have no variable part. • **1-254**: Represents the variable-format record type as defined in the DASDL.
occurs_level	`occ`	This column contains the nesting level of OCCURS in the DMSII database. For example, an OCCURS table created from another OCCURS table has an occurs_level of 2. The original OCCURS table has an occurs_level of 1. NOTE: The Client does not support occurs levels that are greater than 2.
table_number	`T#`	This number is used by the SQL*Loader and bcp scripts. The Databridge Client assigns consecutive numbers to the tables it defines for a data source during the `define` command. Each table within a data source has a unique table number, and the numbers begin with 1. The `redefine` command and the Administrative Console's Customize* command assign numbers to new tables starting with the highest table number plus 1. Existing tables get their old table numbers restored.
active	`A`	The value of this column determines whether or not a table is cloned during a `process` or `clone` command. The default is 1, which indicates that the table will be cloned. If you change this value to 0, the table is not cloned. To disable cloning for an entire set of tables related to a DMSII data set, see DATASETS Client Control Table.
create_suffix	`suf`	The `create_suffix` column enables you to specify a value that identifies the index of the `create_suffix` string defined in the configuration file. For more information, see create_table_suffix in Generate Command Parameters.

Column	Display	Description
index_suffix		The `index_suffix` column enables you to specify a value that identifies the index of the `index_suffix` string defined in the configuration file. For more information, see create_index_ suffix in Generate Command Parameters.
original_name	original_name	The Databridge Client saves the original name of the renamed tables in this column so they can be identified during `redefine` and the Administrative Console's **Customize** commands.
prim_table	P	This column indicates whether or not this is a primary table.

| dt_options | OP | The `dt_options` column uses the following bits: |

Bit and Description

- **1:** DTOPT_Table_Renamed - The table was renamed by the user. This bit is used by the Client configurator to preserve the name change.

- **2:** DTOPT_Index_Renamed - The table's index was renamed by the user. This bit is used by the Client configurator to preserve the name change.

- **4:** DTOPT_User_Table - This table was created by the user. Not supported in Databridge Client 7.0.

- **8:** DTOPT_No_aux_stmts - This option inhibits the use of auxiliary statements for a given table during a `process` or `clone` command when the configuration parameter `aux_stmts` (default 100) is not zero.

- **16:** DTOPT_Occ_Depends - This option, automatically set by the Client during a `define` or a `redefine` command or the Administrative Console's **Customize** command, indicates that an OCCURS table (`occurs_level > 0`) contains an item with an OCCURS DEPENDING ON clause. This bit is used during update processing to properly handle cases where the value of the `dms_depends_num` item of an OCCURS DEPENDING ON clause changes.

- **32:** DTOPT_All_Keys - Indicates that all columns in the table are keys. Do not change this value.

- **64:** DTOPT_No_Unique_Key - Indicates that the table does not have a unique index and therefore it cannot be tracked unless a composite key is created. Do not change this value.

- **128:** DTOPT_Preserve_Deletes - Do not change this value.

- **256:** DTOPT_HistoryTable - This option, which is set by the `define` and `redefine` commands or the Administrative Console's **Customize** command, indicates to the Client that this table is a history table and that all records should be treated as inserts into the history table.

  CAUTION: Clearing this bit can corrupt history tables because it causes the Client to treat records as creates, deletes, and modifies instead of inserts.

- **512:** DTOPT_UserSP - Indicates that the table uses the stored procedure m_*tablename* to perform customized functions instead of using the procedure i_*tablename* for an insert. This procedure is used to merge records rather than insert them into the table.

  This bit is used in Miser databases.

- **1024:** DTOPT_Clustered_Index - This option, which only applies to the SQL Server Client, tells the Databridge Client to create a clustered index for this table. You can globally set this option via the `use_clustered_index` parameter. See use_clustered_index.

- **2048:** DTOPT_Primary_Key - This option tells the Databridge Client to create a primary key (instead of a unique index) for this table. When creating the script to create a primary key constraint, the Microsoft SQL Server Client uses the value of the DTOPT_Clustered_Index to determine whether to add the NONCLUSTERED clause to the SQL. If this second option bit is not set, the NONCLUSTERED clause is added. You can set this option globally via the `use_primary_key` parameter.

- **4096:** `DTOPT_Delete_Seqno` - This option is automatically set by the `define` or `redefine` command or the Administrative Console's **Customize** command when the `delete_seqno` mask is set in the `default_user_columns` parameter value.

- **8192:** DTOPT_Table_Split - This option is automatically set by the `define` or `redefine` command or the Administrative Console's **Customize** command when the table is part of a split table. Do not modify this value.

- **16,384:** DTOPT_ConcatItems - This bit is automatically set by the `define`

Column	Display	Description
changes		These bits are used by the `redefine` command and the Administrative Console's **Customize** command. **1**: CHG_new - New entry • **2**: CHG_modified - Modified entry • **4**: CHG_del_before - One or more entries before this one were removed. • **8**: CHG_del_after - One or more entries after this one were removed. • **32**: CHG_index_changed - This bit indicates that the table's index changed. The `reorganize` command uses this bit as an indication that it must drop the index for the table and recreate it. • **64**: CHG_IndexType_changed - The index type changed primary key versus unique index or in the case of SQL Server unique index versus clustered index. • **128**: CHG_IndexName_changed - The index name changed, the Client needs to drop the index using the old name and create the new index using the new name. The old name is saved in a temporary file by the `redefine` command or the Administrative Console's **Customize** command. The `reorganize` command deletes this file once the new index is successfully created • **256**: CHG_new_hist_tab - History tables were added for the data set. The `redefine` command ot the Administrative Console's **Customize** command sets this bit when they find a new history table. The `reorganize` command then can create these tables and we can continue processing without re-cloning the data set.
update_count		The `update_count` column represents the smoothed average number of updates for the table over the specified period of time.
update_interval		This `update_interval` column represent the period of time (in milliseconds) that the `update_count` spans. The multi-threaded update code uses the `update_count` column to balance the thread load. The `update_interval` column will start out as 0 and increase until it reaches the value representing one hour, after which it never changes, as the average is smoothed to reflect the number of updates for each table over the last hour. The `update_count` and `update_interval` columns were added to hold the table update statistics.
dt_user_bmask		This column, which shadows the `dt_options` column, contains a bit mask that represent customized columns in `dt_options`. This column is used by the `redefine` and the Administrative Console's **Customize** commands to restore the portion of `dt_options` that has been customized while leaving the remaining bits intact.
dflt_options		This column is used to keep track of the initial state of the `dt_options` bits before any customizations are performed. It allows the Client to clear the `dt_user_bmask` bit that were set by customizations that were later undone.

12.4 DMS_ITEMS Client Control Table

The DMS_ITEMS table contains entries for each DMSII item that is visible to the Client after column filtering in the Support Library on the host. The DMS_ITEMS table also contains the name of the DMSII data set of which the item is a member, as well as other DMSII layout information.

The following table contains descriptions of each column in the DMS_ITEMS Client control table. Included is the abbreviated column name that the `display` command writes to the log file.

Column	Display	Description
data_source		This column contains the name of the data source that identifies the DMSII database from which this data was taken.
dataset_name	ds	This column contains the name of the data set in the DMSII database to which this DMSII item belongs.
rectype	/type	This column, which is zero for all tables mapped from fixed-format data sets, contains the record type of a DMSII variable-format data set. For more information on variable-format data sets, see Variable-Format Data Sets. **Record Type and Description** • **0:** For a variable-format data set, this represents records that have no variable part. • **1-254:** Represents the variable-format record type as defined in the DASDL.
dms_item_name	item_name	This column contains the name of the data item for the listed data set. This column is limited to 128 characters.
active	A	This column specifies whether or not the item will be mapped. A value of 1 (default) indicates that the item will be mapped (if this is possible) to an entry in the DATAITEMS Client control table. A value of 0 indicates that the item will not be mapped. The `define` and `redefine` commands and the Administrative Console's **Customize** command change the value in the `active` column to 0 for the global database unless you set the `active` column to True in the corresponding entry in the DATASETS control table.
item_key	K	This column contains a numeric value which specifies the order of the item in the DMSII set (1, 2, 3, and so on). If the item is not a key, this value is 0. **NOTE:** You can edit this column to create a composite key or change the order of the keys in the index. See Creating Indexes for Tables.
dms_item_number	#	This column contains the item number, which indicates the relative position of the item in the original DMSII record.
dms_parent_item	P#	This column contains the `dms_item_number` of the parent item for an item that is a member of a GROUP item. For example, if `dms_item_number` 12 is a DMSII GROUP containing items 13, 14, 15, and 16, the `dms_parent_item` of the last four items will be 12.

Column	Display	Description
dms_item_type	T	Values 10, 14, 21, 27, 29, and 30 through 37 are DMSII data types. This column indicates the type of data item, as follows: **Type and Description** ・**10**: DMSII link ・**14**: Image (alpha data to be stored as binary) ・**21**: variable-format record type ・**27**: Field of Booleans ・**29**: Group ・**30**: Boolean ・**31**: Field ・**32**: Alpha ・**33**: Number (n) ・**34**: Number (n,m) ・**35**: Real (n) ・**35**: Real (n) ・**36**: Real (n,m) ・**37**: Real
dms_decl_length	DL	This column contains the user-declared length of the item in the DMSII DASDL. This length changes according to the data item type selected (alpha, boolean, field, number, or real).
dms_scale	S	This column contains the numeric scaling factor, which is the number of digits to the right of the decimal point, if any.
dms_offset	O	This column contains the item's offset value which indicates where, within the DMSII record, this item begins. This is the location Databridge uses to extract data from DMSII records. For example, in a 400-byte record with an offset of 200, the first 100 bytes are used by other items. This number is in digit size, which is equal to one-half of a byte (four bits).
dms_length	L	This number is the size, in digits, of the data. Digit size is equal to one-half of a byte (four bits).
dms_signed	s	This column contains a boolean value specifying whether the item is signed or unsigned as follows: 0 = unsigned and 1 = signed.
dms_num_occurs	#O	This column indicates the number of times this data item occurs (is present) within the data set. If the item does not have an OCCURS clause, this value is 0.
dms_num_dims	#D	This column contains the number of dimensions for the data item, which is the number of subscripts required to access the item.
dms_depends_num	dep	This column contains the `dms_item_number` value of the item that specifies the number of occurrences in use for an item with an OCCURS DEPENDING ON clause.

Column	Display	Description
dms_subtype	ST	For items mapped to relational database date types, this column contains the format of the date as it is stored in the DMSII database. These are not actual DMSII data types; rather, they represent the formats of dates that might be stored as a DMSII GROUP, a NUMBER, or an ALPHA item. For non-DMSII columns this column identifies the type of the non-DMSII column. For split data items, this column determines the offset of the split. This column is also used to identify columns in tables that pose unique characteristics. For example, MISER databases use this column to identify special columns in history virtual data sets which indicate if this is a resident history record.

di_options   OPTIONS   The following bits, which can be set through data set
                                               mapping customization user scripts or the Administrative
                                               Console's **Customize** command, enable you to control how the
                                               item is mapped.

Bit and Description

- **1:** DIOPT_Flatten_Occurs - This bit specifies that the OCCURS clause of the item
  should be flattened; it is ignored if the item does not have an OCCURS clause.

- **2:** DIOPT_Clone_as_Date - This bit specifies that the item should be mapped to a
  relational database short date (**smalldatetime** on SQL Server and **date** in Oracle).
  The format for the encoded date is specified in the `dms_subtype` column. If you
  set this bit at the same time as bit 128, bit 128 takes precedence.

- **4:** DIOPT_Split_Item - This bit indicates that the item should be split into smaller
  chunks if it cannot be accommodated using a relational database data type (for
  example, ALPHA(4000) in Oracle). The default is to truncate the item.

- **8:** Reserved

- **16:** DIOPT_Clone_as_Tribit - This bit is used in Miser databases to map DMSII
  number(1) items to a field of three Booleans.

- **32:** DIOPT_Clone_as_Binary - For ALPHA items, this bit indicates that items
  should be mapped to a relational database binary data type, rather than a character
  type. Items too large to fit in the corresponding binary type are truncated, unless
  the DIOPT_Split_Item bit is also set, which then maps the item to multiple binary
  type columns.

  For REAL items that contain visible RSNs, this bit indicates that the items should
  be mapped to a relational database binary data type -- BINARY(6) for SQL Server
  and RAW(6) for Oracle.

- **64:** DIOPT_Xlate_Binary - When this bit is set, EBCDIC data is translated to ASCII
  before being stored as binary.

  NOTE: This bit only affects the program when the DIOPT_Clone_as_Binary bit (32)
  is also set.

- **128:** DIOPT_Use_LongDate - This bit, which applies to Microsoft SQL Server only,
  tells the Client to use a **datetime** data type instead of **smalldatetime** for the
  corresponding column in the relational database.
  If the you are cloning timestamps that include seconds as explained in Decoding
  DMSII Date/Times, set this bit.

- **256:** DIOPT_Clone_as_Time - Indicates to the Client that the DMSII items should
  be interpreted as a time and stored on the relational database as an **int** in the SQL
  Client or **number(10)** in Oracle Client in the form hhmnss except for ticks, which
  are stored in the form *ddddhhmnss*.

- **512:** DIOPT_Numeric_Data - This bit, which applies to DMSII ALPHA types only,
  indicates to the Client that the item contains numeric data and should be mapped
  to a numeric type on the relational database.

- **1024:** DIOPT_AlphaNumData - This bit, which applies to DMSII NUMBER types
  only, indicates to the Client that the item should be mapped to a character type on
  the relational database.

- **2048:** DIOPT_VarFormat_Date - This bit specifies that the item should be mapped
  to a relational database date (**smalldatetime**, **datetime**, **datatime2** or **date** on SQL
  Server and **date** on Oracle), using a unique encoding scheme. This bit requires that
  you also set DIOPT_Clone_as_Date (2).
  The format for the encoded date is specified in the `dms_subtype` column, using
  the instructions for Unique DMSII Date/Time Formats Represented as Alpha or
  Number Items.
  If you use the SQL Server Client and are cloning a value for seconds (hexadecimal
  value 7) from the host, also set bit 128 to get a data type of **datatime**.

- **4096:** DIOPT_FlatSecondary - This bit specifies whether occurring items in the

Column	Display	Description
dms_concat_num		Using this column, the Client supports concatenating two non-contiguous columns of the same data type (that is, treat two columns as one). You can concatenate two ALPHA items, or two unsigned NUMBER items. You can also use the Client to store a numeric item as ALPHA and then use the item in a concatenation. You can also store an ALPHA item that contains numeric data as an unsigned NUMBER and concatenate it with an unsigned number. Concatenation is also supported for unsigned numeric columns that represent a date and a time. The date can be any of the supported date formats, while the time must be a NUMBER(6) containing the time as HHMISS. The combined NUMBER can then be interpreted by the Client as date/time. For an example of the layout scripts, see Concatenating Two Items and Cloning the Result as a Date/Time.
changes		These bits are used by the Client Configurator (not by the `redefine` command). • **1:** CHG_new - New entry • **2:** CHG_modified - Modified entry • **4:** CHG_del_before - One or more entries before this one were removed. • **8:** CHG_del_after - One or more entries after this one were removed • **16:** CHG_dms_item_key - This bit indicates that value in the `item_key` column of the entry has changed. • **32:** CHG_dms_item_type - This bit indicates that the DMSII data type of the item changed. • **64:** CHG_dms_decl_length - This bit indicates that the value in the `dms_decl_length` column of the entry has changed. • **128:** CHG_dms_scale - This bit indicates that the value in the `dms_scale` column of the entry has changed. • **256:** `CHG_dms_signed` - This bit indicates that the value in the dms_signed column of the entry has changed.
dms_link_ds_num	lnk	This column holds the structure number of the data set to which a LINK item points. Thus a nonzero value in this column identifies a DMSII LINK. Links that use AA Values have a dms_item_type value of (10).
di_user_bmask		This column, which shadows the `di_options` column, contains a bit mask that represents the bits in `di_options` that were customized. This column is used by the `redefine` command to restore the portion of `di_options` that has been customized while leaving the remaining bits intact.
redef_item_type		This column is used by the Client to redefine a DMSII GROUP, consisting of items that have the same data types (e.g. a GROUP of 4 unsigned NUMBER items), as a single item of the given type.
redef_decl_len		This column is used by the Client to specify the resulting length when redefining a DMSII GROUP consisting of items that have the same data types.

Column	Display	Description
di_options2		The following bits, which can be set through data set mapping customization user scripts, enable you to control how the item is mapped. **Bit and Description** • **1:** DIOPT_Item_Masked - This bit specifies that the item in question is masked in DMSII. • **2:** DIOPT_Item_Encrypted - This bit specifies that the item in question is encrypted in DMSII. • **4:** DIOPT_Split_Table – This bit forces the define and redefine and the Administrative Console's **Customize** commands to split the table before mapping this item. This gives the user more control in handling split tables when the splitting of the table in the middle of an OCCUR clause is undesirable. • **8:** DIOPT_End_Split_TABLE - This bit is used in conjunction with the DIOPT_Split_Table bit to make the Client return to the parent table following a forced split. It must follow an item with the DIOPT_Split_Table bit set and there can be only one outstanding split (i.e. you cannot have two table splits followed by two end table splits).
da_user_bmask2		This column, which shadows the `di_options2` column, contains a bit mask that represents the bits in `di_options2` that were customized. This column is used by the redefine and the Administrative Console's **Customize** commands to restore the portion of `di_options2` that have been customized while leaving the remaining bits intact.
dflt_options		This column is used to keep track of the initial state of the `di_options` bits before any customizations are performed. It allows the Client to clear the `di_user_bmask` bits that were set by customizations that were later undone.
dflt_options2		This column is used to keep track of the initial state of the `di_options2` bits before any customizations are performed. It allows the Client to clear the `di_user_bmask2` bits that were set by customizations that were later undone.

12.5 DATAITEMS Client Control Table

This table duplicates the DMSII information in the DMS_ITEMS table and contains the layout information for the tables in the relational database. This table is not directly linked to the DATASETS table. Instead, it is linked to the DATATABLES Client control table using the `table_name` column as a foreign key.

You can use the DATAITEMS Client control table to specify the data items you do not want to clone by setting their corresponding active column to 0. However, we recommend that you accomplish this by setting the active column to 0 in the DMS_ITEMS table. Using the DATAITEMS table can lead to unnecessary table splits. Unused columns cause the column count and record size computations to be too high.

If data set mapping is already complete, this table can be temporarily used to disable a new column after a DMSII reorganization to avoid recloning. (This is done automatically if the configuration file parameter `suppress_new_columns` is set to True.)

If you want to disable cloning for every data item in a data set (every column in a table), disable cloning for the data set instead of disabling cloning for each individual data item. For details, see DATATABLES Client Control Table.

The following table contains descriptions of each column in the DATAITEMS Client control table. Included is the abbreviated column name that the `display` command writes to the log file.

Column	Display	Description
data_source		This column contains the name of the data source that identifies the DMSII database from which the data was taken.
table_name	table	This column contains the name of the table in the relational database to which this item belongs.
item_number	#	This column contains an internal number that gives each item within a table a unique number. Numbers are assigned consecutively in increments of 10, starting with 10, making it easier to change the order of items using data table customization user scripts.
item_name	item_name	This column contains the name of the item (column) in the relational database table. Typically, this is the same as the lowercase form of the DMSII item name with all dashes changed to underscores. To modify, see Appendix D: Customization Scripts.
active	A	The value in this column specifies whether or not this data item will be cloned. The default is 1, which indicates that the data item will be cloned. 0 indicates that the data item will not be cloned. The define, redefine and the Administrative Console's **Customize** commands change the value in the active column to 0 if the data set is the global dataset, unless the active column is set to true in the corresponding DMS_ITEMS entry. **NOTE**: If the active value for the data set to which this item belongs is 0 (off), this item will not be cloned even if its active value is 1 (on).
item_key	iK	This column contains a numeric value specifying the order of the item in the DMSII set (10, 20, 30, and so on). You can modify this column to make it part of a composite key or change the order of the keys in the index. For details, see Creating Indexes for Tables. If the item is not a key, the value is zero (0).
virtual_key	VK	Do not change this value. This column contains a boolean value specifying if this item is a virtual key; however, it is created only for mapping DMSII items with unflattened OCCURS clauses. When an item is a virtual key, the corresponding value for item_key is a positive number that is one greater than the item_key value of the last key for the data set. The virtual key is not a DMSII key -- its value in the data table is the occurrence number in the occurs clause (starting at 1).
dms_item_number	I#	This column contains the item number, which indicates the relative position of the item in the original DMSII record.
dms_parent_item	P#	This column contains the dms_item_number of the parent item for an item that is a member of a GROUP item. For example, if dms_item_number 12 is a DMSII GROUP containing items 13, 14, 15, and 16, the dms_parent_item columns of the last four items will be 12. This column contains a copy of the dms_subtype in the DMS_ITEMS table.

Column	Display	Description
dms_item_type	TYP	For a description of this column, see "dms_item_type" in DMS_ITEMS Client Control Table. In addition to the types defined in DMS_ITEMS, this column contains the following values: **Type and Description** • **256:** AA Value or RSN, which the Databridge Client generates using the AA Value or RSN of the record in the DMSII database as received from Databridge Engine. You can tell them apart by looking at the `item_name` column, which is `my_aa` for AA Values and `my_rsn` for RSNs. • **257:** Parent AA, which the Databridge Client generates using the AA Value of the parent record of an embedded data set in the DMSII database as received from Databridge Engine. • **258:** External type, which indicates that the data comes from some place other than the DMSII database.
dms_decl_length	DL	This column contains the user-declared length of the item in the DMSII DASDL. This length changes according to the data item type selected (alpha, boolean, field, number, or real).
dms_scale	SC	This column contains the numeric scaling factor, which is the number of digits to the right of the decimal place, if any.
dms_offset	OFF	This column contains the item's offset value which indicates where, within the DMSII record, this item begins. This is the location Databridge uses to extract data from DMSII records. For example, in a 400-byte record with an offset of 200, the first 100 bytes are used by other items. This number is in digit size, which is equal to one-half of a byte (four bits).
dms_length	LEN	This number is the digit size of the data. Digit size is equal to one-half of a byte (four bits).
dms_signed	s	This column contains a Boolean value specifying whether the item is signed or unsigned as follows: 0 = unsigned and 1 = signed.
dms_num_occurs	OCC	This column indicates the number of times this data item occurs (is present) within the data set. If the item does not have an OCCURS clause, this value is 0.
sql_type	TY	This column contains the relational database data type that corresponds to the DMSII data types. See DMSII and Relational Database Data Types.
sql_length	LEN	If the data type for the column has a length specification in the relational database, this column specifies the length to be used. For example, in the case of char(5) the `sql_length` is 5. Conversely, if the data type for the column does not have a length specification in the relational database (for example, **int** in SQL Server or **date** in Oracle) this column has a value of 0.

Column	Display	Description
occurs_level	OLV	This column contains the nesting level of OCCURS in the DMSII database. For example, an OCCURS table created from an in item with an OCCURS clause contained in in a GROUP with an OCCURS clause has an `occurs_level of 2`. The original OCCURS table has an `occurs_level` of 1. NOTE: The Client does not support nested OCCURS that are more than 2 levels deep.
dms_subtype	STY	For items mapped to relational database date types, this column contains the format of the date in the DMSII database. These are not actual DMSII **date** types; rather, they represent the formats of dates that might be stored as a DMSII GROUP, a NUMBER, or an ALPHA item. For non-DMSII columns this column identifies the type of the non-DMSII column. For split items it represents the offset of the split.
sql_scale	SC	This column contains a copy of the `dms_scale` that you can edit. This value is used in the relational database to specify the scale for columns of whose data type is DECIMAL(p,s) on SQL Server or NUMBER(p,s) on Oracle.
dms_depends_num	dep	This column contains the `dms_item_number` of the item that specifies the number of occurrences in use for an item with an OCCURS DEPENDING ON clause.

da_options	OP	The following bits, which you can set through data table customization user scripts or the Administrative Console's **Customize** command, allow you to specify additional properties of the data items:

Bit and Description

- **1: DAOPT_Nulls_Allowed** - This bit is set by the `define`, `redefine` and the Administrative Console's **Customize** commands based on the value of the configuration parameter `allow_nulls`. You can later change this value via user scripts or customization. A value of 1 indicates that the item will be created with the attribute of NULL (except in Oracle where this is the default attribute of a column). A value of 0 indicates that the item will be created with the attribute of NOT NULL (except in SQL Server where this is the default attribute of a column).

- **2: DAOPT_Column_Renamed** - The column was renamed by the user. This column is used by the Administrative Console's **Customize** command to restore changes.

- **4: DAOPT_Type_Changed** - The SQL type of the column was changed by the user. This column is used by the Administrative Console's **Customize** command to restore changes.

- **8: DAOPT_Length_Changed** - The SQL length of the column was changed by user. This column is used by the Administrative Console's **Customize** command to restore changes.

- **16: DAOPT_Scale_Changed** - The SQL scale changed by user. This column is used by the Administrative Console's **Customize** command to restore changes.

- **32:** Reserved

- **64:** Reserved

- **128: DAOPT_Item_Renumbered** - The item number (that is, the location of the column) was changed by the user. This column is used by the Administrative Console's **Customize** command to restore changes.

- **256:** Reserved

- **512: DAOPT_Store_as_Char** - This bit indicates that the item, which is numeric, should be stored in the relational database as a character data type.

- **1024: DAOPT_Xlate_Binary** - This bit determines whether or not character data gets translated from EBCDIC to ASCII before being stored as binary. This bit is copied from the DIOPT_Xlate_Binary bit in the `di_options` column of the DMS_ITEMS table as the `process` and `clone` commands do not load the DMS_ITEMS table.

- **2048: DAOPT_Store_as_Number** - Indicates that the Client is storing the corresponding ALPHA data using the appropriate numeric data type.

- **4096: DAOPT_VarFormat_Date** - Indicates that the `dms_subtype` column contains a mask describing the date format.

- **8192: DAOPT_FixAlphaChar** - This bit applies to data items whose data type is ALPHA, and it indicates that the Client will scan the data for control characters and replace each control character with a space.
  You can set this bit via a user define script or by using the Administrative Console's **Customize** command, or you can set it globally via the `convert_ctrl_char` parameter. See convert_ctrl_char.

  CAUTION: Do not set the `convert_ctrl_char` parameter to True unless you are absolutely certain that eliminating control characters will have no adverse effect on the data. For example, eliminating control characters can cause some fields to be misinterpreted.

- **16,384: DAOPT_ActiveReset** – Internal use only. This bit indicates that the `active` column of items was set to zero by the Client. This happens for concatenated items, which must be present to access the data and are otherwise not processed.

Column	Display	Description
changes		These bits are used by the `redefine` and the Administrative Console's **Customize** commands. • **1:** CHG_new - New entry • **2:** CHG_modified - Modified entry • **4:** CHG_del_before - One or more entries before this one were removed. • **8:** CHG_del_after - One or more entries after this one were removed
dms_link_ds_num		This column holds the structure number of the data set to which a LINK item points. Thus a nonzero value in this column identifies a DMSII LINK. Links that use AA Values have a dms_item_type value of (10).
dms_concat_num		This column is a copy of the DMS_ITEMS column of the same name and is automatically set by the define and redefine commands since the DMS_ITEMS table is not loaded during a `process` or `clone` command. Do not modify this column in your user scripts.
da_user_bmask		This column, which shadows the `da_options` column, contains a bit mask that represents the columns in `da_options` that were customized. This column is used by the `redefine` and the the Administrative Console's **Customize** commands to restore the portion of `da_options` bits that has been customized while leaving the remaining bits intact.
masking_info		This column is used by the SQL Server Client to do data masking. This integer value contains the masking function type (none, default, email, random, partial) and the index of the corresponding parameter data for masking function that have parameters. The `define` command create entries with default mask for columns that have a datamask specification in DMSII DASDL. You can then change the masking type using the Administrative Console's **Customize** command.
dflt_options		This column is used to keep track of the initial state of the `da_options` bits before any customizations are performed. It allows the Client to clear the `da_user_bmask` bits that were set by customizations that were later undone.

12.6 AF_STATS Client Control Table

This table is used to hold the audit file statistics for the last 9999 audit files processed by the client.

Column	Description
data_source	This column contains the name of the data source to which the record belongs.
audit_filenum	The audit file number (1 to 9999).
no_stat_available	This column is set by Client to while processing an audit file to indicate that the record does not contain any statistics.
audit_start_time	First timestamp in the audit file (**datetime2(7)** for SQL Server and **date** for Oracle).
audit_end_time	Last time stamp in the audit file (**datetime2(7)** for SQL Server and **date** for Oracle).
client_start_time	Time when the Client first started processing the audit file (**datetime2(7)** for SQL Server and **date** for Oracle).
client_end_time	Time when the Client finished processing the audit file (**datetime2(7)** for SQL Server and **date** for Oracle).
n_threads	Number of threads configured in the Client
elapsed	Elapsed time (in milliseconds) processing the audit file. This is not always equal to `client_end_time` - `client_start_time` as the audit is sometimes processed in multiple Client runs.
dms_rec_count	Number of DMSII records received.
sql_op_count	Number of SQL updates executed.
sql_rb_op_count	Number of rolled back SQL updates.
sql_suppressed	Number of SQL updates that were suppressed because there were no changes in the BI and AI images for the columns involved in the update.
sql_filtered	Number of SQL updates that were eliminated by occurs table row filtering.
recs_discarded	Count of discarded records.
recs_in_error	Count of records in error.
bytes_received	Number of DMSII data bytes received from the server.
total_bytes_received	Number of DMSII data and protocol overhead bytes received from the server.
bi_bytes_received	Number of MODIFY_BI data bytes received from the server.
create_count	Number of CREATE records received from the server.
delete_count	Number of DELETE records received from the server.
modify_count	Number of MODIFY records received from the server.
modify_bi_count	Number of MODIFY_BI records received from the server.
modify_ai_count	Number of MODIFY_AI records received from the server.
link_ai_count	Number of LINK_AI records received from the server.
state_count	Number of STATE info records received from the server.
doc_count	Number of DOC records received from the server.
commit_count	Number of COMMITS.
rollback_count	Number of ROLLBACKS.

13. Automating Client Operations with the Service

This chapter explains how to automate Client operations by using the service and scripts (that is, command files on Windows; shell scripts on UNIX). It also covers the Batch Console and its role in automation.

13.1 Configuring the Service

To configure update scheduling, error recovery, and other features of the service, use the Administrative Console. Your changes are automatically saved to the service's configuration file `dbcontrol.cfg`, located in the **config** directory of the working directory. See Managing Operations in the Administrative Console Help, available from within the program. For more information about the service's working directory, see the topic The Working Directory in the Databridge Installation Guide.

13.2 Automation Scripts

The Databridge Client 7.0 service uses scripts (command files in Windows; shell scripts in UNIX) to allow the user to gain control at key points of Client operations. Scripts allow the user to perform pre-processing and post-processing for Client runs (typically `process` commands) and supplement the service's built-in error recovery with additional recovery or error-reporting mechanisms.

Additionally, users can start a script from the mainframe by using the BCNOTIFY program. The BCNOTIFY program is designed to make the service start the script whose name it supplies. By inserting BCNOTIFY at key points in a WFL (Work Flow Language), you can trigger a task on the Client machine, such as restarting audit file processing. Scripts can interact with the service via the Batch Console (**bconsole**), a program that interprets source files that contain programs written in a language vaguely resembling to Visual Basic. Scripts can also perform arbitrary tasks that may not be directly related to the service (for example, start a run that generates a Crystal Report).

Automation scripts fall into two basic categories:

- Scripts that are associated with a `process` command. These scripts are associated with a data source and run before or after a Client run. (See Process-Related Scripts.)
- Scripts that are initiated by the BCNOTIFY program on the mainframe. (See BCNOTIFY Initiated Scripts.)

All scripts start with the current directory set to the service's working directory. Additionally, the service sets up the environment variable INSTALLDIR that points to the install directory where the Batch Console program resides. You must use this environment variable as the path when invoking the Batch Console (on Windows, `%INSTALLDIR%\bconsole`; on UNIX, `$INSTALLDIR/bconsole`). You cannot rely on the install directory being in the PATH. For more information about Batch Console, see Introducing the Batch Console.

13.2.1 Process-Related Scripts

This first category of automation scripts includes scripts that are associated with a `process` command. These scripts are referred to as start-of-run and end-of-run scripts. Start-of-run scripts are only applicable to runs started by the service without any outside interference, specifically runs that are started by the service's scheduler. This includes runs that are launched when the service is started in response to the `run_at_startup` parameter for a data source. End-of-run scripts on the other hand are applicable to all runs.

Client runs that are started from the Administrative Console, the Batch Console, and BCNOTIFY do not look for start-of-run scripts.

Both types of scripts follow strict filename conventions, as follows.

Type of Script	Naming Convention
start-of-run	*source*_startofrun.*ext*
end-of-run	*source*_endofrun.*ext*
event-notice	*source*_eventnotice.*ext*

where *source* is the data source name in lowercase letters and *ext* is the file extension (.cmd for Windows or .sh for UNIX).

The service searches for these script files in the `scripts` subdirectory in the service's working directory. (For information about the working directory, see The Working Directory in the Databridge Installation Guide.) Before a scheduled run is started, the service looks for the existence of a start-of-run script. When the service finds a script, it runs it and then starts the `process` command after the script is complete. If the script does not exist, the `process` command is started without issuing any errors or warnings. When the service determines that the `process` command is complete, it checks for the existence of an end-of-run script and runs it.

When a `process` command terminates with an exit code that initiates auto-recovery, the service checks for the existence of an event-notice script and runs it if found. These type of scripts are designed to give the user the ability to generate event notices, such as e-mails when the service enter auto-recovery. In the absence of these scripts the DBA would be totally unaware that the service restarted the Client after a recoverable error.

If a Client run ends because of an error, the end-of-run script will run only after the service's built-in error recovery has been executed. The service has a built-in recovery mechanism for responding to errors and non-zero exit codes. In situations that are temporary or self-resolving, the service will attempt the failed operation after a brief delay. For example, if a connection to the server or database fails, the service will pause for a specified amount of time before attempting to reconnect, and will do so repeatedly until the server or database becomes available. (The parameter `sched_retry_secs` determines the initial retry interval. For connect failures, this interval doubles on each subsequent retry until it reaches a maximum value of 5 minutes. You can set this parameter value in the Client Configuration dialog box. When using the Administrative Console, from the data source's **Settings** menu click on **Processing > Error Recovery**, you will find these parameters in the "Options" group at the top of the page.

Both start-of-run and end-of-run scripts are passed the following set of parameters:

Script Parameter	In the script, referred to as
Data source name	%1
Exit status	%2
Run type (a number that indicates the type of command)	%3
Token used as the password when connecting back to the service using the Batch Console	%4

For start-of-run scripts, *run_type* is always 1, indicating a `process` command. For end-of-run scripts, *run_type* can be 1, 2 (`clone` command), 7 (`redefine` command), 8 (`generate` command) or 4 (Administrative Console **Customize command run).

13.2.2 BCNOTIFY Initiated Scripts

This second category of automation scripts are initiated by BCNOTIFY, a utility included with Databridge Host software.

BCNOTIFY is a host-based Databridge utility you can use to issue remote procedure calls (RPC) to the service and tell it to launch scripts. BCNOTIFY passes the script name and parameters in the RPC. BCNOTIFY can optionally pass a data source name as an additional parameter. If you do not supply a data source name in the RPC, the data source name must be provided within the script that the service launches. The advantage of including the data source name in the RPC is that the service will only launch the script if the data source is idle (versus always launching it).

Scripts initiated by BCNOTIFY are named as follows:

start_*name*.*ext*

where *name* is an arbitrary name, and <ext> is `.cmd` on Windows and `.sh` on UNIX.

When BCNOTIFY launches a script that initiates a `process` command, the service behaves differently when looking for an end-of-run script to execute. It first looks for a script named `end_name.ext` in the scripts subdirectory (where *name* is the name used in the original script and *ext* is the OS dependent file extension). If the service finds this script, it uses the script in place of the standard end-of-run script described earlier. Otherwise, the standard end-of-run script is used if it exists. This allows one to associate multiple end-of-run scripts with a data source, depending on which script started the `process` command.

These script files are passed the following set of parameters. The parameters for these scripts can change, depending on whether the data source is an empty string. For example, if no data source name is provided, parameter one is the AFN and parameter 2 is the token.

Parameters: - Data source name (optional) - Parameters supplied by BCNOTIFY. For example, the current database audit file number (AFN) - A token used as the password when connecting back to the service via the Batch Console

13.3 Introducing the Batch Console

The Batch Console automates routine Client tasks by allowing command files/shell scripts launched by the Databridge Client Manager to interact with the service. It interprets a source file that contains a set of statements written in a language similar to Visual Basic. These statements can initiate a connection, perform rudimentary tests, and issue console requests, to the service. For example, by using the Batch Console in an end-of-run script that runs daily reports, you can restart the Client after the reports are generated.

To use the Batch Console, you must first create a source file for the Batch Console and place it in the `scripts` directory of the service's working directory (also referred to as the Client's global working directory). We recommend that you use a file extension that allows you to easily identify this file as a Batch Console source file (for example, `.bcs`). You can debug this source file by running the Batch Console from the command line, using the source filename (including directory, such as scripts/source_filename) as the first argument of the Batch Console command.

The Batch Console always runs as a background run. Its activity is written to a log file in the current directory. The log filename uses the source filename with the extension `.log` added to it. For example, if your source filename is `sourcefile.bcs` the log file is named `sourcefile.bcs.log.`

13.3.1 Running the Batch Console (bconsole)

The Batch Console program (bconsole) has a command line of the form:

```
bconsole [options] filename [argument list]
```

where *filename* is the name of a text file that contains commands for the Batch Console to interpret.

> ✏️ **Note**
>
> All scripts start with the current directory set to the service's working directory. Additionally, the service sets up the environment variable INSTALLDIR that points to the install directory where the Batch Console program resides. You must use this environment variable as the path when invoking the Batch Console (on Windows, `%INSTALLDIR%\bconsole` ; on UNIX, `$INSTALLDIR/bconsole`). You cannot rely on the install directory being in the PATH. For more information about Batch Console, see Introducing the Batch Console.

You can include additional arguments to pass parameters to the program. This allows you to use generic source files that work with externally-supplied values for the parameters. The command-line parameters in [argument list] are referenced in the script file using the Windows command file conventions (%1 is parameter 1, %2 is parameter 2, and so on). For example, if you invoke bconsole using the statement

```
bconsole /P secret resume.bcs 1234
```

the program substitutes the text "1234" for any occurrence of "%1" in the script file " `resume.bcs` " the same as Windows command files.

The following command file performs the aforementioned task for a fictional data source named MISDB. MISDB uses the service that runs on a machine named "galactica" using port 8001. (Included with the following command are words, such as "data source", which are ignored by the parser but make the script read more like plain English. These words appear in black text.)

```
connect to galactica port 8001
enable data source MISDB
process MISDB
```

If a command fails, the program returns a non-zero exit status indicating a failure. On successful execution of the script, the program returns an exit status of 0. Each executed script is logged; if something fails, you can look at the log file to determine what went wrong.

The user ID of the user that launches the bconsole run is used to sign on to the service. When the Batch Console program starts from a script that the service launches, the script is passed a handle for use as a temporary password. This eliminates any security problems that having user IDs or passwords in the script files might cause. If the service cannot authenticate the user ID password, it verifies that the user is the same one that is running the service. (This is typically the built-in SYSTEM account). If it is, the service verifies that the handle matches the one assigned to the thread that launched the script. (The handle contains the thread number.)

13.3.2 Signing On to the Service

The userid that starts the Batch Console is also used to sign on to the service. This eliminates the security problems that can result from including userids and passwords in script files. When the service launches a script, it passes a handle for the script to use as the Batch Console password. This password is set using the command-line option `/P` .

After the service identifies the userid as being the same as the service's userid, it validates the signon once it determines that the password matches the handle passed to the script. Handles are only valid while the script is running and cannot be reused.

13.3.3 Using Batch Console in Scripts Initiated by BCNOTIFY

You can use the Batch Console to make the service start a `process` command or notify a currently running `process` command to stop after a specified AFN. This mode of operation replaces dbauditwait working in conjunction with the deprecated NOTIFY program on the mainframe and uses the service-initiated DBClient runs instead of the command-line Client. When invoking the Batch Console in a script launched by the service, you must pass the handle to Batch Console using the /P option.

```
bconsole /P %2 sample_script2.bcs  mikera018684 MISDB 1234
```

The following Batch Console source file `sample_script2.bcs` uses command-line parameters similar to a Windows command file, except that the parameters are numbered starting with the one that follows the source filename. The program does a textual substitution by replacing `%n` with the exact text of the corresponding parameter. (This script is located in the `scripts` directory of the service's working directory.)

```
connect to %1 port 8001
if run active %2 then
    stop run %2 after afn %3
else
    process %2 with option "/F %3"
end if
```

In the above example, the text `%1` is replaced by `mikera018640`, `%2` is replaced by `MISDB` and `%3` is replaced by `"1234"` .

13.3.4 Using Batch Console to Get Status Information

You can use the Batch Console in query mode to get status information. This capability is not related to automation, but is provided to let you query the service about the status of your data sources. Query mode connects to the service, gets the status of the specified data sources, and then writes that information to a file.

To use query mode, you must provide the connect parameters and the command using command-line switches. The command line for query mode is as follows:

```
bconsole /s  service_name /p port /P  password /w  filename /q  command
```

where the *service_name*, output *filename*, and *command* can optionally be enclosed in double quotation marks. If the `/w filename` option is omitted, the program will write the output to the file `bconsole.log`. The syntax for the *command* is:

```
status [data_source]
```

If *data_source* is omitted (or is specified as `_all`) the status of all data sources is written to the output file in CSV format. A sample output file is as follows:

```
MISDB,0,0x00000000,0,9999
DEMODB,0,0x00000000,0,9999,,,,disabled
NTSL,0,0x00000000,1,0,2011-03-31@15:26:46,2011-03-31@17:46:52
```

Each line includes (in this order): a) the data source name; b) its state; c) the process-id of the current run (or 0 if there's no active run); d) type (state) of the last run; e) exit code of last run or 9999 if the run is active; f) start time of the active run or the last run (if there's no active run); g) stop time of the last run (0 if there is an active run); h) the next scheduled run (if idle); and i) the flag for the data source. If a Client run crashes, it will have an exit code of 9999 and the data source will be marked as disabled.

13.3.5 Batch Console Commands

The syntax for Batch Console (bconsole) scripts is loosely modeled after Visual Basic. The end-of-line character acts as a statement terminator. This limits you to one statement per line. The following table lists commands in alphabetical order, followed by a list of buzz words that are allowed to improve readability.

Command	Reference
abort	abort [run [for [[data] source]]]...
clone	clone [[data] source]...
connect	connect [to]...
define	define [[data] source]...
disable	disable [[data] source]...
disconnect	
display	display "text"
drop	drop [[data] source]...
dropall	dropall [[data] source]...
enable	enable [[data] source]...
exit	exit (*value*)
generate	generate [[data] source]...
if expression then ... [else ...] end [if]	
launch	[[[data] source]...
process	process [[data] source]
redefine*	redefine [[data] source]...
reorg**	reorg [[data] source]...
reorganize	reorganize [[data] source]...
runscript	runscript "filename"
status	status [[data] source]...
stop	stop [run [for [[data] source]]]...
wait	wait (*value*)

* Synonymous with define

** Synonymous with reorganize

Statements in Detail

Let's look at the individual statements and syntax of a Batch Console script. All statements are confined to a single line, except for the if statement.

This command	Does this
connect [to] *service_name* [port] *number*	Connects to the given service as a console. If the service name is an IP address or contains non alphanumeric characters, it must be enclosed in double quotation marks.
enable [[data] source] *name*	Enables the specified data source. If the data source doesn't exist, an error occurs. If the data source is not disabled, no command is executed and a warning results. This will not cause the script to terminate prematurely. To eliminate the warning, use an "if" statement; this test whether the data source is disabled before trying to enable it.
process [[data] source] *name* [[with] option[s] "*option_list*"] [reclone *ds_list*]	Initiates a `process` command. The options are specified the same as on the dbutility command line, using either slashes or a hyphen (depending on the operating system) followed by a letter and an argument, if applicable. When specifying options, you must include the keyword option to indicate that a list of options follows the command. Make sure that you separate each option with a space and enclose the entire set of options with double quotation marks. The program will validate the options before passing them to the service. You can force all of the specified data sets to be re-cloned by adding the keyword reclone followed by a list of data sets to the `process` command. This sets the ds_mode to 0 for all the specified data sets in a single command. If you use the name "all", all data sets will be re-cloned; you don't need to name them individually.
clone [[data] source] *name* [[with] option[s] "*option_list*"] *ds_list*	Initiates a `clone` command. The options are specified like you would in dbutility, using slashes or dashes (depending on the operating system) followed by a letter and an argument when applicable.
[re]define [[data] source] *name* [[with] option[s] "*option_list*"]]	Initiates a `redefine` or `define` command, depending on whether the data source exists. This statement causes the service to launch DBClientCfgServer for the data source (unless it is already running) and then execute a `define`/`redefine` command.
reorg[anize] [[data] source] *name* [[with] option[s] "*option_list*"]]	Initiates a `reorganize` command. This statement causes the service to launch DBClientCfgServer for the data source (unless it is already running) and then execute a `reorg` command.
generate [[data] source] *name* [[with] option[s] "*option_list*"]]	Initiates a `generate` command. The statement causes the service to launch DBClientCfgServer for the data source (unless it is already running) and then execute a generate command.
stop [run [for [[data] source]]] *name* [{after [afn] *number*	at [time] *hh:mm*}]
abort [run [for [[data] source]]] *name*	This is equivalent to the DBConsole Abort command. The Client run is terminated immediately by closing the TCP/IP connection to the server. This will cause the last transaction group to be rolled back by the Client.

This command	Does this
status [[[data] source] name]	This command writes the status of a data source to the Batch Console (bconsole) log file. If the data source name is omitted, or the name "_all" is used, the status of all data sources is written to the log file.
display "*string*"	This command writes the given string to the log file. It is mainly intended to help debugging.
if expression then ... [else ...] end [if]	The block of commands following the "if" line, which must end with the keyword "then", are executed if the expression evaluates to true. This block ends with either an "else" or "end [if]" keyword (in the absence of an else clause). The else clause starts a new block of commands that will be executed if the expression in the "if" statement evaluates to false. In all cases "end [if]" ends the block that is started by a "then" or "else".
launch [[[data] source]] *name cmd_file params*	This command makes the service launch an arbitrary command file. It is only useful in debugging BCNOTIFY scripts, as this is the easiest way to launch them. For name, list the data source name. If the run is not associated with a specific data source, use "_none" (quotation marks not required). The cmd_fileis the filename of the script in the `scripts` subdirectory of the service's working directory that you want the service to launch *.cmd_file and params* must be enclosed in double quotation marks if they contain non alphanumeric characters, such as a period (.).
disconnect	This command tells the program to disconnect from the given service it is connected to. This command will not normally be needed, as the program will automatically issue it when it reaches the end of the script file.
wait (integer value)	This command injects a delay, (measured in seconds) in execution of the script file. It is mainly intended for debugging purposes.
exit (integer value)	This command stops the program and returns the specified exit code. The command is only needed to return a non-zero exit code or to stop the flow of execution within an "if" statement.

If Statements

Use the "if" statement to test for the following three conditions for a data source:

1. Whether it is **disabled**

2. Whether a run is **active**

3. Whether a run is **scheduled**

The keywords "disabled", active, and "scheduled" are used to indicate the condition being tested. You must follow these keywords with a data source name and the keyword "then". Optionally, you can precede keywords with the buzzwords "run", "data source", or "source".

To reverse the test, you can place the keyword "not" in front of expressions that follow the keyword "if". The syntax of these expressions is summarized as follows:

```
[not] {[run]  | [[data] source]} active name
[not] {[run]  | [[data] source]} disabled name
[not] {[run]  | [[data] source]} scheduled name
```

Command-Line Options

> ✏️ **Note**
>
> Options are case sensitive. `-p` and `-P` are separate options.

This Switch	Argument	Does this
-d		Enables debug output
-o		Overwrites the log file (versus appending to it)
-p	port	Specifies the port on the command line.
-q		Switches into single query mode (status command only)
-s	name	Specifies the domain name or IP address of the service machine on the command line.
-t		Enables RPC traffic tracing.
-w	filename	Sets the name of the log file.
-p	password	Specifies the password to be used when connecting to the service.
-T		Specifies that the user is a trusted user*

* When you run the batch console (bconsole) from a command file that is not launched by the service, you need to specify a password using the -P option. Since the password is not encoded, some sites may find this objectionable. In order to solve this problem we implemented the -T option, which requires that the userid being used be registered as the trusted user. The batch console will then read the Windows Registry and determine if the userid is registered as the trusted user (there can only be one in the current implementation). To facilitate the registration process, the program `setbcuserid.exe` was implemented. This program registers the userid you enter as the login userid. You must be an administrator to run this program and the userid you specify must be a valid Windows user.

14. Glossary of Terms

absolute address (AA) value

AA is a DMSII term that stands for absolute address. An absolute address value is an A Series WORD (48-bits in length). In the Databridge Client, AA is the hexadecimal representation (12 character strings containing the characters 0--9 and A--F) of the AA Value on the host. Databridge Client uses the AA Values to implement unique keys for the parent structures of embedded data set records. It also uses AA Values to reference the records of data sets that do not have DMSII SETS with the NO DUPLICATES ALLOWED attribute.

AA Values are not constant. Any DMSII reorganization (record conversion, file format, or garbage collection) changes these values.

Databridge Client supports numeric AA Values that are stored as NUMBER(15) in Oracle and BIGINT in SQL Server. It also supports binary AA Values that are stored as RAW(6) in Oracle and BINARY(6) in SQL Server.

Audit Files

An 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.

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 set the Databridge Engine Read Active Audit option, Databridge can access the current audit file. If you do not set Read Active Audit = true in the Databridge Engine parameter file, Databridge can access audit information up to and including the current audit file minus one. 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.

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

• A Databridge accessory closed the file in preparation for the fixup phase after extracting records from a DMSII database.

• The current audit file could not be found.

• A file reorganization was executed to modify the DMSII structure.

audit trail

The audit trail contains all of the audit files generated for a database. The Databridge Engine reads the audit files to extract updates. It then passes the updates to the Client to be applied to the relational database. After the updates have been successfully extracted, the Client saves the state information, which includes the location in the audit trail from which the last group of updates for the data set were read.

Batch Console

The Batch Console automates routine Client tasks by allowing command files/shell scripts launched by the Databridge Client Manager to interact with the service.

caching

A process that filters files before they\'re requested by the Databridge Client. Caching allows Databridge Enterprise Server to send Client data requests quickly and without placing an additional resource burden on the mainframe.

client

The Client is the computer system that will receive DMSII records from the primary database. The Client could be a Windows computer, a UNIX computer, or an MCP server. The Client can have a relational or a DMSII database.

cloning

Cloning is the one-time 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 synchronized 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.

DASDL

Data and Structure Definition Language (DASDL) is the language that defines DMSII databases. The DASDL must be compiled to create a DESCRIPTION file.

data set

A data set is 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.

Databridge Director

Databridge Director (also referred to as DBDirector) is a Windows Service installed with Enterprise Server that starts Enterprise Server whenever a connection request is received.

When you start your computer, DBDirector starts and reads the ListenPort registry value to determine which TCP/IP port communicates with Databridge Clients.

Databridge Engine

Databridge Engine is a generic term that can refer to either DBEngine or the engine component of Databridge Enterprise Server. The two are interchangeable as far as the Databridge Client is concerned.

Databridge Server

Databridge Server is a generic term that can refer to either DBServer or Databridge Enterprise Server. The two are interchangeable as far as the Databridge Client is concerned.

DBClntCfgServer

A program that handles all requests from the Client Console specific to a data source. These requests include updating the Client configuration file, providing access to the Client control tables, and handling the Client Configurator. Like DBClient, this program is run by the Client Manager service as a background run.

DBClntCfgServer

A program that handles all requests from the Client Console specific to a data source. These requests include updating the Client configuration file, providing access to the Client control tables, and handling the Client Configurator. Like DBClient, this program is run by the Client Manager service as a background run.

DBServer

DBServer is a Databridge Host accessory that responds to Databridge Client requests for DMSII data or DMSII layout information and provides communications between the following components:

- Databridge Engine and Databridge Enterprise Server
- Databridge Engine and the Databridge Client

> ✎ **Note**
>
> When Enterprise Server is used with the Databridge Client, Enterprise Server takes over much of the functionality of DBServer and Databridge Engine.

direct disk

A replication method that allows Databridge Enterprise Server to clone and track DMSII data sets without using any significant mainframe resources. Direct disk replication requires a SAN (Storage Area Network) or Logical Disks configured to make MCP disks visible in Windows.

entry point

A procedure in a library object.

extraction

Extraction is 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 type of DMSII reorganization affects file size values (for example, AREASIZE, BLOCKSIZE, or TABLESIZE), but it does not change the layout of the records in a DMSII database.

flat files

A flat file is a plain text or mixed text and binary file which usually contains one record per line. Within the record, individual fields may be separated by delimiters, such as commas, or have a fixed length and be separated by padding. An example of a flat file is an address list that contains fields for *Name* and *Address*.

---

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.

---

lag time

The lag time is defined as the elapsed time between the time a record in the DMSII database is updated and the time where this update appears in the relational database. This value accounts for any difference between the clock on the mainframe and that on the Client machine.

---

mutex

A mutex is an operating system resource that is used to implement a critical section and prevent multiple processes from updating the same variables at the same time.

---

null record

A record for a data set where every data item is null.

---

null value

The value defined in the DASDL to be NULL for a 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 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 quiet point is a point in the audit trail when the DMSII database is quiet and no program is in transaction state. This can occur naturally, or it can be forced by a DMSII sync point.

---

record format conversion

A type of DMSII reorganization that occurs when a data set or set (group of keys) is reordered or reformatted. It indicates that changes were made to a data set format, or to data items, such as changing the length of an item, for example, BANK-ID NUMBER (10) to BANK-ID NUMBER (15).

---

record serial number (RSN)

Record sequence numbers (RSN) are 48-bit quantities used by the Databridge Engine, in the case of DMSII XE, to uniquely identify a record. RSNs will always be used instead of AA Values when available except for data sets having embedded data sets. RSNs are always static; they will not change after a garbage collection reorganization.

reorganization

Structural or formatting changes to records in the DMSII database, which may require parallel changes to (or re-cloning of) records in the secondary, or relational, database. See also file format conversion and record format conversion.

replicated database

The replicated database is the database that usually resides on the Client machine and contains records cloned from the DMSII database. 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

Replication is the ongoing process of cloning and tracking changes to a DMSII database.

rollback

A systematic restoration of the primary or secondary database to a previous state in which the problem or bad data is no longer found.

secondary database

The replicated database. The replicated database is the database that usually resides on the Client machine and contains records cloned from the DMSII database. 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.

semaphores

Operating system resources that are mainly used to implement thread synchronization and signaling.

service

The service (Windows) or daemon (UNIX) that automates most Client operations. It handles operator requests from the Client Console and routes all log and informational messages to the consoles.

set

An index into a data set. A set has an entry (key + pointer) for every record in the data set.

state information

Data that reflects information about the cloned data, such as the audit location and format level.

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

Tracking is an ongoing process for propagating changes made to records in the DMSII primary database to the replicated database after the initial clone. The Databridge Engine performs extraction as well as tracking.

visible RSN

An RSN (record serial number) that is declared in the DASDL. These appear as an item in the data set and are therefore visible to the database user.

15. Legal Notice

© Copyright 2022 Micro Focus or one of its affiliates.

The only warranties for products and services of Micro Focus and its affiliates and licensors ("Micro Focus") are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Contains Confidential Information. Except as specifically indicated otherwise, a valid license is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

For information about legal notices, trademarks, disclaimers, warranties, export and other use restrictions, U.S. Government rights, patent policy, and FIPS compliance, see https://www.microfocus.com/about/legal/.

16. Appendix

16.1 Appendix A: Troubleshooting

This appendix provides instructions for troubleshooting problems you may experience with Databridge Client.

16.1.1 General Troubleshooting Procedures

If you have problems using the Databridge Client, complete the following steps:

1. Check to see that your system meets the minimum hardware and software requirements. For details, see the *Databridge Installation Guide*.

2. Check that you've selected the correct configuration options for connecting to the relational database server:

   • The relational database name

   • Your user ID and password to log in to the relational database server. Does your user ID to the relational database server have the correct privileges?

   • If you use configuration file parameters or environment variables to supply the signon parameters, did you enter them correctly?

   • If you use command-line options, did you enter them in their correct uppercase or lowercase? Did you enter them with each **dbutility** command? See dbutility Command-Line Options.

   • If you use a UNIX Client, make sure that the ORACLE_HOME, and LD_LIBRARY_PATH variables point to the correct directory, (for example, LD_LIBRARY_PATH=/opt/oracle/product/19.0.0/dbhome_1/lib:/home/dbridge/db70/lib).

3. Check that you've selected the correct configuration options for connecting to the host.

   • Is Databridge Server running on the host?

   • Did you use the data source name as it is defined in the DBServer control file? For more information, refer to the Databridge Host Administrator's Guide.

   • Did you enter the correct host name or IP address?

   • Did you enter the TCP/IP port number as it is defined in the DBServer control file?

   • If there is a password defined in the DBServer parameter file, did you enter the correct password?

4. Make sure that the PATH environment variable contains the Databridge Client's directory and the appropriate relational database bin directory (named bin for Oracle and binn for Microsoft SQL Server).

5. Check your cable connections to make sure that they are securely attached.

6. Determine whether the problem is caused by the host and DMSII (versus Databridge Client) by using Databridge Span on the host to clone a data set from the DMSII database in question.

   • If you cannot clone the data set, the problem is most likely on the host.

   • If you can clone the data, the problem is most likely occurring between the DBServer and Databridge Client.

7. Resolve any errors. If you receive error messages or status messages that you don't understand, see the *Databridge Error and Message Guide*.

8. If you cannot identify and solve the problem without assistance, contact your product distributor or Micro Focus Technical Support from a location where you have the ability to run **dbutility**.

16.1.2 Troubleshooting Table

The following table lists some common problems and their solutions.

Problem	Solution
You made changes to the Client control tables, such as changing the `active` column value, but none of your changes are taking effect.	This problem, which only occurs when using SQL*Plus in an Oracle database, is an indication that your SQL statements did not get "committed." The default mode of operations of SQL*Plus is transaction mode. SQL statements only get committed when you explicitly issue a commit or when you exit SQL*Plus. You can make the program automatically issue a `commit` after every SQL statement by typing `set auto[commit] on`.
You changed one or more table names, but the new tables are empty after you do a clone or an update.	Most likely you did not update the `table_name` columns in the DATAITEMS Client control table.
You have the correct host name, port number, and data source name, but you still cannot connect to the host.	Make sure the domain name server is running. If the domain name server is down, change the host name in the DATASOURCES table to the IP address and try the **dbutility** command again.
You get a "constraint violation" error when you run the `process` command to update the relational database.	Most likely you have placed a constraint on one of the columns in the Databridge data tables. When this occurs, remove the constraint and re-clone the data set to get all of the records. IMPORTANT: You must not place constraints or other restrictions on any Databridge data table. If you do, Databridge will not work. Instead, filter rows on the host using the DBGenFormat utility.

The Databridge Client becomes unresponsive at the following message:
 	
Begin populating/updating database from AFN=afn, ABSN=absn, INX=inx, SEG=seg, DMSII Time=time_stamp
	Check the host ODT for a waiting entry from Databridge Server, similar to the following:
	
	`(usercode) DBSERVER/WORKER-n`
	`NO FILE (usercode)databasename-AUDITnnnn`
	
	In this case, make the audit file available to the Databridge Engine. For example, if the file is on tape, copy it to the usercode indicated for the AUDITnnnn file. Once you make the audit file available, the Databridge Engine automatically begins processing again.
	
	If for some reason you cannot make the audit file available, stop running the Databridge Client by typing QUIT NOW on the Client system.

You are running multiple Databridge Clients, and all of them seem to stop processing.	Most likely, only one of the Databridge Clients has stopped processing because of a problem, and the other Databridge Clients have stopped not because of a processing problem, but because of a resource contention problem on the host or network. To correct this situation, look at the ODT and at the Windows Event Viewer for messages related to the Databridge Client. (The previous two problem descriptions in this table list possible messages.) When you locate and respond to the message for the problem Client, the other Clients start processing automatically from where they left off.
You are unable to execute the **dbutility** program.	Make sure you have included the Databridge Client program directory in the operating system's PATH environment variable.

Problem	Solution
The Databridge Client gets an index creation error for a table that uses a legitimate DMSII SET as an index.	There is no guarantee that the Databridge Engine will always produce tables without duplicate records at the end of the data extraction phase. Most of the time, duplicate records occur when records are deleted and later reinserted into the data set (this sometimes occurs in environments where the DMSII applications use delete/create pairs or in compact data sets). If a record ends up in a location that is different from the original one, the Databridge Engine sees it twice, resulting in a duplicate record. The Client normally runs the script "script.clrduprecs.tablename" when an index creation fails. This script removes all occurrences of duplicate records, as they will be reinserted during the fixup phase. You can inhibit the running of this script by resetting the bit DSOPT_Clrdup_Recs (32768) in the `ds_options` column of the DATASETS table entry. This must be done manually if you have disabled this bit. When this problem occurs, use the procedure described in "Using SQL Query to Find Duplicate Records" to query for duplicate records and remove them. Alternatively, you can clone the data set when the database is inactive or clone the data set offline (the *Databridge Host Administrator's Guide* provides information about cloning offline).
The Databridge Client stops at the start of the fixup phase with the following error: `Stopping: Errors occurred during data extraction`	The Databridge Client stops at this point if records were discarded. There are two types of discards: • Discards created by the Databridge Client because of data errors in items used as keys. • Discards created by the bulk loader because of internal errors. This type of error typically does not occur. If it does occur, it indicates that the program failed to detect a data error. The Databridge Client stops so that you can review these errors. You can fix the data in the discard files that the Databridge Client creates and load the records using a relational database query tool. Alternatively, you can fix the bad data on the mainframe and let the normal update processing take care of things. If you restart the `process` command, the fixup phase proceeds normally.
The Databridge Client stops at the start of the fixup phase with the following error: `Stopping: Errors occurred during index creation`	The Databridge Client stops at this point if one or more index creations fail. You need to determine why the index creation failed and remedy the situation, if possible. For example, if you did not have a large enough TEMP SEGMENT in Oracle, increase its size and execute the index creation scripts using SQL*Plus. Once the indexes are created, you can change the `ds_mode` of the affected data sets to 1 and resume the `process` command, which proceeds normally. Tables that do not have indexes do not cause the Databridge Client to stop at the beginning of the fixup phase. The Databridge Client deselects such data sets and sets their `ds_mode` column to 11 before entering the fixup phase. Any subsequent `process` commands will not select such data sets unless you fix the problem and set their `ds_mode` columns to 1. You can re-clone such data sets at any time.

Problem	Solution
The Databridge Client stops at the start of the fixup phase with the following error: `Stopping: Errors occurred during data extraction and index creation`	This message indicates that both of the last two conditions have occurred.

16.1.3 Using SQL Query to Find Duplicate Records

Use the following SQL query to list the keys and the record counts for duplicate records in a table. Duplicate records result when the given combination of keys is used as the index. This query is also useful when trying to determine if certain key combinations produce a unique key.

```
SELECT key_1, key_2,...key_n, COUNT(*) FROM tablename
GROUP BY key_1, key_2,...key_n
HAVING COUNT(*) >1
```

Where	Is
key_1 key_2 key_n	The list of columns that make up the index for the table.
tablename	The name of the table for which the error occurs.

If no records are duplicated, the output within the relational database query tool will indicate that no rows have been affected. If the SQL query returns a GROUP of duplicates, do the following:

1. Manually delete the extra record or records for each combination of duplicate records.

2. Execute a **dbutility** `runscript` command for each table that contained duplicate records, specifying the index creation script as follows:

   ```
   dbutility -n runscript dbscripts\script.index.*tablename*
   ```

3. Set ds_mode = 1 for each data set that contained duplicate records.

4. Execute a **dbutility** `process` command.

> ✏️ **Note**
>
> If the query routine returns an unusually high number of duplicates, there may be more serious problems with your keys or the process that creates them. For more information about how Databridge uses keys, see Creating Indexes for Tables.

16.1.4 Log and Trace Files

The Databridge Client produces log files and trace files. This topic describes these files and the differences between them.

Log Files

The log file contains information about errors that the Client encounters and statistics that are useful in tracking performance problems. Additionally the log contains messages that are useful when reporting problems to Micro Focus Technical Support (for example, versions of the various host components). When a command is executed for a data source, one or more messages appear onscreen and are written to the log file for that data source. Log files are created in the **logs** subdirectory of the data source's working directory. Log files are named

`dbyyyymmdd.log`

where db is a configurable prefix that can be redefined in the configuration file and *yyyymmdd* is the date the log file was created. A time (_hhmnss) is appended to the filename if the filename is already in use. (For details about configuring the log via the file see Export or Import a Configuration File.)

If more than one log file is created for a data source on the same date, the time of day is included after the date to make the filename unique (for example, *dbyyyymmdd_hhmnss*`.log`).

Some messages are written only to the log file. These messages generally include information that may be useful when reporting problems to Micro Focus Technical Support, such as version information for the various host and Client components, the OS version, the database version and in the case of Microsoft SQL Server the ODBC driver version. We recommend you use the ODBC driver version 17.4 or newer.

When sending log files to Micro Focus Technical Support always send the entire log file (do not screen shots or segments of the file), as we capture a lot of information about the environment in which the Client was run at the beginning of the log file. In version 7.0 we repeat most of this information if a log switch occurs during the Client run. Knowing exactly what version of the software we are dealing with is very important when trouble shooting.

Trace Files

Tracing is a powerful option that provides details on the internal processing of the Databridge Client.

> ✏️ **Note**
>
> Trace files are only required if you experience a problem that requires further diagnostics by Micro Focus Technical Support. Do not enable tracing during routine operations as the trace files tend to be huge. You can delete these files when you no longer need them.

Trace files are named

traceyyyymmdd`.log`

where *trace* is a user configurable prefix and *yyyymmdd* is the date the trace file was created. The file extension is `.log`. If more than one trace file is created on the same date, the time is added after the date to make the filename unique. Trace files are written to the working directory for the data source.

16.1.5 Using Log and Trace Files to Resolve Issues

When an error or problem occurs, use log and trace files to troubleshoot the cause.

- Review the log file, which contains a record of all data errors.

- To prevent problems caused by excessive trace and log file size, use the `max_file_size` parameters to limit file size. On UNIX, the Client will crash if the trace file exceeds the system imposed file size limit.

- If you are having problems and contact Micro Focus Technical Support, they may request a copy of the log file. We recommend that you use a compression utility before sending the log file.

- If Micro Focus Technical Support requests a trace, make sure that the old trace files are deleted before starting the Client with the `-t nnn` (or `-d`) option. You will need to use a compression utility (such WinZip on Windows and gzip on UNIX) before sending the trace file (which can be quite large). You can use the splitter utility to break up big trace files into smaller, more manageable files. For help on running the splitter program, type `splitter` with no parameters.

  The splitter program can also split binary files (for example, WinZip® files) that are too large to ship as an e-mail attachment. The original file can be reconstructed from the split files by using the copy /B Windows command. When splitting binary files, you must specify the -B option for the splitter program.

16.1.6 Enabling Tracing

> ✏️ **Note**
>
> We recommend that you enable trace options only when directed to do so by Micro Focus Technical Support. Specifically, avoid full tracing, SQL tracing, protocol tracing, or API tracing. The volume of logging data is so large it can dramatically slow performance of the Client and fill up your hard disk. Compress files using a compression utility before you send them to Micro Focus Technical Support for analysis. Very large trace files should be broken into manageable pieces with the splitter utility. For help on running the splitter utility, type `splitter` with no parameters.

The trace option controls the volume and type of information written to the trace file.

To enable a trace using dbutility

1. Determine the type of trace you want. Then, add the value for each tracing option (see the table below), and use the result for *nnnn*.

2. Specify the `-t nnnn` (or the `-d`) option using the following syntax:

   dbutility -t *nnnn* command *arguments*
   dbutility -d command *arguments*

   where *nnnn* is a bit mask that specifies the tracing option. You can prefix it with 0x to provide the value in hex.

   If you are not sure which tracing masks to use, use the `-d` option. This is the equivalent of `-t 0xB7F`, which enables the most useful trace options.

   You can enter other command-line options, such as `-U`, `-P`, and `-D` with the trace option. The order is not important as long as all dash (-) options precede each command line argument. (See dbutility Command-Line Options.)

3. (Optional) To analyze performance, you can use an additional command line option, `-m`. This option includes a five-digit millisecond timer in all output messages. The timer is appended to the timestamp as (*mmmmm*).

4. (Optional) To change the trace option when the Databridge Client is running, use the commands explained in Controlling and Monitoring dbutility.

To enable a trace from the Administrative Console

To create a trace file, you can use the available options in the Administrative Console by clicking on the "Trace and Log Options" item in the data source's **Advanced** menu. If there is no active run for the data source the trace option you select will be applied to the next launched run, and if there is an active run the tracing will be dynamically enabled for the run in question. The tracing options are not persistent once they are used the Administrative Console clears them. If you want to start a run with tracing a simpler option is to use the **Process (with options)** item in the **Advanced** menu of the data source and select the `-d` options, which will give the default tracing, which is why you should use unless we tell you otherwise.

To enable tracing for a `clone` command only, the **Clone** item in the **Advanced** menu of the data source also allows you to select the `-d` option. Alternatively you can clicking on the "Trace and Log Options" item in the data source's **Advanced** menu and select the desired trace option.

To stop tracing, click on "Select None" in the **Trace and Log Options** dialog and push OK.

16.1.7 Trace Options

Decimal	Hexadecimal	Description
0	0	Disables tracing.
1	0x1	Writes log messages to the trace file in addition to trace information.
2	0x2	Traces all SQL commands as the Databridge Client passes them to the relational database. Typically, these messages are SELECT or UPDATE SQL statements and stored procedure calls.
4	0x4	Traces all DBServer or DBEnterprise communications and key actions associated with Databridge on the host, including RPC calls such as DB_SELECT and DB_READ and their responses.
8	0x8	Traces information on the Databridge Client control tables as they are loaded from the relational database (that is, load tracing).
16	0x10	Enables relational database API tracing, which traces calls from the Databridge Client to the ODBC, OCI or CLI APIs.
32	0x20	Traces the records that are written to temporary data files (or UNIX pipes) and used by the bulk loader utility during the data extraction phase of cloning.
64	0x40	Traces information exchanged between the Databridge Server and the Databridge Client. The blocks of data are traced as they are read and written to the TCP interface. The messages are listed in DEBUG format, which is an offset followed by 16 bytes in hexadecimal, followed by the same 16 bytes interpreted as EBCDIC text. The non-printable EBCDIC characters are displayed as periods (.).
128	0x80	Traces all messages that are routed through the Databridge Client Manager (primarily messages from the Client Console and Client Configurator to the Client, DBClient).
256	0x100	Traces debugging output that is temporarily added to the Databridge Client (primarily engineering releases).
512	0x200	Displays the configuration file parameters as they are processed.
1024	0x400	Enables exchange of traces information between DBClient (or DBClntCfgServer) and the service. The output looks like a DBServer protocol trace, except for the fact that all the data is ASCII.
2048	0x800	Enables SQL tracing while running user scripts during `define` and `redefine` commands.
4096	0x1000	Prints the Read_CB exit line in the trace file. This option is useful only for determining when the execution of a SQL statement ends because the start of the subsequent wait for TCP input is not traced.
8192	0x2000	Traces DOC records. This option provides the same information you would get by setting trace option bit 4, which traces all messages used in server communications. This bit allows you to trace only the DOC records. When used in conjunction with 4 bit, this bit is redundant.
16,384	0x4000	This bit is reserved for internal use only.
32,768	0x8000	This bit is reserved for internal use only.

Decimal	Hexadecimal	Description
65,536	0x10000	Enables verbose tracing.
131,072	0x20000	Enables thread tracing.
262,144	0x40000	Enables DMSII buffer management tracing.
524,288	0x80000	Enables row count tracing.
1,048,576	0x100000	Enables SQL buffer size calculations.
2,097,152	0x200000	Enables load balancing tracing.
4,194,304	0x400000	Enables host variable tracing.

Examples

Following are different ways you can set the logging options.

Log Option Example (Decimal and Dexadecimal)	Result
dbutility -t 7 dbutility -t 0x7	Traces log data (1), SQL (2) and host events (4)
dbutility -t 2943 dbutility -t 0xB7F dbutility -d	Traces the most commonly desirable options. **NOTE**: Whenever Micro Focus Technical Support asks
you for a trace, use the `-d` option, unless you are told otherwise.	

16.1.8 Trace Messages

Any of the messages in this section may appear in the trace file, depending on which options you select when you execute dbutility. See Enabling a Trace. Successful executions of dbutility are separated by a line of 132 equal signs (=).

Database API Trace

Database API tracing is available via the `-t 16` or `-t 0x10` command-line option. The API trace messages trace calls to ODBC (Microsoft SQL Server) or OCI (Oracle). The following messages may appear when you use database API tracing:

Message	Description
Abort_Transaction:	This message indicates that the Databridge Client is making an API call to rollback the current transaction group.
Begin_Transaction:	This message indicates that Databridge Client is starting a transaction group.
BindColumnPtr, stmt=*nnnn*:	This message only appears when the configuration parameter `aux_stmts` has a nonzero value. It indicates that the columns involving a host variable in the SQL statement that was just parsed are being bound to a memory address. This message follows every Parse_SQL message.
Bind_Record: col=*number*, name=*colname*, ofs=*number*, size=*number*, type=*number*	This message appears when the various data columns referenced explicitly or implicitly in a select statement are bound to fields in a program structure. This messages lists the column number (col=*number*), item name (name=*colname*), offset of the field in the structure expressed as a hexadecimal number (entry ofs=*number*), size of the field (in bytes) expressed as a decimal number (size=*number*), and code for the `sql_type` of the column (type=*number*).
Cancel_SQL:	This message indicates that Databridge Client canceled a SQL statement that failed to complete in the designated time. The timer thread performs this operation when it determines that the threshold specified by the configuration parameter `sql_exec_timeout` has been reached.
Close_Database:	This message indicates that a database session has been closed. The Databridge Client typically uses two database sessions at a time.
Commit_Transaction:	This message indicates that the Databridge Client is making an API call to commit the current transaction group.
Execute_PreParsed_ SQL for stmt number, table '*name*'	This message is immediately followed by the SQL_DATA message, which displays the actual values of the host variables for the pre-parsed SQL statement that is being executed.
Execute_SQL:	This message indicates that the Databridge Client called the Execute_SQL procedure, which executes most SQL statements not involving host variables. This call is preceded by one or more calls on Process SQL, which constructs the SQL statements in a temporary buffer.
Execute_SQL_Direct:	This message indicates that the Databridge Client called the Execute_SQL_Direct procedure, which executes SQL statements directly (versus from the buffer that Process_SQL creates).
Fetch_Results: No more rows	This message appears when the Databridge Client loads the Client control tables and indicates the no more rows are available in the select statement result.
Fetch_Results: Row retrieved	This message appears when the Databridge Client loads the Client control tables and indicates that the Databridge Client successfully read the row when it retrieved the results of a select statement.

Message	Description
OCIBindByPosition: col_no= *nn*, addr=0x*hhhhhhhh*, len =0x*hhhh*, ind=*nn*	This message, which is limited to the Databridge Client for Oracle, indicates that the column in the given position in the parsed SQL statement was bound to a host variable at the given address and length.
Open_Database: user =userid, pwd=**, {db=database	data source=src}, rslt= *dbhandle*
Open_Stmt: Opened stmt *nnnn*	This message indicates that the Client allocates a new stmt structure associated with a SQL statement that uses host variables. The Client allocates a maximum number of auxiliary statements (configuration file parameter `aux_stmts`) before it starts reusing these structures. The Client reuses the least recently used (the oldest) stmt in this case.
Oracle NLS parameter name= *value*	This message appears when the Databridge Oracle Client connects to the database. One of the first things it does is to read the NLS parameters to determine the language and decimal character being used. The Client then automatically adjusts the connection so the Client operates properly in the given environment. The `bcp_delim` parameter is automatically set the value that SQL*Loader expects.
Parse_SQL: SQL[number]=stmt	This message indicates that the SQL statement involving a host variable is being parsed using the stmt in question. Using host variables improves performance by only parsing statements, binding the host variables to specific columns, and executing the statement multiple time after setting the host variables to the desired values.
Procedure_Exists(*name*)	This message indicates that the Databridge Client called the procedure Procedure_Exists, which reads the data dictionary to determine if the given stored procedure exists.
Process_SQL: SQL=SQLText	This message, which should not be confused with a similar SQL tracing message, overrides the SQL trace when both SQL and API tracing are enabled. This avoids having duplicate entries in the trace.
SQLBindParameter: col_no=nn , addr=0x*hhhhhhhh*, len=0x*hhhh*, ind_addr=0x*hhhhhhhh*, ind=*nn*	This message, which applies to all ODBC Clients, indicates that the given column in the prepared SQL statement was bound to a host variable at the given address and the given length. The ind column is an indicator that is used to mark columns as being null.
SQL_DATA[number]= ...|...|...	This message, which should not be confused with a similar SQL tracing message, overrides the SQL trace when both SQL and API tracing are enabled.
Table_Exists (*name*)	This message indicates that the Databridge Client called the procedure Table_Exists, which reads the data dictionary to determine if the given table exists.

Bulk Loader Trace

Bulk loader tracing is available via the `-t 32` or `-t 0x20` command-line option. Bulk loader data tracing results in records of the bulk loader data files (or UNIX pipes) being written to the trace file during the data extraction phase of cloning. Bulk loader data trace messages are in the following form:

Message	Description
Build_Pipe_Stream: table=name, record=data	where *data* is the actual ASCII data that is written to the temporary data file (or UNIX pipe) used by the bulk loader utility.

Configuration File Trace

The configuration file trace is available via the `-t 512` or `-t 0x200` command-line option. These messages log configuration file parameters as they are being processed.

For example:

`CONFIG: nnn. Config_file_line`

If a binary configuration file is used, the Client uses the same output procedure as the export command to write the text version of configuration file into the trace file.

DBServer Message Trace

Databridge Server message tracing is available via the `-t 4` or `-t 0x4` command-line option. This trace highlights pertinent information during communications with Databridge Server on the host. These messages are listed in the trace file and may include the following:

Message	Description
Common_Process: DBDeSelect Table=*name*, stridx= *nnnn*, rslt= *errorcode*	The DBDeselect RPC call is used to deselect data sets that need to be excluded from change tracking. An example would be a data set whose AA Values are invalidated by a garbage collection reorganization. This message shows the name of the data set and its related structure index. If *errorcode* is nonzero, this message is followed by a Host message.
Common_Process: DBSelect Table=*name*, stridx=*nnnn*, rslt=*errorcode*	The DBSelect RPC call is used to select data sets when the Databridge Client starts a `process` or a `clone` command. This message shows the name of the data set and its related structure. If *errorcode* is nonzero, this message is followed by a Host message.
Datasets_CB: *dataset_name* [/rectype] (strnum), subtype = *dd*, ds_options=*0xhhhhhhhh*, misc_flags = *0xhhhhhhhh*	CB stands for callback. This message shows the receipt of a data set information record from the Databridge Server during the execution of a `define` or `redefine` command.
Define_Table_Items: table= *name*, item=*name data_type (sql_length)*	This message shows the data type and SQL length of data items as they are inserted in the Client control tables. This occurs during execution of the `define` or `redefine` command.
Get_Response: Req=*req* Rslt=*rslt* Len=*len*	where *req* is the request type (RPC name), rslt is the returned status (typically OK), and *len* is the number of bytes of data that follow the status in the response packet. This message indicates that the Databridge Client received a response to a remote procedure call other than DBREAD or DBWAIT.
Layout_CB: DataSet = *name*[/rectype], item (*number*) = *name*, data_type = *dd*, dlen = *dd*, scaling = *dd*	CB stands for callback. This message shows the receipt of a data set item layout information record from the Databridge Server during the execution of a `define` or `redefine` command.
Read_CB: Type=*typename* StrIdx=*iii*, aa= *hhhhhhhhhhhh*	This message indicates that the Databridge Client received a response from the Databridge Server in response to a DBREAD or DBWAIT remote procedure call. *typename* is the response name (CREATE, DELETE, MODIFY, STATE, DOC, MODIFY_BI, or MODIFY_AI) *iii* is the structure index assigned to the structure when it is selected via the DBSELECT call *hhhhhhhhhhhh* is the value of the absolute address of the DMSII record (For protocol levels greater than 6, this value is all zeros unless the data set uses the AA Value as a key.)

Message	Description
Read_CB: Type=DOC[AF_HEADER], Afn=*afn*, RectoQPT=*dd*, UpdateLev=*ul*, TS='ts', DMSRel=*nnn*, DMSBuild=*nnn*, AudLev=*nnn*, AFSize=*nnn*, AFOrigin=*orig*,firstABSN=*absn1*, lastABSN=*absn2*	This message is always sent to the Client when the Databridge Engine opens a new audit file. It contains information about the audit file, including the audit file number afn, the update level *ul*, and the audit file origin *orig*. This last item is particularly useful when using DBEnterprise as it allows the Client to detect what access method is being used to read the audit file (i.e. direct-disk, indirect-disk or cache).
Read_CB: Type=DOC *[type]*, . . .	This message is printed only when the enable_doc_records parameter is set to Yes in the configuration file. The Databridge Client uses the DOC record only for debugging purposes. DOC records are documentation records that are optionally sent by the Databridge Engine to document the events that occur while Databridge Engine is reading the audit files.
	The various types include BEG_TRAN, CLOSE, END_TRAN, OPEN, REORG. The rest of the message varies based on the DOC record type. In the case of BEG_TRAN and END_TRAN, the message includes the transaction count, while OPEN and CLOSE messages give information about the job number, the task number and the task name of the program that accessed the DMSII database. REORG DOC records are sent to Client to notify it that some sort of reorganization has occurred for the specified structure index, which is printed out in the message. The remaining DOC records are only identified by type with no additional information.
Read_CB: Type=LINK_AI StrIdx= *number*	This message indicates that the Databridge Client received a DMSII LINK after image from the Databridge Server in response to a DBREAD or DBWAIT remote procedure call.

Information Trace

Information tracing occurs via the default `-t 1` or `-t 0x1` command-line option. The information messages include the following messages that are not displayed on the screen, as well as all messages that are displayed on the screen.

Message	Description
command line echo	Everything you type at the command line is echoed in the trace file.
Current date is: *day month year*	This is the date you ran the Client. It is used to identify sections of the trace file as there might be several runs of dbutility logged to the same trace file.
Negotiated Protocol level = *n*, Host version *n.n*	This is the negotiated protocol level that the Databridge Client and the Databridge Server are using to communicate. For example, a protocol level 7 Databridge Client and a protocol level 6 server use a negotiated protocol level of 6 in all communications.

Load Trace

Load tracing is available via the `-t 8` or `-t 0x8` command-line option. Load tracing messages refer to the Client control tables. To check these tables, use the dbutility `display` command. See dbutility Commands.

The Load External messages are displayed only during a dbutility `define` or `redefine` command. They indicate that the Databridge Client is reading table names defined in other data sources to make sure that any newly-defined tables and indexes do not duplicate table names or index names defined previously in other data sources.

The following messages may appear when you use load tracing:

Message	Description
Load: DataSet = *name[/rectype]*, strnum = *number*, AFN = *afn*, ABSN = *absn*	This message appears for every data set loaded from the DATASETS Client control table. The message lists the data set name (and the record type for variable-format data sets) as well as the structure number, the audit file number, and the audit block serial number. For most commands, this message appears for only those data sets whose active column is 1.
Load: dms_item = *name*, item_number = *number*, DataSet = *name[/rectype]*	This message appears for every DMS item loaded from the DMS_ITEMS Client control table. The message lists the data set name (and the record type for variable-format data sets) as well as the DMSII item name and the corresponding item number.
	This message does not appear during the process and clone commands because all of the information the DMS_ITEMS entries contain is in the DATAITEMS Client control table.
Load: datatable = *name*, DataSet = *name[/rectype]*	This message appears for every data table loaded from the DATATABLES Client control table. The message lists the data set name (and the record type for variable-format data sets) and the table name.
Load: dataitem = *name*, datatable = *name*	This message appears for every data table loaded from the DATAITEMS Client control table. The message also displays the table name to which the item belongs.
Load External: DataSource = *name*, TableName = *name*, IndexName = *name*	The Load External messages appear during a dbutility `define` or `redefine` command only. They indicate that the Databridge Client is reading table names defined in other data sources to make sure that any newly-defined tables and indexes do not duplicate table names or index names defined previously in other data sources.
Load: global_dataset = Global_DataSet, AFN = *afn*, ABSN = *absn*	This message appears when the global data set is loaded from the DATASETS Client control table. Under normal circumstances, the AFN and the ABSN is 0 as the Databridge Client sets these entries to 0 after it propagates the global stateinfo for all data sets that have a value of 1 in their in_sync columns before the process command terminates.

If an OCCURS table filter is being used the Load Trace also includes a display of the filter data, which can also be generated by using the `display` command of the **makefilter** utility. This immediately follows the log message "Loading binary filter file "config\dbfilter.cfg".

```
Filter:  NumFilters = nnn, NumFilterEntries = nnn, ConstantPoolSize=0xhhhh
Constant Pool:
0000 hh hh hh . . .
Table 'name', filter_start = nnn, num_entries = nnn
    Type = ColumnName: item_name = 'name'
    Type = Constant: associated item,_name = 'name', offset = ddd, length = lll
    Type = Operator: op
    Type = Operator: END
. . .
```

Each OCCURS table that is being filtered has a starting index and a count that represents the number of tokens associated with the table. Constants are associated with an item, whose properties they share. Constants are put into a global constant pool that is shown in debug format. Individual constants are represented in DMSII native form (i.e. binary data). The offset into the constant pool is used to reference a constant, its length is the same as that of the associated data item. An offset of -1 is used to denote a NULL. The filters are represented in reverse polish form. The various operators are represented by 2 or 3 letter terms such as EQL, NEQ, AND, OR and so on. Every filter ends with an END operator.

Protocol Trace

Protocol tracing is available via the `-t 64` or `-t 0x40` command-line option. Protocol traces display the data that is read from or written to the TCP/IP interface during all communication with the Databridge Server.

Message	Description
read: number_of_bytes_read	Received data. These messages are followed by a hexadecimal dump of data in DEBUG format, with all data interpreted as EBCDIC text. Non-printable characters are displayed as periods (.).
write: number_of_bytes_read	Sent data. These messages are followed by a hexadecimal dump of data in DEBUG format, with all data interpreted as EBCDIC text displayed in ASCII. (Non-printable characters are displayed as periods (.).

SQL Trace

SQL tracing is available via the `-t 2` or `-t 0x2` command-line option. The following SQL messages may appear in the log file:

Message	Description
SQL=*sqltext*	Indicates general SQL tracing where sqltext is the actual SQL command sent to the relational database.
SQL[number]=*sqltext*	Indicates SQL tracing that involves host variables in the Databridge Client for Oracle when the configuration parameter `aux_stmts` has a nonzero value. *Number* is the stmt number, and sqltext is the actual SQL command sent to the relational database.
SQL_DATA[number]= ...|...|...	This message shows the data being passed to the database API when executing updates involving previously parsed SQL statements that use host variables. *Number* is the stmt number.

User Script Trace

User script tracing is available via the `-t 2048` or `-t 0x800` command line options. This causes the SQL statements in user scripts to be traced only during a `define` or `redefine` command. This option provides a subset of the SQL Trace. This option has no effect if SQL tracing is enabled.

Read Callback Exit Trace

Read callback exit tracing is available via the `-t 4096` or `-t 0x1000` command line options. This causes the Client to display the message shown below when it exits the read call back procedure. This indicates that the Client is done processing a data buffer and is ready to read the next one. This is only useful when looking for reasons why the Client is running slow. In such cases we recommend that the command line option `-m` be used, as this will give you a finer granularity timestamp.

```
Read_CB: Exit
```

DOC Record Trace

DOC record tracing is available via the `-t 8192` or `-t 0x2000` command line options. This causes the DOC records received from the Databridge Engine to be traced during a `process` or `clone` command. This option is redundant when the Databridge Server message tracing is enabled, see DBServer Message Trace.

Verbose Trace

Verbose tracing is available via the `-t 65536` or `-t 0x10000` command line options. These messages are described in the *Databridge Errors and Messages Guide* and identified by using the TR_VERBOSE bit, which is the above-mentioned bit in the trace mask.

Thread Trace

Thread tracing is available via the `-t 131072` or `-t 0x20000` command line options. These messages include the following:

Message	Description
`Bulk_loader thread[nn] {started	ready	exiting}`	(Windows only) These messages indicate a change in the state of the bulk loader thread(s). • `started` indicates that the thread was started. The thread is only started when there are tables to be bulk loaded. • `ready` indicates that the thread is ready to process requests to run the bulk loader. The bulk loader thread gets the load request from its work queue. If there is none, it blocks until one becomes available. • `exiting` indicates that the thread is no longer needed and is exiting. At this point, the Client is ready to start processing audit files, as soon as the index thread finishes creating indexes for all of the tables that were cloned.
`Bulk loader thread[nn] starting {sql*loader	bcp} for table 'name'`	(Windows only) This message indicates that the bulk loader thread in question is launching the bulk loader for the specified table.
`Console_Reader thread {starting	ready	exiting }`	These messages indicate a state change in the Console thread. The command line Client uses this thread to read console commands from the keyboard. The service-based Client (DBClient) uses this thread to handle console commands that originate in the GUI Console and are passed to the Client as RPCs. The various states indicate the following: • `starting` indicates that the thread was successfully started. • `ready` indicates that the thread is waiting for keyboard input in the case of dbutility and waiting for an RPC in the case of DBClient. • `exiting` means that the thread is about to exit.
`Index_creator thread {started	ready	exiting}`	These messages indicate a state change in the index creator thread. • `started` indicates that the thread was started because there are tables for which indexes must be created. • `ready` indicates that the thread is ready to process requests to create indexes for tables. The index creator thread gets the index creation request from its work queue. If there is none, it blocks until one becomes available. • `exiting` indicates that the thread is no longer needed and is exiting. At this point, the Client is ready to start processing audit files.
`Update Worker thread [nn] empty_work_queue, EOT=n, SDW=n, n_active_threads=nn`	This message, which is only seen when using multi-threaded updates,.indicates that the specified update worker is performing an update for the given table. It shows the address of the work descriptor storage block that is used to queue the request. This information is only useful if you are diagnosing a problem that deals with the management of work descriptor storage blocks.
`Update Worker thread [nn] {started	ready	exiting}`	These messages, which are only seen when using multi-threaded updates, indicate a state change in one of the update worker threads. > `started` indicates that the thread was started. The update threads are started at the start of the process command. • `ready` indicates that the thread is ready to process requests to execute updates. The update worker threads get the update requests from their work queues. If there is no request in the queue, the thread blocks until one becomes available. • `exiting` indicates that the thread is no longer needed and that it is exiting. This only happens when the Client is shutting down.

Message	Description
Waiting for bulk_loader thread to finish	(Windows only) This message indicates that the bulk_loader thread is not finished loading tables. The main thread, which is ready to enter the fixup phase, must wait for these operations to complete before updates can be processed. When the bulk loader thread is finished it displays the message "`Bulk_loader thread exiting`."
Waiting for index_creator thread to finish	(Windows only) This message indicates that the index_creator thread is not finished. The main thread, which is ready to enter the fixup phase, must wait for these operations to complete before updates can be processed. When the index creator thread is finished, it displays the message "`Index_creator thread exiting`."

DMS Buffer Trace

Buffer size tracing is available via the `-t 262144` or `-t 0x40000` command line options. This causes the Client to display the following messages when a DMS buffer is gotten from the free buffer list or when it is returned to the list.

Message	Description
XDR_Get_Buffer: buf=*0xhhhhhhhh*, buf_cnt=*dd*, sem_cnt=*dd*	This line is printed every time a DMS buffer is gotten off the free list; buf is the address of the buffer, `buf_cnt` is the number of DMS buffer that have been allocated and sem_cnt is the number of buffers that are available (note that all of these may not yet have been allocated).
XDR_Return_Buffer: buf=*0xhhhhhhhh*, sem_cnt=*dd*	This line is printed every time a DMS buffer is returned to the free list; buf is the address of the buffer, and sem_cnt is the number of buffers that are available (note that all of these may not yet have been allocated).

Row Count Trace

Row count tracing is available via the `-t 524288` or `- t 0x80000` command line options. This causes the Client to display the following message when the Client fetches the row count following the execution of a SQL statement. Note that in the case of user scripts, using the `-v` option causes the exact same output to appear in the log file when a user script executes an update statement.

`Rows updated = dd` The value *dd* represents the number of rows updated.

Buffer Size Trace

Buffer size tracing is available via the `-t 1048576` or `-t 0x100000` command line options. This causes the Client to display the following messages at startup when the control tables are being loaded.

Message	Description
Item name: hv_len=*dd*, sqlcmd(oh=*dd*, gr=*dd*), ins=(*dd*,*dd*), upd(*dd*, *dd*); total ins=(*dd*,*dd*), upd=(*dd*,*dd*)	This line is printed every time a data item is processed. It shows the contributions of the item to the various SQL buffer sizes.
Computed SQLcmd lengths for table name: [hv_len = *dd*,], sqlbuf_len = *dd*, sql_buf2_len = *dd*, sql_buf_size = *dd*, [thr_sql_buf_size = *dd*,] sql_buf2_size = *dd*	At the end of the processing of the items in a table this summary line is displayed. In the case of the Flat File Client the sections enclosed in square brackets are not present.
Buffer sizes are gSQLcmd/SQLCMDLEN = *dd/dd*, gSQLcmd2 = *dd*	When all the tables have been processed this line is displayed. It shows the sizes for the two SQL buffers used by the main thread. When using multi-threaded updates refer to the previous message to see what the size of the update thread SQL buffers are.

16.2 Appendix B: dbutility Commands and Options

This appendix provides a list of all dbutility commands and command-line options. For a complete reference of command-line options paired with their equivalent environment variables and configuration file parameters, see Reference Tables.

> ✏️ **Note**
>
> The hyphen is used for all command options and is valid for Windows and UNIX. Windows users can substitute the slash (/) for the hyphen.

16.2.1 dbutility Commands

The following table lists all of the dbutility commands and their related command-line options.

Example

Assume you want to override the environment variable for the relational database name (DBDATABASE) and enter a blank instead (which is the same as the using the default database name). To do this, you could enter either of the following:

```
dbutility -U usera -P secret -D "" configure
dbutility -U usera -D -P secret configure
```

Command	Purpose and Result
dbutility clone *datasource dataset1* *[dataset2… datasetn]*	Related command-line options: Signon options, `-c`, `-f`, `-l`, `-m`, `-o`, `-s`, `-t`, `-u`, `-v`, `-x`, `-z`, `-A`, `-F`, `-K`, `-L`, `-N`, -T Run this command to `clone` or `reclone` (not track changes) a list of data sets. Using the dbutility clone command is a convenient way of cloning a few data sets without having to update the DATASETS Client control table. For recloning with dbutility clone, see Recloning.
dbutility configure	Related command-line options: Signon options, `-f`, `-m`, `-t`, `-u`, `-L`, `-T` Run once for each set of Client control tables you want to create. The result is empty Client control tables and their indexes in the relational database. See Creating Client Control Tables. **NOTE:** The only time you would run dbutility configure again for the same relational database is if you previously executed a dbutility dropall command.
dbutility define *datasource host port*	Related command-line options: Signon options, `-f`, `-m`, `-t`, `-u`, `-v`, `-L`, `-T` Run once for each data source you want to define except when customizing with user scripts. See Customizing with User Scripts. The result is a data source entry in the DATASOURCES Client control table and all other Client control tables containing the DMSII database layout and corresponding relational database table schema information. See Defining a Data Source.
dbutility display *datasource*	Related command-line options: Signon options, `-a`, `-f`, `-m`, `-t`, `-B`, `-L`, `-T` Run this command to create a report of the Databridge Client control tables for the specified data source. The report is written to the log file in the logs directory. For more information about log files, see Log and Trace Files. Use this command to check the results of the dbutility define command or script customization. **NOTE:** When you use dbutility display, the column names for the Client control tables are abbreviated. The actual column names and the abbreviated column names are listed for each Client control table in Chapter 6, Databridge Client Control Tables.

Command	Purpose and Result
dbutility drop *datasource*	Related command-line options: Signon options, `-m`, `-t`, `-v`, `-L`, `-T` Run this command to "undo" the results of a dbutility define, generate, process, and clone for a specified data source. dbutility drop does the following: • Drops tables and their associated stored procedures • Removes the script files in the current directory • Deletes the DMSII record layout and relational database table schema information (for the specified data source) from the Client control tables CAUTION: We recommend that you create a separate directory for each data source. When you must drop a data source, make sure that the current directory is the directory you created for the data source. Then, use the drop (not dropall) command to drop each individual data source. Failure to do this results in dbutility not being able to locate the required scripts, which causes it to terminate with an error.
dbutility dropall	Related command-line options: Signon options, `-m`, `-t`, `-u`, `-L`, `-T` Run this command to drop all tables that have been created by the Databridge Client, as well as to remove the script files in the current directory. Note that your other non Databridge tables are not affected. If you are executing dbutility commands from more than one directory, the dbutility dropall command locates scripts in the current directory only. In this case, it drops the scripts that it can find and then refrains from removing the Client control table entries for those data sources that it could not properly delete (that is, the data sources whose scripts are in other directories). Therefore, we recommend that you do either of the following: • Change the directory and repeat the dbutility dropall command. • Drop each data source via the drop command, then use dbutility dropall for the final data source. Typically, you do not need to use this command.
dbutility options export *filename*	Related command-line options: `-E`, `-u` Exports the binary Client configuration file to an editable text file (`dbridge.ini`, by default) that can then be imported, using the import command, for use with the Databridge Client. See Export or Import a Configuration File.
dbutility generate *datasource*	Related command-line options: Signon options, `-f`, `-m`, `-t`, `-u`, `-v`, `-L`, `-T` Generates the Databridge Client script files required to populate the Databridge data tables in the relational database. The result is a set of scripts in the dbscripts subdirectory of the working directory. There are approximately five scripts for each DMSII data set. See Generating Databridge Client Scripts.

Command	Purpose and Result
dbutility options import *filename*	Related command-line options: `-E`, `-f` *filename*, `-u`
	Reads the specified input file and writes it as a binary Client configuration file (`dbridge.cfg`, by default). See Export or Import Configuration Files.
dbutility process *datasource*	Related command-line options: Signon options, `-f`, `-l`, `-m`, `-o`, `-s`, `-t`, `-v`, `-w`, `-z`, `-C`, `-K`, `-N`, `-L`, `-T`
	Run the first time to populate the Databridge tables in the relational database with the DMSII database data. Run subsequent times to update the relational database with only the changes that have been made to the DMSII database since the last time you ran dbutility process.
	NOTE: dbutility process can also re-clone instead of update if ds_mode=0 when you run dbutility process.
	See Populating the Databridge Data Tables and Updating the Databridge Data Tables.
dbutility redefine *datasource*	Related command-line options: Signon options, `-f`, `-m`, `-t`, `-u`, `-v`, `-r`, `-R`, `-L`, `-T`
	The `redefine` command compares the old and new layouts of all the tables generated for data sets whose status_bits columns indicate a structural reorganization.
	The `redefine` command also does the following:
	• If a new data set appears, the redefine command defines it with its corresponding active column set to 0 in the DATASETS Client control table (unless the `suppress_new_datasets` parameter is set to False). When the active column is set to 0, the redefine command will not perform any mapping for it unless you set the active column in the DATASETS entry to 1 in the corresponding data set mapping customization user script.
	• If a data set no longer exists, the `redefine` command deletes all the associated Client control table entries, but does not drop the data tables and their associated stored procedures. You must delete them by running the corresponding scripts (these are not removed either).
	• The `redefine` command refreshes the data set mapping in three instances. First, the mapping is refreshed when the data set's DS_Needs_Remapping bit is set (value 4). Use this method when you modify the DATASETS and DMS_ITEMS tables. Because the data set mapping customization scripts are not run in this instance, you must execute the runscript command prior to executing the redefine command. Secondly, mapping is refreshed if a data set's active column is set to 1, and the DS_Needs_Mapping bit is set (value 1) in the status_bits column. Thirdly, mapping is refreshed when you set the DS_Needs_Redefining bit (value 8). In this case, the `redefine` command refreshes the DMSII layout as well.
	• If a data set has an active column set to 0, and the DS_Needs_Mapping bit is set (value 1) in the status_bits column, the layout information is refreshed, but no mapping is performed.
	• The `redefine` command sets the active columns of the Client control tables equal to zero for data sets that contain global data. No other data sets are affected by the `redefine` command. You must execute a generate command after a `redefine` command to update the scripts.

Command	Purpose and Result
dbutility [options] refresh *datasource dataset*	Related command-line options: Signon options The `refresh` command enables you to drop and recreate all of the stored procedures for the tables associated with the given data set in the specified data source. It is a variation of the `runscript` command that is designed to run portions of the Databridge Client scripts (`script.drop.tablename` and `script.create.tablename`). This command is useful when you want to add a new column to a table after a DMSII reorganization. If _ALL is specified for *dataset*, the program refreshes the stored procedures for all active tables. If a specific data set is specified, only the stored procedures for that data set refreshed. All data sets specified must already exist. NOTE: When variable-format data sets are involved, the tables for all the record types that have their active column set to 1 in the DATA SETS Client control table are refreshed.
dbutility reload datasource *backupfile* [*dataset, dataset2…*]	Related command-line options: Signon options, `-f`, `-k`, `-m`, `-t`, `-L`, `-T` Restores the Client control tables from a file that the `unload` command created. If a *datasource* of _ALL is specified, all data sources contained in the backup file are restored. If a specific data source is specified, only the entries for that data source are restored from the file. The reload operation is sensitive to the version of the program that wrote the backup file. As an option, you can provide a list of data sets to be loaded. If such a list does not exist, all data sets for the given data source are reloaded. The `-k` option preserves the stateinfo for data sets whose format levels and item counts remain unchanged.
dbutility rem . . .	A dummy command that opens the log file and echoes the command line into it. The purpose of this command is to make it possible for script files or operators to create a log file entry to document the action that was taken. For example: `dbutility rem accidentally killed the Client - JaneDoe`
dbutility reorg *datasource*	Related command-line options: Signon options Generates new scripts for stored procedures and refreshes the relational database stored procedures. The `reorg` command resets ds_mode to 2 (indicating that the data set is in tracking mode). Typically, you would use the `reorg` command after the `redefine` command when a reorganization has occurred on the DMSII database.
dbutility rowcounts *datasource*	Related command-line options: Signon options, `-f`, `-m`, `-t`, B, `-L`, `-T` Creates a report in the log file with all the row counts for all the active tables associated with the data source.

Command	Purpose and Result
dbutility runscript *filename*	Related command-line options: Signon options, `-m`, `-n`, `-t`, `-L`, `-T` Use this command to run user scripts (for example, script.user_define.primary_tablename) or Databridge Client scripts (for example, `script.create.tablename`). The Databridge Client expects the user scripts to be located in the directory specified by `user_script_dir` in the Databridge Client configuration file. To override this directory specification, use the `-n` option, as follows: `dbutility -n runscript drive:\directory\scriptfilename` The `runscript` command automatically enables SQL tracing and logging (similar to setting the `-t` 3 option). The `runscript` command runs in transaction mode so that if an error occurs, all changes are rolled back. You can then fix your scripts and run them again.
dbutility switchaudit *datasource*	Related command-line options: Signon options, `-f`, `-k`, `-m`, `-t`, `-v`, `-L`, `-T` Run this command to close an audit file on the host. This ensures that you get the most current information possible because the Databridge Engine does not read the currently open audit file. DMSII audit files are explained in detail in the *Databridge Host Administrator's Guide*. IMPORTANT: Do not use this command unless you check with the DMSII database administrator first.
dbutility tcptest *datasource [host port]* *length count*	Related command-line options: Signon options, `-f`, `-m`, `-t`, `-L`, `-T` Run to test the TCP/IP interface between the Databridge Client and the server. You can use this command as a diagnostic tool to help troubleshoot slow network connections. If the data source is already defined, you do not have to specify the host and the port parameters; the program reads them from the DATASOURCES table entry instead. *Length* is the size of the message to use and count is the number of iterations that should be executed. 8000 and 1000 are standard values with these options.
dbutility unload *datasource backupfile*	Related command-line options: Signon options, `-f`, `-k`, `-m`, `-t`, `-L`, `-O`, `-T` Creates a file containing a backup of the Client control tables. If a *datasource* of _ALL is specified, all of the data sources that are found in the Client control tables are written to the backup file *backupfile*. If any other data source is specified, only the entries for that data source are written to the file.

16.2.2 dbutility Command-Line Options

This section explains the command-line options you can enter with dbutility commands, with all lowercase options first in alphabetical order and all uppercase options following. Use the following syntax to include the command-line options:

```
dbutility [options] command
```

where *[options]* begin with the forward slash (/) or hyphen (-) and are followed by a letter and a possible argument, as listed in the following table. If you use a UNIX Client, all options must start with a hyphen (-). Note the following guidelines for using command-line options:

- All options are case-sensitive.
- The options can be used in any order.
- When you enter any of these command-line parameters, do *not* type the [brackets]. The [brackets] indicate that the command-line parameter is optional.
- Following the option letter, you can enter option arguments with or without a space. For example, `-tl` and `-t l` are equivalent.
- If an argument is blank (an empty character string), you can omit it if the next entry on the command line is another option (for example, -D). Otherwise, you must enter the blank argument as `" "` (quotation marks) with both leading and trailing spaces.

Examples

Assume you want to override the environment variable for the relational database name and enter a blank instead (which is the same as the using the default database name). To do this, you could enter either of the following:

```
dbutility -U usera -P secret -D "" configure
dbutility -U usera -D -P secret configure
```

(Both of these examples override the environment variable DBDATABASE.) For a complete reference of command-line options paired with their equivalent environment variables and configuration file parameters, see Reference Tables.

This option	Does this
-?	Displays short help, which includes dbutility command syntax and parameters but not options.
-a	Toggles the setting of the `display_active_only` parameter.
-c	Toggles the setting of the `defer_fixup_phase` parameter during a `clone` command.
-d	When used with any dbutility command, this option enables full tracing. This is the same as entering `-t 8191` (or `-t 0x1FFF`). If you are not sure whether to use the `-d` option or the `-t` option, you may want to use the `-d` option. It is often better to have too much tracing information than not enough.
-f filename	Specifies an input configuration file other than the default filename when used with an import command. If *filename* doesn't start with a backslash () on Windows or a forward slash (/) on UNIX, it is assumed to be in the config subdirectory of the working directory. Conversely, if filename starts with the appropriate slash, it is taken to a full file specification.
-h	Displays long help, which includes dbutility command options, syntax, and parameters.
-k	Used with a `reload` command to preserve the stateinfo for data sets whose format levels and item counts remain unchanged. Used by the `process`, `clone` and `drop` commands in a multi-source environment to force the Client drop tables rather than running the cleanup script. This is designed to be used after a reorg that requires a re-clone, one would otherwise have to physically drop the affected table(s) to get the clone to recreate the table(s) with the new layout. Now you can use the `-k` option on the first data source that gets re-cloned; from thereon the `-k` option must obviously not be used.
-l	(SQL Server only) forces the Client to use the bcp utility instead of the BCP API.
-m	Includes a 5-digit millisecond timer in all trace messages. The millisecond timer is appended to the timestamp in the trace file. This option does not affect log file output.
-n	Used with the `runscript` command to override your entry for `user_script_dir` so that you can run any script that is not located in that directory.
-o	Overrides shutdown periods, including those initiated by the stop_time and end_stop_time values in the DATASOURCES Client control table for the data source entry when the `controlled_execution` configuration file parameter is enabled.

This option	Does this
-r	Forces the parameter `use_dbconfig` to be treated as False when running a redefine command. The `use_dbconfig` parameter is internal and not directly editable. However, this parameter is set to True in the following situations: • When a data source is created using the Client Configurator • When a define command creates the data source and no users scripts are encountered • When you run the dbscriptfixup program In all other cases, including after an upgrade using the migrate program, the `use_dbconfig` parameter is set to False. Its purpose is to ensure that the Client Configurator isn't run with improperly setup Client control tables. This would cause all changes that were made via user scripts to be lost.
-s	Loads relational database tables that cannot be loaded via the bulk loader utility (SQL*Loader for Oracle and bcp for Microsoft SQL Server). *The* -s *option inhibits the use of the bulk loader utility during the data extraction phase of cloning. The Databridge Client loads the table using SQL statements instead.* *CAUTION:** Use this option only when certain tables will not load via the bulk loader utility. Do not use it under normal circumstances. The -s option slows the cloning process considerably. The `-s` option is also used by the createscripts command to add the data source name in all where clause of the SQL statements that are created.
-t *mask*	Enables trace options designated by mask. See Enabling a Trace. If you are unsure whether to use the `-d` option or the `-t` option, you may want to use the `-d` option. It is often better to have too much tracing information than not enough.
-u	Creates override conditions that dbutility would otherwise interpret as a possible user error. These situations include the following: • Creating a second set of Client control tables within one relational database. In this case, the second set of tables must be owned by a different user ID. • Starting over by dropping and creating the Client control tables, even though this removes all of the state information associated with the user tables. • Attempting to define a data source that already exists. With the dbutility `dropall` command, use this option to drop Databridge Client tables that still contain data.
-v	Causes the Databridge Client to write some additional information to the log file and sometimes to the screen. The most useful one is causing user scripts executed by the Client to write the number of rows affected by INSERT, UPDATE and DELETE SQL statements to the log file.
-w	Toggles the setting of the `use_dbwait` parameter.
-x	Makes the `clone` command clone all active data sets *except* for those specified at the command line.
-y	Forces the Client to re-clone data sets with a mode of 11 and 12.

This option	Does this
-z	**CAUTION:** This option is for troubleshooting only in a test environment. **Do not use it in a production environment.** Allows dbutility to simulate a dbutility `process` or `clone` command without actually storing data. For troubleshooting purposes. After the Client control tables are loaded for the specified data source, the program sets a global flag that disables all SQL execution by the ODBC (Microsoft SQL Server) or OCI (Oracle) interface routines. Using this option with statistics enabled (`show_statistics` and `show_perf_stats` both set to True) to determine the rate at which the Databridge Engine and the Databridge Client can process data without including any of the additional delays caused by the relational database. If this rate is significantly slower than the rate you get when the `-z` option is set, you can conclude that the slowness is caused by the actual relational database, which might need to be tuned.
-A	Prevents the Client from deleting the tables when cloning virtual data sets that have the `DSOPT_Ignore_Dups` option bit (value 32) set in the `ds_options` column of the DATASETS Client control table. Instead, it drops their indexes and appends the new records to the tables.
-B	Causes the display command to report only the second half of the DATASETS Client control table to the trace.log file, then causes the program to quit. (In most cases, the information in the second half of the DATASETS Client control table is the only information you actually need from that table.)
-C	Toggles the `inhibit_console` parameter. On UNIX, this doesn't apply if you run `dbutility` as a background run.
-D *databasename*	(Oracle only) Specifies the relational database you are using.
-F *afn*	Makes the Client act as if a `QUIT AFTER afn` command had been executed. Applies only to `process` and `clone` commands. The range of values allowed are 1 through 9999.
-K	Inhibits audit file removal WFLs from running on the host during a process command. This option is also implied during a `clone` command and when used with the `-z` option.
-L	Forces the Client to start using a new log file.
-N	Toggles the setting of the `enable_optimized_sql` parameter.
-O *ODBCdatasource*	Specifies an ODBC data source that the Client uses to connects to the Microsoft SQLServer database.
-P *password*	Defines the password for accessing the relational database.
-R	Treats every data set as if its DS_Needs_Redefining status bit is set and allows the program to rebuild the control tables when you change a configuration parameter such as `optimize_updates` or `read_null_records` .
-T	Forces the Client to start using a new trace file. See Log and Trace Files.
-U *userid*	Specifies the user ID defined in the relational database.

This option	Does this
-V	Used with the unload command, this option lets you specify the control table version you want. To create tables that you can use with an older Client version, use the value that corresponds to that version. Values for the Databridge Client are as follows: • **34**: Version 7.0 SP1 • **33**: Version 7.0 • **31**: Version 7.0 • **30**: Version 6.5 SP1 • **29**: Version 6.5 • **26**: Versions 6.2 and 6.3 • **25**: Version 6.1 SP3 • **24**: Version 6.1 • **23**: Version 6.0 • **22**: Version 5.2.0.12 • **21**: Version 5.2.0.3 • **20**: Version 5.2 (base release) • **19**: Version 5.1
-W	**NOTE**: Uses Integrated Windows Authentication to connect to the SQL Server database.
-X *password*	Used to define a host password. The host password is required only when it is configured on the host in the DBServer control file. **CAUTION**: The password is currently not encrypted in communications between the Databridge Client and Databridge Server.
-Y reclone_all	Causes a process command to re-clone all data sets. The text reclone_all is required and prevents you from accidentally recloning everything, when you actually wanted to specify the -y option. When using the console, specify this option by selecting a check-box labeled "Reclone all active data sets." To find this option, click the data source in Administrative Console **Explorer** view to activate the **Data Source** menu, then click **Data Source** > **Advanced** > **Process with options**.
-Z	Forces a `process` or `clone` command to drop and create the tables of all re-cloned data sets, regardless of the use of the `deleted_record` or expanded `update_type` columns.

16.3 Appendix C: Client Configuration

16.3.1 Client Configuration Files

The Databridge Client 6.1 and later versions use binary configuration files. Binary configuration files are compatible with both service-initiated operations and command-line operations. However, if you use the service, you must use binary configuration files. (Command-line operations can use either binary or text configuration files. For information about creating text configuration files, see Export or Import a Configuration File.)

The Databridge Client software uses the following configuration files:

- The service configuration file (`dbcontrol.cfg`). This file contains settings for the service (Windows) or daemon (UNIX) that specify scheduling, passwords, logs, and more. For more information, see Appendix F: Service Configuration.
- Data source configuration files (`dbridge.cfg`) used by the Databridge Client programs (**DBClient**, **DBClntCfgServer** and **dbutility**). Each data source has its own configuration file, which can be updated using the Administrative Console's **Configure** command in the data source's menu. The Client configuration file overrides any equivalent parameter settings on the host.

In a new installation, the service creates the service configuration file in the **config** directory of the service's working directory (also referred to as the global working directory) the first time the service is started. The service also creates the **logs** and **scripts** sub-directories of the service's working directory at that time. When you add a data source in the Administrative Console, the service creates a binary configuration file "dbridge.cfg" for the data source. Data source configuration files are stored in the **config** subdirectory of the data source's working directory.

In an upgrade, as long as you are upgrading from version 6.1 SP3 or newer, you should be able to use the same working directory. If this is not possible, rename the old working directory and use the Migrate utility to recreate it using the old name (you must use a different working directory). You will also want to use this utility if you are upgrading from software older than 6.1 or you are switching from command line operations to service based operations. The Migrate utility takes your existing configuration files and creates new data source configuration files from them. It also creates a new service configuration file and adds your preexisting data sources to it.

16.3.2 How Do I Edit the Configuration File?

> ⚠ **Caution**
>
> You should **never** directly modify a binary configuration file. This will corrupt the file.

Each time you change your configuration settings in the Administrative Console, you update the binary configuration files. If you need to change a parameter that is not supported by the Administrative Console, you can export the binary configuration file to a readable text file. After you edit the text file, you can import the updated file to create a new binary configuration file. The `import` command performs all the necessary checks to ensure that your changes are valid. If you don't like the idea of using binary files for command-line operations, you can force the `export` command to replace the binary file with an equivalent text file.

Because passwords are encoded in the configuration file, there is no way to read them. If a password is wrong, export the configuration file and reenter the password as plain text. Then, import the file and export it again to remove the unencoded passwords from the text configuration file. Alternatively, you can use the Administrative Console or the **dbpwenc** utility to change passwords.

16.3.3 Export or Import a Configuration File

Use the `export` command to create an editable text file from your configuration file. If no configuration file exists, the `export` command creates a text file with the default configuration settings. After you make your changes, the `import` command will convert the text file to binary for use with the Client. Text configuration files can only be used with the command-line Client.

The `export` and `import` commands are typically used from a command line; the Administrative Console supports only the `export` command (the **Export Configuration** item in the data source's **Advanced** menu.). When you export the configuration file, the Databridge Client creates a text file that reflects the current values of the configuration parameters. Any passwords in the file are automatically encoded.

> ⚠️ **Caution**
>
> If you overwrite an existing text configuration file with this file, any comments you had in the previously existing file will be lost. To change or encode a password that was manually entered in a text configuration file, use the password encoding utility **dbpwenc** from a command line. See Change or Encode a Password.

To export the configuration file for a data source

- Open a command session and run the following command:

  dbutility [*options*]export [*filename*]

  where *[filename]* is an optional parameter to name an exported file something other than the default "dbridge.ini".

  The exported text file is written to the **config** subdirectory of the data source's working directory.

Option	Description
`-u`	Use this option if you export a text file named "dbridge.cfg". This allows the Client to overwrite the existing binary configuration file "dbridge.cfg" with a text configuration file of the same name. **For example:** `dbutility -u export dbridge.cfg`

To import the configuration file for a data source

Use this procedure to create a binary configuration file from a text Client configuration file.

- Open a command session and run the following command:

  dbutility [*options*] import [*filename*]

  where *[filename]* is an optional parameter to specify a filename other than the default, "dbridge.ini". When no option or filename is specified, the `import` command processes the text file `dbridge.ini` in the **config** directory and creates an equivalent binary configuration file, "dbridge.cfg", in the same directory. If the file "dbridge.ini" does not exist in this location, the `import` command creates a binary configuration file with the default values. If the text file contains errors, the Client returns an error to help you identify the problem and no binary file is created.

Option	Description
`-f` *filename*	Use this option to specify a filename or path other than the default. If this option is omitted, the Client tries to read the file "dbridge.cfg" in the config directory of the data source's working directory. To indicate a different location, type a backslash (Windows) or forward slash (UNIX) followed by the full path, including filename. For example, `/home/user/xyz/foo/myconfig.cfg`
`-u`	This option is required to allow the existing configuration file to be overwritten with a new file with the same name. Otherwise, the Client will try to read (import) the configuration from a file named "dbridge.ini". For example, the following command: `dbutility -u -f dbridge.cfg import` imports (reads) a file named "dbridge.cfg" and creates the binary configuration file "dbridge.cfg" regardless of whether the imported file is a text or binary file.

To export the service configuration file

- Open a command session and from the Client's global working directory, run the following command:

```
dbctrlconfigure export
```

This command reads the binary configuration file "dbcontrol.cfg" in the **config** sub-directory of the global working directory and creates an editable text configuration file "dbcontrol.ini" in the same location.

To import the service configuration file

- Open a command session and run the following command:

```
dbctrlconfigure import
```

This command reads the text configuration file "dbcontrol.ini" in the **config** sub-directory of the global working directory and creates a binary configuration file named "dbcontrol.cfg" in the same location.

16.3.4 Change or Encode a Password

Use this procedure for any of the following situations:

- To change the password for the user ID that you use to sign on to the database in your text or binary configuration file
- When the KEY (host password) on the host has changed and you need to update and encode the `hostpasswd` value in the Client configuration file. The KEY can only be changed by editing the DBServer control file (DATA/SERVER/CONTROL) on the host.

Passwords in the Client configuration file are automatically encoded when you use the export command to export the file (see Export or Import a Configuration File).

To change a password

1. When using a text Client configuration file, make sure that the password and hostpassword entries are uncommented.

```
[signon]
user         = user1
password     =
datasource   = BANKDB_S
hostpasswd   =
```

2. Open a command prompt session and set the directory to the working directory of the appropriate data source.
3. To change or encode the password, enter the following command:

```
dbpwenc -p
```

dbpwenc	Description
-h	Displays help for the password encoding utility (**dbpwenc**).
-p	Changes the relational database password and encodes it. You must supply both the new and the old passwords.
-q	Changes the Databridge Server password and encodes it. You must supply both the new and the old passwords.
-f filename	filename is the name you have given the Databridge Client configuration file. Use this option only when you have changed the configuration file name from its default ("dbridge.cfg"). If necessary, you can also include the directory path.

4. Make sure that the host password matches the password in the Databridge Server control file (DATA/SERVER/CONTROL) on the host.

The Databridge Client configuration file is updated with the encoded password, similar to the following example:

```
[signon]
user         = user1
password     = 9610ac320e9571e0d35020d15190610412131816
datasourse   = BANKDB_S
hostpasswd   = e617dc2316120d0a371003031c00230067c99551
```

16.3.5 Command-Line Options

The following command-line options have no equivalent configuration parameter:

Option	dbutility Command	Description
?		Short help
-d	All	Full Tracing
-f *filename*	All	Specifies
-h		Long help
-k	reload	Makes the command preserve the stateinfo of data sets that have a ds_mode of 2 and have not been reorganized.
-m	All	Includes a 5-digit millisecond timer in all output messages.
-r	redefine	Toggles the setting of the parameter use_dbconfig, which determines whether the command uses user scripts.
-t *mask*	All	Log file and tracing options
-u	configure, define, redefine, generate and dropall	Unconditionally performs the requested command, overriding any warnings that would be displayed without this option
-w	clone or process	Toggles the setting of the use_dbwait parameter.
-x	clone	Clones all active data sets except those specified at the command line
-y	process	Instructs the Client to re-clone all data sets whose ds_mode has a value of 11 or 12.
-z	clone or process	Instructs dbutility to not update the relational database during a clone or process command. This option is useful in determining how much non-database time is required to extract data for a data set.
-A	clone or process	Prevents the Databridge Client from dropping a table during a clone (the Databridge Client drops only the index).
-B	display	Causes the display command to quit after displaying the DATASETS Client control table records.
-D *database*	All	Specifies the name that identifies the Oracle instance or the Net8 service that is being accessed.
-F *afn*	process	Use this option make the Client act as if a QUIT AFTER *afn* command had been executed. It applies to process and clone commands only. The range of values allowed for *afn* are 1-9999.

Option	dbutility Command	Description
-K	`process`	Prevents the audit file removal WFL from being run on the mainframe after the Engine finishes processing an audit file.
-L	All	Forces the Client to start using a new log file.
-O ODBCdatasource	All	Specifies the ODBC data source to connect to (SQL Server Client only).
-P password	All	Sets the password associated with the user ID for the relational database. The password is limited to 30 characters.
-R	`redefine`	Forces all data sets to be redefined.
-T	All	Forces the Client to create a new trace file when tracing is enabled.
-U userid	All	Specifies the user ID for the relational database. The user ID must have the appropriate resource privileges for the designated relational database.
-W	All	Specifies that the configuration parameter `use_nt_authen` should be set to .
-Y	`process`	Causes all active data sets to be re-cloned.
-Z	`clone` or `process`	Forces the Client to drop and create the tables of all re-cloned data sets, regardless of the use of the `deleted_record` or expanded `update_type` columns.

16.3.6 Syntax

Follow these conventions in the configuration file:

- For hexadecimal values, use the 0x *nnnn* format.

- A semicolon (;), except within double quoted strings, indicates that the remainder of the current line is a comment.

- Section headers are enclosed in square brackets.

- Section headers and parameter names are not case-sensitive.

- Spaces and tabs between entries are ignored; however, spaces within double quoted values (for example, password values) are read.

- If you are not using a parameter, either comment the parameter out or delete the corresponding line in the configuration file. Do not leave an uncommented parameter without a value after the equal sign (=). Doing so results in syntax error.

You can specify some of these parameters only in the Client configuration file. Other parameters have equivalent command-line options and environment variables. For a complete list of configuration file parameters, their equivalent command-line options, and their related Client command, see Reference Tables.

16.3.7 Sample SQL Server Client Configuration File

You can view the configuration file for SQL Server by using the Export command. See Export or Import a Configuration File.

To use a parameter that is commented out, delete the semi-colon (;) and after the equals sign (=), enter a value that is appropriate for your site. Boolean parameters can be represented by True or False.

In the example below, some of the commented-out parameters have a value of -1. These parameters include the Databridge Engine control file parameters that can be overridden by the Client (commit frequency parameters and engine workers). This value indicates that the corresponding parameter in the Databridge Engine (or Server) control file will not be overridden by the Client. Do not uncomment these lines, unless you want to supply an actual value. Otherwise, the Client will issue an error.

```
;
; Databridge Client version 7.0 SQL Server configuration file -- generated programmatically
;

[Signon]
;user                   = USERID
;password               = PASSWORD
;datasource             = DATASOURCE
use_nt_authen           = false
;hostpasswd             = HOSTPASSWD

[Log_File]
file_name_prefix        = "db"
;max_file_size          = 0
logsw_on_size           = false
logsw_on_newday         = false
newfile_on_newday       = true
single_line_log_msgs    = false

[Trace_File]
file_name_prefix        = "trace"
;max_file_size          = 0

[Bulk_Loader]
bcp_batch_size          = 100000
bcp_code_page           = "ACP"
;bcp_copied_msg         = "rows copied"
bcp_packet_size         = 0
;bulk_loader_path       = "str"
max_bcp_failures        = 5
max_errors              = 10
max_temp_storage        = 400M
use_bcp                 = true
verify_bulk_load        = 1

[Params]
;
;  (1) define/redefine command parameters
;
allow_nulls             = true
auto_mask_columns       = true
automate_virtuals       = false
bracket_tabnames        = false
clr_dup_extr_recs       = true
convert_ctrl_char       = false
default_user_columns    = 0x00000000
dflt_history_columns    = 0x00000000
enable_dms_links        = false
enable_dynamic_hist     = false
;external_column[n]      = ["name"][,[sql_type][,[sql_length][,"default"]]]
extract_embedded        = false
flatten_all_occurs      = false
force_aa_value_only     = 0
history_tables          = 0
inhibit_required_opt    = false
;maximum_columns        = 0
min_varchar             = 4
minimize_col_updates    = false
miser_database          = false
optimize_updates        = false
read_null_records       = true
reorg_batch_size        = 50000
sec_tab_column_mask     = 0x00000000
split_varfmt_dataset    = false
strip_ds_prefixes       = false
suppress_dup_warnings   = false
suppress_new_columns    = false
suppress_new_datasets   = true
use_bigint              = false
use_binary_aa           = false
use_clustered_index     = false
use_column_prefixes     = false
use_date                = false
use_datetime2           = false
use_dbconfig            = true
use_decimal_aa          = false
use_internal_clone      = false
use_nullable_dates      = false
```

```
use_primary_key          = false
use_stored_procs         = false
use_time                 = false
use_varchar              = true
;
; (2) process/clone command parameters
;
alpha_error_cutoff       = 10
aux_stmts                = 100
;batch_job_period        = 00:00, 00:00
century_break            = 50
;commit_absn_inc         = -1
;commit_idle_database    = -1
;commit_longtrans        = -1
;commit_time_inc         = -1
;commit_txn_inc          = -1
;commit_update_inc       = -1
controlled_execution     = false
;convert_reversals        = -1
correct_bad_days         = 0
dbe_dflt_origin          = direct
defer_fixup_phase        = false
discard_data_errors      = false
display_bad_data         = false
enable_af_stats          = false
enable_doc_records       = false
enable_minimized_col     = false
enable_optimized_sql     = true
;engine_workers          = -1
error_display_limits     = 10,100
inhibit_8_bit_data       = false
inhibit_console          = false
inhibit_ctrl_chars       = false
inhibit_drop_history     = false
keep_undigits            = 0
linc_century_base        = 1957
;masking_parameter[n]    = "str"
max_clone_count          = 10000
max_discards             = 0,100
max_retry_secs           = 20
max_srv_idle_time        = 0
max_wait_secs            = 3600,60
min_check_time           = 600
n_dmsii_buffers          = 0
n_update_threads         = 8
null_datetime_value      = 19010101
null_datetime2_value     = 19010101
null_digit_value         = 9
numeric_date_format      = 23
preserve_deletes         = false
set_blanks_to_null       = false
set_lincday0_to_null     = false
show_perf_stats          = true
show_statistics          = true
show_table_stats         = true
sql_exec_timeout         = 180,0
sql_heart_beat           = 0
statistics_increment     = 100000,10000
stop_after_fixups        = false
stop_after_gc_reorg      = false
stop_after_given_afn     = false
stop_on_dbe_mode_chg     = false
suppress_delete_msgs     = false
track_vfds_nolinks       = true
use_ctrl_tab_sp          = true
use_dbwait               = false
use_latest_si            = false
;
; (3) Server options
;
;shutdown {until | for} hh:mm after stop
;stop {before | after} task "name"
;stop {before | after} time hh:mm[:ss]
;
; (4) generate command parameters
;
;global_table_suffix    = "str"
;create_table_suffix[n] = "str"
;global_index_suffix    = "str"
;create_index_suffix[n] = "str"
;user_column_suffix[n]  = "str"
;
; (5) miscellaneous command parameters
;
display_active_only      = true
;
; (6) user scripts
;
user_script_bu_dir       = ""
user_script_dir          = "scripts"
;
; (7) external data translation parameters
;
use_ext_translation      = false
eatran_dll_name          = "DBEATRAN.DLL"


[Scheduling]
```

```
;
; dbutility process command only
;
;daily                = 08:00, 12:00, 17:00, 24:00
;exit_on_error        = false
;sched_delay_secs     = 600
;sched_minwait_secs   = 3600
;sched_retry_secs     = 3600
;blackout_period      = 00:00, 02:00

[EbcdicToAscii]
; e1 = a1
; e2 = a2
;  ...
; en = an
;

[DBConfig]
default_date_fmt      = 21
global_type0_changes  = true

[Encryption]
ca_file               = ""
ca_path               = ""
certify_server_name   = false
enable_encryption     = false
tls_host_name         = ""
```

16.3.8 Sample Oracle Client Configuration File

```
;
; Databridge Client version 7.0 Oracle configuration file -- generated programmatically
;

[Signon]
;user                 = USERID
;password             = PASSWORD
;database             = DATABASE
;hostpasswd           = HOSTPASSWD

[Log_File]
file_name_prefix      = "db"
;max_file_size        = 0
logsw_on_size         = false
logsw_on_newday       = false
newfile_on_newday     = true
single_line_log_msgs  = false

[Trace_File]
file_name_prefix      = "trace"
;max_file_size        = 0

[Bulk_Loader]
;bcp_code_page        = <code_page>
;bcp_decimal_char     = -1
;bulk_loader_path     = "str"
enable_parallel_mode  = false
inhibit_direct_mode   = false
max_bcp_failures      = 5
max_errors            = 10
max_temp_storage      = 400M // Windows only
sqlld_bindsize        = 65536
sqlld_rows            = 10000
verify_bulk_load      = 1

[Params]
;
;  (1) define/redefine command parameters
;
allow_nulls           = true
automate_virtuals     = false
clr_dup_extr_recs     = true
convert_ctrl_char     = false
default_user_columns  = 0x00000000
dflt_history_columns  = 0x00000000
enable_dms_links      = false
enable_dynamic_hist   = false
enable_extended_types = false
;external_column[n]   = ["name"][,[sql_type][,[sql_length][,"default"]]]
extract_embedded      = false
flatten_all_occurs    = false
force_aa_value_only    = 0
history_tables        = 0
inhibit_required_opt  = false
;maximum_columns      = 0
min_varchar           = 4
minimize_col_updates  = false
miser_database        = false
optimize_updates      = false
read_null_records     = true
reorg_batch_size      = 50000
sec_tab_column_mask   = 0x00000000
```

```
    split_varfmt_dataset   = false
    strip_ds_prefixes      = false
    suppress_dup_warnings   = false
    suppress_new_columns    = false
    suppress_new_datasets   = true
    use_binary_aa          = false
    use_dbconfig           = true
    use_clob               = false
    use_column_prefixes    = false
    use_decimal_aa         = false
    use_internal_clone     = false
    use_nullable_dates     = false
    use_primary_key        = false
    use_stored_procs       = false
    use_varchar            = true
    ;
    ; (2) process/clone command parameters
    ;
    alpha_error_cutoff     = 10
    aux_stmts              = 100
    ;batch_job_period       = 00:00, 00:00
    century_break          = 50
    ;commit_absn_inc        = -1
    ;commit_idle_database   = -1
    ;commit_longtrans       = -1
    ;commit_time_inc        = -1
    ;commit_txn_inc         = -1
    ;commit_update_inc      = -1
    controlled_execution   = false
    ;convert_reversals      = false
    correct_bad_days       = 0
    dbe_dflt_origin        = direct
    defer_fixup_phase      = false
    discard_data_errors    = false
    display_bad_data       = false
    enable_af_stats        = false
    enable_doc_records     = false
    enable_minimized_col   = false
    enable_optimized_sql   = true
    ;engine_workers         = -1
    error_display_limits   = 10,100
    inhibit_8_bit_data     = false
    inhibit_console        = false
    inhibit_ctrl_chars     = false
    inhibit_drop_history   = false
    keep_undigits          = 0
    linc_century_base      = 1957
    max_clone_count        = 10000
    max_discards           = 0,100
    max_retry_secs         = 20
    max_srv_idle_time      = 0
    max_wait_secs          = 3600,60
    min_check_time         = 600
    n_dmsii_buffers        = 0
    n_update_threads       = 8
    null_digit_value       = 9
    numeric_date_format    = 23
    preserve_deletes       = false
    ;rollback_segment_name  = ""
    set_blanks_to_null     = false
    set_lincday0_to_null   = false
    show_perf_stats        = true
    show_statistics        = true
    show_table_stats       = true
    sql_exec_timeout       = 180,0
    sql_heart_beat         = 0
    statistics_increment   = 100000,10000
    stop_after_fixups      = false
    stop_after_gc_reorg    = false
    stop_after_given_afn   = false
    stop_on_dbe_mode_chg   = false
    suppress_delete_msgs   = false
    track_vfds_nolinks     = true
    use_ctrl_tab_sp        = true
    use_dbwait             = false
    use_latest_si          = false
    ;
    ; (3) Server options
    ;
    ;shutdown {until | for} hh:mm after stop
    ;stop {before | after} task "name"
    ;stop {before | after} time hh:mm[:ss]
    ;
    ; (4) generate command parameters
    ;
    ;global_table_suffix    = "str"
    ;create_table_suffix[n] = "str"
    ;global_index_suffix    = "str"
    ;create_index_suffix[n] = "str"
    ;user_column_suffix[n]  = "str"
    ;
    ; (5) miscellaneous command parameters
    ;
    display_active_only    = true
    ;
    ; (6) user scripts
    ;
```

```
user_script_bu_dir      = ""
user_script_dir         = "scripts"
;
; (7) external data translation parameters
;
use_ext_translation     = false
eatran_dll_name         = "DBEATRAN.DLL"


[Scheduling]
;
; dbutility process command only
;
;daily                  = 08:00, 12:00, 17:00, 24:00
;exit_on_error          = false
;sched_delay_secs       = 600
;sched_minwait_secs     = 3600
;sched_retry_secs       = 3600
;blackout_period        = 00:00, 02:00

[EbcdicToAscii]
; e1 = a1
; e2 = a2
;  ...
; en = an
;

[DBConfig]
default_date_fmt        = 21


[Encryption]
ca_file                 = ""
ca_path                 = ""
certify_server_name     = false
enable_encryption       = false
tls_host_name           = ""
```

16.3.9 Processing Order

Configuration file options override environment variables. Command-line options override both environment variables and configuration file options.

The parameter processing order is as follows:

- The operating system login name (user ID) is used as the lowest level default for the database user ID.

- Environment variables (DBUSERID, DBPASSWD, DBDATABASE, and DBHOSTPW).

- Command-line options `-d` (for full tracing), `-v` (for verbose messages), `-t` (for creating a Databridge Client trace file) and `-T` (for forcing the Client to start a new trace file), and `-f` (for specifying a configuration file other than the default `dbdridge.cfg`). These options are processed in the order in which they appear on the command line.

- Parameters specified in the configuration file. You can specify the configuration file via the `-f` option. If you do not specify a configuration file name via the `-f` option, **dbutility** tries to open the default configuration file (`dbridge.cfg` in the **config** subdirectory of the data source's working directory); if the file does not exist, the Databridge Client uses the default values for all the configuration file parameter. The absence of a configuration file is not treated as an error only when running the command-line Client. If you use the service or daemon, the absence of a configuration file named "dbridge.cfg" is treated as an error.

- All remaining command-line options. In the final pass, a command-line option with a configuration file equivalent overrides the configuration file entry.

16.3.10 Parameter Descriptions

Accessing configuration parameters in the Administrative Console

The Administrative console allows you to directly update all the parameters described in this section, except for those in the signon section.

The parameters in the signon section are specified when adding a new data source. When using the Administrative Console to update configuration parameters, you need to take the following actions:

1. Navigate to the Client Managers page, `Datadridge Servers > Client Managers`.

2. Click on the desired Client Manager. This displays the data sources page.

3. From the **Settings** menu click on **Configure**, this opens the Client Parameters dialogs.

4. We provide the navigation steps to get to the parameter in question in the line that starts with " `Console:` " for most parameters. The first item is the page name in the left pane. Some of these items have an associated "+" icon, which when clicked on display the additional items. Clicking on any of these items opens the corresponding dialog page. The directions for items within the page are enclosed in parentheses and typically include a group name followed by the start of the item name within the group. For example: **PROCESSING > DMSII Data Error Handling** (Character data error > Change to space). In this case you need to click on either **PROCESSING** or the "+" after it and then click on **DMSII Data Error Handling** and go to the "Character data error" group and look for "Change to space".

5. Change the parameters you need to change and Click **Save** when done, unless you want to make changes in other property pages. You can navigate to other property pages and make changes until you are ready to save your changes by pushing the **Save** button at which point all your changes will be applied.

6. When you push the **Save** button, if there are errors in your changes you will get a pop-up telling what you did wrong. You can then go back in the Client Configuration parameter dialog pages and correct the errors and push **Save** again when done.

[signon]

Use the [signon] section of the "dbridge.cfg" file to enter information for signing on to the relational database and Databridge Server on the host.

The configuration file must include the data source (or database, if using Oracle), signon parameters to access the relational database, and a user and a password (unless you use the SQL Server Client with Integrated Windows authentication).

When using the Administrative Console, you need to supply these parameters at the time you create the data source. To do so, right-click on the service in the tree view and click Add Data Source from the pop-up menu to open the dialog and enter these parameters.

Parameter	Description
database	**Default:** None **Command-line option:** `-D` (Oracle) This parameter is the name that identifies the Oracle instance or the Oracle Net Services node that is being accessed. If the name contains non-alphanumeric characters, you must enclose it in double quotation marks, as follows: `database = "orcl.cin.microfocus.com"`
datasource	**Default:** None **Command-line option:** `-O` (Microsoft SQL Server) This parameter is the name that identifies the ODBC data source used to access the SQL database. This ODBC data source is configured using the Control Panel during the Client installation.

hostpasswd	**Default:** None
	Range: 17 alphanumeric characters
	Command-line option: `-X`
	
	Use the host password parameter to specify the password associated with Databridge Server on the host. This parameter must match exactly the KEY parameter defined in the Host Server control file. For example:
	DBServer
	`KEY = "Secret"`
	dbridge.cfg
	`hostpasswd = Secret`

password	**Default:** None **Command-line option:** `-P` Use the password parameter to specify the password associated with the user ID for the relational database. The password must be valid for the user ID or the connection to the relational database server will fail. Passwords are limited to 30 characters. If your password contains non alphanumeric characters other than the underscore, you must enclose it in double quotes, as follows: `password = "a$bb%"` **NOTE:** Passwords starting with Oracle 11g release 2 are case-sensitive. The password is always encoded in both text and binary versions of the Client configuration file. For more information, see Export or Import a Configuration File or in the Databridge Administrative Console Help, see "Export the Client Configuration to a File." Passwords that are communicated between the Databridge Client and Databridge Server are not encoded.
user	**Default:** None **Command-line option:** `-U` Use the user parameter to specify the user ID for the relational database. The user ID must have the appropriate resource privileges for the designated relational database, as explained in Setting Up a User ID (Windows) in the Databridge Installation Guide.

Parameter	Description
use_nt_authen	**Default:** False **Range:** True or False **Command-line option:** `-W` The `use_nt_authen` parameter applies to Microsoft SQL Server Clients only. Use Windows ODBC Data Source Administrator to set the required ODBC data source authentication method. The SQL Server database must be installed with the proper authentication mode selected; either SQL Server, Integrated Windows, or Mixed Mode (that is, using both methods). When using Integrated Windows authentication, Windows Administrators are automatically included in the SQL Server user list. The SYSTEM account is only included in versions of SQL Server older than 2012. For more information, see the *Databridge Installation Guide*. Use this parameter as follows: • Set it to True when Microsoft SQL Server is set to use Integrated Windows Authentication for access to the SQL Server database. • Set it to False when Microsoft SQL Server is set to use its own SQL Server authentication. The SQL Server verifies the authenticity of the login ID with SQL Server authentication using a Login ID and password entered by the user.

[signon] parameters with equivalent environment variables

[signon] Parameter	Environment Variable	Option	dbutility Command
database	DBDATABASE	`-D`	All (only applies to Oracle)
datasource		`-O`	All (does not apply to SQL Server)
hostpasswd	DBHOSTPW	`-X`	`define`, `redefine`, `process`, `clone`, and `switchaudit`
password	password	`-P`	All
user	DBUSERID	`-U`	All

[Log_File]

Use the [Log_File] section to control the various options for the log file that is created in the **logs** subdirectory of the working directory for a data source.

When using the service, two sets of Client log files are generated. The **DBClient** program and the command-line Client **dbutility** use log files, whose default names are of the form `dbyyyymmdd.log`. The **DBClntCfgServer** program uses log files, whose names are of the form `db_cfgyyyymmdd.log`. The prefix "db" can be changed by specifying a `file_name_prefix` in the log section of the Client configuration file.

When using the Administrative Console, click on the **LOGGING** item to get the "Client Log parameters" page of the dialog, enter prefix in the edit box titled "File name pefix".

Parameter	Description
file_name_prefix	**Default:** "db" **Range:** 1 to 20 characters **Recommended value:** data source name Use this parameter to change the prefix of the log files for this data source. We recommend using the name of the data source as the prefix as this ensures that log files created on the same date but for different data sources have unique names. The log files have names in the form *dbyyyymmdd*.log or when necessary, *dbyyyymmdd_hhmiss*.log (This command allows you to replace the prefix "db" by any character string, provided that it results in a legal file name.)
logsw_on_newday	**Default:** False **Range:** True or False This parameter determines whether or not the Client uses a new log file, when the date changes. You may want to set this parameter to False, if your log files are small and use the `logsw_on_size` parameter to manage the log files.
logsw_on_size	**Default:** False **Range:** True or False **Recommended value:** True (when running real/time) Use this parameter to control whether or not the Client should check the log file size to see if it has reached the size defined by the `max_file_size` parameter. If the size of the log file exceeds this parameter the log file is closed and a new one is opened. If the current date is different than the creation date of the old file, which is part of its name, the new log file will be of the form *dbyyyymmdd*.log otherwise the time component will be added to the file name to ensure that the name is unique.
max_file_size	**Default:** 0 **Range:** numeric value, optionally followed by K, M, or G **Recommended value:** 1M Use this parameter to limit the size of log files. The default value of 0 indicates that no limit is imposed on the size of log file. The suffixes of K, M and G allow you to specify the maximum file size in kilobytes, megabytes, or gigabytes. A value on the order of 1 MB is a reasonable value to use. The file size is always checked when you start the Client, regardless of the setting of the `logsw_on_size` parameter. When the `logsw_on_size` parameter is set, the log file size is also checked when the Client starts processing a new audit file.
newfile_on_newday	**Default:** True **Range:** True or False This parameter forces the Client to create a new log file when it starts if the existing log file was created on an earlier date. You may want to set this parameter to False, if your log files are small and use the `logsw_on_size` parameter to manage the log files.

Parameter	Description
single_line_log_msgs	**Default:** False **Range:** True or False The `single_line_log_msgs` parameter tells the Client to make all of its log file output messages single-line messages. When this parameter is set to True, the end-of-line character of all multi-line outputs are replaced by a space. This parameter exists to assist some log file analysis programs that fail to parse multi-line output messages.

[Trace_File]

Use the [Trace_File] section to control the various options for the trace file, created in the trace subdirectory of the working directory for a data source.

When using the Administrative Console, click on **LOGGING > Trace Log** to get the Trace parameters page of the dialog.

Parameter	Description
file_name_prefix	Specifies a string (up to 20 characters in length) to change the default prefix "trace".
max_file_size	Specifies the size limit of trace files. You can enter a number with a suffix of K, M and G to indicate the unit of measure (kilobytes, megabytes, or gigabytes).

[Bulk_Loader]

The bulk loader parameters apply to the bulk loader utility for your relational database -- **SQL*Loader** for Oracle and **bcp** SQL Server.

When using the Administrative Console, click on the `BULK LOADER` to get to the "Bulk loader parameters" page of the dialog.

Parameter	Description
bcp_batch_size	**Default:** 100,000 rows per batch **Range:** 0 or 1000-10000000 rows per batch **bcp and BCP API:** SQL Server Specifies the batch size used during the bulk loader operations. This is parameter supplied to the **bcp** utility, to make it load the table in several batches of the given size. When using the BCP API after `bcp_batch_size` rows are loaded, the Client calls bcp_batch to commit these rows. Permitted values are 0 or 1000-10000000 (rows per batch). A value of zero causes the bcp utility to load the entire group of records in the data file in one batch. Copying all of the rows of a very large table in one batch may require a high number of locks on the Microsoft SQL Server database. When you specify a nonzero value, the Databridge Client adds the `-b` batch_size option to the bcp command line. A value of 0 omits the -b option.
bcp_code_page	**Default:** "" **Range:** "String" **Bulk Loader utility:** SQL Server and Oracle Adds the line "CHARACTERSET `<code_page>`" to the **SQL*Loader** control file. Consult the Oracle documentation for the exact names of the code pages as Oracle uses their own notation. The typical code page for 8-bit character data is "WE8ISO8859P1". You need to specify a `bcp_code_page` when dealing with a UTF8 database.
bcp_copied_msg	**Default:** NULL (omitted) **Range:** Any "quoted string" **Bulk Loader utility:** SQL Server Enables the **bcp_auditor** utility to determine whether or not a bulk loader was successful in cases where the database language is not English. For example, in German, this parameter is "Zeilen kopiert", but in English, it is "rows copied". If this parameter is not set correctly, the **bcp_auditor** reports bulk loader failures even though the bulk loader worked correctly. The bcp_auditor program also accepts the `bcp_copied_message` in binary format expressed as the text "HEX_" followed by a string of hexadecimal values representing the values of each byte. This allows you to circumvent code page related problems, which can sometimes corrupt the `bcp_copied_message` when it is passed to the **bcp_auditor** program as a command-line argument. For example, the string "HEX_6C69676E657320636F7069E965732E" can represent the French message "lignes copiées." (The character "é" does not cause any problems when expressed as "E9".)

Parameter	Description
bcp_decimal_char	**Default:** -1 (This parameter is commented out.) **Range:** a period (.) or a comma (,) **SQL * Loader: Oracle Clients only** This parameter is normally auto-detected by the Client that gets its value by reading the of Oracle database's NLS_NUMERIC_CHARACTERS parameter. This method will work correctly when the Client and the database reside in the same machine. However, if the Client is run outside the database machine, there is no guarantee that the Oracle Client software, that the Databridge Client uses, will have the same NLS settings as the target database. For example it is possible to have a US Oracle Client software in the Client machine that connects to a Brazilian database. In this rather unusual situation you would have to set the `bcp_decimal_character` to '.' as it will default to ',' which will lead to **SQL*Loader** errors in for all records that have numeric data with a decimal point.
bcp_delim	**Default: Tab (SQL Server)** **bcp** utility: SQL Server This parameter works as follows with the various Clients: **Oracle**: The string "delim" is not configurable, the Client always uses the vertical bar as the delimiter. The Client sets the bcp `decimal_character` by reading the database's NLS parameters. **SQL Server:** The string `bcp_delim` can be longer than one character. This is useful if the data contains alpha fields with TAB characters that need to be preserved. (A possible delimiter value in this case would be "|" or "||"; see `inhibit_ctrl_chars`)
bcp_packet_size	**Default:** 0 (which omits the -a option) **Range:** 0 or 512-65535 (decimal or hexadecimal) **bcp** utility: SQL Server (remote servers only) Defines the network packet size value for the **bcp** utility. Use this parameter when you have wide tables. For wide tables, setting this parameter to a packet size larger than the **bcp** default (4096) can speed up loading the data into the table. When you specify a nonzero value, the Databridge Client adds the "`` `-a pkt_size ``" option to the **bcp** command line in the .CMD scripts. If you omit this parameter, or if you specify a value of 0, the Databridge Client omits the "`-a pkt_size`" option and the **bcp** utility uses the default network packet size of 4096.

Parameter	Description
enable_parallel_mode	**Default:** False **Range:** True or False **SQL*Loader:** Oracle **Related parameters:** `inhibit_direct_mode` This parameter, which is only meaningful when DIRECT mode is enabled, causes the generate command to add the specification "parallel = true" to the **SQL*Loader** command line. Parallel mode makes the **SQL*Loader** run faster at the expense of additional system resources.
inhibit_direct_mode	**Default:** False **Range:** True or False **SQL*Loader:** Oracle **Related parameters:** `enable_parallel_mode`, `sqlld_rows`, and `sqlld_bindsize` Controls whether the `generate` command adds the specification "direct=true" to the **SQL*Loader** command line. If your Oracle database is on the same machine as the Databridge Client, you would let this parameter assume its default value of False, as DIRECT mode is much faster than conventional mode. Conversely, if your Databridge Client accesses a remote Oracle database using `SQL*Net` between two dissimilar architectures (for example, Windows and UNIX), you must use conventional mode by setting this parameter to True. Setting `inhibit_direct_mode` to True inhibits the use of the direct=true option when invoking **SQL*Loader** in the command files. It is provided for your convenience so that you do not to have to remove the string "direct=true" from every call on **SQL*Loader**. When you enable `inhibit_direct_mode`, we recommend that you increase the size of `sqlld_bindsize` for better performance.
max_errors	**Default:** 10 **Range:** 0-1000 **Bulk loader utility and BCP API:** All Controls the bulk loader's tolerance to records that are discarded due to data errors. Use this parameter when you have many bulk loader errors. Increasing the maximum error count allows you to gather all the errors in one run rather than finding 10 errors and then having to start over again. For example, if you are having problems cloning a table, you may want to increase the count to 1000 or more to get all the errors in one cloning or process session. Knowing the type of errors helps you to solve the problems. The default value for this parameter is 10, which means that the bulk loader aborts after encountering 10 bad records. These bad records are written to the discard file and information about the error is written to the bulk loader log file. For information about these files, see Files Related to **SQL*Loader** and Files Related to BCP.

Parameter	Description
max_bcp_errors	**Default:** 5 **Range:** 0-1000 This parameters is designed to prevent the Client from blindly extracting all the selected data sets when the number of tables with load failures reaches the specified threshold. A value 0 indicates that the Client should proceed regardless of the number of failed loads.
max_temp_storage	**Default:** 400 MB **Range:** 10 MB-3 GB (or 0) **Bulk loader utility:** bcp for SQL Server or **SQL*Loader** for Oracle **Applies to:** Windows Clients This parameter activates the segmented bulk load feature, which allows you to specify the maximum amount of storage that the Client should use for temporary files. Because the Client cannot stop in the middle of a record, you can expect it to use slightly more storage than the value you specify. Therefore, select a value less than the total amount of free space available on the disk. We recommend that you keep this value low as there is no real advantage to attempting to load large tables all at once. If you set the value too high, thye Client can run out of storage while it is writing temporary files. You can specify the max_temp_storage value as an integer with any of the following suffixes: K (or KB) for kilobytes (default) M (or MB) for megabytes G (or GB) for gigabytes The space between the number and the suffix is optional. **NOTE:** The valid range for this parameter is 10 MB to 3 GB (0xC0000000). You must specify values greater than 0x7FFFFFFF without a suffix. The value you enter for max_temp_storage must be a whole number.
sqlld_bindsize	**Default:** 64K bytes **Range:** 0x10000-0x400000 (decimal or hexadecimal) **SQL*Loader:** Oracle **Related parameters:** inhibit_direct_mode , sqlld_rows Defines the value to be used for the BINDSIZE parameter for **SQL*Loader** operations. Increasing this value can speed up **SQL*Loader** operations when using conventional mode (for example, running remote to a database on a UNIX system. Use sqlld_rows and sqlld_bindsize when you are running the Client for a remote Oracle database running on UNIX or Windows. A larger bind size and row size can increase the speed of the load across Oracle Network Services at the expense of using more memory.

Parameter	Description
sqlld_rows	**Default:** 100 **Range:** 10-100,000 rows **SQL*Loader:** Oracle **Related parameters:** `inhibit_direct_mode` , `sqlld_bindsize` Defines the value to be used for the ROWS specification for **SQL*Loader** operations. Use `sqlld_rows` and `sqlld_bindsize` when you are running the Client for a remote Oracle database running on UNIX or Windows. A larger bindsize and row size can increase the speed of the load across Oracle Network Services at the expense of using more memory.
use_bcp	**Default:** true **Range:** True or False **Bulk Loader utility:** SQL Server The SQL server Client can operate with the **bcp** utility or the BCP API. This parameter determines the default value used by the `define` and `redefine` commands when setting the `ds_options` for the various data sets. We recommend using the **bcp** utility as it is more reliable than the BCP API. If you have a lot of record types with some variable format data sets, you should definitely set them to use the **bcp** utility, as you may end up with a lot database connections during their data extraction, as each table being loaded requires each own database connection when using the BCP API.
verify_bulk_load	**Default:** 1 **Range:** 0, 1, or 2 **Bulk loader utility:** All Determines how you want the Databridge Client to handle the results of the bulk loader operations during data extraction, as follows: **Setting and Description** • **0:** The Databridge Client does not verify the results of bulk loader operations. • **1:** The Databridge Client retrieves the number of rows in the table and compares it to the number of rows handled by the bulk loader. If the two counts differ, the Databridge Client displays a warning message. • **2:** This setting is the same as the preceding setting 1, except that the Databridge Client terminates so that you can investigate the reason for the mismatch.

[params]

The [params] section of the configuration consists of the following groups of command parameters:

For this group	See this topic
`define` and `redefine`	Define and Redefine Command Parameters
`process` and `clone`	Process and Clone Command Parameters
Server options	Server Option Parameters
`generate`	Generate Command Parameters
`display`	Display Command Parameter
User scripts	User Scripts Parameters
External data translation DLL support	[Bulk_Loader]

Define and Redefine Command Parameters

The following parameters are included in the [params] section of Databridge Client configuration file. The parameters listed in this section affect only the `define` and `redefine` commands.

ALLOW_NULLS

Default: False
Range: True or False
Console: CUSTOMIZING (General > Allow NULLs)

The `allow_nulls` parameter specifies whether or not the `define` and `redefine` commands should set the DAOPT_Nulls_Allowed bit (value 1) in the `da_options` column of the DATAITEMS Client control table. This means that both DMSII null data and data items that contain bad values (excluding keys) will be stored as relational database NULLs.

You can set this parameter in the Client Configuration dialog box of the Administrative Console or by using data table customization scripts. To avoid storing NULL data as values that are possibly legitimate (0 or 999), keep this parameter set to True.

AUTO_MASK_COLUMNS

Default: True
Range: True or False
Console: CUSTOMIZING (DMSII related parameters > Preserve DMSII MASKING option)

The parameter `auto_mask_columns` specifies whether the Databridge Client should automatically mask columns whose corresponding items in DMSII have DATAMASK specifications in the DASDL.

AUTOMATE_VIRTUALS

Default: False
Range: True or False
Console: PROCESSING > Advanced (General > Automated virtual data sets)

This parameter enables code that automatically handles virtual data sets that must be linked with their parent data sets using the `virtual_ds_num`, `real_ds_num`, and `real_ds_rectype` columns in the DATASETS Client control table. These links are currently set up via user scripts. When this option is enabled, you simply issue a `process` command. When issuing a `clone` command, the virtual data sets do not have to be explicitly specified on the command line.

BRACKET_TABNAMES

Default: False
Range: True or False
Applies to: SQL Server Client only
Console: CUSTOMIZING (General > Use brackets ...)

The parameter `bracket_tabnames` specifies whether the Databridge Client should allow data set names that are TRANSACT_SQL reserved words to be used as table names or they should be renamed. If this parameter is set to True all such table names are enclosed in square brackets in all SQL statements used by the Client. If the parameter is set to False the Client renames them by adding "_x" to the data set name.

CLR_DUP_EXTR_RECS

Default: True
Range: True or False
Console: CUSTOMIZING > Advanced (Global data set options > Clear duplicate ...)

This parameter defines the initial value of the DATASETS table `ds_options` bit DSOPT_Clrdup_Recs. When this bit is set, the Databridge Client runs a script to remove false duplicate records after the index creation fails. These duplicate records are caused by long cloning of an active DMSII database where the extraction process can see the same record more than once if it moves. In addition to indirectly affecting the `process` and `clone` commands, this parameter indirectly affects the `generate` command.

The `ds_options` bit DSOPT_Clrdup_Recs causes the following actions:

- When set to False, the Databridge Client ignores false duplicate records. If there are false duplicate records, the index creation will fail. In this case you must manually remove the false duplicate records and recreate the index before the fixup phase can continue.

- When set to True, the `generate` command creates a script (named "script.clrduprecs.*tabname*) that removes records with duplicate key values. This script will run only if the create index step fails. After the duplicate records are deleted, the index creation and fixup phases continue as normal.

CONVERT_CTRL_CHAR

Default: False
Range: True or False
Related parameters: `alpha_error_cutoff` , `discard_data_errors` , `display_bad_data`
Console: PROCESSING > DMSII Data Error Handling (Character data error > Control character)

The `convert_ctrl_char` parameter applies to DMSII data items of type ALPHA.

> ✎ **Note**
>
> - Do not set the `convert_ctrl_char` parameter to True unless you are absolutely certain that eliminating control characters will have no adverse effects on the data. For example, eliminating control characters can cause some fields to be misinterpreted.
> - This parameter and the parameter `inhibit_ctrl_chars` are mutually exclusive. If you attempt to set them both to True, the configuration file scanner will generate an error.

Use this parameter as follows:

- Set `convert_ctrl_char` to True if you want the Databridge Client to replace all control characters in ALPHA data with spaces. This is usually the result of host applications whose record layouts are out-of-sync with the DMS database.

- Set `convert_ctrl_char` to False if you want the Databridge Client to not change control characters to spaces. Depending on your setting for `alpha_error_cutoff`, the column that contains control characters may be set to NULL, but at least the problem field will be identified. Then, you can decide whether to set this parameter to True and ignore the bad data.

In summary, before you set this option to True, set `alpha_error_cutoff` to a low value and set `display_bad_data` to True to determine whether or not it is safe to ignore the control characters.

When using the Administrative Console, these two parameters are presented as 3 radio buttons ("Change to space", "Change to ?""" and "Translate if possible"). This prevents you from setting both mutually exclusive parameters to True.

DEFAULT_USER_COLUMNS

Default: 0
Range: 0 - 16383 (Some bit combinations are not allowed.)
Console: CUSTOMIZING > User Columns Section Two

The `default_user_columns` parameter adds non-DMSII columns (user columns) to all the tables mapped from data sets in the Client database. You can add additional user column to history tables using the `dflt_history_columns` parameter, see dflt_history_columns. You can also exclude certain user columns from secondary tables using the `sec_tab_column_mask` parameter, see sec_tab_column_mask.

When using the Administrative Console use the checkboxes in the `Primary Tables` column to set corresponding the bits for the `dflt_user_columns` parameter.

DFLT_HISTORY_COLUMNS

Default: 0
Range: 0 - 16383 (Some bit combinations are not allowed.)
Console: CUSTOMIZING > User Columns Section Two

The `dflt_history_columns` parameter adds more non-DMSII columns (user columns) to all history tables in the Client database. By default, history tables are created with three non-DMSII columns. The `dflt_history_columns` parameter is intended to simplify user scripts at sites where the same non-DMSII columns (user columns) are added to all (or most) history tables. When you use this parameter to add user columns to history tables, the specified non-DMSII columns are added to all history tables. If you do not want to add all of these columns to all history tables, you must use a user script to set the `active` column to 0 for the unwanted columns in the DATAITEMS Client control table.

For more information about history tables, see `history_tables` and `ds_options` in the DATASETS Client control table.

When using the Administrative Console use the checkboxes in the `History Tables` column to enable the corresponding bits for the `dflt_history_columns` parameter.

ENABLE_DMS_LINKS

Default: False
Range: True or False
Console: CUSTOMIZING (DMSII related parameters > Enable DMSII links)

This parameter must be set to True when running a `define`, `redefine` or the Administrative Console's **Customize** commands if you want to replicate the links in the DMSII database. In addition to setting this parameter to True, you must also enable DMSII link support in the Databridge Engine control file.

ENABLE_EXTENDED_TYPES

Default: False
Range: True or False
Applies to: Oracle Client only
Console: CUSTOMIZING > SQL Data Types

This parameter makes the Oracle Client read the Oracle database's `max_string_size` parameter and check if it is set to "extended" (as opposed to "standard"). If it finds the parameter set to "extended", it allows `varchar2` and `raw` columns to have a maximum length of 32K.

Using this parameter, to ensure that DMSII ALPHA columns that are longer than 4000 characters do not get split, is much more efficient than setting the parameter `use_clob` to True.

ENABLE_DYNAMIC_HIST

Default: None
Range: True or False
Console: CUSTOMIZING > History Tables (Options > Enable dynamic history)

This parameter allows the user to add history tables without having to re-clone all the affected data sets. To do this, specify the default history columns (if any) using the `default_history_columns` configuration file parameter. Then, set the DSOPT_SaveUpdates (8) bit for all data sets for which history tables are to be kept, and run a `redefine` command with the `-R` option, forcing all data sets to be remapped. Finally, run a `reorganize` command, which will creates the history tables and their indexes. The new history tables will populate when audit files are processed.

EXTERNAL_COLUMN[N]

Default: N/A
Range: N/A
Console: CUSTOMIZING > User Columns Section One

This parameter allows you to globally change the `item_name`, `sql_type`, or `sql_length` of the non DMSII columns described in Numeric Date and Time in Non-Contiguous Columns. The syntax is as follows:

```
external_column[n] = ["name"[,sql_type[,sql_length]]]
```

Where	Is
n	The corresponding bit number (`dms_subtype` value) for the non-DMSII column. **NOTE:** The brackets and value are required syntax.
name	Custom column name
sql_type	An integer value that represents the internal code for the SQL type that you want to use. The Client only accepts data types that make sense for a particular column. For instance, you cannot set the data type for the AFN to **bit** or **char**, but you can set it to **int** or **dec(10)**. For details, see DMSII and Relational Database Data Types.
sql_length	A value that represents the length of the data type. Specify this value only if the data type requires it. If the data type does not have a length specification, specifying a value may cause an error.

The following table shows allowable `sql_type` values for `external_column` .

DMS Subtype	Mask Value (hex)	Default Column Name	Allowable SQL Types (SQL Server)	Allowable SQL Types (Oracle)
1	0x0001	update_type	tinyint, shortint, int, bigint	number(n)
2	0x0002	update_time	datetime, smalldatetime, datetime2	date
3	0x0004	update_ts	timestamp	N/A
4	0x0008	audit_ts	datetime, smalldatetime	date, timestamp
5	0x0010	audit_filenum	shortint, int, dec(n), bigint	number(n)
6	0x0020	audit_block	int, dec(n), bigint	number(n)
7	0x0040	source_name	varchar(n), char(n)	varchar(n), char(n)
8	0x0080	source_id	tinyint, shortint, int, bigint	number(n) (where n >=3)
9	0x0100	my_id	int, dec(n), bigint	N/A
10	0x0200	deleted_record	int, bigint	number(n) (where n >=9)
11	0x0400	source_name	varchar(n), char(n)	varchar(n), char(n)
12	0x0800	source_id	tinyint, shortint, int, bigint	number(n)(where n >=3)
13	0x1000	audit_ts	datetime, smalldatetime, datatime2	date, timestamp
14	0x2000	user_column1	char(n), varchar(n), tinyint, shortint, int, float, datetime, dec(n), smalldatetime, datetime2, date, time	char(n), varchar(n), number(n), float, date
15	0x4000	sequence_no	int, bigint	number(n)(where n >=9)
16	0x8000	delete_sqno	shortint, int, bigint	number(n)(where n >=5)
17	0x10000	create_time	datetime, smalldatetime, datetime2	date, timestamp
18	0x20000	user_column2	char(n), varchar(n), tinyint, shortint, int, float, datetime, dec(n), smalldatetime, datetime2, date, time	char(n), varchar(n),number(n), float, date

DMS Subtype	Mask Value (hex)	Default Column Name	Allowable SQL Types (SQL Server)	Allowable SQL Types (Oracle)
19	0x40000	user_column3	char(n), varchar(n), tinyint, shortint, int, float, datetime, dec(n), smalldatetime, datetime2, date, time	char(n), varchar(n), number(n), float, date
20	0x80000	user_column4	char(n), varchar(n), tinyint, shortint, int, float, datetime, dec(n), smalldatetime, datetime2, date, time	char(n), varchar(n), number(n), float, date

> ✏️ **Note**
>
> For Oracle, if you choose the **tinyint** value for `sql_type` you get **number(3)**, if you choose the **smallint** value you get **number(5)** and so on, as the data types in question are not defined for Oracle. Oracle has only one type of data type for integer values **number(n)**.

For example, the entry below causes the `audit_filenum` column to be renamed AFN (the double quotation marks are optional since no special characters are involved); the `sql_type` and `sql_length` remain unchanged.

```
    external_column[5] = "AFN"

    default_user_columns  = 0x0010
```

In the example below, the data type of the `audit_block` column changed to **dec(12)**.

```
external_column[6] = ,11,12
```

EXTRACT_EMBEDDED

Default: False
Range: True or False
Console: CUSTOMIZING (DMSII related parameters > Extract embedded data sets)

Use the `extract_embedded` parameter when the DMSII INDEPENDENTTRANS option is reset. If INDEPENDENTTRANS is set, the `extract_embedded` parameter is not needed because the Databridge Client can clone and update embedded datasets.

When INDEPENDENTTRANS is reset, use this parameter as follows:

- Set `extract_embedded` to True if you want the Databridge Client to extract embedded data sets during cloning when INDEPENDENTTRANS is reset. However, the Databridge Client cannot apply fixups or updates to these extracted embedded data sets.

- Set `extract_embedded` to False if you want the Databridge Client to ignore embedded data sets.

FLATTEN_ALL_OCCURS

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Table layout > Flatten all OCCURS)

This parameter simplifies writing user scripts when you want to flatten a lot of OCCURS clauses. Setting this parameter to True causes the Client to initialize the DIOPT_Flatten_Occurs bit to 1 in the `di_options` column of the DMS_ITEMS Client control table for all items that have an OCCURS clause. This avoids having to do this using user scripts. If you do not want to flatten certain OCCURS clauses, you can set the corresponding bit to 0 for those specific items by using customization user scripts or by using the Administrative Console's **Customize** command (see the *Databridge Administrative Console Help*).

FORCE_AA_VALUE_ONLY

Default: 0
Range: 0-2
Console: CUSTOMIZING > Advanced (Global data set options > Force AA Values...)

When set to 1, this parameter globally sets the DSOPT_Use_AA_Only bit in the `ds_options` column for the DATASETS table entries that have valid AA Values or RSNs. When set to 2, this action is only performed for data sets that have RSNs, because AA Values aren't preserved when a garbage collection or structural reorganization occurs.

If you want to exclude certain data sets, you can set the DSOPT_Use_AA_Only bit to 0 by using user scripts or the Administrative Console's **Customize** command (see the *Databridge Administrative Console Help*).

HISTORY_TABLES

Default: 0
Range: 0-2
Console: CUSTOMIZING > History tables (Data set history tables)

This parameter is designed to simplify script writing. It allows you to make the `define` command globally set the DSOPT_Save_Updates and DSOPT_History_Only bits. A value of 0 indicates that neither bit should be set for data sets. A value of 1 indicates that the DSOPT_Save_Updates bit should be set for all data sets. Finally, a value of 2 indicates that the DSOPT_Save_Updates and the DSOPT_History_Only bits should both be set for all data sets.

INHIBIT_REQUIRED_OPT

Default: False
Range: True or False
Console: CUSTOMIZING (General > Ignore REQUIRED attribute)

The 6.6 Client honors the REQUIRED attribute of DMS items that indicates that the item cannot be NULL because it is a member of a SET. This option tells the Client no to pay attention to the REQUIRED attribute and handle things the way older Clients did. We already have provisions in place that inhibit data sources created by older Clients from honoring the REQUIRED attribute. Without these provisions, columns that have the REQUIRED attribute would be changed to NOT NULL, which would make them incompatible with the table layouts used by older Clients. This parameter allows such column to have the NULL attribute so that items that have bad values can be stored as NULL, rather than an artificial value that is used in place of NULL.

This parameter has no effect on items that are keys, which always have the NOT NULL attribute.

MAXIMUM_COLUMNS

Default: Dependent on the database
Console: CUSTOMIZING > Advanced (Table layout > Maximum columns ...)

The `maximum_columns` parameter enables you to reduce the column count when a table split occurs because of the maximum column limitation of the relational database. For example, if you want to add a column containing the value of the audit timestamp file to the first table of a split table, you can set the `maximum_columns` parameter to 1023 instead of 1024. By doing so, you avoid moving an item from a full table to a secondary table to make room for the new column. The table below shows the maximum columns and ranges for different relational databases.

Database	Default	Range
Oracle	1000	1-1000
SQL Server	1024	1-1024

MIN_VARCHAR

Default: 4
Range: 0 to 255
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use varchar ...)

This parameter supplements the `use_varchar` configuration file parameter by adding the condition that the length must be at least equal to the value of this parameter. Setting this parameter value to 4 would force columns whose data types would have been VARCHAR(1), VARCHAR(2), or VARCHAR(3) to instead be CHAR(1), CHAR(2), and CHAR(3) if `use_varchar` is set to True.

When using the Administrative Console the `Use varchar` slider must be enabled before you can change the value of this parameter.

> ✏️ **Note**
>
> Setting this parameter to a non-zero value when `use_varchar` is set to False has no effect.

MINIMIZE_COL_UPDATES

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Global data set options > Update changed columns only)

The `minimize_col_updates` parameter specifies whether the `define`, `redefine` and the Administrative Console's **Customize** commands should set the DSOPT_Optimize_4_CDC bit in the `ds_options` column of the DATASETS table. This bit, when set, causes the Client to create update statements that only assign values to columns whose values are changed. To do this, stored procedures are abandoned in favor of pure SQL without the use of host variables. This slows down the Client considerably, but the overall process may ultimately take less time because SQL Server or Oracle replication sends significantly less data to the remote database. See the parameter enable_minimized_col, which allows the user to disable this option without having to run a `redefine` command.

> ⚠️ **Caution**
>
> Using this parameter will significantly slow down update processing by the Client. If you are replicating your relational database, enabling this feature may provide some benefits if replication is very slow.

See also the `enable_minimized_col` parameter, which allows the user to disable this option without running a `redefine` command.

MISER_DATABASE

Default: False
Range: True or False
Related parameters: `automate_virtuals`, `use_nullable_dates`
Console: CUSTOMIZING (DMSII related parameters > MISER database)

This parameter is for MISER database sites. When set to True, it sets the default date format to be a MISER date. It also sets the following parameters (required for MISER sites) to True, if they aren't already set to True:

- automate_virtuals

- flatten_all_occurs

- use_nullable_dates

OPTIMIZE_UPDATES

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Global data set options > Optimize SQL updates)

The `optimize_updates` parameter specifies whether the `define`, `redefine` and the Administrative Console's **Customize** commands should set the DSOPT_Use_bi_ai bit (1) (in the `ds_options` column of the DATASETS table) for data sets that have items with OCCURS clauses that are not flattened. The Client uses this bit, which you can modify using user scripts, to determine if it should request the Databridge Engine to send all updates for the data set as BI/AI pairs. The Databridge Client then compares the before and after images to determine if an update has any effect, and suppresses all redundant updates. Depending on the data, this can greatly increase performance when you do not flatten OCCURS clauses. See the parameter enable_optimized_sql, which allows the user to disable this option without having to run a `redefine` command.

READ_NULL_RECORDS

Default: True
Range: True or False
Console: CUSTOMIZING (DMSII related parameters > Read NULL record values)

This parameter determines whether or not the Client should request the NULL VALUES for data set records from the Databridge Engine during the `define`, `redefine` and the Administrative Console's **Customize** commands. The NULL VALUES are then stored in the binary file "datasource_NullRec.dat" from which they are retrieved at the beginning of `process` or `clone` commands. When this parameter is enabled, the testing for NULL is more accurate; however, this feature generates a small amount of overhead, particularly with a large database where these records use more memory. Note that this parameter does *not* imply that NULLS are allowed in the relational database; this is still specified using the `allow_nulls` parameter.

REORG_BATCH_SIZE

Default: 50000
Range: 5000 - 100000
Console: CUSTOMIZING > Advanced (Table reorganization options > Reorg command ...)

This parameter determines the size of the transactions that the Client uses during a `reorganize` command to set the value of newly-added columns to their initial value, as defined in the DASDL. The `redefine` command creates a reorg script that uses a stored procedure to do the updates in batches that are executed as transactions. For a large table, this process can take quite long, but it does not run the database out of log space. Consider using the internal clone option instead (see use_internal_clone.

SEC_TAB_COLUMN_MASK

Default: 0
Range: 0 - 16383
Console: CUSTOMIZING > User Columns Section Two

The parameter `sec_tab_column_mask` eliminates a set of user columns from secondary tables without having to write extensive user scripts to set the `active` column in DATAITEMS to 0 for the unwanted columns. To remove those columns, the Client removes the bits you specified in `sec_tab_column_mask` from the value represented in `external_columns` and uses the resulting value to determine which user columns to add to secondary tables during `define`, `redefine` and the Administrative Console's **Customize** commands.

This parameter is intended to allow the adding the audit timestamp, the audit file number, or the audit block to primary tables without adding them to secondary tables. The default value of this parameter is 0, which indicates that no user columns should be removed from secondary tables.

When using the Administrative Console uncheck the checkboxes in the `Secondary Tables` column to set the corresponding bits in the `sec_tab_column_mask` parameter.

SPLIT_VARFMT_DATASET

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Global data set options > Split variable format data set)

This parameter makes the `define` and `redefine` and the Administrative Console's **Customize** commands set the bit DSOPT_Split_Varfmt_ds in the `ds_options` column for the DATASETS table globally. See Split Variable Format Data Sets Option

---

STRIP_DS_PREFIXES

Default: False
Range: True or False
Console: CUSTOMIZING (General > Strip Data Set prefixes)

This parameter makes the `define`, `redefine` and the Administrative Console's **Customize** commands set the `item_name_prefix` column in the DATASETS table to the data set name. This is useful when all DMSII data item names use the data set name followed by a dash as common prefix. The `strip_ds_prefixes` parameter provides a quick way of stripping those common prefixes without writing any user scripts or using the Administrative Console's **Customize** command (as renaming every column requires a lot of work).

If the prefix is an abbreviated form of the data set name (e.g. SVHIST instead of SV-HISTORY), use a user script or the Administrative Console's **Customize** command to set the `item_name_prefix` column in the DATASETS table to this value (do not include the trailing dash).

SUPPRESS_DUP_WARNINGS

Default: False
Range: True or False
Console: PROCESSING > DMSII Data Error Handling (General error handling > Suppress duplicate ...)**

The parameter `suppress_dup_warnings` controls whether or not duplicate insert and failed update warnings are displayed during update processing. The bit DSOPT_Ignore_Dups (32) in the `ds_options` column of the DATASETS table can be used instead when you want to apply this only for certain data sets.

SUPPRESS_NEW_COLUMNS

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Global Data Set Options > Ignore new columns)

The `suppress_new_columns` parameter indicates that the `redefine` and the Administrative Console's **Customize** commands set the `active` columns to 0 in the DATAITEMS and DATATABLES entries resulting from DMSII reorganizations that add DMSII items. The `suppress_new_columns` parameter is useful when you want to keep your relational database tables intact after a DMSII reorganization, particularly if the added column will cause existing application to fail. If this is the case, set `suppress_new_columns` to True.

SUPPRESS_NEW_DATASETS

Default: True
Range: True or False
Console: CUSTOMIZING (General > Ignore new data sets)

This parameter indicates whether or not the Client maps new data sets created during a DMSII reorganization. If this parameter is set to True new data sets get their `active` columns set to 0 in the DATASETS table which causes them not go get mapped to the relational database.

If you decide that you want to replicate the new data set after running the `redefine`, you must set the `active` column to 1 for the data set in the DATASETS Client control table. You can do this by updating the user script "script.user_datasets.*datasource*" in the scripts directory. You then need to run a second `redefine` command to get the data set mapped. When you run back-to-back redefine commands the Client will automatically reloads the control tables from the unload file it creates at the start of the first `redefine` command.

USE_BIGINT

Default: False
Range: True or False
Recommended Value: True
Applies to: SQL Server Client only
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use bigint ...)

This parameter is only applicable to the SQL Server Client. It indicates that the Databridge Client should map DMSII numeric data that is too large to fit in the **int** data type (32-bit integer), to **bigint** (64-bit integer). If this parameter is set to False, such data items would be mapped to **decimal(n)**. Items that are too large to fit in a **bigint** are still mapped to **decimal(n)**. This parameter makes it easy to standardize the Client to use **bigint** instead of **decimal(n)**, without having to write user scripts. It also allows you to maintain backward compatibility with older databases that do not use **bigint**, by setting this parameter to False.

USE_BINARY_AA

Default: False
Range: True or False
Console: CUSTOMIZING (AA Values and RSNs > Use binary AA Values)

This parameter maps AA Values, Parent_AA Values, RSNs (including Visible RSNs) and DMSII Links to **binary(6)** or **raw(6)** instead of **char(12)** to reduce their storage requirements by half.

AA Values (and RSNs), which are 48-bit values, are stored in 6 bytes when using binary data, as opposed to 12 bytes when using character data.

The data types used for these columns depend on the value of the `sql_type` column in the DATAITEMS Client control table. The purpose of this parameter is to define how these items are to be mapped by default to avoid changing the `sql_type` of all such columns.

USE_CLOB

Default: False
Range: True or False
Applies to: Oracle Client only
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use clob ...)

It indicates that DMSII ALPHA data that is too large to fit in a **varchar2** column, which is limited to 4000 characters, should be mapped to a data type of **clob** instead of being truncated or split into two columns.

USE_CLUSTERED_INDEX

Default: False for index. True for primary key.
Range: True or False
Applies to: SQL Server Client only
Console: CUSTOMIZING > Advanced (Indexes > Use clustered indexes)

The `use_clustered_index` parameter applies to all data tables. You can override its setting on a table-by-table basis via the DTOPT_Clustered_Index bit in the DATATABLES control table's `dt_options` column.

Use this parameter as follows:

- Set `use_clustered_index` to True if you want a clustered index for all or most tables.

- For all tables, just set this parameter to True.

- For most tables, set this parameter to True and then reset DTOPT_Clustered_Index bit in `dt_options` for those tables for which you do not want a clustered index.

- Set `use_clustered_index` to False if you want no clustered indexes on all tables, or if you want clustered indexes on only a few tables.

- For no clustered index on all tables, just set this parameter to False.

- For clustered indexes on only a few tables, set this parameter to False and then set the DTOPT_Clustered_Index bit in `dt_options` for those tables for which you do want a clustered index.

To reset or set DTOPT_Clustered_Index, see `dt_options` in DATATABLES. Typically you would do this via user scripts.

USE_COLUMN_PREFIXES

Default: False
Range: True or False
Console: CUSTOMIZING (General > Use column prefixes)

This parameter extends the `tab_name_prefix` specified in the DATASOURCES Client control table to the columns of the user tables. If the `tab_name_prefix` column of the data source is blank, this parameter has no effect. For more details, see DATASOURCES Client Control Table.

USE_DATE

Default: False
Range: True or False
Related parameters: use_datetime2
Applies to: SQL Server Client
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use date ...)

Use this parameter to make the `define`, `redefine` and the Administrative Console's **Customize** commands interpret the DIOPT_Clone_as_Date bit in the `di_options` column of the DMS_ITEMS table as a request to use a data type of **date** instead of **smalldatetime**. This eliminates the need to set the `di_options` bit DIOPT_Use_Date for every item that is to be mapped to a data type of **date**.

USE_DATETIME2

Default: False
Range: True or False
Related parameters: use_date
Applies to: SQL Server Client
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use datetime2 ...)

Use this parameter to make the `define`, `redefine` and the Administrative Console's **Customize** commands interpret the DIOPT_Use_LongDate bit in the `di_options column` of the DMS_ITEMS table as a request to use a data type of **datetime2** instead of **datetime**. This eliminates the need to set the `di_options` bit DIOPT_Use_LongDate2 bit for every item that is to be mapped to a data type of **datetime2**.

USE_DBCONFIG

Default: False (hidden when False)
Range: True or False
Applies to: Clients launched by the service
Console: Automatically set for a new data source when you use the **Customize** command

A successful run of the `upgrade` command of the **dbscripfixup** program automatically sets this parameter to True. You can later set it to False if you want to revert to using user scripts by running a **dbutility** `import` command. You should first run a **Create Users Scripts** command from the data source's **Advanced** menu, or run a **dbutility** `creatscripts` command to save all your customizations in user scripts. We recommend saving the customizations in user scripts in case you have to drop and recreate it. The `define` command will always run user scripts regardless of the setting of the parameter `use_dbconfig`, while the `redefine` command will only do this when the `use_dbconfig` parameter is set to False, or if you use the `-r` option to toggle this parameter so it is treated as False when set.

> ⚠️ **Caution**
>
> Do not change the `use_dbconfig` parameter to True when the data source was not created using the **Customize** command. If you created the data source using the **Define/Redefine** command, you will be able to use **Customize** command, unless there were user scripts involved in the underlying `define` command. If you are not sure run, the **dbscriptfixup** program's `upgrade` command to check for use scripts and fixup the control table and automatically set this parameter to True when the command is successful.

USE_DECIMAL_AA

Default: False
Range: True or False
Console: **CUSTOMIZING** (AA Values and RSNs > Use decimal AA Values)

> ✏️ **Note**
>
> This parameter is mutually exclusive with the `use_binary_aa` parameter.

This parameter maps AA Values, Parent_AA Values, RSNs (including Visible RSNs) and DMSII LINKS to a numeric data type instead of **char(12)**. The data type varies from database to database. In the case of SQL Server, **bigint** is used and in the case of Oracle, **number(15)** is used.

USE_INTERNAL_CLONE

Default: False
Range: True or False
Console: **CUSTOMIZING > Advanced** (Table reorganization options > Use internal clone ...)

This parameter affects the `redefine` and `reorganize` commands. Instead of using ALTER commands to add, delete or modify new columns to tables, the Client uses a combination of scripts and table renaming commands to create new copies of the tables with the new layouts. The Client copies the data using SELECT INTO in the case of SQL Server and CTAS (Create Table As Select) in the case of Oracle. This operation works like the bulk loader and is faster than using ALTER and UPDATE commands, but more importantly the command is not logged. The only drawback of this method is that it requires sufficient free disk storage to hold a second copy of the table for the duration of the operation.

USE_NULLABLE_DATES

Default: False
Range: True or False
Console: CUSTOMIZING (General > Allow NULL dates)

This parameter forces all MISER dates, including keys, to have the DAOPT_Nulls_Allowed bit (value 1) in the `da_options` column of the DATAITEMS Client control table. This parameter should only be set to True if you are using a MISER database. Only one MISER date is allowed as a key. The Client generates custom stored procedures that handle the cases where the MISER date that is part of the index is NULL.

USE_PRIMARY_KEY

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Indexes > Use Primary Keys)

This parameter tells the Databridge Client to create a primary key instead of a unique index for all tables. You can override its setting on a table-by-table basis via the DTOPT_Primary_Key bit in the DATATABLES Client control table's `dt_options` column.

- Set `use_primary_key` to True if you want a primary key for all or most tables.

- For all tables, just set this parameter to True.

- For most tables, set this parameter to True and then reset DTOPT_Primary_Key for those tables for which you do not want a primary key.

- Set `use_primary_key` to False if you want no primary keys on all tables, or if you want primary keys on only a few tables.

- For no primary key on all tables, just set this parameter to False.

- For primary keys on only a few tables, set this parameter to False and then set DTOPT_Primary_Key for those tables for which you do want a primary key.

To reset or set DTOPT_Primary_Key, see `dt_options` in DATATABLES. Typically you would do this via user scripts.

USE_STORED_PROCS

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Global Data Set Options > Use stored procedures in updates)

This parameter makes the `process` and `clone` commands generate actual SQL commands instead of stored procedure calls to perform updates. The Client still uses host variables, as was the case with stored procedures calls. Executing the SQL directly eliminates some overhead and makes processing the update faster. If you change this parameter, you must propagate the change to the `ds_options` columns of the DATASETS Client control table. The easiest and safest way to do this is to run a `redefine` command using the `-R` option (when using the Administrative Console, click on **Advanced > Redefine with Options** for the data source and then enable "All Data Sets" slider). The `redefine` command will ask you to run a `reorganize` command, which creates a new set of scripts for creating the tables. It also will refresh the stored procedures for all data sets by dropping them if they exist and then creating them if they are needed.

USE_TIME

Default: False
Range: True or False
Applies to: SQL Server Client
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use time ...)

Use this parameter to make the `define`, `redefine` and the Administrative Console's **Customize** commands interpret the DIOPT_Clone_as_Time bit in the `di_options` column of the DMS_ITEMS table as a request to use a data type of **time** instead of a numeric time. This eliminates the need to set the `di_options` bit DIOPT_Use_Time in the DMS_ITEMS Client control table for every item that is to be mapped to a data type of **time**.

USE_VARCHAR

Default: True
Range: True or False
Console: CUSTOMIZING > SQL Data Types (Default SQL data types > Use varchar ...)

Set `use_varchar` to True to cause the `define`, `redefine` and the Administrative Console's **Customize** commands to map DMSII ALPHA data to **varchar** (Microsoft SQL Server) or **varchar2** (Oracle) instead of **char**.

> ✏️ **Note**
>
> The Databridge Client suppresses trailing blanks from all character data constructed from DMSII ALPHA data.

Process and Clone Command Parameters

The following parameters are included in the [params] section of the Databridge Client configuration file. The parameters listed in this section affect only the `process` and `clone` commands.

ALPHA_ERROR_CUTOFF

Default: 10
Range: 0 - 100
Related parameters: `discard_data_errors`, `display_bad_data`
Console: PROCESSING > DMSII Data Error Handling (Character data error > Set item to NULL ...)

This parameter specifies the percentage of data errors in any given ALPHA item that are tolerated before the field is declared bad and treated as NULL (or simulated NULL if the column does not allow NULLS). The default value for this parameter is 10 (10%); the allowable values are in the range 0 (fail on first error) to 100 (ignore all errors).

AUX_STMTS

Default: 100
Range: 0 - 200
Console: PROCESSING (General > Number of auxiliary (ODBC/OCI) statements ...)

Use the `aux_stmts` parameter to set the number of database API (that is, ODBC or OCI) STMT structures that can be assigned to individual SQL statements. Using multiple database API STMT (statement) structures allows SQL statements to be parsed once and executed multiple times, provided the STMT structure is not reassigned to hold another SQL statement. Increasing the number of database API statements significantly improves processing time, if your system has enough memory.

> ✏️ **Note**
>
> When using the Oracle Client, make sure that the `open_cursors` parameter defined in the database initialization file for the Oracle instance ("init *SID*.ora", where *SID* is the name of the instance) is set to a high enough value.

BATCH_JOB_PERIOD

Default: ;00:00, 00:00
Range: 00:00 to 24:00 (The two time values cannot be equal.)
Console: PROCESSING > Engine and Enterprise Server (Batch commit parameters > Batch job ...)

The `batch_job_period` parameter specifies the block of time during which batch jobs typically run. For example "batch_job_period = 22:00, 01:00" indicates that batch jobs run between 10:00 pm and 1:00 am. The syntax for the four commit checkpoint parameters, that can be overridden by the Client, was modified to allow an optional second value to be specified. The second value represents

the alternate value to be used during the batch period. The Client was then modified implement the automatic switching of commit parameters between the two periods. The switching is based on the value of the audit time stamp rather than the time when the Client is run. Commit parameters that use low values do not work well for batch jobs, but the might work well for on-line periods to keep the lag time low. This enhancement, which was added to Databridge 7.0, allows you optimize performnace for both periods by picking appropriate commit parameters for each period.

CENTURY_BREAK

Default: 50
Range: 0 - 99 or -1
Console: PROCESSING > Date and Time (Date parameters > Century break)

Values for the `dms_subtype` column in the DMS_ITEMS Client control table that are in the 30s, 60s, and 90s have 2-digit years (*yy*) which represent dates in the 20th and 21st centuries. The `century break` parameter is used to determine the century for a 2-digit year. For example, if this parameter is set to 50, values < 50 are 21st century years (20*yy*); values >= 50 are 20th century years (19*yy*).

When the `century_break` value is set to -1, the century break value is dynamically calculated based on the current year giving the two digit years a range of "current_year - 49" to "current_year + 50". In the case of the year 2021, this range is 1972 to 2071.

You can find DMSII date formats that are affected by the `century_break` parameter at Decoding DMSII Dates, Times, and Date/Times.

COMMIT_ABSN_INC

Default: --1 (This parameter is commented out.)
Range: 0 -- 200,000
Related parameters: `commit_update_inc` , `commit_time_inc` , `commit_txn_inc`
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters)

The `commit_absn_inc` parameter allows the Databridge Client to override the Databridge Engine CHECKPOINT CLIENT EVERY *nnn* AUDIT BLOCKS parameter setting. This parameter causes the Databridge Engine to generate a commit at the next quiet point after *nnn* audit blocks have been processed since the last commit. This parameter determines one of many conditions under which Databridge Engine generates a commit.

When the `commit_absn_inc` parameter is not included in the configuration file, or it is commented out, the Databridge Client uses the default value of --1. This value indicates that the Client won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is 100). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

A value of 0 disables the use of this parameter by Databridge Engine. A value that exceeds the value specified in Databridge Engine control file is ignored.

When the `batch_job_period` parameter is enabled, you need to add a second value, preceded by a comma, to specify the alternate value to be used when the Client is processing audit blocks that were created during the batch period.

When using the Administrative Console the second value is provided in the **Batch commit parameters** group, after enabling the **Batch job** slider and entering the time interval for the batch job period.

If `commit_absn_inc` , `commit_update_inc` , `commit_time_inc` , and `commit_txn_inc` are specified, Databridge Engine commits at the next quiet point after one or more of the conditions are satisfied.

COMMIT_IDLE_DATABASE

Default: --1 (This parameter is commented out.)
Range: True or False
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters > commit during idle ...)

This parameter allows the Client to override the COMMIT DURING IDLE DATABASE in the Databridge Engine's Control File. It makes the Databridge Engine commit at the next quiet point when it encounters an update while the data base is idle.

COMMIT_LONGTRANS

Default: --1 (This parameter is commented out.)
Range: True or False
Related parameters: `commit_absn_inc` , `commit_update_inc` , `commit_time_inc` , `commit_txn_inc`
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters)

> ⚠️ **Warning**
>
> Setting this parameter to True (that is, overriding the Databridge Engine CHECKPOINT LONG TRANSACTIONS parameter) can result in problems and is therefore not recommended. By default, this parameter is commented out.

This parameter determines one of many conditions under which Databridge Engine should generate a commit. When this value is not specified, as in the case of the default setting, Databridge Client uses an internal value of --1. This value indicates that it won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is False). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

A value of 0 disables the use of this parameter by Databridge Engine. A value that exceeds the value specified in the Databridge Engine control file is ignored.

COMMIT_TIME_INC

Default: --1 (This parameter is commented out.)
Range: 0 - 300 seconds
Related parameters: `commit_absn_inc` , `commit_update_inc` , `commit_txn_inc`
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters)

The `commit_time_inc` parameter allows the Databridge Client to override the Databridge Engine CHECKPOINT CLIENT EVERY *n* SECONDS parameter setting by causing Databridge Engine to generate a commit at the next quiet point after *n* seconds have elapsed in the current transaction. This parameter determines one of many conditions under which Databridge Engine should generate a commit.

When the `commit_time_inc` parameter is not included in the configuration file, or it is commented out, the Databridge Client uses the default value of --1. This value indicates that it won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is 0). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

A value of 0 disables the use of this parameter by Databridge Engine. A value that exceeds the value specified in the Databridge Engine control file is ignored.

When the `batch_job_period` parameter is enabled, you need to add a second value preceded by a comma, to specify the alternate value to be used when the Client is processing audit blocks that were created during the batch period.

When using the Administrative console the second value is provided in the "Batch commit parameters" group, after enabling the "Batch job" slider.

If `commit_absn_inc` , `commit_update_inc` , `commit_time_inc` , and `commit_txn_inc` are specified, Databridge Engine commits at the next quiet point after one or more of these conditions are satisfied.

COMMIT_TXN_INC

Default: --1 (This parameter is commented out.)
Range: 0 -- 200,000
Related parameters: commit_absn_inc, commit_update_inc, commit_time_inc
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters)

The `commit_txn_inc` parameter allows the Databridge Client to override the Databridge Engine CHECKPOINT CLIENT EVERY *n* TRANSACTIONS parameter setting by causing Databridge Engine to generate a commit at the next quiet point after *n* transaction

groups have been processed. This parameter determines one of many conditions under which Databridge Engine should generate a commit.

When the `commit_txb_inc` parameter is not included in the configuration file, or it is commented out, the Databridge Client uses the default value of --1. This value indicates that it won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is 0). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

A value of 0 disables the use of this parameter by Databridge Engine. A value that exceeds the value specified in the Databridge Engine control file is ignored.

When the `batch_job_period` parameter is enabled, you need to add a second value preceded by a comma, to specify the alternate value to be used when the Client is processing audit blocks that were created during the batch period.

When using the Administrative console the second value is provided in the "Batch commit parameters" group, after enabling the "Batch job" slider.

If `commit_absn_inc`, `commit_update_inc`, `commit_time_inc`, and `commit_txn_inc` are specified, Databridge Engine commits at the next quiet point after one or more of these conditions are satisfied.

COMMIT_UPDATE_INC

Default: -1 (This parameter is commented out.)
Range: 0 -- 200,000
Related parameters: `commit_absn_inc`, `commit_time_inc`, `commit_txn_inc`
Console: PROCESSING > Engine and Enterprise Server (COMMIT parameters)

The `commit_update_inc` parameter allows the Databridge Client to override the Databridge Engine CHECKPOINT CLIENT EVERY *nnn* UPDATE RECORDS parameter setting. It does this by causing Databridge Engine to generate a commit at the next quiet point after *nnn* updates have been sent to the Databridge Client. This parameter determines one of many conditions under which Databridge Engine should generate a commit.

When the `commit_update_inc` parameter is not included in the configuration file, or it is commented out, the Databridge Client uses the default value of --1. This value indicates that it won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is 1000). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

A value of 0 disables the use of this parameter by Databridge Engine. A value that exceeds the value specified in the Databridge Engine control file is ignored.

When the `batch_job_period` parameter is enabled, you need to add a second value, preceded by a comma, to specify the alternate value to be used when the Client is processing audit blocks that were created during the batch period.

When using the Administrative console the second value is provided in the "Batch commit parameters" group, after enabling the "Batch job" slider.

If `commit_absn_inc`, `commit_update_inc`, `commit_time_inc`, and `commit_txn_inc` are specified, Databridge Engine commits at the next quiet point after one or more of these conditions are satisfied.

CONTROLLED_EXECUTION

Default: False
Range: True or False
Related command-line option: `-o`
Related parameters: `min_check_time`
Applies to: Command-line Client (**dbutility**) only
Console: N/A

> ✏️ **Note**
>
> This parameter is only used by the command-line Client dbutility. The blackout_period parameter in the scheduling section of the configuration file renders this method obsolete.

The `controlled_execution` parameter forces the Client to check the values of the `stop_time` and `end_stop_time` columns of the DATASOURCES table. These columns enable an application external to Databridge to specify a block of time during which Databridge Client operations are disallowed. If the Databridge Client determines that this period of time exists, update processing is stopped. Any attempts you make to restart the Databridge Client also fail until the blackout period is over or the `stop_time` and `end_stop_time` columns are set to 0.

CONVERT_REVERSALS

Default: -1 (This parameter is commented out.)
Range: True or False
Console: PROCESSING > **Engine and Enterprise Server** (General > Convert reversals ...)

The `convert_reversals` parameter allows the Client to override the Databridge Engine control file parameter CONVERT REVERSALS. Refer the Databridge Host Administrator Guide for more details on this parameter. When this value is not specified, as in the case of the default setting, Databridge Client uses an internal value of -1. This value indicates that it won't attempt to override the settings for the corresponding Databridge Engine parameter (whose default value is false). The value -1 is not a valid setting, per se, and will result in a "value out of range" error. Comment the line out instead by adding a ";" at the start of the line.

CORRECT_BAD_DAYS

Default: 0
Range: -1 to 2
Console: PROCESSING > **Date and Time** (Date parameters > Correct invalid date values)

The parameter `correct_bad_days` specifies whether the Databridge Client should treat a DMSII date with a bad day (or month) value as an error or attempt to correct it by setting the value to last day for the given month and year.

This parameter does not apply in the following circumstances:

- Dates whose day values are greater than 31 (unless the parameter is set to 2)
- DMSII Julian dates (`dms_subtype` values 17, 18, 27, 28, 37, 38)
- MISER dates, Linc dates, DMSII dates and DMSII timestamps

Set this parameter as follows:

- Set `correct_bad_days` to 1 if you want the Databridge Client to set bad DMSII dates to the last day for the given month. In this case, a bad date would be February 29, 2002 because 2002 is not a leap year. The Databridge Client would correct this date to February 28, 2002. Likewise, a date of September 31 would be corrected to September 30, regardless of the year because September always has 30 days. A day value greater than 31 is not corrected in this case. However, a day value of 0 is always silently changed to 1, regardless of the setting of the setting of this parameter.

- Set `correct_bad_days` to 2 if you want the Databridge Client to perform the following corrections in addition to the ones for the case where `correct_bad_days` is set to 1. Day values greater than 31 are set to the last legal day of the month, month values greater than 12 are set to 12 and a month value of 0 is set to 1.

- Set `correct_bad_days` to 0 if you want the Databridge Client to store bad dates as NULL. If the DAOPT_Nulls_Allowed bit in the `da_options` column of the corresponding DATAITEMS entry is not set, the bad date is stored as 1/1/1900 in the SQL Server Client and 1/1/0001 in the Oracle Client.

- Set `correct_bad_days` to -1 if you want the Databridge Client to store bad dates (including dates with a day value of 0, which normally gets changed to 1) as NULL. If the DAOPT_Nulls_Allowed bit in the `da_options` column of the corresponding DATAITEMS entry is not set, the bad date is stored as 1/1/1900 in the SQL Server Client and 1/1/0001 in the Oracle Client.

DBE_DFLT_ORIGIN

Default: direct
Range: direct, indirect, cache
Console: PROCESSING > Engine and DBEnterprise Server (General)

The `dbe_dflt_origin` parameter specifies the expected origin for Enterprise Server audit files during normal operations. The Client issues a WARNING if Enterprise Server sends it a different value whenever it starts processing a new audit file.

DEFER_FIXUP_PHASE

Default: False
Range: True or False
Console: PROCESSING > Stop Conditions

The `defer_fixup_phase` parameter prevents the Databridge Client from entering the fixup phase, which is deferred to the next `process` command.

> ✎ **Note**
>
> Version 6.1 and later Databridge Clients do not support parallel clones, which was one of the reason for the existence of this command.

DISCARD_DATA_ERRORS

Default: False
Range: True or False
Related parameters: `alpha_error_cutoff`, `display_bad_data`
Console: PROCESSING > DMSII Data Error Handling (General error handling > Discard records ...)

The parameter `discard_data_errors` instructs the Client to write all records with data errors to the discard file *tablename*.bad, located in the **discards** subdirectory of the working directory. If you set this parameter to False, the Client loads the record into the database with the affected column set to NULL or with the affected characters changed to question marks (?). Setting this parameter to True forces the `alpha_error_cutoff` parameter to 0 so that no errors are tolerated before the Client declares the field bad. For more information, see alpha_error_cutoff.

DISPLAY_BAD_DATA

Default: False
Range: True or False
Related parameters: `alpha_error_cutoff`, `discard_data_errors`
Console: PROCESSING > DMSII Data Error Handling (General error handling > Display data errors ...)

The `display_bad_data` parameter is a debugging aid for users that encounter many data errors. Enabling this parameter makes the Databridge Client display the raw DMSII data in a field that is found to have a data error. This output, which immediately follows the data error messages, is suppressed whenever the number of errors exceeds the maximum number of errors to be logged (as defined by the `error_display_limits` configuration file parameter).

ENABLE_AF_STATS

Default: False
Range: True or False
Console: PROCESSING > Statistics (Audit file statistics)

This parameters enables the writing of the audit file statistics to the AF_STATS Client control table. These statistics are the incremental statistics that are written to the log file every time the Client start processing a new audit file. The only difference being that when the processing of an audit file spans multiple Client runs, the statistics are combined into a single record that spans the audit file. See the section on AF_STATS Chapter 8 titled "Databridge Control Tables" for a complete description of the columns of the AF_STATS Client control table.

ENABLE_DOC_RECORDS

Default: False
Range: True or False
Console: PROCESSING > Engine and Enterprise Server (General)

The `enable_doc_records` parameter requests DOC records from the Databridge Engine. Enable this parameter only when you are troubleshooting Databridge Engine problems. These records help diagnose the various block types that the Engine encounters while processing audit files.

> ✏️ **Note**
>
> These records are recorded in the trace file only when full debugging is enabled (`-d`) or if you enable the DOC Record Tracing option as described in the section DOC Record Trace .

ENABLE_FF_PADDING

Default: False
Range: True or False
Console: PROCESSING > DMSII Data Error Handling (Character Data > Enable High Value Padding)

This parameter enables an option that lets you mark items as padded with high values to achieve left justification. This parameter applies to ALPHA items and unsigned numeric items that are stored as ALPHA data. When set to False, this parameter does not appear in the exported configuration file.

ENABLE_MINIMIZED_COL

Default: True
Range: True or False
Console: PROCESSING > Advanced (General > Override changed columns only option)

When the `minimize_col_updates` parameter is applied during a `define` or `redefine` and the Administrative Console's **Customize** commands, the DSOPT_Optimize_4_CDC bit is set in all data sets. Set this parameter to False to override the DSOPT_Optimize_4_CDC bit during the change tracking phase and avoid having to run a `redefine` command to clear the DSOPT_Optimize_4_CDC bit.

Minimized SQL is not generally very useful, as it slows down update processing by not using host variables. It might be useful when the relational database is replicated to a secondary database that is remote. This option reduces the size of the changes by only updating column that are changed rather than using canned SQL that update all columns regardless of whether they were changed or not.

ENABLE_OPTIMIZED_SQL

Default: True
Range: True or False
Console: PROCESSING > Advanced (General > Override optimized SQL updates option)

When the `optimize_updates` parameter is applied during a `define` , `redefine` and the Administrative Console's **Customize** commands, the DSOPT_Use_bi_ai bit is set in all data sets containing secondary OCCURS tables. Set this parameter to False to override the DSOPT_Use_bi_ai bit during the change tracking phase and avoid having to run a `redefine` command to clear the DSOPT_Use_bi_ai bit. The DSOPT_Use_bi_ai bit is documented under `ds_options` in the DATASETS Client control table.

ENGINE_WORKERS

Default: -1 (This parameter is commented out.)
Range: 1-10
Console: PROCESSING > Engine and Enterprise Server (General)

The `engine_workers` parameter allows the Databridge Client to override the Databridge Engine WORKERS = *n* parameter setting to control the number of extract workers Databridge Engine can use during the data extraction phase.

This value can only be lower than Host parameter (DATA/ENGINE/CONTROL), never higher.

The default value of -1 indicates that the Client does not attempt to override the settings for the corresponding Databridge Engine parameter whose default value is 1.

ERROR_DISPLAY_LIMITS

Default: 10 errors for the display; 100 errors for the log file
Range: 0-1000, 0-10000
Console: PROCESSING > DMSII Data Error Handling (General error handling > Error display limits)

The `error_display_limits` parameter allows you to control the number of screen output messages and log file entries for data errors. All data error counts are maintained for individual tables. This parameter prevents Databridge from filling the disk with meaningless errors when a large number of the records in a data set are in error.

INHIBIT_8_BIT_DATA

Default: False
Range: True or False
Console: PROCESSING > DMSII Data Error Handling (Character data > Change 8-bit characters to ? ...)

Use the `inhibit_8_bit_data` parameter for data validation. Do not set this parameter if your data contains international characters.

For example, if your valid alpha data consists of 7-bit characters, set inhibit_8_bit_data to True. The Databridge Client then changes all 8-bit characters to a question mark (?) and issues a warning message on the *first* occurrence of the bad data. The message contains the keys of the record with the invalid data, as in the following:

```
WARNING: Item 'cm_addr_line_2' in table 'customers' has 8-bit characters in alpha data - Keys: cm_number=00101301
```

> ✏️ **Note**
>
> If an item containing 8-bit characters or control characters happens to be a key, the record is discarded as it attempts to change the bad characters to ? (question marks), potentially resulting in duplicate records. All discarded records are written to the file "*tablename*.bad" in the **discards** subdirectory of the working directory for the data source.

INHIBIT_CONSOLE

Default: False
Range: True or False
Related command-line parameter: `-C` (toggle)
Applies to: Command-line Client (dbutility) only
Console: N/A

When set to True, this parameter disables the console commands for the command-line Clients (**dbutility**). The console commands are explained in Controlling and Monitoring dbutility.

INHIBIT_CTRL_CHARS

Default: False
Range: True or False
Console: PROCESSING > DMSII Data Error Handling (Character data > Control character)

When this parameter is set to true, the Databridge Client treats all control characters as errors and converts them to a question mark (?) when set to True. When it is set to False, it supports all control characters except NUL, CR, LF, and TAB (in some cases). The Client for Oracle accepts TAB, and the Microsoft SQL Server Client accepts TAB characters if the **bcp** delimiter is not the TAB character.

> ✏️ **Note**
>
> This parameter and the parameter `convert_ctrl_char` are mutually exclusive. If you attempt to set them both to True, the configuration file scanner will generate an error.

INHIBIT_DROP_HISTORY

Default: False
Range: True or False
Console: CUSTOMIZING > History Tables (Options > Inhibit Drop)

Use this option to prevent the Databridge Client from inadvertently dropping history tables during a `clone`, `process`, or `drop` command or to prevent the clean-up scripts from running.

This is a safeguard to prevent the user from making an unrecoverable error. If you want the tables dropped and are sure of that, you can change this setting and rerun the Client. However, make sure to set it back to True for the next time.

- If the data source is dropped it cannot be reprocessed because the Databridge Client attempts to drop the history table, and the option prevents this from happening.
- Cleanup scripts deal with tables that are partially re-cloned. In the case of multiple source tables, they are re-cloned one data source at a time. In the case of tables that preserve deleted records, the deleted records are preserved during a clone. In the case of MISER data sets that hold history and resident records, the table is re-cloned without dropping the history records (which is different than Databridge Client history tables).

INHIBIT_INIT_VALUES

Default: False
Range: True or False
Console: CUSTOMIZING > Advanced (Table reorganization options > Do not set initial values ...)

This parameter allows you to disable new columns added after a DMSII reorganization from getting set to their initial values. If there are large tables and your applications can deal with the new columns being NULL, setting this parameter to True will save time.

> ✏️ **Note**
>
> If you ever re-clone such tables these columns will no longer be NULL.

KEEP_UNDIGITS

Default: 0
Range: 0 - 2
Console: PROCESSING > DMSII Data Error Handling (Character data ...)

This parameter, when set to 1, allows you to keep the undigits in numeric fields that are stored as character data. These characters will have a value of 'A' through 'F' based on the value of the corresponding undigit.

Setting this parameter to 2 also causes undigits in numeric fields to be treated as 9s.

LINC_CENTURY_BASE

Default: 1957
Range: 1800 and up
Console: PROCESSING > Date and Time Parameters (Date parameters)

This parameter allows you to configure the base year for Linc dates, which is site-specific parameter in Linc databases.

MASKING_PARAMETER[*N*]

Default: N/A
Range: "string"
Applies to: SQL Server Client using SQL Server 2016 or newer
Console: CUSTOMIZING > SQL Suffixes (Data Masks)

This array of parameters is used to hold the parameters for the random and partial masking functions. Data masking is defined using the `masking_info` column of DATAITEMS, which defines the masking function and the index of the corresponding parameter string (which does not include the parentheses). The format of the `masking_info` column (which is an **int**) is 0x00nn000m, where m is the masking function code and nn is the index into the table of masking parameters.

The following masking codes are defined: 0 -- no masking, 1 -- default() masking function, 2 -- email() masking function, 3-- random() masking function, 4 -- partial masking function. The last two masking functions have 2 and 3 parameters respectively. These parameters are represented in the left half of the `masking_info` by the index into the table of masking parameters (or example 0x00010003 would be a random() masking function with its parameters represented by the `masking_parameter[1]` entry in the configuration file. This parameter could be "0,100" which would result in the masking function "random(1,100)" being used in defining the data mask for the column.

You can reuse `masking_parameter` entries as many times as needed. The index must be between 1 and 100. Refer to the SQL Server documentation for details on how data masking works.

The figure below shows how to set up masking parameters using the Administrative Console's **Customize** command. This example results in the accountno column having the following attributes:

```
accountno varchar(16) masked with (function='partial(0,"***********",4)') NULL,
```

MAX_CLONE_COUNT

Default: 10000
Range: 1000-100000 SQL insert statements before a commit
Related command-line option: `-s`
Console: N/A

In most cases you do not need to use this parameter. This parameter is used only when you enter the `-s` option at the command line or set the bit DSOPT_No_Loader (2) in the `ds_options` column of corresponding row in the DATASETS table.

The `max_clone_count` parameter applies to the dbutility `process` and `clone` commands for cloning only, not updates, when the command-line option `-s` is set. It defines the maximum number of rows that the Client can insert into a table before a commit is required.

The value of this parameter has no effect on the commit frequency during the processing of updates, which is controlled by Databridge Engine.

MAX_DISCARDS

Default: 0,100
Range: 0-10000, 0-1000
Console: PROCESSING > DMSII Data Error Handling (General error handling > Discard record ...)

This is a two-part parameter that controls how the Client handles discarded records. The first number represents the total number of discards the Client will tolerate before abending. The second number represents the maximum number of discards records for a table that are written to the discard file. Discards that exceed this number are ignored.

If either of these values are set to zero, no limits are imposed for the corresponding actions, and the Client will behave the way it did before this parameter was implemented.

The first value must be greater than the second value, unless the second value is zero, indicating that it's unlimited. Otherwise, the Client will always abend before the second value goes into effect.

MAX_RETRY_SECS

Default: 20
Range: 1 - 36000 seconds
Related parameters: `use_dbwait`, `max_wait_secs`
Console: PROCESSING (Audit unavailable action > Retry interval ...)

The `max_retry_secs` parameter works only when you enable max_wait_secs so be sure to set both.

The `max_retry_secs` parameter applies when you use the `process` command to track changes. It defines the value for the retry time (in seconds) for the DBWAIT API call for Databridge Engine, which is called when the `use_dbwait` parameter is set to True. This value defines the amount of time to wait before reading an audit file again.

For example, if you set `max_wait_secs` to 3600 seconds (same as 1 hour) and `max_retry_secs` to 60 seconds, Databridge Engine checks for new updates in the audit file once a minute for an hour before giving up and returning an audit file unavailable status.

Note that when you supply a second value for the parameter `max_wait_secs`, the value of `max_retry_secs` must be less than that value, as the Client expects to get control back within the time specified by the second value of `max_wait_secs`. Ideally, the second value of `max_wait_secs` should be an exact multiple of `max_retry_secs` to ensure that Client gets control back after the correct amount of time. For example, if using the default value of 60 for the second value of `max_wait_secs`, we recommend you set this parameter to 20 or 30 seconds, which ensures that the Client gets control back in 60 seconds.

MAX_SRV_IDLE_TIME

Default: 0
Range: 15 -- 600 minutes
Console: PROCESSING > Advanced (Server inactivity timeout)

This parameter allows the timer thread to time out a server connection after several inactivity warnings. When this parameter is set to a non-zero value, which represents the timeout value in minutes, the Client stops if the length of an inactivity period exceeds this value.

The Client stops with an exit code of 2059. If using the service, this will cause it to restart the Client after a brief delay. This parameter provides an alternative to the TCP keep-alive mechanism to detect situations where we have a dead connection. This situation is most likely to occur if the MCP is HALT LOADED.

When using the Administrative Console enabling the slider labeled "Server inactivity timeout ..." activates the edit box for supplying the value for this parameter.

MAX_WAIT_SECS

Default: 3600,60
Range: 0--36000 seconds for the first value, 0 or 60-300 seconds for the second value
Related parameters: `use_dbwait` , `max_retry_secs`
Console: **PROCESSING** (Audit unavailable action > Maximum wait time ...)

The `max_wait_secs` parameter works only when you enable [use_dbwait](). When you set `max_wait_secs` , also set [max_retry_secs]().

The `max_wait_secs` parameter applies when you use the dbutility `process` command to track changes. It defines the maximum wait time (in seconds) for the DBWAIT API call for Databridge Engine, which is called when the `use_dbwait` parameter is set to True. This is the maximum amount of time that Databridge Engine waits before returning an audit file unavailable status.

The `max_wait_secs` value and the `max_retry_secs` value are the DBWAIT API input parameters. The maximum wait time (`max_wait_secs`) specifies the cutoff point for the retries (`max_retry_secs`). DBWAIT gives up when the total amount of time elapsed since the last successful attempt to read the audit file is greater than or equal to the `max_wait_secs` .

The optional second value for this parameter is used to break up large wait times into smaller increments by making the Client repeatedly issue DBWAIT calls using this second value, which must be smaller than the first value (unless the first value is 0).

For example setting `max_wait_secs` to 3600,60 will result in the Client issuing a DBWAIT remote procedure call with a `max_wait_secs` value of 60 seconds. Upon getting a "no more audit available" return status, the Client will issue another DBWAIT call until it has received no data for the amount of time indicated by the first parameter. This way of doing things ensures that an idle line has some traffic on it, which makes it possible to detect situations where the network goes down and neither side knows about it.

Upon receiving data the Client resets the timer that keeps track of idle during which no updates are received. A value of 0 for the second parameter makes the Databridge Engine handle the wait-and-retry loop without any involvement by Client.

Note that when you supply a second value for the parameter `max_wait_secs` , the value of `max_retry_secs` must be less than that value, as the Client expects to get control back within the time specified by the second value of `max_wait_secs` . Ideally the second value of `max_wait_secs` should be an exact multiple of `max_retry_secs` to ensure that Client gets control back after the correct amount of time. For example, if using the default value of 60 for the second value of `max_wait_secs` , we recommend you set this parameter to 20 or 30 seconds, which ensures that the Client gets control back in 60 seconds.

> ✏️ **Note**
>
> A value of 0 indicates that Databridge Engine continually waits.

MIN_CHECK_TIME

Default: 600 (expressed in units of seconds)
Range: 10--1200
Related parameters: controlled_execution
Applies to: Command-line Client (dbutility) only
Console: N/A

The `min_check_time` parameter is used in with the `controlled_execution` parameter to reduce the number of times the program reads the corresponding entry in the DATASOURCES table. After a quiet point, which ends a transaction group of updates, the Client only reads the DATASOURCES table if `min_check_time` has elapsed since the last read. If you set this parameter to 60 seconds, the Client reads the DATASOURCES table no more than once a minute, even if quiet points are only a few seconds apart.

MONTHS

Default: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
Range: A list of exactly 12 three-character entries
Console: N/A

Use the `months` parameter when you want to use month name abbreviations that are not in English. This parameter applies only when you are using DMSII date encoding methods that use three-character abbreviations for months.

For more information on DMSII date encoding methods, see Decoding DMSII Dates, Times, and Date/Times.

To make an entry for the `months` parameter, enter your three-character month names in order and separated by commas.

N_DMSII_BUFFERS

Default: 0
Range: 0, 2 -- 64
Related parameters: n_update_threads
Console: PROCESSING > Advanced (Multi-threaded updates > Number of DMSII buffers ...)

Use this parameter to configure the number of RPC buffers to be used by the Client. If you let this parameter default or set it to 0, the Client uses 4 times `n_update_threads` RPC buffers or 2 buffers when `n_update_threads` is 0. When you have DMSII links enabled, this parameter is set to the number of extract workers unless the default value is larger. Raising this value might improve performance by ensuring that there are enough buffers queued to keep the update workers busy at all times.

N_UPDATE_THREADS

Default: 8
Range: 0 -- 16
Applies to: SQL Server (see note) and Oracle Clients
Console: PROCESSING > Advanced (Multi-threaded updates > Number of update threads ...)

Use this parameter to specify the number of update threads to be used. The update threads are responsible for executing SQL to update the user tables and writing bulk loader temporary files. When using the BCP API in the SQL Server Client these threads are also responsible for making the BCP API calls to load the data. If you have multiple processors and disk arrays, setting this parameter to a high value will increase the update processing speed at the expense of additional memory. Avoid setting this parameter to 1, as this will effectively pass off all updates to the single worker thread, when executing them directly would be preferable.

It is recommended to always use multi-threaded updates, as it improves performance considerably.

> ✏️ **Note**
>
> This parameter requires the use of SQL Native Client in ODBC. The SQL Server driver doesn't support MARS, which is required for multithreaded updates. If MARS cannot be enabled, the Client automatically reverts to using single-threaded updates.

NULL_DATETIME_VALUE

Default: 19010101
Range: 17530101 to 99991231
Applies to: SQL Server Client
Console: PROCESSING > Date and Time (Null date values ... > Datetime)

Use this parameter to change the value used to represent a NULL date in a **datetime** column that does not allow nulls. For example, you could change the value to 18991231 if the default value of 190001001 is meaningful.

NULL_DATETIME2_VALUE

Default: 19010101
Range: 00010101 to 99991231
Applies to: SQL Server Client
Console: PROCESSING > Date and Time (Null date values ... > Datetime2)

Use this parameter to change the value used to represent a NULL date in a **datetime2** column that does not allow nulls. For example, you could change the value to 00010101 if the default value of 190001001 is meaningful.

NULL_DIGIT_VALUE

Default: 9
Range: 0 or 9
Related parameters: allow_nulls
Console: PROCESSING (Store NULL DMSII numbers as)

Use this parameter when your DMSII data contains NULL values that you do not wish to store as NULL. This parameter applies only to items that have the DAOPT_Nulls_Allowed bit reset in the `da_options` column of the corresponding DATAITEMS table entry.

- If you set `null_digit_value` to 0, all NULL values encountered in DMSII NUMBER data types get stored as zeros.
- If you set `null_digit_value` to 9, all NULL values encountered in DMSII NUMBER data types get stored as high values (999 or 999.999).

NUMERIC_DATE_FORMAT

Default: 23 (format mmddyyyy)
Range: Any legal numeric date format value (dms_subtype values 11--16, 21--26, 31--36)
Console: PROCESSING > Date and Time (Default formats > Numeric date)

The `numeric_date_format` parameter enables you to store DMSII dates as relational database numbers written out in the specified, allowable, DMSII numeric date format. To configure the `numeric_date_format`, you need to set the DMS_ITEMS Client control table DIOPT_Clone_as_Date bit and set the `sql_type` to 13, which represents a numeric date. The date is stored as an **int** data type in Microsoft SQL Server and a **number(10)** in Oracle.

This feature is useful in converting a DMSII MISER date or Linc date as a readable, numeric date. Note that the use of relational database **date** data type is a much better alternative.

PRESERVE_DELETES

Default: False
Range: True or False
Console: PROCESSING > Advanced (General > Preserve deleted records ...)

Setting this parameter to True causes records that contain an extended `update_type` column (type or bit 11) whose value is 2 (DELETE) to survive a re-clone of the data set. Instead of dropping the table, all non-deleted records are removed from the table during the re-clone.

This parameter has no effect on the handling of tables that have a non-DMSII column of type 10 (named `deleted_record` by default). Deleted records are unconditionally preserved when such tables are re-cloned.

ROLLBACK_SEGMENT

Default: NULL string
Range: rollback_segment_name
Applies to: Oracle Clients only
Console: **PROCESSING** (General)

This parameter makes the Client use the specified rollback segment by executing the SQL "SET TRANSACTION USE ROLLBACK SEGMENT `Rollback_segment_name` " at the start of every transaction.

SET_BLANKS_TO_NULL

Default: False
Range: True or False
Console: **CUSTOMIZING** (General > Set blank columns to NULL)

This parameter causes the Client to store zero-length character data (that is, "") as NULL instead of a single space. This parameter only applies to columns that are not part of the index.

SET_LINCDAY0_TO_NULL

Default: False
Range: True or False
Console: **PROCESSING > Date and Time** (Date parameters > Linc date base year ...)

This parameter causes the Client to treat a Linc date of 0 as NULL rather than 1/1 of the Linc base year.

SHOW_PERF_STATS

Default: True
Range: True or False
Console: **PROCESSING > Statistics** (Logging options > Show performance statistics)**

The `show_perf_stats` parameter enables the displaying and looging of performance statistics at the end of the data extraction phase when the AFN value changes (for example, when the processing of audit files is completed) and when the `process` or `clone` command terminates.

SHOW_STATISTICS

Default: True
Range: True or False
Related command-line option: `-v`
Related parameter: `statistics_increment`
Console: **PROCESSING > Statistics** (Logging options > Show statistics)

The `show_statistics` parameter, when set to True, causes the Databridge Client to display record count statistics at the intervals specified by the `statistics_increment` parameter. The statistics lines are useful in monitoring the progress of lengthy operations.

The `show_statistics` parameter applies to both the `process` and `clone` commands.

SHOW_TABLE_STATS

Default: True
Range: True or False
Console: **PROCESSING > Statistics** (Logging options > Show table statistics)

This parameter, when set to True, causes the Client to log the record counts for each cloned table at the end of the data extraction phase. During update processing it causes the Client log the update counts and average update times when the Client starts

processing a new audit file. These statistics are reset after being logged, as they are incremental statistics that span the last audit file processed. Tables that have no updates are omitted from these report.

SQL_EXEC_TIMEOUT

Default: 180,0
Range: 15-1200 for the first value, 0 or 30-3600 for the second value
Console: PROCESSING > Advanced (SQL execution timeout values)

The `sql_exec_timeout` parameter applies to update processing only. The first value allows the user to override the default setting of 180 seconds (3 minutes), which is used to determine when the timer thread should issue a WARNING about the query taking too long to complete. The optional second parameter, which defaults to 0 when omitted, allows the user to set the secondary timeout value for a long query after which time the query is aborted. A value of 0 disables this timeout. The value of the second parameter must be greater than that of the first parameter, except if it is 0.

SQL_HEART_BEAT

Default: False
Range: True or False
Console: PROCESSING > Advanced (Generate SQL heartbeats ...)s

This parameter was implemented as a work-around for the situation where long clones resulted in the Client's connections to the database getting closed because of long periods of inactivity. When this parameter is set to a non-zero value, the Client periodically executes a dummy SQL update on the Client connection to keep the connection alive during the data extraction where the only activity is on the bulk loader connection.

When using the Administrative Console, enabling the slider labeled "Generate SQL heartbeats ..." activates the edit box for the value for this parameter.

STATISTICS_INCREMENT

Default: 100000,10000
Range: 1--10000000 (10 million) for the first value, 1--1000000 (1 million) for the second value
Related command-line option: `-v`
Related parameter: `show_statistics`
Console: PROCESSING > Statistics (Record count display intervals)

The `statistics_increment` parameter applies when `show_statistics` is set to True *or* when the `-v` option is in effect. The `statistics_increment` parameter lets you set the display interval for record counts that occur during cloning and updating. For example, a setting of 1 indicates that the Databridge Client will display when *every* record is processed. A setting of 1000000 indicates that the Databridge Client will display a line after one million records have been processed.

Setting the `statistics_increment` parameter to a low number slows processing time, especially during cloning.

Enter a value using the following syntax:

```
statistics_increment = ccc[,uuu]
```

Where	Is
ccc	The record count before displaying the record statistics. This record count is used during cloning.
uuu	The record count before displaying the record statistics. This record count is used during updating.

STOP_AFTER_FIXUPS

Default: False
Range: True or False
Console: PROCESSING > Stop Conditions (Stop after fixup phase)

Setting this parameter to True causes the Client to stop as soon as all the tables are synchronized. This is a useful stopping point in a data warehousing environment, as the warehouse can be loaded at this point. It is also helpful if you want to validate data before declaring things to be in working order.

STOP_AFTER_GC_REORG

Default: False
Range: True or False
Console: **PROCESSING > Stop Conditions** (Stop after garbage collection reorganization)

Setting this parameter to True causes the Client to stop at the first quiet point after a garbage collection reorganization occurs. The program acts as if the operator issued a console `QUIT` command (or a SIGTERM signal in the case of UNIX) at the point when the garbage collection reorganization was detected. The Client exit status is 2034 if garbage collection or a file format reorganization is encountered in the audit trail.

STOP_AFTER_GIVEN_AFN

Default: False
Range: True or False
Applies to: Command-line Client (dbutility) only
Console: N/A

> ✏️ **Note**
>
> This parameter is specific to replication and applies only to the command-line Client. The command-line `-F` option, which allows you to specify the AFN after which to stop, overrides this parameter.

The `stop_after_given_afn` parameter enables you to stop processing after an externally specified audit file has been processed. Note that you must store the value of the audit file number in the `stop_afn` column of the DATASOURCES entry using data source tools external to dbutility. The `stop_after_given_afn` parameter forces the Client to check the values of the `stop_afn` column of the DATASOURCES table. If a non-zero value is found in this column, the Client sets the stop AFN value and stops reading the DATASOURCES table.

To automate this functionality using a script launched by the service, see Automate Client Operations with the Service.

STOP_ON_DBE_MODE_CHG

Default: False
Range: True or False
Console: **PROCESSING > Stop Conditions** (Stop on Enterprise Server audit file origin change)

Setting this parameter to True causes the Client to stop as soon as it detects that the Databridge Enterprise Server access mode changes from the value specified in the parameter `dbe_dflt_origin`. If this parameter is set to "direct" and Enterprise Server switches to "indirect", this will result in the Client stopping at the next quiet point.

SUPPRESS_DELETE_MSGS

Default: False
Range: True or False
Console: **PROCESSING > DMSII Data Error Handling** (Suppress messages for delete operations with bad keys)

Setting this parameter to True causes the Client to stop reporting data errors for DELETE operations that result in discards. The Client totally ignores such discards, as the target records cannot be in the relational database because they have keys with data errors, they are not included in the discard count and they are not written to the table's discard file.

TRACK_VFDS_NOLINKS

Default: True
Range: True or False
Console: CUSTOMIZING (DMSII related parameters > Track Variable Format datasets ...)

When set to True, this parameter causes the Client to track variable-format data sets that contain links; however, the links themselves are not tracked. When a record is created in a variable-format data set, links are set to null. If the application assigns the links to point to other records, the Client database will not contain these new link values until the variable-format data set is re-cloned. This parameter is selected, by default.

When this parameter is set to False, variable-format data sets are set to have a `ds_mode` value of 11 after initial cloning, which makes them ineligible for change tracking.

USE_DBWAIT

Default: False
Range: True or False
Related parameters: max_wait_secs, max_retry_secs
Console: PROCESSING (Audit unavailable action > Wait and retry)

Use this parameter to select the Databridge RPC to use during update processing, as follows:

- Set to False to use the DBREAD RPC. The DBREAD RPC returns an audit file unavailable status when all available audit files have been processed.

- Set to True to use the DBWAIT RPC. The DBWAIT RPC waits for an audit file to become available. *This is the required setting if the reading of the active audit file is enabled (* READ ACTIVE AUDIT *parameter in the Engine control file).*

The difference between the DBWAIT RPC and the DBREAD RPC is that DBWAIT waits for updates to become available rather than returning an audit file unavailable status.

This parameter applies only to the `process` command for updates. The Databridge Client ignores it for a `clone` command, which always uses the DBREAD RPC.

> ✎ **Note**
>
> You can temporarily toggle this parameter by using the `-w` command-line option.

USE_LATEST_SI

Default: False
Range: True or False
Console: PROCESSING > Engine and Enterprise Server (General > Include latest StateInfo ...)

If the `use_latest_si` parameter is set to True, the Client will request that the server include the latest StateInfo in all the data records sent during audit file processing. The overhead of doing this is 24 bytes per record. This parameter is mainly intended as a debugging tool when chasing audit file processing problems. In addition to making the Client print up-to-date audit locations instead of the audit location of the last quiet point that was used as a COMMIT, this option may be useful when you use the audit timestamp as an external column for data tables. Enabling this parameter will make the values used in such columns much more accurate. Not all DMSII audit file records have an associated timestamp, so the timestamp will still not be 100% accurate.

Server Option Parameters

The following parameters are included in the [params] section of the Databridge Client configuration file. The parameters listed in this section affect how the Databridge Client processed updates.

SHUTDOWN

Console: N/A

This parameter applies only to the command-line Client (**dbutility**). It inhibits update processing for a given period of time after a LIMIT_NAME or LIMIT_TIME condition (normally initiated by a STOP parameter) is encountered. The format of the shutdown parameter is as follows:

```
shutdown {until | for} hh:mm after stop
```

The first form specifies the time of day at which the shutdown period ends, while the second form specifies the length of the shutdown period. The command-line option `-o` can override this parameter.

STOP

Console: PROCESSING > Stop Conditions (Dynamic stop conditions)

This parameter allows you to specify a condition for the Databridge Engine to stop processing updates as follows:

```
stop {before | after} {task "name" | time hh:mm[:ss]}
```

For example, you would enter the following:

```
stop before task "name"
- or-
stop after time 12:30:15
```

Generally, you should include only one stop specification in the configuration, but using two stop specifications is legal. When more than one task or one time is specified in the configuration file, the program honors only the last one. However, when a task specification is coupled with a time specification, the Client honors the task specification only if it occurs on the date specified in the time specification.

Generate Command Parameters

The `generate` command parameters include `decimal_aa_length` (Oracle Clients only) and a number of SQL statement suffixes (Oracle and SQL Server Clients).

Parameter	Description
`decimal_aa_length`	**Default:** 15 **Range:** 15 – 38 **Applies to:** Oracle Client Use this parameter to control the size of the data type that represents a decimal AA Value—by default, this is NUMBER(15). If you set this parameter to 16, the Client will use NUMBER(16) for all decimal AA Values.

SQL STATEMENT SUFFIXES

Console: CUSTOMIZING > SQL Suffixes ({Table | Index} SQL suffixes ...)

The following parameters determine which extra clauses are added to the create table and create index SQL statements in the scripts generated by the Databridge Client.

Suffixes must be entered on a single line and be enclosed in double quotation marks. Suffixes can be up to 256 characters in length.

Parameter	Description

create_index_suffix	**Default**: None
	Range: "suffix"
	Applies to: Oracle and SQL Server Clients
	
	The `create_index_suffix` parameter enables you to define extra attributes (a suffix) for create index SQL statements that the Client generates for any given table. Each attribute list is defined with a number or index n so you can reference it. Up to 100 different suffixes can be defined. Individual indexes can select one of the suffixes by specifying this value in the `index_suffix` column of the corresponding DATATABLES Client control table entry. The index suffix is then concatenated to all create index SQL statements for this table.
	
	Here's an example suffix for a SQL Server database which specifies file groups for create index statements:
	
	`create_index_suffix [1]="ON filegroup"`
	
	Here's an example suffix for an Oracle database:
	
	`create_index_suffix [1]="TABLESPACE name STORAGE MINEXTENTS 1 NEXT 10 MAXEXTENTS UNLIMITED"`

create_table_suffix	**Default**: None
	Range: "suffix"
	
	The `create_table_suffix` parameter enables you to define a suffix for create table SQL statements that the Client generates for any given table and to assign a number to this suffix so you can reference it. The index n allows for up to 100 different suffixes to be defined. Individual tables can select one of the suffixes by specifying this value in the `create_suffix` column of the corresponding DATATABLES Client control table entry. The table suffix is then concatenated to all create table SQL statements that specify the given suffix number.
	
	Here's an example suffix for an SQL Server database which specifies filegroups for create table statements:
	
	`create_table_suffix [1]="ON filegroup"`
	
	Here's an example suffix for an Oracle database:
	
	`create_table_suffix [1]="TABLESPACE tablename"`

global_index_suffix	**Default**: None **Range**: "suffix" **Applies to**: Oracle and SQL Server Clients The `global_index_suffix` parameter enables you to add a filegroup (SQL Server) or a tablespace (Oracle) or any other SQL command specification to all create index SQL statements that the Client generates except those that have a suffix associated with the create_index_suffix parameter.

Parameter	Description
global_table_suffix	**Default:** None **Range:** "suffix" The **global_table_suffix** parameter allows you to add a filegroup (SQL Server) or a tablespace (Oracle) or any other SQL command specification to all the create table SQL statements that the Client generates, except for statements whose suffix is associated with the create_table_suffix parameter.
user_column_suffix	**Default:** None **Range:** "suffix" The `user_column_suffix` parameter allows you to add a suffix to the column definition created by the generate command for external columns of type user_column1 through user_column4. This is particularly useful for adding default clauses.

In the case of index suffixes (both global and specific) for the Oracle Client, you can use the string $(INDEX_NAME) as an environment variable that the Client replaces by the actual index name for the table when using the suffix. You can also insert new line characters into the suffix by using "\n"; this is sometimes necessary when the suffix contains a SQL statement that must be executed separately after the index creation completes. An example for this is enabling parallel mode for index creations, which speeds up the index creation significantly. You can use the following index suffix to do this:

```
"parallel (degree 8)\n/***/\nalter index $(INDEX_NAME) parallel 1"
```

Once the index is created, the alter index statement sets the parallel degree back to 1, it needs the index name to be able to do this, using the $(INDEX_NAME) environment variable makes this possible without having to write separate scripts for each table. The /***/ is inserted into the SQL suffix to force the Client to execute the create index statement before executing the alter index statement. Using a semicolon causes an OCI error. Inserting "\n/***/\n" makes the Client break up the line into two separately executed SQL statements.

DATA MASKING STRINGS

Applies to: SQL Server Client (using SQL Server 2016 or newer version)
Console: CUSTOMIZING > SQL Suffixes (Data Masks)

These strings provide the data for the arguments of the random() and partial() data masking functions. They do not include the parentheses. See the `masking_parameter` in the [params] section for details on how to setup data masking for columns with sensitive data.

Display Command Parameters

The following parameter is included in the [params] section of the Databridge Client configuration file. It affects the `display` command only.

When using the Administrative Console this parameter can be found in the **PROCESSING** page of the Client Configuration property pages.

---

display_active_only	**Default:** True
	Range: True or False
	Related command-line option: `-a`
	Console: PROCESSING (General)

Use the `display_active_only` parameter to affect the display command, as follows:

- Set `display_active_only` to True to show only the Client control table entries for data sets whose `active` column is 1. This is particularly useful if your site clones a small number of data sets.
- Set `display_active_only` to False to show all Client control table entries, regardless of the data set `active` column setting.

You can temporarily override this parameter by using the `-a` command-line option.

User Scripts Parameters

The following parameters are included in the [params] section of the Databridge Client configuration file. The parameters listed in this section affect what the Databridge Client does with user scripts.

When using the Adminstrative Console these parameters can be found in the **CUSTOMIZING** page of the Client Configuration property pages.

> ✏ **Note**
>
> We highly recommend that you set these parameters. As long as you have made sure that each user script includes all of the changes for the specified data set, the user scripts ensure that the Databridge Client can handle DMSII reorganization changes.

---

check_user_scripts	**Default:** False
	Range: True or False

Set this parameter to True to let the Databridge Client inform you if a user script for a table is missing. In this case, the Databridge Client returns the following message:

`ERROR: Unable to open script file filename`

This parameter is especially useful if you have created data table creation user scripts and index creation user scripts for every table in your relational database. The Databridge Client runs these scripts immediately after it completes its own scripts for creating tables and table indexes.

NOTE: This parameter does not apply to data set selection user scripts and data table customization scripts.

[Scheduling]

The Scheduling parameters section only applies to the dbutility `process` command. You must run the `process` command once before the scheduling takes effect. For more information, see Scheduling dbutility Updates.

To schedule Client runs that are initiated from the Administrative Console, click **PROCESSING** > **Scheduling** to open the Client Configuration property pages and set these parameters. For more information, see the Databridge Administrative Console Help.

Parameter	Description
blackout_period	**Default:** `00:00, 00:00` **Range:** `00:00` to `24:00` (The two time values cannot be equal.) Use this parameter to specify a fixed block of time during which the Client cannot run. This parameter is useful for operations, such as database backups, that can only take place when the Client is inactive. For example, if you want to back up the database daily between 1:00 a.m, and 2:30 a.m. daily, define a blackout period from 0:55 to 2:30. The extra 5 minutes ensures that the Client finishes any long transactions before the database backup begins. If the Client is running when the blackout period starts, the Client automatically stops. If the Client is waiting for an idle host to send it updates when the blackout period starts, the Client resets the TCP/IP connection and aborts the run if it hasn't received any updates after 15 seconds. If you try to run the Client during a blackout period, nothing happens. During a blackout period the service will not start the Client. If the scheduler tries to schedule a **DBClient** run at a time that falls within a blackout period, the start of the run will be delayed until the blackout period ends. When this parameter is updated using the Administrative Console, it is set to the same value in both the service and Client configuration files.
daily	**Default:** `daily = 08:00, 12:00, 17:00, 24:00` **Range:** 12 entries in ascending order from `00:00` to `24:00` **NOTE:** The `daily` parameter is mutually exclusive with the `sched_delay_secs` parameter. If you specify both `daily` and `sched_delay_secs` in the [scheduling] section of the configuration file, `sched_delay_secs` overrides `daily` regardless of the order in which they are specified. *Enter the times you want the dbutility** `process` command to wake up and gather updates from the DMSII database. You must specify 24-hour time (for example, 5:00 for 5:00 a.m. and 17:00 for 5:00 p.m.). The range for minutes is 00–59. You can specify up to 12 times for the daily parameter. However, you must specify the times in ascending order. Note the following: • The values 00:00 and 24:00 are equivalent for midnight. • 24:00 is allowed only so that you can put it at the end of the list of times in ascending order. • 24:01 is not allowed; instead, specify, 00:01.
exit_on_error	**Default:** True **Range:** True or False The `exit_on_error` parameter indicates that the scheduling should be terminated if an error occurs. If this parameter is set to false, the process command is retried at the next scheduled time.

Parameter	Description
sched_delay_secs	**Default:** 0 **Range:** 0–86,400 seconds (24 hours) **NOTE:** The `sched_delay_secs` parameter is mutually exclusive with the daily parameter. If you specify both daily and fixed_delay in the [scheduling] section of the configuration file, `fixed_delay` *overrides daily regardless of the order in which they are specified*. Use the `sched_delay_secs` parameter to specify a time delay between successive executions of the process command. The `sched_delay_secs` parameter does use the `retry_time` parameter. To disable the `sched_delay_secs` parameter, comment it out or set its value to 0.
sched_minwait_secs	**Default:** 0 **Range:** 0–86,400 seconds (24 hours) This parameter ensures that next scheduled process command is delayed by the specified interval and doesn't occur too soon after the current scheduled time.
sched_retry_secs	**Default:** 3600 seconds (1 hour) **Range:** 0–86,400 seconds (24 hours) The `sched_retry_time` parameter only applies after a failed process command. A value of 0 means that **dbutility** schedules the next run at the next regularly scheduled time without any retries. For example, if the mainframe is down when **dbutility** attempts to run a `process` command using the scheduling option, **dbutility** will retry the operation after the specified amount of time has elapsed. If the retry time value is larger than the next scheduled time, **dbutility** retries at the next scheduled time.

[EbcdictoAscii]

Use the [EbcdictoAscii] section of the configuration file to customize character translation tables.

When using the Administrative Console, you can customize the translation table by clicking **CUSTOMIZING** > **Translations** to open this section of the Client Configuration property pages.

> ✏️ **Note**
>
> If you plan to customize character translation tables, you must modify the configuration file before you run dbutility `process` or dbutility `clone` to populate the Databridge data tables in the relational database. In addition, if you customize the character translation tables when you populate the data tables the first time, you must use them on all subsequent updates. If you don't, the data will be invalid.

TRANSLATION TABLE

The Databridge Client uses the ISO standard translation tables to translate EBCDIC data received from the host to ASCII data. You can adjust the translation tables to work with national character sets, which typically redefine characters such as { } [ ] | to represent national characters.

REDEFINING A CHARACTER

To redefine a character, alter the EBCDIC to ASCII translation table by entering the pair of numeric values representing the EBCDIC character code and the corresponding ASCII character code in the [EbcdictoAscii] section of the configuration file. You can use decimal or hexadecimal (for example, 124 for decimal or 0x7C for hexadecimal) to represent the EBCDIC and ASCII character codes.

The Databridge Client does not allow you to change the values of characters that are constant across national characters, including the space, hyphen (-), single quote (\'), digits 0--9, and the letters of the alphabet (A--Z and a--z). Changing any of these characters causes an error unless you set the `restrict_translation` parameter appropriately.

Example

The following example shows EBCDIC to ASCII translation using hexadecimal characters. Note that this file is for example only; it does not represent any national character set.

```
;hexadecimal format
[EbcdictoAscii]
0x7C = 0x7E      ; remapping of @ to ~
0xE0 = 0x7C      ; remapping of \ to |
0xC0 = 0x5B      ; remapping of { to [
0xD0 = 0x5D      ; remapping of } to ]
```

EXTERNAL DATA TRANSLATION DLL SUPPORT

The following parameters are included in the [params] section of the Databridge Client configuration file.

When using the Administrative Console, you can change the translation DLL name by clicking **CUSTOMIZING > Translations** to open this section of the Client Configuration property pages.

Parameter	Description
eatran_dll_name	**Default:** "DBEATRAN.DLL" **Range:** "dllname" **NOTE:** You must include quotation marks around the filename. The parameter `eatran_dll_name` allows you to rename the external translation file DBEATRAN.DLL.
use_ext_translation	**Default:** False **Range:** True or False The `use_ext_translation` parameter enables you to translate 16-bit character sets from EBCDIC to ASCII. When this parameter is enabled, the Databridge Client accesses an alternate data translation routine that uses an external DLL, named `DBEATRAN.DLL` (`dbeatran.so` for UNIX), instead of the standard translation procedure (for example [EbcdictoAscii] Section). The `DBEATRAN.DLL` contains the EBCDIC_to_ASCII entry point. This DLL is dynamically loaded when you enable the `use_ext_translation` option.

[DBConfig]

This section contains parameters that are related to the Administrative Console's **Customize** command.

`default_date_fmt`

Default: 21
Range: 1-296
Console: PROCESSING > Date and Time Parameters (Default date formats)

This parameter specifies the default format for numeric columns that are clones as dates. For a MISER database this should be set to 1.

`global_type0_chnages`

Default: True
Range: True or False
Console: CUSTOMIZING (Customizing General)

The version 7.0 **DBCIntCfgServer** program together with the Administrative Console applies all customizations done to the fix part of a variable format data set to all the records types, as they all contain the exact same fixed part. This option is provided as a safeguard for the unlikely situation where users do not want to do this. Some sites have variable format data sets that have a large number of record types, if you are customizing a date in the fixed part using the Administrative Console's **Customize** command you only have to do this once and it gets applied to all the records types. The only reason we did not implement this parameter in the Administrative Console is that it is highly unlikely that anyone will want to change it.

[Encryption]

This section contains parameters that are related to the data encryption in Client/Server communications using SSL/TLS.

`enable_encryption`

Default: False
Range: True or False
Console: Encryption (Enable ENCRYPTION ...)

Enabling this parameter is the first step towards using SSL/TLS encryption between the Client and DBServer. See the section on setting up encryption in the Install Guide to find out more about how to do this. If using the Administrative Console simply more the slider the on position, This will make the next two parameters visible.

`ca_file`

Default: ""
Range: String
Console: ENCRYPTION (CA file)

This parameter is a full file specification for the file that contains the certificate to be used.

`ca_path`

Default: ""
Range: String
Console: ENCRYPTION (CA path)

This parameter is the path of a directory that contains the bundle of certificate to be used.

`certify_server_name`

Default: False
Range: True or False
Console: ENCRYPTION (Check server name in certificate)

This parameter indicates whether or not the server certificate will be checked to determine if the server is the node we think we are connecting to.

`tls_host_name`

Default: ""
Range: String
Console: N/A

This parameter is intended to specify the server name to check for when the `certify_server_name` parameter is set to True. The code does not currently use this parameter.

16.3.11 Reference Tables

The following reference tables show all of the configuration file parameters, and *as applicable*, their associated environment variables, command-line options, and **dbutility** commands. Additionally, the tables show relationships between configuration file parameters that work together.

Because these tables do not explain each configuration file parameter, environment variable, and command-line option in detail, we recommend that you use it for reference only.

Bulk Loader Parameters

The following parameters from the [Bulk_Loader] section of the Databridge Client configuration file apply only to the dbutility `clone` and `process` commands and have no associated command-line options.

[Bulk_Loader] Parameter	Bulk Loader Utility
bcp_batch_size	SQL Server
bcp_code_page	Oracle and SQL Server (**bcp** only)
bcp_copied_msg	SQL Server (**bcp** only)
bcp_delim	SQL Server (**bcp** only)
bcp_packet_size	SQL Server (**bcp** only)
enable_parallel_mode	Oracle
inhibit_direct_mode	Oracle
max_bcp_failures	SQL Server and Oracle
max_temp_storage	SQL Server (**bcp** only) and Oracle (Windows only)
sqlld_bindsize	Oracle
sqlld_rows	Oracle
verify_bulk_load	All

Scheduling Parameters

The following [Scheduling] parameters from the Databridge Client configuration file have no associated command-line parameter, and they apply to the `process` command only when using the command-line Client (dbutility):

- `daily`
- `exit_on_error`
- `sched_delay_secs`
- `sched_retry_secs`

EBCDIC to ASCII Parameters

EBCDIC to ASCII translation applies only to the `clone` and `process` commands and has no associated command-line options.

Params Parameters

The following parameters are from the [params] section of the Databridge Client configuration file:

[params] Parameter	Option	dbutility Command	Notes
allow_nulls		define and redefine	
alpha_error_cutoff		clone and process	
auto__mask_columns	define , generate , process , redefine and clone	SQL Server 2016 and newer	
automate_virtuals		clone and process	
aux_stmts		clone and process	This parameter applies to Oracle and SQL Server ODBC Clients only.
batch_job_period		clone and process	
bracket_tabnames		clone and process	SQL Server only
century_break		clone and process	
check_user_scripts		clone and process	
clr_dup_extr_recs		generate	
commit_absn_inc		clone and process	
commit_idle_database		clone and process	
commit_longtrans		clone and process	
commit_time_inc		clone and process	
commit_txn_inc		clone and process	
commit_update_inc		clone and process	
controlled_execution	-o	clone and process	
convert_ctrl_char		clone and process	
correct_bad_days		clone and process	
create_index_suffix [n]		generate	
create_table_suffix [n]		generate	
dbe_dflt_origin		clone and process	
decimal_aa_length		define and redefine	Oracle only
default_user_columns		define and redefine	
defer_fixup_phase	-c	clone	Toggle
dflt_history_columns		define and redefine	
discard_data_errors		clone and process	
display_active_only	-a	display	Override
display_bad_data		clone and process	
eatran_dll_name		clone and process	
enable_af_stats		clone and process	
enable_dms_links		define , redefine , process and clone	

[params] Parameter	Option	dbutility Command	Notes
enable_doc_records		clone and process	
enable_dynamic_hist		redefine	
enable_ff_padding		clone and process	
enable_minimized_col		clone and process	
enable_optimized_sql	-N	clone and process	Toggle
engine_workers		clone and process	
error_display_limits		clone and process	
external_column [n]		define and redefine	
extract_embedded		define , redefine , process and clone	
flatten_all_occurs		define and redefine	
force_aa_only		define and redefine	
global_index_suffix		generate	
global_table_suffix		generate	
history_tables			
inhibit_8_bit_data		clone and process	
inhibit_console	-C	clone and process	Toggle
inhibit_ctrl_chars		clone and process	
inhibit_drop_history		clone and process	
inhibit_init_values		redefine	
inhibit_required_opt		define and redefine	
linc_century_base		clone and process	
masking_parameter[n]		generate	
max_clone_count	-s	clone and process	
max_discards		clone and process	
max_retry_secs		process	Requires use_dbwait and works with max_wait_secs
max_srv_idle_time		clone and process	
max_wait_secs		process	Requires use_dbwait and works with max_retry_secs
maximum_columns		define and redefine	
min_check_time		clone and process	
min_varchar		define and redefine	
minimize_col_updates		define and redefine	
miser_database		define , redefine , process and clone	
months		clone and process	

[params] Parameter	Option	dbutility Command	Notes
null_datetime_value		clone and process	
null_datetime2_value		clone and process	
null_digit_value		clone and process	
numeric_date_format		clone and process	
n_dmsii_buffers		clone and process	
n_update_threads		clone and process	
optimize_updates		define and redefine	
preserve_deletes		clone and process	
read_null_records		define and redefine	
reorg_batch_size		redefine and reorganize	
rollback_segment		All	Oracle only
sec_tab_column_mask		define and redefine	Requires default_user_columns
set_blanks_to_null		clone and process	
set_lincday0_to_null		clone and process	
show_perf_stats		clone and process	
show_statistics	-v	clone and process	Works with statistics_increment
show_table_stats		clone and process	
shutdown	-o		Override
split_varfmt_dataset		define and redefine	
sql_exec_timeout		clone and process	
sql_heart_beat		clone and process	
statistics_increment	-v	clone and process	Works with show_statistics
stop			
stop_after_fixups		clone and process	
stop_after_gc_reorg		clone and process	
stop_after_given_afn		clone and process	
strip_ds_prefixes		define and redefine	
suppress_dup_warnings		clone and process	
suppress_new_columns		redefine	
suppress_new_datasets		redefine	
use_binary_aa		define and redefine	
use_bigint		define and redefine	SQL Server only
use_clob		define and redefine	Oracle only

[params] Parameter	Option	dbutility Command	Notes
use_clustered_index		define and redefine	This parameter applies to SQL Server. See use_decimal_aa.
use_column_prefixes		define and redefine	The tab_name_prefix column of the DATASOURCES Client control table must contain an entry.
use_date		define and redefine	SQL Server only
use_datetime2		define and redefine	SQL Server only
use_dbwait	-w	process	Toggle Works with max_wait_secs and max_retry_secs
use_decimal_aa		define	
use_ext_translation		clone and process	This parameter applies to Windows.
use_internal_clone		redefine and reorg	
use_latest_si		clone and process	
use_nullable_dates		define and redefine	This parameter applies only to MISER databases.
use_primary_key		define	
use_stored_procs		define , redefine , process and clone	
use_time		define and redefine	SQL Server only
use_varchar		define and redefine	
user_script_dir	-n	define , redefine , process and clone	Override
user_column_suffix[n]		generate	

16.4 Appendix D: Customization Scripts

This appendix is intended as a quick reference for writing user scripts. For more information about user scripts, see Customizing with User Scripts.

The user scripts described in this Appendix differ significantly from program-generated user scripts (that is, user scripts created by the `Create Scripts` command in the Administrative Console or the **dbutility** `createscripts` command). Program-generated user scripts set additional bits in the control tables. These bits allow the `redefine` command and the Administrative Console's **Customize** command to restore changes to the Client control tables. The code in **DBClntCfgServer** that support the Customize command is fully compatible with the `redefine` command.

If you use the Administrative Console's **Customize** command and want the ability to restore the Client control tables, you'll need to set some additional fields whenever you make a change.

16.4.1 Customization Rules for Client Configurator

All of the Client control tables except DATASOURCES have a column named *xx*_user_bmask (where *xx* is "ds", "di", "dt" or "da", depending on the table where it resides). This column, which parallels *xx*_options, is used to indicate whether the bits were changed by the user script or by the Client Configurator. Additionally, some of the bits in the *xx*_options columns are set by the Client or are set by changing an item to a special Client data type, such as a date.

The `redefine` command, when run in the Administrative Console's **Customize** command mode (`use_dbconfig` = True), will restore the bits in *xx*_options that are referenced by *xx*_user_bmask, while leaving the remaining bits unchanged. Several bits in *xx*_options that were previously unused are now used to indicate that a specific field in the record was modified by a user script or the Administrative Console's **Customize** command.

Parameters that affect ds_options

The global parameters that affect `ds_options` settings are as follows:

`history_tables = { 0 | 1 | 2}`	0 - No history tables will be created. 1 - Creates history tables for all data sets. The bit DSOPT_Save_Updates (8) is automatically set for all data set table entries. (If you used a data set user script to do this, remove it and set `history_tables` to 1 in the Client configuration file using either the Administrative Console's **Configure** command or the editor. If you use binary configuration files, you must export the file before editing the file. See Export or Import a Configuration File. 2 - The same as a value of 1, except that it also sets the bit DSOPT_History_Only (0x2000 or decimal 8192).
`clr_dup_extr_recs = {true | false}`	Defines the initial value of the new `ds_options` bit DSOPT_Clrdup_Recs (0x8000 or decimal 32768). This parameter is no longer checked by the `process` and `clone` commands, which only look at the `ds_option` bit.
`split_varfmt_dataset = {true | false}`	Defines the initial value of the new `ds_options` bit DSOPT_Split_Vfmt_ds (0x10000 or decimal 65536). It makes the Client treat variable format data sets in a slightly different manner by putting all the fixed parts of records in the table normally used for type 0 records. The fixed parts of records in all other tables are not included, except for the items that are keys.
`force_aa_value_only = {0 | 1 | 2}`	Defines the initial value of the `ds_options` bit DSOPT_Use_AA_Only, which forces the data set to use AA Values or RSNs as keys if the data set has a valid AA Value or RSN. RSNs always take precedence over AA Values unless an embedded data set or a DMSII link is involved. A value of zero sets the bit to 0 for all data sets. A value of 1 sets the bit to 1 for all data sets that have a valid AA Value or an RSN. A value of 2 sets the bit 1 for data sets that have an RSN.

> ✏️ **Note**
>
> Any time you explicitly change the value of a bit in `ds_options`, you must set the corresponding bit in `ds_user_bmask`. If you set a bit that had a default value of 1 to 0, you must set the corresponding bit in `ds_user_bmask` to 1 to indicate that the value of this bit should be preserved by the `redefine` command.
>
> Be aware that some bits in `ds_options` may already be set. For SQL Server, use the \"|\" operator. For Oracle, use the BITOR function with the BITAND function to perform logical OR and logical And functions. For best results, avoid directly setting `ds_options` or using the + operator. The following example uses the BITOR function when updating the `ds_options` column of DATASETS to set the bit DSOPT_Select_Only (64) while leaving the rest of the bits intact:
>
> ```
> ds_options=BITOR(ds_options,64)
> ```
>
> When using the Client Configurator, if you change the value of `external_columns` for a single data set, you must also set the new bit DSOPT_ExtCols_Set (0x2000 or decimal 131072) in both `ds_option`s and `ds_user_bmask`. This ensures that the Administrative Console's **Customize** command retains the change.

Sample script for setting a ds_options bit in DATASETS

This script sets the `ds_options` bit DSOPT_Ignore_Dups (32) for the data set SVHIST without changing any of the other bits in the column. We provide both a SQL Server version and Oracle version of this script.

Filename: `script.user_layout.svhist`:

SQL Server version:

```
update DATASETS set ds_options = ds_options | 32
where dataset_name = 'SVHIST'
```

Oracle version:

```
update DATASETS set ds_options = BITOR(ds_options, 32)
where dataset_name = 'SVHIST'
```

16.4.2 Changes By Table

DATAITEMS Control Table Changes

Besides the addition of the column `da_user_bmask`, several `da_options` bits are used to indicate that a specific field in the record was changed by the Client Configurator or a user script. These new `da_options` bits are described in the following table.

> ✏️ **Note**
>
> Any time you explicitly change the value of a bit in `da_options`, you must set the corresponding bit in `da_user_bmask`. If you set a bit that had a default value of 1 to 0, you must set the corresponding bit in `da_user_bmask` to 1 to indicate that the value of this bit should be preserved by the `redefine` command.

`DAOPT_Column_Renamed` (2)	This bit indicates that the column was renamed by changing the `item_name` column of the item. The `redefine` command uses this bit to determine if the `item_name` value should be preserved.
`DAOPT_Type_Changed(4)`	This bit indicates that the column's data type was changed by changing the value in the `sql_type` column. The `redefine` command uses this bit to determine if the `sql_type` value should be preserved.
`DAOPT_Length_Changed` (8)	This bit indicates that the column's data type length specification was changed by changing the value in the `sql_length` column. The `redefine` command uses this bit to determine if the `sql_length` value should be preserved.
`DAOPT_Scale_Changed` (16)	This bit indicates that the column's data type scale was changed by changing the value in the `sql_scale` column. The `redefine` command uses this bit to determine if the `sql_scale` value should be preserved.
`DAOPT_User_Column (32)`	This bit indicates that the column was added by the user. The `redefine` command uses this bit to determine if the column should be preserved.
`DAOPT_Item_Renumbered` (128)	This bit indicates that the column was renumbered by the user. The `redefine` command uses this bit to determine if the `item_number` should be preserved.
	CAUTION: This will not always work because item numbers may change as a result of a DMSII reorganization. If you do this, you'll need to use Administrative Console's **Customize** command to get the column into the proper place.

DATASETS Control Table Changes

Besides the addition of the column `ds_user_bmask`, some `ds_options` bits are used to indicate that a specific field in the record was changed by the Client Configurator or a user script. These new `ds_options bits` are described in the following table.

> ✏️ **Note**
>
> If you explicitly change the value of a bit in `ds_options`, you must set the corresponding bit in `ds_user_bmask`. If you set a bit that has a default value of 1 to 0, you must set the corresponding bit in `ds_user_bmask` to 1 to indicate that the value of this bit should be preserved by the `redefine` command.

DSOPT_SetNameChange (262144)	This bit must be set for any data set whose `set_name` column is modified by the Client Configurator or a user script. The `redefine` command uses this bit to determine if the value of the `set_name` should be preserved.

DATATABLES Control Table Changes

Besides the addition of the column `dt_user_bmask`, several `dt_options` bits are used to indicate that a specific field in the record was changed by Client Configurator or a user script. These new `dt_options` bits are described in the following table.

> ✏️ **Note**
>
> If you explicitly change the value of a bit in `dt_options`, you must set the corresponding bit in `dt_user_bmask`. If you set a bit that had a default value of 1 to 0, you must set the corresponding bit in `dt_user_bmask` to 1 to indicate that the value of this bit should be preserved by the `redefine` command.

DSOPT_SetNameChange (262144)	This bit must be set for any data set whose `set_name` column is modified by the Client Configurator or a user script. The redefine command uses this bit to determine if the value of the `set_name` should be preserved.

DMS_ITEMS Control Table Changes

Besides the addition of the column `di_user_bmask`, several `di_options` bits are used to indicate that a specific field in the record was changed by Client Configurator or a user script. These new `di_options` bits are described in the following table.

> ✏️ **Note**
>
> If you explicitly change the value of a bit in `di_options`, you must also set the corresponding bit in `di_user_bmask`. If you set a bit that has a default value of 1 to 0, you must set the corresponding bit in `di_user_bmask` to 1 to indicate that the value of this bit should be preserved by the `redefine` command.

DTOPT_Table_Renamed (1)	This bit indicates that the table was renamed by changing the `table_name` column of the `item_name` columns and all the DATAITEMS that belong to the table. The `redefine` command uses this bit to determine if the `table_name` value should be preserved.
DTOPT_Index_Renamed (2)	This bit indicates that the index was renamed by changing the `index_name` column of the table. The `redefine` command uses this bit to determine if the `index_name` value should be preserved.
DTOPT_User_Table (4)	This bit indicates that the table was created by the user. The `redefine` command uses this bit to determine if the `index_name` value should be preserved. (This bit is not fully implemented)

16.4.3 Sample Scripts for Customizing Data Set Mapping

This section is intended as a quick reference for writing data set mapping customization user scripts. Therefore, it lists sample scripts without background explanation. If you are unfamiliar with the Databridge Client, refer to the indicated sections for more information.

Sample Data Set Global Mapping Customization Script

The following example updates the `dms_subtype` value for every occurrence of the time value TS in the DMSII database whose data source name is CMDB. Create only one of these scripts for each data source.

File name: `script.user_datasets.cmdb`

```
update DMS_ITEMS set dms_subtype = 6
where dms_item_name = 'TS'
```

For more information about the `dms_subtype` column of the DMS_ITEMS Client control table, see DMS_ITEMS Client Control Table

---

Sample Data Set Selection Script

This script selects the data sets that we want to clone. Following is a sample user script for a DMSII customer database whose data source name is CMDB. This script turns cloning off (by setting the `active` column value to 0) for two data sets. We used the data set global customization script rather than the scripts for individual data sets in this example.

File name: `script.user_datasets.cmdb`

```
update DATASETS set active = 0
where data_source = 'CMDB'
/***/
update DATASETS set active = 0
where dataset_name = 'EMPLOYEE' and data_source='CMDB'
/***/
update DATASETS set active = 0
where dataset_name = 'CUSTOMER' and data_source='CMDB'
/***/
update DATASETS set active = 0
where dataset_name = 'INVENTORY' and data_source='CMDB'
/***/
update DATASETS set active = 0
where dataset_name = 'BILLING' and data_source='CMDB'
```

For a complete explanation of specifying data sets for cloning, see Tips for Efficient Cloning.

Selecting DMSII Items

The following script disables the cloning of two DMSII items in the data set named ORDER by setting the value of the active column to 0 in the corresponding DMS_ITEMS table entries.

File name: `script.user_layout.order`

```
update DMS_ITEMS set active=0
where dms_item_name = 'SPECIAL-ORDER-DATE' or
      dms_item_name = 'SPECIAL-ORDER-AMOUNT'
      and dataset_name = 'ORDER'
```

Multiple data sets can contain items with the same name. Adding the data set name to the WHERE clause ensures that you update only the items in question.

For more information, see Tips for Efficient Cloning.

Cloning a Numeric Field as a Date

The following script causes the `define` command to map a DMSII item of type NUMBER(8) to a relational database date data type where the number contains a date in the *mmddyyyy* format.

File name: `script.user_layout.payments`

```
update DMS_ITEMS set dms_subtype=23,di_options=2
where dms_item_name = 'PAYMENT-DATE' and dataset_name='PAYMENTS'
```

Cloning an Alpha Field as a Date

The following script causes the `define` command to map three DMSII items of type ALPHA(10) to a relational database date data type, where those items contain a date in the *mm/dd/yyyy* format.

File name: `script.user_layout.order`

```
update DMS_ITEMS set dms_subtype=53,di_options=2
where dms_item_name = 'ORDER-DATE' or
      dms_item_name = 'DUE-DATE' or
      dms_item_name = 'DATE-SENT'
      and dataset_name = 'ORDER'
```

Cloning an Alpha or Number Field as a Time

The following script causes the `define` command to map a DMSII ALPHA or NUMBER time item as a relational database time item.

File name: `script.user_layout.payment`

```
update DMS_ITEMS set di_options=256, dms_subtype=3
where dms_item_name='TIME11' and dataset_name = 'BILLING'
```

Cloning an Alpha or Number Field as a Date/Time

The following script causes the `define` command to map a DMSII ALPHA or NUMBER date/time item as a relational database date/time item.

File name: `script.user_layout.payment`

```
update DMS_ITEMS set di_options=128, dms_subtype=121
where dms_item_name='PAY_REC_TIME' and dataset_name = 'PAYMENTS'
```

Flattening OCCURS Clause

The following script causes the `define` command to map an item with an OCCURS clause as a series of columns in the corresponding relational database table instead of mapping each occurrence of the items to a separate column in an OCCURS (secondary) table.

File name: `script.user_layout.billing`

```
update DMS_ITEMS set di_options=1
where dms_item_name = 'MONTHLY-BILLS' and dataset_name='BILLING'
```

For details see Flattening OCCURS Clauses.

Flattening OCCURS Clause for Item Cloned as Dates

The following script directs the `define` command to map an item with an OCCURS clause as a series of columns, whose data type is a relational database date type, in the corresponding primary table. Furthermore, it specifies that the DMSII item, which is of type NUMBER(8), contains a date in the *mmddyyyy* format.

File name: `script.user_layout.billing`

```
update DMS_ITEMS set di_options=3, dms_subtype=23
where dms_item_name = 'BILLING-DATES' and dataset_name = 'BILLING'
```

Flattening OCCURS Clause for Three Bit Numeric Flags

MISER systems store certain flags as arrays of single-digit numbers, where each number is used to hold three Boolean values. The Databridge Client can be directed to map these items as a series of Booleans data items (bit in SQL Server). This requires the setting of the DIOPT_Flatten_Occurs bit (1) and the DIOPT_Clone_as_Tribit bit (16) in the `di_options` column of the corresponding DMS_ITEMS record.

Following is an example for the item L-LOCK-FLAG in the data set LOAN.

File name: `script.user_layout.loan`

```
update DMS_ITEMS set active=1, di_options=17, dms_subtype=0
where dataset_name = 'LOAN' and rectype=0 and dms_item_name = 'L-LOCK-FLAG'
```

In this example, if the L-LOCK_FLAG has an OCCURS 20 TIMES clause, 60 items of type bit named l_lock_flag_01 to l_lock_flag_60 are created.

Splitting an Unsigned Number Item into Two Items

If you have NUMBER(12) items whose first two digits represent an account type and the remaining ten digits represent the account number, you might want to split this item into two columns. You can then rename the two columns as described in Renaming Columns.

In the following scripts, the NUMBER(12) item is named L_APPL_ACCT and is part of the data set LOAN. This item is mapped into two columns, the first of which contains 2 digits while the second one contains 10 digits. When the Client splits an item it appends "x1" and "x2" to the column names it creates to avoid having to deal with duplicate names.

File name: `script.user_layout.loan`

```
update DMS_ITEMS set di_options = 1048576, dms_subtype = 2
where dms_item_name = 'L-APPL-ACCT' and dataset_name = 'LOAN'
```

For SQL Server, this results in `columns l_appl_acct_x1` (data type tinyint) and `l_appl_acct_x2` (data type bigint).

You can also make the Client convert the first column to CHAR by setting bit 1024 in `di_options` to force the data to be stored using a data type of CHAR(2) in the relational database.

File name: `script.user_layout.loan`

```
update DMS_ITEMS set di_options = 1049600, dms_subtype = 2
where dms_item_name = 'L-APPL-ACCT' and dataset_name = 'LOAN'
```

Merging Two Neighboring Items

The following example merges the items SHIPPING-DATE and the item SHIPPING-TIME (which immediately follows it) in the data set SHIPMENTS.

File name: `script.user_layout.shipments`

```
update DMS_ITEMS set di_options = 0x1000000
where dms_item_name = 'SHIPPING-DATE' and dataset_name = 'SHIPMENTS'
```

The Client automatically skips the second item after it performs the merge, so you do not need to set its active column to 0.

> ✏️ **Note**
>
> This example is only valid for SQL Server. If you are using Oracle, you have to use decimal values.

Merging a Date and Time to Form a Date/Time

We extend the previous example to map the result to a relational database date/time data type. Assuming that these items have data types of NUMBER(8) and NUMBER(6) respectively in DMSII, we then treat the resulting value as a date/time of the form "*yyyymmddhhmiss*" (a date format value of 121).

File name: `script.user_layout.shipments`

```
update DMS_ITEMS set di_options = 0x1000080, dms_subtype = 121
where dms_item_name = 'SHIPPING-DATE' and dataset_name = 'SHIPMENTS'
```

The Client automatically skips the second item after it performs the merge, so you do not need to set its active column to 0.

> ✏️ **Note**
>
> This example is only valid for SQL Server. If you are using Oracle you have to use decimal values.

Concatenating Two Items and Cloning the Result as a Date/Time

This script allows you to combine numeric date and time data in non-contiguous columns. When the two columns are not contiguous, use the `dms_concat_num` column to append the time part of the combined item to the date part. This column must be set to the item number of the item containing the time value. The Client will effectively treat these two items as if the second one were concatenated to the first one. You must also set the `di_options` bit 524288 (0x80000) to make the Client include the second item in DATAITEMS with its active column set to 0. This is a lot more efficient than using DBGenFormat to perform this operation.

Filename: `script.user_layout.dttest` :

```
update DMS_ITEMS
    set dms_concat_num =(select dms_item_number from DMS_ITEMS
                         where dms_item_name='SALE-TIME' and dataset_name='DTTEST'),
       di_options = 0x82,
       dms_subtype = 111
where dms_item_name='SALE-DATE' and dataset_name ='DTTEST'
/***/
update DMS_ITEMS set di_options = 0x80000
where dms_item_name='SALE-TIME' and dataset_name='DTTEST'
```

This script combines the columns SALE-DATE and SALE-TIME into a column that effectively replaces SALE-TIME and is to be cloned as a long date with a date format of 111. The column sales_time needs to be present in the DATAITEMS control table, as the Client needs to access the DMSII data for the corresponding DMS item when performing the concatenation.

The second SQL statement in the script sets an option bit that tells the Client to map this item to DATAITEMS with its active column set to 0.

Adding a Composite Key to Tables Mapped from a Data Set

The following example inserts a composite key named user_set_shipping_detail into the data set SHIPPING-DETAIL, which does not have a SET defined in DMSII.

File name: `script.user_layout.shipping_detail`

```
update DATASETS set set_name='user_set'
where dataset_name = 'SHIPPING-DETAIL'
/***/
update DMS_ITEMS set item_key=1
where dms_item_name = 'SD-PO-NUMBER' and dataset_name = 'SHIPPING-DETAIL'
/***/
```

```
update DMS_ITEMS set item_key=2
where dms_item_name = 'SD-LINE-ITEM' and dataset_name = 'SHIPPING-DETAIL'
```

> ✏️ **Note**
>
> If the `set_name` is either "aa_set" or "user_set", the Databridge Client appends the `table_name` to the `set_name`. The above script takes advantage of this feature.

Specifying How to Handle Alpha Items That Are Too Long

The following script splits the item NOTES in the data set EMPLOYEE into multiple columns rather than truncating it at 4000 characters. The item is declared as ALPHA(4095) in DMSII. This script applies to Oracle.

File name: `script.user_layout.employee`

```
update DMS_ITEMS set di_options=4
where dms_item_name = 'NOTES' and dataset_name = 'EMPLOYEE'
```

16.4.4 Sample Data Table Customization Scripts

This section is intended as a quick reference for writing data table customization user scripts. Therefore, it lists sample scripts without any background explanation. If you are unfamiliar with the Databridge Client, make sure that you refer to the indicated sections for more information.

Sample Data Table Global Customization Script

The following example shows how to use one statement to rename all occurrences of the column name ts to time_stamp in the `item_name` column of the DATAITEMS Client control table for the DMSII database whose data source name is CMDB. Create only one of these scripts for each data source.

File name: `script.user_datatables.cmdb`

```
update DATAITEMS set item_name = 'time_stamp' where item_name = 'ts'
```

Disabling the Cloning of Secondary Tables

The following script disables the cloning of the secondary table, order_amounts for the data set named ORDER, by setting the active column value to 0 in the corresponding DATATABLES entry. In the case of an OCCURS table, the same result can be achieved by disabling the DMSII item instead. This is much more efficient because it does not create numerous unnecessary entries in DATATABLES and DATAITEMS.

File name: `script.user_define.order`

```
update DATATABLES set active=0 where table_name='order_amounts'
```

For more information, see Tips for Efficient Cloning.

Renaming a Table

Use the DATATABLES Client control table to rename tables in the relational database. The `dataset_name` column shows the DMSII data set name and the `table_name` column shows the name of the table as it appears in the relational database. For an explanation of how the DMSII data set and data items are mapped to the relational database, see Relational Database Table and Column Names.

You can change one or more relational database table names before you clone DMSII data sets. If you use the `clone` command, keep in mind that you must specify the DMSII data set name with the `clone` command, not the relational database table name. This means that if a DMSII data set is named ORD-YEAR-TOTAL and you rename the equivalent relational database table to total, you must still reference the DMSII data set by its name ORD-YEAR-TOTAL.

When you rename a table, make sure to do the following:

- The new table name must not be used by any other table. After the relational database has been created by the `define` or `redefine` command, the Databridge Client *does not* verify that renamed tables have unique names.
- The table name is no longer than 28 characters. Using table names longer than 28 characters causes SQL syntax errors when the Databridge Client executes the corresponding stored procedures.

Example

The following script changes the name of the table derived from the data set named EMPLOYEE to be full_time_employees. Both the DATATABLES and DATAITEMS Client control tables must be updated as all data items have a column that points back to the table to which they belong.

File name: `script.user_define.employee`

```
update DATATABLES set table_name='full_time_employees'
where table_name='employee'
/***/
update DATAITEMS set table_name='full_time_employees'
where table_name='employee'
```

Renaming Columns

Use the DATAITEMS Client control table to rename the columns that appear in the relational database. The data_item column shows the DMSII data item (column) name and the `item_name` column shows the name of the column as it will appear in the relational database. For an explanation of how the DMSII data set and data items are mapped to the relational database, see Relational Database Table and Column Names.

You can change one or more column names before or after cloning, as follows:

- If you change the relational database column name immediately after you run a `define` command, continue with the remaining commands. Keep in mind, however, that the DMSII data item retains its original name in the DMSII database. We recommend that you make this change via user scripts during the `define` and `redefine` command to ensure that your changes are not lost.
- If you change the column name after you have already cloned a DMSII database, you must mark the table to be re-cloned and then rerun the `generate` command to create new scripts that contain the new column name.

> ✏️ **Note**
>
> Column names in Oracle are limited to 28 characters. Using a column name longer than 28 characters results in a SQL syntax error when the Databridge Client executes the corresponding stored procedures.

Example

The following script changes the names of two columns in the table derived from the data set named ORDERS.

File name: `script.user_define.orders`

```
update DATAITEMS set item_name='order_amount'
where item_name='order_amt' and table_name='orders'
/***/
update DATAITEMS set item_name='order_date'
where item_name='order_dt' and table_name='orders'
```

Changing SQL Data Types

The following user script changes the sql_type for a packed decimal (`sql_type` of 11) data item named order_amount to be a floating point number (`sql_type` of 6).

File name: `script.user_define.transaction`

```
update DATAITEMS set sql_type=6
where item_name='order_amount' and table_name='orders'
```

Cloning a Number as a Character Type

This operation requires that you set the DAOPT_Store_as_Char bit (512) in the `da_options` column of the corresponding DATAITEMS record. Additionally, you must change the value of the `sql_type` column to the appropriate character type (such as 1 for char, 2 for varchar, and so on). Finally, in the case of SQL Server, you must also change the value of the `sql_length` column, as this column has a value of zero for the **int** and **smallint** data types. An example for the item l_appl_code in the table loan follows.

File name: `script.user_define.loan`

```
update DATAITEMS set sql_type=1, sql_length=2, sql_scale=0, da_options=512
where item_name='l_appl_code' and table_name='loan'
```

Adding a Non DMSII Column

The following script demonstrates how to add a non DMSII column to a relational database table.

This script adds three non DMSII columns (`update_type` , `audit_ts` , and `deleteD_record`) to the ORDERS data set and preserves all deletes, including multiple deletes with the same key value, since bit column 10 becomes a new key item with a unique value.

File name: `script.user_layout.orders`

```
update DATASETS set external_columns = 521 where dataset_name ='orders'
```

16.5 Appendix E: Client Exit Codes

If the Databridge Client terminates with an error, an exit code appears in the `last_run_status` column of the DATASOURCES Client control table. The value 9999 indicates that the last run status is no longer available (typically when dbutility is running). These status messages apply only to the `process` and `clone` commands.

The Client exit codes are as follows:

> ✏️ **Note**
>
> On UNIX, exit statuses are restricted to 8 bits (a range of 0 to 255). The Client uses the exit status specified in the 8-bit Exit code column instead of the actual code, which is longer than 8 bits. This only affects shell scripts that test the exit status.

Exit Code	8-bit Exit Code	Description
2001	150	**Indicates an error in the command line when invoking the Client**
2002	151	**The control table check failed** This means one of two things—either the Client control tables do not exist, or they are not the right version. In the latter case, the Client issues a control table version mismatch error and suggests that you run the dbfixup program.
2003	152	**The data source is locked, indicating that a Client is currently running** If you try to run a `process` or `clone` command while there is a DBClient `process` command running, the run will return this exit code. The `-u` option will not work in this situation; you must wait for the run to finish. If the run hangs, you can release the data source lock by terminating the run.
2004	153	**An error occurred while loading the control tables**
2005	154	**The data source specified on the command line does not exist in the DATASOURCES table**
2006	155	**The process or clone command failed because the DS_Needs_Redefining(8) bit in the `status_bits` column was set for an active data set** This status indicates that normal operations can only be resumed after a `redefine` command is executed.
2007	156	**The Client could not connect to the Databridge Server or to Databridge Enterprise Server either during initialization or data transmission**
2008	157	**The `clone` command failed because one of the data set names specified on the command line is invalid**
2009	158	**A data set has an invalid value in the `ds_mode` column of DATASETS** Any other value causes the Client to abend with this exit code.
2010	159	**An error occurred while creating or cleaning up a data table at the start of the data extraction phase of a process or clone command**
2011	160	**An error occurred while dropping the index of a data table at the start of the data extraction phase of a process or clone command**
2012	161	**A bad structure index was received**
2013	162	**A system error occurred while attempting to allocate memory**
2014	163	**No active structures were found at the start of a `process` or `clone` command**
2015	164	**No active structures remain after a structure was deselected during a process or clone command**
2016	165	**The Client is stopping at the start of the fixup phase because errors occurred during data extraction**
2017	166	**The Client is stopping at the start of the fixup phase because errors occurred during data extraction and index creation**
2018	167	**The Client is stopping at the start of the fixup phase because errors occurred during index creation**
2019	168	**The Client is stopping at the start of the fixup phase because of the `defer_fixup_phase` parameter setting (or `-c` option)**

Exit Code	8-bit Exit Code	Description
2020	169	**Client operations are being inhibited by the `stop_time` settings** You can override this situation by specifying the `-o` option on the command line. The applies only to the command line Client (dbutility).
2021	170	**The console operator issued a `QUIT` command, which stops the Client at the next quiet point** If you stop the Client using a DBServer `AX QUIT` command for the worker or an Enterprise Server `Quit` command, a different exit code results (1015 for DBServer) and (1135 for Enterprise Server).
2022	171	**The Client encountered a SQL error while updating the control tables** **NOTE:** Some SQL errors generate an exit code of 2099.
2023	172	**An error occurred while executing a COMMIT TRANSACTION for the relational database**
2024	173	**An error occurred while executing a ROLLBACK TRANSACTION for the relational database**
2025	174	**The Client is stopping because it finished processing the audit file specified in the `stop_afn` column of the DATASOURCES tables** You can do one of the following: • Specify the stop AFN using the " `-F <afn>` " command line option. • Use the Stop After AFN command from the command line or the Administrative Console.
2026	175	**An error occurred in the EBCDIC to ASCII translation**
2027	176	**The command terminated because the Client encountered improperly linked virtual data sets while loading the control tables** This status applies only when the configuration file parameter `automate_virtuals` is set to True.
2028	177	**The clone command terminated because the operator tried to reclone a data set that is the primary source for the virtual data set without recloning the data set that is the secondary source for the virtual data set** This status only applies when the configuration file parameter `automate_virtuals` is set to True. For example, if the data sets SV-HISTORY (primary source) and SAVINGS (secondary source) provide input to the virtual data set SV-HISTORY-REMAP, you must reclone SAVINGS when you reclone SV-HISTORY.
2029	178	**The Client discarded records during audit processing** Any other fatal error or reorganization indication overrides this exit code.
2030	179	**The Client was unable to sign on to the relational database** To find the cause, locate the corresponding OCI Error in the log file or the log output.

Exit Code	8-bit Exit Code	Description
2031	180	**The `process` or `clone` command failed because some records were not loaded during the data extraction phase** When the verify_bulk_load parameter is set to 2, the Client compares the number of records loaded to the actual count of records in the table. If these do not match, the program fails with this exit code. If the `verify_bulk_load` parameter is set to 1, the program doesn't fail and errors are reflected in the final exit code, unless a more serious error occurs and overrides this exit code.
2032	181	**The `process` or `clone` command failed because the DS_Needs_Generating(4) bit in the `status_bits` column was set for an active data set** This status indicates that normal operations can only be resumed after a generate command is executed. You can also get this exit code from a redefine command when a generate command is required to create scripts for tables in the relational database that were affected by a DMSII reorganization.
2033	182	**You need to run a reorg command before resuming normal processing** You will get this exit code from a `redefine` command when a `reorg` command is needed to alter tables in the relational database affected by a DMSII reorganization. Note that a `reorg` command implicitly does a `generate` command.
2034	183	**The Client stopped because a DMSII garbage collection reorganization that affects one or more datasets was encountered during audit file processing and the configuration `parameter` `stop_after_gc_reorg` was set to True.**
2035	184	**A clone was aborted by Enterprise Server and the operation was never restarted.** This is a special case of a failed clone.
2036	185	**Client stopped** This exit status indicates that the operator issued a dbutility QUIT NOW command or an abort command from the Administrative Console, which stopped the Client by closing the TCP connection to the server.
2037	186	**A relational database deadlock was detected** This error causes the Client to exit. When using dbutility, the program tries to restart the process command 3 times before exiting. When using DBClient, the service automatically retries the process command, as specified by the Error Recovery parameters set for the data source in the service configuration file.
2038	187	**The Client is stopping at the end of the fixup phase because of the `stop_after_fixup` parameter setting** You can resume processing by issuing another process command when you are ready.

Exit Code	8-bit Exit Code	Description
2039	188	**The Client is unable to continue because the global working directory specified in the Windows registry or in the UNIX file /etc/Micro Focus/DATABridge/globalprofile.ini cannot be found** `Even if you do not use the service, before you can run the Client, the working directory (which includes the locks subdirectory) must be created.`
2040	189	**Client encountered an error when trying to open the lock file for the data source** `Look at the log file or the log output to determine the nature of the error, which might be security related. You must always run the Client using the same user. Failure to do so can result in this error.`
2041	190	**Databridge Client for Microsoft SQL Server is unable to continue because the install directory specified in the Windows registry cannot be found** `Reinstall the Client using the installer so that the Client can access this directory (and the bcp_auditor program) without having to use the full path. Copying the files from the DVD will result in this exit code.`
2042	191	**The Client command failed because the DS_Needs_Mapping(1) or the DS_Needs_Remapping(4) bit in the `status_bits` column was set for an active data set** `This exit code indicates that normal operations can only be resumed after a redefine command is executed. You would typically get this status if you try to run a process or generate command after an error occurs when using the Client Configurator. You need to rerun the Client Configurator to fix the problem.`
2043	192	**File IO error caused the Client to terminate**
2044	193	**DMSII link improperly set up in the control tables**
2045	194	**`Reorg` command script in error**
2046	195	**Attempt to refresh stored procedure failed**
2047	196	**The Client abended because of one or more bulk loader errors.**
2048	197	**Client did not find a binary configuration file** `The DBClient and DBClntCfgServer programs get this exit status when the configuration file is not binary.`
2049	198	**An I/O error occurred while reading the configuration file** `For details, see error messages in the log.`
2050	199	**Computed checksum does not match the value in the binary configuration file** `This error occurs if you try to patch the file using a hex editor. Use the export command, edit the exported configuration file, and then import it.`
2051	200	**Errors found while processing a text configuration file** `See the log file for details.`

Exit Code	8-bit Exit Code	Description
2052	201	**User_scripts directory not contained within the Client's working directory when security is enabled** Databridge security prevents users scripts from residing outside the Working Directory, as we have no control over such a directory and could therefore be vulnerable to unauthorized users modifying user scripts, which could have rather dire consequences.
2053	202	**Client encountered an I/O error while trying to write to a discard file** This exit status indicates that either the discard file is too large or the machine is running out of disk space. You should periodically clean up the working directory for the Client along with the discards and logs folders.
2054	203	**Total discards threshold has been reached** See the max_discards parameter in Appendix C for details.
2055	204	**Client encountered an error while trying to update a user table** This exit status indicates that the audit file original for Databridge Enterprise has changed. Setting the configuration parameter `stop_on_dbe_mode_chg` to true causes the Client to stop when the audit file origin changes.
2056	205	**DBEnterprise audit file origin changed** This exit status indicates that the audit file original for Databridge Enterprise has changed. Setting the configuration parameter stop_on_dbe_mode_chg to true will cause the Client to stop when the audit file origin changes.
2057	206	**Client control table version mismatch** This exit status indicates that the Client control tables need to be upgraded before you can resume Client operations. When using the service this happens automatically. However, if you are using the command line Client dbutility, you have to manually run dbfixup for a data source in each relational database.
2058	207	**SQL update took longer than the maximum allowable time specified by the `sql_exec_timeout` parameter** See Appendix C for details.
2059	208	**Errors found while processing a text configuration file** See the log file for details. The most likely cause of this error is that the table is locked by another application. The Databridge Client cannot operate if you prevent it from doing its job by locking tables.
2060	209	**Effective CHECKPOINT FREQUENCY parameters for the Databridge Engine are all 0** The most likely cause for this is COMMIT parameters set to 0 in the Client configuration file. Rather than attempting to process audit with this rather ridiculous setting, the Client stops and gives you a chance to rectify the situation.

Exit Code	8-bit Exit Code	Description
2061	210	Error in loading a DLL or finding its entry points
2062	211	Error in updating control table
2063	212	Error creating control table
2064	213	Error dropping control table
2065	214	Malformed unload file This exit status indicates that the reload command encountered a malformed unload file and could not complete the operation.
2066	215	Error dropping user table
2067	216	Control tables are incompatible with DBConfig You need to run dbscriptfixup to fix this situation.
2068	217	Unable to create directory
2069	218	Unable to allocate a STMT Try reducing the value of aux_stmts
2070	219	Client got an error while attempting to create a file
2071	220	User script in error
2072	221	Bad DMSII database timestamp This exit status indicates that the DMSII database timestamp does not match the one the Client is using. This is taken to mean that the Client is not using the same DMSII database as it was earlier.
2073	222	History table error
2074	223	Data source already defined This exit status indicates that the Client is attempting to define a data source that is already defined.
2075	224	Index for user table has too many columns
2076	225	Mismatched AFNs in control tables The redefine command requires that all active data sets point to the same audit file.
2077	226	Protocol Error
2078	227	File does not exist
2079	228	IO error reading filter file
2080	229	Malformed binary filter file
2081	230	Bad checksum in binary filter file
2082	231	Syntax error in filter source file
2083	232	Filter generation failed
2084	233	Unsupported table encountered in filter source file

Exit Code	8-bit Exit Code	Description
2085	234	Data source already exists in relational database
2086	235	Running a redefine command after customizing a data source is not recommended
2087-2088		Not currently used
2089	238	Client lost connection to database This usually means that the database was taken down without stopping the Client first. The service/daemon recognizes this error and enters its error recovery, which keeps trying to connect every so often, until the database comes back up again.
2090	239	Reserved
2091	240	Client lost connection to Client Manager service This usually means that the service crashed.
2092	241	Connection to the Databridge Server closed by the host or Enterprise Server system The Client is forced to exit. This indicates that the server closed the connection because of an operator command. This exit code indicates that the connection was closed in an orderly manner.
2093	242	Connection to the Databridge Server reset by the host or Enterprise Server system The Client is forced to exit. This exit code indicates that the server was forcibly terminated or that it crashed.
2094	243	Transport error for connection to server
2095	244	RPC timeout
2096	245	Initialization error
2097	246	Error processing RPC data
2098	247	Communications or protocol error
2099	248	Internal error This code is used for all errors that cause the dbread and dbwait callback routine to terminate prematurely.
9701	245	Encryption DLL not found
9702	245	SSL/TLS handshake failed
9703	245	Invalid user certificate
9704	245	SSL/TLS client failed to find user credentials
9705	245	Connection lost due to corrupted message
9706	245	Missing a DLL for SSPI
9707	245	Missing a DLL for CryptoAPI
9708	245	Security.dll not found
9709	245	Did not finish encryption handshake with host
9710	245	Local SSPI does not support crypto key length

Exit Code	8-bit Exit Code	Description
9711	245	**IP or hostname from server cert does not match connection**
9712	245	**Wrong client OS; no Security Support**

> ✏️ **Note**
>
> For more detailed information on how exit codes are used in custom programs to automate control of Databridge operations, see dbutility Exit Status Values.

16.6 Appendix F: Service Configuration

This appendix lists the parameters for the Client Manager service that automate most Client operations. In most cases, you'll use the Client Console to configure scheduling and other features of the service. See Configuring the Service.

16.6.1 Sample Client Manager Service Configuration File

sample of service configuration file

For information about Databridge Client configuration files, see Client Configuration Files.

```
;
; Databridge control program version 7.0 configuration file -- generated programmatically
;
[control_program]
ipc_port               = 8001
userid                 = "dbridge", "", administrator
startup_delay          = 1
sess_start_timeout     = 2
n_scr_threads          = 1
enable_status_file     = false
data_sources           = BANKDB, DEMODB

[Log_File]
file_name_prefix       = "cp"
;max_file_size          = 0
logsw_on_size          = false
logsw_on_newday        = false
newfile_on_newday      = true

[BANKDB]
working_dir            = "d:\\dbridge_work\\bankdb"
client_dir            = "c:\\Program Files\\Micro Focus\\DATABridge\\7.0\\SQLServer"
;sched_delay_secs       = 0
;daily                  = 10:00, 14:00, 18:00
sched_retry_secs       = 60
max_retries            = 3
blackout_period = 00:00, 00:00
;disable_on_exitcode = 93, 94
run_at_startup         = false
auto_redefine          = false
auto_generate          = false
disabled               = false

[DEMODB]
working_dir            = "d:\\dbridge_work\\demodb"
client_dir            = "c:\\Program Files\\Micro Focus\\DATABridge\\7.0\\SQLServer"
;sched_delay_secs       = 0
;daily                  = 10:00, 14:00, 18:00
sched_retry_secs       = 60
max_retries            = 3
sched_minwait_secs     = 18000
run_at_startup         = false
auto_redefine          = false
auto_generate          = false
disabled               = false
```

16.6.2 [Control_Program]

This section, which must always be present in the configuration file, is used to define various service parameters. It also contains a list of all the data sources that are configured for the service.

data_sources

Default: `<empty list>`
Range: Comma separated list of no more than 32 data sources (maximum of 256 characters)
Console: N/A (Handled Automatically)

Use the migrate utility to create the configuration during an upgrade or use the **Add Data Source** and **Remove Data Source** commands in the Administrative Console to manage this list rather than manually adding data sources to the configuration file.

If the line of data sources is long and must wrap, the export command inserts a backslash (\) after a comma to indicate that the list continues on a next line.

enable_status_file

Default: True
Range: True or False
Console: Property sheet for the service
Applies to: Clustered Windows systems

When set to True, this parameter causes the service to maintain a status file containing information about the state of the various data sources it controls. This file is named "dbstatus.cfg" and resides in the **config** sub-directory. It is used to restart runs that were active before the service was restarted. The difference between using this method and setting the configuration parameter `run_at_startup` to True for a data source is that latter causes the run to always be started, even if the data source was not active when the service was taken down.

> ✏️ **Note**
>
> If you are not on a Clustered Windows system or you have not installed the Cluster option package, this parameter has no effect as the service ignores it. The Cluster option is separately licensed from the Client software.

---

ipc_port

Default: 8001
Range: 16-bit unsigned integer
Console: Not yet implemented

This parameter specifies the TCP/IP port number on which the service listens for connection requests from the Client Console or Client runs. If the default port is used by some other application on the Client machine, you can change the port.

When the service creates a new configuration file, it sets the value for this parameter using the port number specified at the time of installation, which is then saved to the Windows Registry (or the `globalprofile.ini file` in UNIX). After the `ipc_port` value is set, the service refers only to the `ipc_port` parameter in the service configuration file for this value (not the Windows Registry (or the `globalprofile.ini file` in UNIX).

---

n_scr_threads

Default: 1
Range: 1 - 4
Console: Not yet implemented

This parameter specifies the size of the pool of threads the service uses to start externally launched scripts and end-of-run scripts. If all of the threads are busy, the execution of a script might be delayed until a thread become available. To update this parameter from the Client Console, go to the Explorer view, right-click the service node, click Properties, and then modify the value.

---

sess_start_timeout

Default: 2 (seconds) **Range:** 2-10 **Console:** Not yet implemented

This parameter specifies the length of time that the service waits for input from a new connection before forcing a disconnect. The reason for doing this is to protect against a flood of rogue connection requests that would otherwise cripple the service. In some cases, the default value of 2 seconds might be too low. This parameter allows you to adjust the value to best suit your environment.

---

startup_delay

Default: 1
Range: 0-15 (seconds)
Console: Not yet implemented

This parameter ensures that process commands for all data sources do not launch simultaneously, which can result in the mainframe failing to start all workers and lead to failed runs. In most cases, the default value of 1 second is adequate.

---

userid = , ,

Default: dbridge, "", administrator **Range:** character string **Console: Client Managers**

When an Administrative Console session connects to the service, it provides a userid that has been authenticated without a password. The batch console and BCNOTIFY continue to work they way they did in 6.6 and need to provide a password in addition to the userid. The userid is not encoded, but the password is. You can have up 30 userids and each can be assigned one of the following roles:

Role	Alternate Spelling	Access Privileges
administrator	admin	Full privileges to use the Client Console.
operator	oper	Can perform only tasks related to daily operations, such as starting or stopping a run.
user		Can monitor runs and perform status command.

When an Administrative Console session connects to the service, it provides the userid of the browser user that has been authenticated. This does not guarantee that the user will be tables to connect to the service, as we require that the userid of the user that allowed to access the service have their userid included in the service' configuration file. If the LDAP credential of the user were good, that does not mean that he is allowed to use the console. For batch console users we still use a password, like we did in 6.6. The service can determine whether the connection originated in Administrative Console server. It will not accept a blank password from bconsole users. Password in the service's configuration file are encoded, like they were in 6.6

To manage the configured userids in the service select the desired Client Manager from the Client managers page and click on **Manage Users** in the **Actions** menu. This will allow to add, modify or remove userid in the service. The **Set bconsole Password** item in the **Actions** menu allows to set the password for a **bconsole** userid. If you use the same userid for both the **bconsole** and Administrative Console, this is not a problem as the password is not checked from Administrative Console users as they have already been authenticated.

16.6.3 [Log_File]

Use the [Log_File] section to control the various options for the log file, which is created in the logs subdirectory of the working directory.

---

file_name_prefix

Default: "cp"
Range: 1 to 20 characters **Console: Settings > Configure > LOGGING > Service Log** (File name prefix ...)

Use this parameter to change the prefix of the log files. The log files have names in the form `cp*yyyymmdd*.log`, or, when necessary, `cp*yyyymmdd_hhmiss*.log`. This command allows you to replace the prefix "cp" with any character string (up to 20 characters in length), provided that it results in a legal filename.

---

logsw_on_newday

Default: False
Range: True or False **Console: Settings > Configure > LOGGING > Service Log** (Switch log daily)

This parameter determines whether the program uses a new log file when the date changes. You may want to set this parameter to False if your log files are small and use the `logsw_on_size` parameter to manage the log files.

logsw_on_size

Default: False
Range: True or False
Recommended value: True **Console: Settings > Configure > LOGGING > Service Log** (Switch log on size)

Use this parameter to control whether the program should check the log file size to see if it has reached the size defined by the `max_file_size` parameter. If the size of the log file exceeds this parameter, the log file is closed and a new one is opened. If the current date is different from the creation date of the old file (which is part of its name), the new log file will be of the form `dbyyyymmdd.log` ; otherwise the time component will be added to the filename to ensure that the name is unique.

max_file_size

Default: 0
Range: numeric value optionally followed by K, M
Recommended value: 1M **Console: Settings > Configure > LOGGING > Service Log** (Maximum file size)

Use this parameter to limit the size of log files. The default value of 0 indicates that no limit is imposed on the size of log file. The suffixes of K, M and G allow you specify the maximum file size in kilobytes, megabytes, or gigabytes. A value on the order of 1 MB is a reasonable value to use. The file size is always checked when you start the program regardless of the setting of the `logsw_on_size` parameter.

newfile_on_newday

Default: True
Range: True or False **Console: Settings > Configure > LOGGING > Service Log** (Switch log on new day)

This parameter forces the program to use a new log file, when it starts up and the log file was created on an earlier date. You may want to set this parameter to False, if your log files are small and use the `logsw_on_size` parameter to manage the log files.

16.6.4 [data_source_name]

To modify the parameters for each data source, open the Databridge Client Manager service and right-click the data source you wish to modify. This will activate the Client Configuration menu. Each parameter's listing below lists the clicks from here to reach its section of the dialog box. If you don't select a data source first, the menu item will be grayed out.

Each data source that is defined in the `data_sources` parameter of the [Control_Program] section has its own section in the configuration file by that name. The data source section name must follow the first two sections. Do not move these sections before the [Control_Program]; this will result in an error and cause the service program to exit immediately. When an error occurs, you can examine the log file to determine the cause of the problem.

auto_generate

Default: False
Range: True or False
Console: Processing > Scheduling

This parameter causes the service to automatically launch a `generate` command if a (service-initiated) `process` or `redefine` command gets a return status indicating that a `generate` command is required. This parameter is designed to be combined with the `auto_redefine` parameter to allow operations to continue when a DMSII reorganization is detected.

auto_redefine

Default: False
Range: True or False
Console: Processing > Scheduling

This parameter causes the service to automatically launch a `redefine` command after a DMSII reorganization is detected (that is, when a service-launched process gets a return status).

When combined with the `auto_generate` parameter, this parameter allows operations to continue after a DMSII reorganization. If the `redefine` command finds nothing to do, the service launches a `process` command and operations resume. If the return status indicates that a `generate` command is required, the service will launch a `generate` command and upon successful completion of this command, will launch a `process` command. If the exit status of the `redefine` command indicates that a `reorganize` command is required, no action is taken. Manual intervention is required to examine the new scripts before they're executed to make sure that they don't corrupt the relational database.

If, after an automatic `redefine` command, tables in the relational database need to be altered, you can customize the data source and resume processing. The `redefine` command is fully compatible with customization features in the Client Console.

---

blackout_period

Default: 00:00, 00:00 **Range:** 00:00 to 24:00 (The two time values cannot be equal.) **Console:** Processing > Scheduling

Use this parameter to specify a fixed block of time during which the Client cannot run. This parameter is useful for operations, such as database backups, that can only take place when the Client is inactive. For example, if you want to back up the database daily between 1:00 a.m, and 2:30 a.m. daily, define a blackout period from 0:55 to 2:30. The extra 5 minutes ensures that the Client finishes any long transactions before the database backup begins.

If the Client is running when the blackout period starts, the Client automatically stops. If the Client is waiting for an idle host to send it updates when the blackout period starts, the Client resets the TCP/IP connection and aborts the run if it hasn't received any updates after 15 seconds. If you try to run the Client during a blackout period, nothing happens.

During a blackout period the service will not start the Client. If the scheduler tries to schedule a **DBClient** run at a time that falls within a blackout period, the start of the run will be delayed until the blackout period ends.

When this parameter is updated using the Client Console or Client Configurator, it is set to the same value in both the service and Client configuration files.

---

client_dir

Default: none (this line must be present)
Range: Double-Quoted string
Console: N/A (Handled automatically)

This parameter contains the full filename of the Client directory. In the case of Windows, all double slashes must be represented using two double slashes. In the case of UNIX, which uses forward slashes, this is not the case as the forward slash character has no special meaning for the configuration file scanner.

The Client directory is the database-specific subdirectory of the install directory.

In the case of Windows, the registry key INSTALLDIR is the Databridge entry point to this directory. The database specific sub-directories are SQLServer, Oracle or FlatFile.

---

daily

Default: daily = 08:00, 12:00, 17:00, 24:00 **Range:** 12 entries in ascending order from 00:00 to 24:00 **Console:** Processing > Scheduling

> ✏️ **Note**
>
> The daily parameter is mutually exclusive with the `fixed_delay` parameter. If you specify both `daily` and `fixed_delay` in a data source section of the configuration file, `fixed_delay` overrides `daily` regardless of the order in which they are specified. The service notifies you of this situation by writing a message to the log file.

Enter the times you want the service to launch a `process` command for the data source. You must specify 24-hour time (for example, 5:00 for 5:00 A.M. and 17:00 for 5:00 P.M.). The range for minutes is 00-59. You can specify up to 12 times for the daily parameter. However, you must specify the times in ascending order.

- The values 00:00 and 24:00 are equivalent for midnight.

- 24:00 is allowed only so that you can put it at the end of the list of times in ascending order.

- 24:01 is not allowed; instead, specify, 00:01.

---

disable_on_exitcode

Default: empty list
Range: a list of up to 3 exit codes
Console: Processing > Error Recovery (Disable ...)

Specify exit codes that cause the service to disable the data source. Allowable values include: 93 (stop before or after task); 94 (stop before or after time); and 2025 (stop after audit file number).

---

max_retries

Default: 3
Range: 0-20
Console: Processing > Error Recovery (Options)

The `max_retries` parameter is intended to specify the maximum number of times the service launches a Client `process` command after a failed `process` command. Not all exit conditions are recoverable. After it unsuccessfully tries to relaunch the Client the specified maximum number of times, the service disables the data source. You must enable the data source using the Client Console before you can launch another `process` command.

The `max_retries` parameter is ignored for a few exit codes, where the condition that causes the problem is expected to self-correct or change over time. (Retrying forever eliminates the need for manual intervention, which would be required if the data source were to be disabled.) Such situations include connect problems to the server or the database, which are often symptomatic of the host, the server, or the database being down.

---

run_at_startup

Default: False
Range: True or False
Console: Processing > Scheduling

This command is only meaningful during startup. It indicates whether the service should launch a Client `process` command for the data source when the service starts. If the process returns with a "`database not up`" error, the service retries the launch until the database is up.

sched_delay_secs

Default: 0 (indicating that this parameter is disabled)
Range: 1-86,400 seconds (24 hours)
Console: Processing > Scheduling

> ✏️ **Note**
>
> The `sched_delay_secs` parameter is mutually exclusive with the `daily` parameter. If you specify both `daily` and `sched_delay_secs` in a data source section of the configuration file, `sched_delay_secs` overrides `daily` regardless of the order in which they are specified.

Use the `sched_delay_secs` parameter to specify a fixed delay between the time a launched Client, running a `process` command for the data source, terminates and the launching of the next `process` command for the data source. To disable the `sched_delay_secs` parameter, comment it out or set its value to 0.

---

sched_minwait_secs

Default: 0
Range: 0-86,400 (24 hours)
Console: Processing > Error Recovery (Options)

This parameter ensures that a next scheduled `process` command is delayed by the given interval, when a `process` commands finishes right around the next scheduled time and would otherwise start too soon. This parameter delays the start of the next run for the specified amount of time.

---

working_dir

Default: none (this line must be present)
Range: A string of any length enclosed with quotation marks
Console: N/A (Handled automatically)

This parameter contains the full file name of the working directory. In the case of Windows, all double slashes must be represented using two double slashes. In the case of UNIX, which uses forward slashes, this is not the case as the forward slash character has no special meaning for the configuration file scanner.