Purpose

This book provides information about Teradata MultiLoad (MultiLoad), which is a Teradata®
Tools and Utilities product. Teradata Tools and Utilities is a group of products designed to
work with Teradata Database.

MultiLoad provides an efficient way to deal with batch maintenance of large databases.
MultiLoad is a command-driven utility for fast, high-volume maintenance on multiple tables
and views of a Teradata Database.

Audience

This book is intended for use by:
- System and application programmers
- System administrators

Supported Releases

This book supports the following releases:
- Teradata Database 12.00.00
- Teradata Tools and Utilities 12.00.00
- Teradata MultiLoad Version 12.00.00

Note: See “In Interactive Mode” on page 30 to verify the Teradata MultiLoad version
number.

To locate detailed supported-release information:
2. Navigate to General Search>Publication Product ID.
3. Enter 3119.
4. Open the version of the Teradata Tools and Utilities 12.00.00 Supported Versions
   spreadsheet associated with this release.

The spreadsheet includes supported Teradata Database versions, platforms, and product
release numbers.
Prerequisites

The following prerequisite knowledge is required for this product:

- Computer technology and terminology
- Relational database management systems
- Utilities that load and retrieve data

Changes to This Book

The following changes were made to this book in support of the current release. Changes are marked with change bars. For a complete list of changes to the product, see the Release Definition associated with this release.

<table>
<thead>
<tr>
<th>Date and Release</th>
<th>Description</th>
</tr>
</thead>
</table>
| July 2007 12.00.00 | - Added BOM UTF8 restriction for MVS platform.  
- Added multi-byte character support with UTF8 and UTF16.  
- Clarified that concurrent load utility task limit is set by MaxLoadTask parameter.  
- Removed support for ASF2 Tape Reader.  
- Removed reference to WinCLI, which is discontinued.  
- Documented 30 byte limitation of logon string when using multi-byte character sets.  
- Added support for multi-character delimiters.  
- Documented possible errors that occur in Windows environment when ctrl+c is used multiple times.  
- Added SET QUERY_BAND as supported SQL statement. |

Additional Information

Additional information that supports this product and Teradata Tools and Utilities is available at the web sites listed in the table that follows. In the table, mmyx represents the publication date of a manual, where mm is the month, y is the last digit of the year, and x is an internal publication code. Match the mmy of a related publication to the date on the cover of this book. This ensures that the publication selected supports the same release.
<table>
<thead>
<tr>
<th>Type of Information</th>
<th>Description</th>
<th>Access to Information</th>
</tr>
</thead>
</table>
| Release overview    | Use the Release Definition for the following information:  
|                     | • Overview of all of the products in the release  
|                     | • Information received too late to be included in the manuals  
|                     | • Operating systems and Teradata Database versions that are certified to work with each product  
|                     | • Version numbers of each product and the documentation for each product  
|                     | • Information about available training and the support center | 1 Go to www.info.teradata.com.  
|                     | | 2 Select the General Search check box.  
|                     | | 3 In the Publication Product ID box, type 2029.  
|                     | | 4 Click Search.  
|                     | | 5 Select the appropriate Release Definition from the search results. |
| Late information    | | |
| Additional product information | Use the Teradata Information Products Publishing Library site to view or download specific manuals that supply related or additional information to this manual. | 1 Go to www.info.teradata.com.  
|                     | | 2 Select the Teradata Data Warehousing check box.  
|                     | | 3 Do one of the following:  
|                     | | • For a list of Teradata Tools and Utilities documents, click Teradata Tools and Utilities and then select a release or a specific title.  
|                     | | • Select a link to any of the data warehousing publications categories listed.  
|                     | | Specific books related to Teradata MultiLoad are as follows:  
|                     | | • Teradata MultiLoad Reference B035-2409-mmyA  
|                     | | • Teradata Tools and Utilities Command Summary B035-2401-mmyA |
| CD-ROM images       | Access a link to a downloadable CD-ROM image of all customer documentation for this release. Customers are authorized to create CD-ROMs for their use from this image. | 1 Go to www.info.teradata.com.  
|                     | | 2 Select the General Search check box.  
|                     | | 3 In the Title or Keyword box, type CD-ROM.  
|                     | | 4 Click Search. |
| Ordering information for manuals | Use the Teradata Information Products Publishing Library site to order printed versions of manuals. | 1 Go to www.info.teradata.com.  
|                     | | 2 Select the How to Order check box under Print & CD Publications.  
|                     | | 3 Follow the ordering instructions. |
The Teradata home page provides links to numerous sources of information about Teradata. Links include:

- Executive reports, case studies of customer experiences with Teradata, and thought leadership
- Technical information, solutions, and expert advice
- Press releases, mentions, and media resources

<table>
<thead>
<tr>
<th>Type of Information</th>
<th>Description</th>
<th>Access to Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>General information about Teradata</td>
<td>The Teradata home page provides links to numerous sources of information about Teradata. Links include:</td>
<td>1 Go to Teradata.com.</td>
</tr>
<tr>
<td></td>
<td>• Executive reports, case studies of customer experiences with Teradata, and thought leadership</td>
<td>2 Select a link.</td>
</tr>
<tr>
<td></td>
<td>• Technical information, solutions, and expert advice</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Press releases, mentions, and media resources</td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

Preface

Purpose ................................................................. 3
Audience ............................................................... 3
Supported Releases .................................................. 3
Prerequisites .......................................................... 4
Changes to This Book ............................................... 4
Additional Information ............................................ 4

Chapter 1: Overview

MultiLoad Utility ................................................... 15
   Description ....................................................... 15
   What it Does .................................................... 16
   How it Works .................................................... 16
Operating Features and Capabilities ............................. 17
   Operating Modes .............................................. 17
   Character Sets ................................................ 17
   Task Status Reporting ....................................... 17
   Data Conversion Capabilities ............................... 19
   Checkpoints ................................................... 20
MultiLoad Commands .............................................. 20
   MultiLoad Command Input ................................... 20
   MultiLoad Support Activity Commands .................. 20
   MultiLoad Task Activity Commands ...................... 21
   Teradata SQL Statements .................................... 22
   A MultiLoad Example ........................................ 24

Chapter 2: Using MultiLoad

Invoking MultiLoad ................................................ 29
   File Requirements ............................................ 29
In Interactive Mode .................................................30
In Batch Mode .....................................................30
Runtime Parameters .............................................31
VM Example ..........................................................36
MVS Example ..........................................................37
UNIX and Windows Examples ......................................37
Terminating MultiLoad ............................................38
Normal Termination ...............................................38
Abort Termination ..................................................39
After Terminating a MultiLoad Job ..............................39
Reinitializing After Terminating a MultiLoad Job ............39
Implications of Dropping Required MultiLoad-Created Tables .................................................40
Restarting a Paused MultiLoad Job ...............................42
After Using the PAUSE ACQUISITION Command ............43
After a MultiLoad Job Script Error ...............................43
After an Unrecoverable I/O Error ...............................43
After a Down AMP is Recovered ..................................43
After a Teradata Database Restart...............................44
After a Teradata Database System Reconfiguration ..........44
After a Job Abort or Client System Failure ....................44
After an Application Processor Reset Condition ...............45
Programming Considerations .....................................46
MultiLoad Configuration File ......................................46
MultiLoad Command Conventions ................................48
Data Conversion Factors .........................................49
Variables ..............................................................49
ANSI/SQL DateTime Specifications .................................53
Comments .............................................................53
Character Set Specification ........................................54
Graphic Data Types ...............................................59
Graphic Constants ..................................................59
Import Tasks ..........................................................59
Delete Tasks ..........................................................62
Dealing with AMPS ................................................62
Using Locks ..........................................................65
Restrictions and Limitations .......................................66
Termination Return Codes ........................................68
UNIX Signals ........................................................68
Using INMOD and Notify Exit Routines .........................69
Overview ............................................................69
Programming Considerations for Using Routines ..........69
Programming Structure ............................................70
Chapter 3: MultiLoad Commands

Syntax Notes ................................................................. 91
ACCEPT ................................................................. 92
BEGIN MLOAD and BEGIN DELETE MLOAD .................................. 95
DATEFORM ............................................................ 108
DELETE ................................................................. 109
DISPLAY .............................................................. 112
DML LABEL .......................................................... 114
END MLOAD .......................................................... 118
FIELD ................................................................. 119
FILLER ................................................................. 130
IF, ELSE, and ENDIF .................................................... 133
IMPORT .............................................................. 135
INSERT .............................................................. 144
LAYOUT .............................................................. 148
LOGDATA .......................................................... 151
LOGMECH .......................................................... 152
LOGOFF ............................................................. 153
LOGON .............................................................. 155
LOGTABLE ........................................................ 158
PAUSE ACQUISITION ................................................... 160
RELEASE MLOAD .................................................... 162
ROUTE MESSAGES .................................................... 165
RUN FILE .......................................................... 167
## Table of Contents

**SET** ................................................................. 169  
**SYSTEM** ............................................................. 171  
**TABLE** ............................................................... 172  
**UPDATE** .............................................................. 174

---

**Appendix A:**  
**How to Read Syntax Diagrams** ........................................ 177  
Syntax Diagram Conventions .................................................. 177  
Abbreviations ...................................................................... 179

---

**Appendix B:**  
**Invocation Examples** ..................................................... 181  
VM .............................................................................. 181  
MVS ............................................................................. 184  
UNIX and Windows ................................................................ 188

---

**Appendix C:**  
**INMOD and Notify Exit Routine Examples** ......................... 195  
VM and MVS .................................................................. 195  
UNIX ............................................................................ 215  
Windows ......................................................................... 226

---

**Appendix D:**  
**MultiLoad Job Script Examples** ........................................ 235  
Import Task ...................................................................... 235  
Delete Task ...................................................................... 236  
Upsert Task ...................................................................... 236  
DDL for PERSONNEL Database Tables ................................. 237
Glossary ................................................................. 239

Index .................................................................... 265
Table of Contents
List of Tables

Table 1: Phases of MultiLoad Operation .............................................. 16
Table 2: MultiLoad Support Activity Commands ................................... 21
Table 3: MultiLoad Task Activity Commands ....................................... 21
Table 4: Supported Teradata SQL Statements in MultiLoad ................. 22
Table 5: Alternatives to Using MultiLoad ........................................... 27
Table 6: Runtime Parameters (Channel-Attached Configurations) .......... 32
Table 7: Runtime Parameters (Network-Attached Configurations) .......... 33
Table 8: System Variables .................................................................. 49
Table 9: Notes About MultiLoad Comments ........................................ 53
Table 10: Character Sets Supported by MultiLoad ............................... 54
Table 11: Methods for Specifying Character Sets ............................... 55
Table 12: Commands Impacting Multibyte Character Sets ................... 58
Table 13: Import Task Considerations for MultiLoad ......................... 60
Table 14: Delete Task Considerations for MultiLoad ........................... 63
Table 15: Types of Table Locks Imposed by MultiLoad ....................... 65
Table 16: Feature Restrictions ............................................................ 66
Table 17: Languages Supported by Platform and Type of User-Developed Routine .................................................. 70
Table 18: Programming Routines by Language .................................... 70
Table 19: Entry Points for Notify Exit Routines .................................... 72
Table 20: MultiLoad-to-INMOD Status Codes ..................................... 72
Table 21: INMOD-to-MultiLoad Interface Status Codes ...................... 73
Table 22: Events Passed to the Notify Exit Routines ............................ 75
Table 23: Acquisition Error Table Format ........................................... 86
Table 24: Application Error Table Format ........................................... 87
Table 25: Sequence Type and Number for Example Statements ........... 88
Table 26: Events that Create Notifications ......................................... 104
Table 27: ANSI/SQL DataTime Specifications ..................................... 125
This chapter provides an introductory overview of the Teradata MultiLoad utility. Topics include:

- MultiLoad Utility
- Operating Features and Capabilities
- MultiLoad Commands
- A MultiLoad Example

## MultiLoad Utility

This section describes MultiLoad and its features, and explains how the utility operates.

### Description

The MultiLoad utility is an efficient way to deal with batch maintenance of large databases. MultiLoad is a command-driven utility for fast, high-volume maintenance on multiple tables and views of a Teradata Database.

A single MultiLoad job performs a number of different import and delete tasks on database tables and views:

- Each MultiLoad import task can do multiple data insert, update, and delete functions on up to five different tables or views.
- Each MultiLoad delete task can remove large numbers of rows from a single table.

Use MultiLoad to import data from:

- Disk or tape files (using a custom Access Module) on a channel-attached client system
- Input files on a network-attached workstation
- Special input module (INMOD) programs you write to select, validate, and preprocess input data
- Access modules
- Any device providing properly formatted source data

The table or view in the database receiving the data can be any existing table or view for which you have access privileges for the maintenance tasks you want to do.

**Note:** Full tape support is not available for any function in MultiLoad for network-attached client systems. If you want to import data from a tape, you will need to write a custom access
module that interfaces with the tape device. For information about how to write a custom access module, see the *Teradata Tools and Utilities Access Module Programmer Guide*.

**What it Does**

When you invoke MultiLoad, the utility executes the MultiLoad commands and Teradata SQL statements in your MultiLoad job script. Table 1 describes the phases of MultiLoad operations.

Table 1: Phases of MultiLoad Operation

<table>
<thead>
<tr>
<th>Phase</th>
<th>MultiLoad Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary</td>
<td>1 Parses and validates all of the MultiLoad commands and Teradata SQL statements in your MultiLoad job</td>
</tr>
<tr>
<td></td>
<td>2 Establishes sessions and process control with the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>3 Submits special Teradata SQL requests to the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>4 Creates and protects temporary work tables and error tables in the Teradata Database</td>
</tr>
<tr>
<td>DML Transaction</td>
<td>Submits the DML statements specifying the insert, update, and delete tasks to the Teradata Database</td>
</tr>
<tr>
<td>Acquisition</td>
<td>1 Imports data from the specified input data source</td>
</tr>
<tr>
<td></td>
<td>2 Evaluates each record according to specified application conditions</td>
</tr>
<tr>
<td></td>
<td>3 Loads the selected records into the worktables in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>(There is no acquisition phase activity for a MultiLoad delete task.)</td>
</tr>
<tr>
<td>Application</td>
<td>1 Acquires locks on the specified target tables and views in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>2 For an import task, inserts the data from the temporary work tables into the target tables or views in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>3 For a delete task, deletes the specified rows from the target table in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>4 Updates the error tables associated with each MultiLoad task</td>
</tr>
<tr>
<td>Cleanup</td>
<td>1 Forces an automatic restart/rebuild if an AMP went offline and came back online during the application phase</td>
</tr>
<tr>
<td></td>
<td>2 Releases all locks on the target tables and views</td>
</tr>
<tr>
<td></td>
<td>3 Drops the temporary work tables and all empty error tables from the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>4 Reports the transaction statistics associated with the import and delete tasks</td>
</tr>
</tbody>
</table>

**How it Works**

MultiLoad processes a series of commands and Teradata SQL statements you usually enter as a batch mode job script. The MultiLoad commands perform session control and data handling of the data transfers. The Teradata SQL statements do the actual maintenance functions on the database tables and views.
Serialization of Changes

In certain uses of MultiLoad (or TPump), it is possible to have multiple changes to one row in
the same job. For example, the row may be inserted and then updated during the batch job, or
it may be updated and then deleted. In any case, the correct ordering of these operations is
obviously very important. MultiLoad automatically guarantees that this ordering of
operations is maintained correctly. By using the serialization feature, TPump can also
guarantee that this ordering of operations is maintained correctly.

Operating Features and Capabilities

This section describes the operating modes and character sets supported by MultiLoad. It also
discusses the utility’s capabilities.

Operating Modes

MultiLoad runs in the following operating modes:

- Interactive
- Batch

Character Sets

MultiLoad supports the character sets shipped with Teradata; this includes Latin, Japanese,
Chinese, and Korean character sets, along with ASCII, EBCDIC, UTF8, and UTF16.
MultiLoad supports user-defined session character sets. For additional information about
character-set support and definition, see “Character Set Specification” on page 54.

Task Status Reporting

MultiLoad has three ways to provide information about the status of jobs that are still in
progress and those that have just completed:

- MultiLoad utility messages provide job status reports, including:
  - Options messages that list the settings of MultiLoad task parameters.
  - Logoff/disconnect messages that list key run statistics.
  - The QrySessn Utility provides real-time, phase-oriented progress reports at selected
    intervals during the MultiLoad job.
- Statistics messages present information at the end of the MultiLoad job.

The following sections describe each MultiLoad task status reporting method.

Options Messages

MultiLoad options messages list the options specifications for each task:

- SESSIONS
- CHECKPOINT
• TENACITY
• SLEEP
• ERRLIMIT
• AMPCHECK

Logoff/Disconnect Messages
Issued in response to the LOGOFF command, the MultiLoad logoff/disconnect message lists:
• The time that the LOGOFF command was executed
• Whether the disconnect operation was successful
• Whether the restart log table was dropped or kept, depending on the success or failure of the job
• The total processor time used
• The job start/end time and date
• The highest return code encountered by the job

QrySessn Utility
QrySessn is a separate utility that can be used to monitor the progress of your MultiLoad job. QrySessn reports different sets of status information for each phase of the MultiLoad job:
• Preliminary
• Acquisition
• Application

QrySessn also reports on:
• Action values for processing data
• Data counts for the total activity
• I/O accesses
• Optional data counts and row/sector counts, by session
• Phase and CPU time-of-day information
• Row count statistics for:
  • Non-unique secondary indexes (NUSIs)
  • Primary data
  • Rows deleted
  • Rows inserted in target tables
  • Target table rows processed
You can execute QrySessn from either:

- A system console using the Database Window interface on a system running Teradata Database for UNIX or Teradata Database for Windows
  
or
  
- The Teradata Manager application

For complete information about using the QrySessn utility, see the QrySessn chapter in the Utilities Quick Reference.

**Statistics Messages**

For each import task, a MultiLoad statistics message reports the total number of:

- Data records sent to the Teradata Database, including a subtotal of the number of records from each data source
- Records skipped from each data source because of false WHERE conditions
- Rows inserted, updated, or deleted
- Errors recorded in the first error table for each target table
- Errors recorded in the second error table for each target table, including the number of duplicate rows

MultiLoad writes options and logoff/disconnect messages, and the statistics facility report to either:

- The customary output destination for your client system (SYSPRINT/`stdout` or the redirected `stdout`).
  
or
  
- An alternate destination specified in a ROUTE MESSAGES command.

The utility also writes the statistics information in the restart log table so it can be restored after a system restart operation.

**Data Conversion Capabilities**

MultiLoad can redefine the data type specification of numeric, character, and date input data so it matches the type specification of its destination column in the MultiLoad table on the Teradata Database.

For example, if an input field with numeric type data is targeted for a column with a character data type specification, MultiLoad can change the input data specification to character before inserting it into the table.

You use the `datadesc` specification of the MultiLoad FIELD command to convert input data to a different type before inserting it into the MultiLoad table on the Teradata Database.
The types of data conversions you can specify are:

- Numeric-to-numeric (for example integer-to-decimal)
- Character-to-numeric
- Character-to-date
- Date-to-character

**Note:** Redundant conversions, such as integer-to-integer, are legal and necessary to support the zoned decimal format. For more information about the zoned decimal format, data types, and data conversions, see *SQL Reference: Data Types and Literals*.

### Checkpoints

MultiLoad supports the use of checkpoints. Checkpoints are entries posted to a restart log table at regular intervals during the MultiLoad data transfer operation. If processing stops while a MultiLoad job is running, you can restart the job at the most recent checkpoint.

For example, assume you are loading 1,000,000 records in a table and have specified checkpoints every 50,000 records. Then MultiLoad pauses and posts an entry to the restart log table whenever multiples of 50,000 records are successfully sent to the Teradata Database.

If the job stops after record 60,000 has been loaded, you can restart the job at the record immediately following the last checkpoint—record 50,001.

You enable the checkpoint function by specifying a checkpoint value in the BEGIN MLOAD command. For more information, see **BEGIN MLOAD** and **BEGIN DELETE MLOAD**.

### MultiLoad Commands

MultiLoad accepts both MultiLoad commands and a subset of Teradata SQL statements. These are described in the following sections:

#### MultiLoad Command Input

The MultiLoad commands perform the following two types of activities.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support</td>
<td>Support commands establish MultiLoad sessions with the Teradata Database and establish the operational support environment for MultiLoad.</td>
</tr>
<tr>
<td></td>
<td>Support commands are <em>not</em> directly involved in specifying a MultiLoad task.</td>
</tr>
<tr>
<td>Task</td>
<td>The MultiLoad task commands specify the actual processing that takes place for each MultiLoad task.</td>
</tr>
</tbody>
</table>

### MultiLoad Support Activity Commands

Table 2 lists the MultiLoad commands that perform support activities.
### MultiLoad Command Activity Commands

The MultiLoad commands that perform task activities are listed in Table 3.

#### Table 2: MultiLoad Support Activity Commands

<table>
<thead>
<tr>
<th>MultiLoad Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>Allows the value of one or more utility variables to be accepted from either a file or an environment variable.</td>
</tr>
<tr>
<td>DATEFORM</td>
<td>Specifies the form of the DATE data type specifications for the MultiLoad job.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Writes messages to the specified destination.</td>
</tr>
<tr>
<td>ELSE</td>
<td>Introduces commands and statements that execute when a preceding IF condition is false.</td>
</tr>
<tr>
<td>ENDIF</td>
<td>Delimits the group of MultiLoad commands that were subject to previous IF or ELSE conditions.</td>
</tr>
<tr>
<td>IF</td>
<td>Introduces a conditional expression whose value initiates execution of subsequent commands.</td>
</tr>
<tr>
<td>LOGOFF</td>
<td>Disconnects all active sessions and terminates MultiLoad on the client system.</td>
</tr>
<tr>
<td>LOGON</td>
<td>Specifies the LOGON command string to be used in connecting all sessions established by MultiLoad.</td>
</tr>
<tr>
<td>LOGTABLE</td>
<td>Identifies the table to be used to journal checkpoint information required for safe, automatic restart of MultiLoad when the client or Teradata Database system fails.</td>
</tr>
<tr>
<td>ROUTE MESSAGES</td>
<td>Identifies the destination of output produced by MultiLoad support environment.</td>
</tr>
<tr>
<td>RUN FILE</td>
<td>Invokes the specified external source as the current source of commands and statements.</td>
</tr>
<tr>
<td>SET</td>
<td>Assigns a data type and a value to a utility variable.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Suspends operation of MultiLoad and executes any valid local operating system command.</td>
</tr>
</tbody>
</table>

### MultiLoad Task Activity Commands

The MultiLoad commands that perform task activities are listed in Table 3.

#### Table 3: MultiLoad Task Activity Commands

<table>
<thead>
<tr>
<th>MultiLoad Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN MLOAD and BEGIN DELETE MLOAD</td>
<td>Specifies:</td>
</tr>
<tr>
<td></td>
<td>• The kind of MultiLoad task to be executed</td>
</tr>
<tr>
<td></td>
<td>• The target tables in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>• The parameters for executing the task</td>
</tr>
<tr>
<td>DML LABEL</td>
<td>Defines a label and error treatment options for a following group of DML statements.</td>
</tr>
</tbody>
</table>
Teradata SQL Statements

Teradata SQL statements define and manipulate the data stored in the Teradata Database.

MultiLoad supports a subset of Teradata SQL statements. As a result, other utilities do not have to be invoked to perform routine database maintenance functions before executing MultiLoad utility tasks. For example, you can use the supported Teradata SQL statements to:

- Create the table that you want to load
- Establish a database as an explicit table name qualifier
- Add checkpoint specifications to a journal table

MultiLoad supports only the Teradata SQL statements listed in Table 4. To use any other Teradata SQL statements, you must exit MultiLoad and enter them from another application, such as Basic Teradata Query (BTEQ).

Table 4: Supported Teradata SQL Statements in MultiLoad

<table>
<thead>
<tr>
<th>Teradata SQL Statement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER TABLE</td>
<td>Changes the column configuration or options of an existing table</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>Adds a checkpoint entry to a journal table</td>
</tr>
</tbody>
</table>
Table 4: Supported Teradata SQL Statements in MultiLoad (continued)

<table>
<thead>
<tr>
<th>Teradata SQL Statement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECT STATISTICS</td>
<td>Collects statistical data for one or more columns of a table</td>
</tr>
<tr>
<td>COMMENT</td>
<td>Stores or retrieves a comment string associated with a database object</td>
</tr>
<tr>
<td>CREATE DATABASE</td>
<td>Creates a new database</td>
</tr>
<tr>
<td>CREATE MACRO</td>
<td>Creates a new macro</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>Creates a new table</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Creates a new view</td>
</tr>
<tr>
<td>DATABASE</td>
<td>Specifies a new default database for the current session</td>
</tr>
<tr>
<td>DELETE</td>
<td>Removes rows from a table</td>
</tr>
<tr>
<td>DELETE DATABASE</td>
<td>Removes all tables, views, and macros from a database</td>
</tr>
<tr>
<td>DROP DATABASE</td>
<td>Drops the definition for an empty database from the Data Dictionary</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>Removes a table</td>
</tr>
<tr>
<td>GIVE</td>
<td>Transfers ownership of a database to another user</td>
</tr>
<tr>
<td>GRANT</td>
<td>Grants access privileges to a database object</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts new rows to a table</td>
</tr>
<tr>
<td>MODIFY DATABASE</td>
<td>Changes the options of an existing database</td>
</tr>
<tr>
<td>RELEASE MLOAD</td>
<td>Removes the access locks from the target tables in the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>This inhibits any attempts to restart the MultiLoad job when a MultiLoad task has been suspended or aborted.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> A RELEASE MLOAD statement must be entered from BTEQ. You cannot enter a RELEASE MLOAD statement from MultiLoad.</td>
</tr>
<tr>
<td>RENAME</td>
<td>Changes the name of an existing table, view, or macro</td>
</tr>
<tr>
<td>REPLACE MACRO</td>
<td>Redefines an existing macro or view</td>
</tr>
<tr>
<td>REPLACE VIEW</td>
<td></td>
</tr>
<tr>
<td>REVOKE</td>
<td>Rescinds access privileges to a database object</td>
</tr>
<tr>
<td>SET QUERY_BAND</td>
<td>Identifies type and source of query to determine prioritization of queries</td>
</tr>
<tr>
<td>SET SESSION COLLATION</td>
<td>Overrides the collation specification for the current session</td>
</tr>
<tr>
<td>SET SESSION OVERRIDE</td>
<td></td>
</tr>
<tr>
<td>REPLICATION ON/OFF</td>
<td>Turns replication services on or off</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Changes the column values of an existing row in a table</td>
</tr>
</tbody>
</table>
Chapter 1: Overview
A MultiLoad Example

Caution: Specifying any DML statements (insert/delete/update) before specifying the BEGIN MLOAD command will use the non-fast path and processing will be done in the Transient Journal (very slow and may exceed the maximum perm space). Specifying the DML statement after the BEGIN MLOAD command will use the fast path and processing will be done in the transaction phase (very fast).

For syntax and a complete description of each Teradata SQL statement, see SQL Reference: Data Definition Statements and SQL Reference: Data Manipulation Statements.

A MultiLoad Example

This topic provides an example of a small MultiLoad job that you can quickly set up and run. The example shows you how to:

- Create a data file to use as the input source for a MultiLoad import task
- Use a MultiLoad job script to import data into a newly created table
- Select data from the table to verify the import task

This example is for UNIX or Windows on a network-attached client system. For additional UNIX and Windows examples, and for VM and MVS examples on channel-attached systems, see the following appendices:

- Appendix B: “Invocation Examples”
- Appendix C: “INMOD and Notify Exit Routine Examples”
- Appendix D: “MultiLoad Job Script Examples”

Dropping the Employee Table

The table in this example is named Employee. Use the Teradata SQL DROP TABLE command to delete any existing version of the Employee table from your database:

```sql
bteq
ologon tdpid/username,password
DROP TABLE employee;
.quit
```

Creating the Source Data File

Create and save a six-record text file named insert.input:

```
|10021 |Brown, Jo |200|2312|Development |63000.00 |20|Jan 01 1955|F| |M|16| 0|
|10001 |Jones, Bill |100|5376|President |83000.00 |15|Jan 01 1960|M| |M|14| 0|
|10002 |Smith, Jim |100|4912|Sales |73000.00 |10|Jan 01 1970|M| |M|13| 1|
|10028 |Lee, Sandra |200|5844|Support |77000.00 | 4|Jan 01 1971|F| |M|18| 0|
|10029 |Berg, Andy |200|2312|Test |67000.00 |10|Jan 01 1967|M| |M|15| 0|
|10023 |Ayer, John |300|4432|Accounting |52000.00 | 8|Jan 01 1965|M| |M|13| 0|
```
Use this file as the input source for a MultiLoad import task.

**Note:** The file example uses field delimiter characters (|) for readability, to help distinguish one field from another.

Field delimiter characters are not required in files.

**Writing the MultiLoad Job Script**

Create and save a MultiLoad job script file named `insert.mload` that loads the six-record `insert.data` file into the Employee table:

```plaintext
.logtable inslogtable;
.logon tdpid/username,password;

CREATE TABLE employee (  
    EmpNo SMALLINT FORMAT '9(5)' BETWEEN 10001 AND 32001 NOT NULL,  
    Name VARCHAR(12),  
    DeptNo SMALLINT FORMAT '999' BETWEEN 100 AND 900 ,  
    PhoneNo SMALLINT FORMAT '9999' BETWEEN 1000 AND 9999,  
    JobTitle VARCHAR(12),  
    Salary DECIMAL(8,2) FORMAT 'ZZZ,ZZ9.99' BETWEEN 1.00 AND 999000.00 ,  
    YrsExp BYTEINT FORMAT 'Z9' BETWEEN -99 AND 99 ,  
    DOB DATE FORMAT 'MMMbDDbYYYY',  
    Sex CHAR(1) UPPERCASE,  
    Race CHAR(1) UPPERCASE,  
    MStat CHAR(1) UPPERCASE,  
    EdLev BYTEINT FORMAT 'Z9' BETWEEN 0 AND 22,  
    HCap BYTEINT FORMAT 'Z9' BETWEEN -99 AND 99 )  
UNIQUE PRIMARY INDEX( EmpNo )  
INDEX( Name );

.begin import mload tables employee;  
.layout inslayout;  
.field EmpNo 2 char(9);  
.field Name 12 char(12);  
.field DeptNo 25 char(3);  
.field PhoneNo 29 char(4);  
.field JobTitle 34 char(12);  
.field Salary 47 char(9);  
.field YrsExp 57 char(2);  
.field DOB 60 char(11);  
.field Sex 72 char(1);  
.field Race 74 char(1);  
.field MStat 76 char(1);  
.field EdLev 78 char(2);  
.field HCap 81 char(2);  
.dml label insdml;  
insert into employee.*;

.import infile insert.input  
format text  
layout inslayout  
apply insdml;  
.end mload;  
.logoff;
```
Chapter 1: Overview
A MultiLoad Example

Comments

1. For syntax and descriptions of the following MultiLoad utility commands used in the preceding example, see Chapter 3: “MultiLoad Commands.”

   - BEGIN MLOAD
   - DML LABEL
   - END MLOAD
   - FIELD
   - IMPORT
   - LAYOUT
   - LOGOFF
   - LOGON
   - LOGTABLE

2. The CREATE TABLE statement creates a new table on the Teradata Database:
   - Named employee
   - Containing the following columns:
     - DeptNo
     - DOB
     - EdLev
     - EmpNo
     - HCap
     - JobTitle
     - MStat
     - Name
     - PhoneNo
     - Race
     - Salary
     - Sex
     - YrsExp

   - Indexed by:
     - EmpNo (UNIQUE PRIMARY)
     - Name

   For a description of the CREATE TABLE statement, see SQL Reference: Data Definition Statements.

3. The LAYOUT command and the series of FIELD commands specify each field of the data records that are sent to the Teradata Database. (The insert.input data file that you created in “Creating the Source Data File” on page 24.)

   The specified layout name, inslayout, is referenced in the subsequent MultiLoad IMPORT command.

   Each FIELD command specifies the name, starting position, and data type description for each field in the input data records. In making the field declarations, note that the 1-character and multi-character delimiter fields are optional. They are not required in your example.

4. The DML LABEL command introduces the INSERT statement.

5. The IMPORT command starts the import task, using the insert.input file and the insdml insert statement.
Running the MultiLoad Job

Use the following command to invoke MultiLoad and run the `insert.mload` job script:

```
mload < insert.mload
```

**Note:** The redirection mechanism is required for MultiLoad job scripts on network-attached client systems.

Verifying the Import Task

Use the following BTEQ commands to verify the MultiLoad import task by selecting newly imported data from the Employee table:

```
bteq
    .logon tdpid/username,password
    .width 120
    select * from employee where salary > 65000.00;
    .quit
```

The response indicates that the Employee table was successfully loaded with the data from the `insert.input` file:

```
*** Query completed. 4 rows found. 13 columns returned.
*** Total elapsed time was 6 seconds.

EmpNo  Name       DeptNo  PhoneNo  JobTitle  Salary       YrsExp DOB    Sex  Race  MStat  EdLev  HCap
-------  ---------  ------  -------  ---------  --------- ------  ------  ---  ----  -----  -----  ----
10028    Lee, Sandra  200     5844  Support 77,000.00 4 JAN 01 1971  F          M         18     0
10001    Jones, Bill  100     5376  President 83,000.00 15 JAN 01 1960  M          M         14     0
10002    Smith, Jim   100     4912  Sales  73,000.00 10 JAN 01 1970  M          M         13     1
10029    Berg, Andy   200     2312  Test   67,000.00 10 JAN 01 1967  M          M         15     0
```

Other Alternatives

The MultiLoad example is a simple, straightforward exercise that you can use to quickly create a job script and run MultiLoad. Table 5 explains some of the more significant functional alternatives that MultiLoad provides, and points to where they are described in greater detail.

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Channel-attached Client Systems</td>
<td>The MultiLoad example assumes you are using UNIX or Windows on a network-attached client system. To run the example using VM or MVS on a channel-attached client system, you must use the standard VM EXEC or MVS JCL control statements (FILEDEF and DD) to allocate and create the MultiLoad data sets or files before you invoke the utility. For more information about invoking MultiLoad on channel-attached client systems, see “Invoking MultiLoad” on page 29.</td>
</tr>
<tr>
<td>Using the FastLoad Utility</td>
<td>The MultiLoad example shows the simplest possible MultiLoad job—loading data into an empty table. Although the FastLoad utility can be used to perform the same task, with the job running quicker than with MultiLoad, note that FastLoad works only on empty tables. FastLoad does not support inserting, updating, or deleting data rows in existing tables. For these tasks, you would use MultiLoad.</td>
</tr>
</tbody>
</table>
Performing Insert, Update and Delete Operations

If, instead of loading data into a new table, you wanted to either insert additional rows into an existing table, or update or delete specific rows, you could use the same MultiLoad example by:

- Deleting the CREATE TABLE statement from the MultiLoad job script.
- Specifying a properly formatted existing table in the BEGIN MLOAD command.
- Using the appropriate Teradata SQL INSERT, UPDATE, or DELETE statement after the DML LABEL command in the MultiLoad job script.

MultiLoad also provides a delete task where all rows are selected by one delete clause, rather than by matching individual input records. For more information, see “Import Tasks” on page 59 and “Delete Tasks” on page 62, and the INSERT, UPDATE, and DELETE commands described in Chapter 3: “MultiLoad Commands.”

Using different input file formats

The format of the input data source for the MultiLoad example (insert.input) is TEXT, as specified by the IMPORT command. MultiLoad also supports input data source files with the FASTLOAD, BINARY, TEXT, UNFORMAT, and VARTEXT formats. For a description of supported input file formats, see the IMPORT command in Chapter 3: “MultiLoad Commands.”

Using INMOD Routines

In the MultiLoad example, the utility reads the input data records directly from the specified source file (insert.input). An alternative would be to write an INMOD routine that MultiLoad could call to obtain input records. For example, the INMOD routine could:

- Read and preprocess records from a file, or read data from other database systems.
- Generate or validate data records, or convert data record fields.

In this case, you would use the optional INMOD modulename specification of the IMPORT command to identify the name of your INMOD routine. For more information about using INMOD routines, see “Using INMOD and Notify Exit Routines” on page 69, and the IMPORT command described in Chapter 3: “MultiLoad Commands.”
This chapter provides detailed information about using the Teradata MultiLoad utility. Topics include:

- Invoking MultiLoad
- Terminating MultiLoad
- Restarting a Paused MultiLoad Job
- Programming Considerations
- Using INMOD and Notify Exit Routines
- Writing a MultiLoad Job Script
- Estimating Space Requirements
- Handling MultiLoad Errors
- Monitoring MultiLoad Performance

## Invoking MultiLoad

This section describes requirements and options pertaining to invoking the MultiLoad utility.

### File Requirements

MultiLoad accesses the input data source and the data sets/files or input/output devices described in the following table.

<table>
<thead>
<tr>
<th>Data Set/File or Device</th>
<th>Provides</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>optional specification of MultiLoad utility default values</td>
</tr>
<tr>
<td>standard error</td>
<td>destination for MultiLoad errors</td>
</tr>
<tr>
<td>standard input</td>
<td>MultiLoad commands and Teradata SQL statements that make up the MultiLoad job</td>
</tr>
<tr>
<td>standard output</td>
<td>destination for MultiLoad output responses and messages</td>
</tr>
</tbody>
</table>

When running MultiLoad in interactive mode, the terminal keyboard functions as the standard input device and the display screen is the standard output/error device. When running MultiLoad in batch mode, you must specify a data set or file name for each of these...
functions. The method of doing this varies, depending on the configuration of your client system:

- On network-attached client systems, use the standard redirection mechanism (<\textit{infilename} and >\textit{outfilename}) to specify the MultiLoad files when you invoke the utility.
- On channel-attached client systems, use standard VM EXEC or MVS JCL control statements (FILEDEF and DD) to allocate and create the MultiLoad data sets or files before you invoke the utility.

**In Interactive Mode**

To invoke MultiLoad in interactive mode, enter \textit{mload} at your system command prompt:

\texttt{mload}

MultiLoad displays the following message to begin your interactive session:

\begin{verbatim}
====================================================================
= = MultiLoad Utility Release MLOD.mm.mm.mm.mmm =
= = Platform =
====================================================================
= = Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
= =
====================================================================
\end{verbatim}

where \textit{mm.mm.mm.mmm} is the release level of your MultiLoad utility software.

**In Batch Mode**

This section covers invoking MultiLoad in batch mode on network-attached and channel-attached systems.

For a description of how to read the syntax diagrams used in this book, see Appendix A: "How to Read Syntax Diagrams."

**In Batch Mode on Network-Attached Systems**

Refer to the runtime parameter descriptions in Table 7 on page 33 and use the following syntax to invoke MultiLoad on network-attached client systems:
**In Batch Mode on Channel-Attached MVS Systems**

Refer to the runtime parameter descriptions in Table 6 on page 32 and use the following syntax to invoke MultiLoad on channel-attached MVS client systems:

```
// EXEC TDSMLOAD
MLPARM='BRIEF
  CHARSET=character-set-name
  ERRLOG=filename
  MAXSESS=max-sessions
  MINSESS=min-sessions
  "multiload command"
  RVERSION
```

**In Batch Mode on Channel-Attached VM Systems**

Refer to the runtime parameter descriptions in Table 6 on page 32 and use the following syntax to invoke MultiLoad on channel-attached VM client systems:

```
EXEC MLOAD
BRIEF
  CHARSET=character-set-name
  ERRLOG=filename
  MAXSESS=max-sessions
  MINSESS=min-sessions
  "multiload command"
  RVERSION
```

**Note:** On VM, use the following statement before the EXEC MLOAD statement:

"GLOBAL LOADLIB DYNAMIC"

**Runtime Parameters**

This section describes the runtime parameters used by MultiLoad.

Table 6 lists MultiLoad runtime parameters for channel-attached configurations.
### Table 6: Runtime Parameters (Channel-Attached Configurations)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BRIEF</strong></td>
<td>Reduced print output runtime parameter that limits MultiLoad printout to the minimal information required to determine the success of the job:</td>
</tr>
<tr>
<td></td>
<td>- Logon/logoff information</td>
</tr>
<tr>
<td></td>
<td>- Candidate records</td>
</tr>
<tr>
<td></td>
<td>- Insert, update, and delete results</td>
</tr>
<tr>
<td></td>
<td>- Error table counts</td>
</tr>
<tr>
<td><strong>CHARSET</strong></td>
<td>Character set specification for the MultiLoad job</td>
</tr>
<tr>
<td></td>
<td>You can specify any character set installed on the Teradata server, by its character set name. For a listing of character set names, see “Character Set Specification” on page 54.</td>
</tr>
<tr>
<td></td>
<td><strong>Caution:</strong> The character set specification remains in effect for the entire MultiLoad job, even if the Teradata Database resets, causing the MultiLoad job to be restarted. However, the character set specification does not remain in effect if the client system fails, or if you cancel the MultiLoad job. In these cases, when you resubmit the job, you must use the same character set specified on the initial job. If you use a different character set specification when you resubmit such a job, the data loaded by the restarted job will not appear the same as the data loaded by the initial job. If you do not enter a character set specification, the default is the character set specified for the Teradata Database when you invoke MultiLoad. <strong>Note:</strong> For more information about specifying and using character sets, see “Character Set Specification” on page 54.</td>
</tr>
<tr>
<td><strong>ERRLOG</strong></td>
<td>Alternate file specification for MultiLoad error messages</td>
</tr>
<tr>
<td></td>
<td>Specifying an alternate file name produces a duplicate record of all MultiLoad error messages.</td>
</tr>
<tr>
<td></td>
<td>On channel-attached client systems, the alternate file specification is limited to eight characters and:</td>
</tr>
<tr>
<td></td>
<td>- On MVS, it must be a DD name defined in the JCL</td>
</tr>
<tr>
<td></td>
<td>- On VM, it must be an existing file definition (FILEDEF)</td>
</tr>
<tr>
<td></td>
<td>There is no default error log file name specification.</td>
</tr>
<tr>
<td><strong>MAXSESS</strong></td>
<td>Maximum number of MultiLoad sessions logged on to the Teradata Database Maximum specification must be greater than zero and no more than the total number of AMPs on your system.</td>
</tr>
<tr>
<td></td>
<td>Default is one session for each AMP.</td>
</tr>
<tr>
<td><strong>MINSESS</strong></td>
<td>Minimum number of MultiLoad sessions required to run the job Minimum specification must be greater than zero.</td>
</tr>
<tr>
<td></td>
<td>Default is one.</td>
</tr>
</tbody>
</table>
Chapter 2: Using MultiLoad

Invoking MultiLoad

Table 7 lists MultiLoad runtime parameters for network-attached configurations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“multiload command”</td>
<td>Invocation option that can signify the start of a MultiLoad job on a network-attached system. Since you can only specify one multiload command as the multiload command specification, it is usually a RUN FILE command that specifies the file containing your MultiLoad job script. You must use the FILEDEF or DD control statements to specify the input and output files before you invoke the utility.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVERSION</td>
<td>Display version number and stop.</td>
</tr>
</tbody>
</table>

Table 6: Runtime Parameters (Channel-Attached Configurations) (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| -b                 | Reduced print output runtime parameter that limits MultiLoad printout to the minimal information required to determine the success of the job:  
  • Logon/logoff information  
  • Candidate records  
  • Insert, update, and delete results  
  • Error table counts |

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| -c character-set-name| Character set specification for the MultiLoad job  
You can specify any character set installed on the Teradata server, by its character set name. For a listing of character set names, see “Character Set Specification” on page 54.  
**Caution:** The character set specification remains in effect for the entire MultiLoad job, even if the Teradata Database resets, causing the MultiLoad job to be restarted. However, the character set specification does not remain in effect if the client system fails, or if you cancel the MultiLoad job. In these cases, when you resubmit the job, you must use the same character set specified on the initial job. If you use a different character set specification when you resubmit such a job, the data loaded by the restarted job will not appear the same as the data loaded by the initial job.  
If you do not enter a character set specification, the default is the character set specified for the Teradata Database when you invoke MultiLoad.  
**Note:** For more information about how to specify and work with character sets, see “Character Set Specification” on page 54. |
Table 7: Runtime Parameters (Network-Attached Configurations) (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i scriptencoding</td>
<td>Encoding form of the job script</td>
</tr>
</tbody>
</table>

If this parameter is not specified and the client character set is UTF16, MultiLoad interprets the job script to UTF16. If character-type data is also specified in the script, MultiLoad converts the string literals and the corresponding field in the import data to the same character set before comparing or concatenating them. (String literals are specified with .APPLY...WHERE...; LAYOUT...CONTINUEIF...; FIELD...NULLIF...; FIELD...||...commands.)

Valid encoding options are:

- UTF8
- UTF16-BE
- UTF16-LE
- UTF16

The specified encoding character set applies to all script files included by the .RUN FILE commands.

The UTF16 or UTF8 Byte Order Mark (BOM) can be present or absent in the script file.

When UTF16 BOM is present and 'UTF16' is specified, MultiLoad interprets the script according to the endianness indicated by the UTF16 BOM. When the UTF16 BOM is not present, MultiLoad interprets the script according to the endianness indicated by the encoding option.

**Note:** UTF8 BOM is not supported on the MVS platform when using access modules or data files.
### Table 7: Runtime Parameters (Network-Attached Configurations) (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-u <code>outputencoding</code></td>
<td>Encoding form of the job output</td>
</tr>
<tr>
<td></td>
<td>The parameter is valid only when the UTF16 client character set is used. If the client character set being used is not UTF16 and this parameter is specified, MultiLoad reports an error and terminates. When this parameter is used, place it in front of other runtime parameters to ensure that the whole job output is printed in the desired encoding form. If is not placed ahead of other runtime parameters when invoking the job, a warning message will be printed. Available output encoding options are:</td>
</tr>
<tr>
<td></td>
<td>• UTF16-BE</td>
</tr>
<tr>
<td></td>
<td>• UTF16-LE</td>
</tr>
<tr>
<td></td>
<td>• UTF16</td>
</tr>
<tr>
<td></td>
<td>UTF16-BE instructs MultiLoad to print the job output in big endian UTF16 encoding scheme. UTF-LE instructs MultiLoad to print the job output in little endian UTF16 encoding scheme. On big endian client systems, UTF16 instructs MultiLoad to print the job output in big endian UTF16 encoding scheme. On little endian client systems, UTF16 instructs MultiLoad to print the job output in little endian UTF16 encoding scheme. The UTF16 BOM is not printed as a part of job output. If MultiLoad console output needs to be redirected to a log file on network platforms, “-u <code>outputencoding</code>” must be specified.</td>
</tr>
<tr>
<td>-e <code>filename</code></td>
<td>Alternate file specification for MultiLoad error messages</td>
</tr>
<tr>
<td></td>
<td>Specifying an alternate file name produces a duplicate record of all MultiLoad error messages. There is no default error log file name specification.</td>
</tr>
<tr>
<td>-M <code>max-sessions</code></td>
<td>Maximum number of MultiLoad sessions logged on to the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>Maximum specification must be greater than zero and no more than the total number of AMPs on your system. Default is one session for each AMP.</td>
</tr>
<tr>
<td>-N <code>min-sessions</code></td>
<td>Minimum number of MultiLoad sessions required to run the job</td>
</tr>
<tr>
<td></td>
<td>Minimum specification must be greater than zero. Default is one.</td>
</tr>
<tr>
<td>-t <code>multiload command</code></td>
<td>Invocation option that can signify the start of a MultiLoad job on a network-attached system</td>
</tr>
<tr>
<td></td>
<td>Since you can only specify one multiload command as the multiload command specification, it is usually a RUN FILE command that specifies the file containing your MultiLoad job script.</td>
</tr>
</tbody>
</table>
Chapter 2: Using MultiLoad

Invoking MultiLoad

Note: For sample JCL listings, commands, and output samples for the invocation options, see the invocation examples in Appendix B: “Invocation Examples.”

VM Example

The following EXEC procedure invokes MultiLoad on a channel-attached VM client system:

```c
/* Run the MultiLoad program on VM. */
/* Accepts input from terminal (sysin) */
/* Sends output to terminal (sysprint) */
/* Before running this EXEC, the user */
/* must issue the VM LINK and ACCESS */
/* commands to link to the CLI */
/* minidisk and to link to the Sas C */
/* runtime minidisk. */

"GLOBAL LOADLIB DYNAMC"
"GLOBAL TXTLIB CLI LC370 "
"FILEDEF SYSPRINT TERMINAL (LRECL 84 RECFM V"
"FILEDEF SYSIN TERMINAL (LRECL 84 RECFM V"

say "Running MULTILOAD"
mload
exit
```
**MVS Example**

The following procedure invokes MultiLoad on a channel-attached MVS client system:

```plaintext
//MLOADRUN JOB 1,'MULTILOAD',
//   MSGCLASS=X,
//   NOTIFY=MLOAD,
//   TYPRUN=SCAN,
//   CLASS=A
/*/ROUTE PRINT TSO
//**************************************************
//* *
//* MULTILOAD PROC *
//* *
//**************************************************
/*
//MLOAD PROC PRM=,INPUT=
//MLOADSTP EXEC PGM=MLOAD,PARM='&PRM'
//STEPLIB DD DISP=SHR,DSN=TERADATA.APPLOAD
// DD DISP=SHR,DSN=TERADATA.TRLOAD
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//SYSIN DD DISP=SHR,DSN=&INPUT
// PEND
//**************************************************
//* *
//* RUN MULTILOAD *
//* *
//**************************************************
//MLOADRUN EXEC PROC=MLOAD
//MLOADSTP.SYSIN DD DATA,DLM=##
.SET X TO 'FSADFDFSDFDS';
.DISPLAY '&X' TO FILE OUTPUT; ##
//OUTPUT DD SYSOUT=* 
```

**UNIX and Windows Examples**

Following are examples of ways to invoke MultiLoad on network-attached UNIX and Windows systems:

- **mload</home/mluser/tests/test1>/home/mluser/tests/out1**
  
  This command specifies both an input file and an output file:
  - `/home/mluser/tests/test1` is the input file that provides the MultiLoad job script.
  - `/home/mluser/tests/out1` is the destination file for output data.

- **mload</home/mluser/tests/test1**
  
  This command specifies only an input file. In this case, the output is written to the standard output device, which is usually your terminal.

- **mload**
  
  This command specifies neither an input nor an output device. In this case, your terminal provides both the command input and the output data destination.

- **mload -r '.RUN FILE mld.startup;'**
  
  This command uses the `-r` invocation option to specify the MultiLoad RUN FILE command. In this case, the MultiLoad job script is in the `mld.startup` file.
Terminating MultiLoad

This section covers methods of termination and other topics related to terminating MultiLoad.

The following are ways to terminate MultiLoad.

- **Normal Termination**
- **Abort Termination**

Either way ends the MultiLoad sessions and logs you off the Teradata Database. A normal termination, however, does so in an orderly, controlled fashion, and returns messages indicating the status of the MultiLoad job. An abort termination does not.

**Warning:** If you abort a MultiLoad job during the data acquisition phase and inadvertently restart the job with different data, MultiLoad does not recognize that this new data is not valid. You must make sure the same data is used to avoid corrupting the target table.

**Caution:** Make sure to terminate any pending or paused jobs prior to upgrading to a new release of the Teradata Database. Database upgrades can effect changes to the MultiLoad worktable, if a pending job created using a different release restarts after an upgrade, errors might result.

**Normal Termination**

Use the MultiLoad LOGOFF command in your MultiLoad job script to terminate the utility normally on both network-attached and channel-attached client systems.

```
.LOGOFF [retcode ];
```

MultiLoad logs off all sessions with the Teradata Database and returns a status message indicating:

- The total processor time that was used
- The job start and stop date/time
- The highest return code (termination return code) that was encountered:
  - 0 if the job completed normally
  - 4 if a warning condition occurred
  - 8 if a user error occurred
  - 12 if a fatal error occurred
  - 16 if no message destination is available

MultiLoad also:

- Either maintains or drops the restart log table, depending on the success or failure of the job.
- If specified, returns the optional `retcode` value to your client operating system.
For more information about return codes and the conditions that maintain or drop the restart log table, see “LOGOFF” on page 153.

Abort Termination

The procedure for aborting a MultiLoad job depends on whether the utility is running on a network-attached or a channel-attached client system.

To abort a MultiLoad job running on a network-attached client system

✔ Press the Control + C key combination three times on your workstation keyboard.

Note: Some session(s) of the aborted MultiLoad job may remain active until the gateway time-out period expires. Error “2574 Operation not allowed: %DBID.%FSTR is being MLoaded.” may occur if the same MultiLoad job is resubmitted within the time-out period. If this happens, wait until the gateway time out has expired before resubmitting the same MultiLoad job.

To abort a MultiLoad job running on a channel-attached client system

✔ Cancel the job from the client system console.

After Terminating a MultiLoad Job

After terminating a MultiLoad job, you must either:

- Restart the job and allow it to run to completion.
  or
- Reinitialize and restore access to the target tables.

The conditions and procedures for performing these tasks depend on whether the job terminated during the application phase and whether you want to restart and continue or abandon the job.

- If the MultiLoad job terminated before or after the application phase, then to restart and continue the job, use the After a Job Abort or Client System Failure procedure in “Restarting a Paused MultiLoad Job” on page 42.
- If the MultiLoad job terminated during the application phase, then you must restart the job and let it run to completion. Use the After a Job Abort or Client System Failure procedure in “Restarting a Paused MultiLoad Job” on page 42.

Reinitializing After Terminating a MultiLoad Job

When a MultiLoad job terminates either before or after the application phase, and you want to abandon the job, use the following procedure to restore access to the target tables.
1. Log on to BTEQ with a user ID that has drop table privileges on the Teradata Database associated with the aborted MultiLoad job.

2. Use a RELEASE MLOAD statement specifying all of the target tables identified in the TABLES clause of the BEGIN MLOAD command for the aborted MultiLoad job.

   **Note:** An unsuccessful RELEASE MLOAD statement indicates that the MultiLoad job was in the application phase when it terminated. In this case, you cannot abandon the MultiLoad job. You must restart the job and let it run to completion. Use the After a Job Abort or Client System Failure procedure in “Restarting a Paused MultiLoad Job” on page 42.

   Further, if the tables had permanent journals, to recover the table, you must use an insert or select operation (with locking for access) to copy data from the target table into a temporary table. Then drop the target table and rename the temporary table with the original name of the target table.

3. After successfully executing the RELEASE MLOAD statement, use Teradata SQL DROP TABLE statements to drop the work tables and error tables identified in the WORKTABLES and ERRORTABLES options of the BEGIN MLOAD command for the aborted MultiLoad job.

   **Note:** If the tables were not explicitly defined, see the SYSPRINT output following the UTY0812/UTY0815/UTY0817 message sequence.

4. Use a DROP TABLE statement to drop the restart log table identified in the LOGTABLE command for the aborted MultiLoad job.

### Implications of Dropping Required MultiLoad-Created Tables

MultiLoad creates four tables that are required for restarting a paused MultiLoad job:

- Restart Log Table
- Work Table
- Acquisition Error Table
- Application Error Table

If you drop any of these tables while a MultiLoad job is paused, you will not be able to restart the job normally, and the target tables can be left corrupted.

In some limited circumstances, you might be able to restart such a job using special procedures but, in most cases, do not drop these tables for a paused job without careful analysis of the situation.

The following topics describe the operational implications of having dropped the required restart tables for a paused MultiLoad job.

### Restart Log Table

If you drop the restart log table, you will not be able to restart the paused job normally by simply resubmitting it. The recovery procedure depends on whether:

- The job entered the pause mode before or during the application phase
- You have dropped any of the other required restart tables
If the job paused before the application phase, then you can restart the job by:

- Dropping the work table and error tables
- Using BTEQ to enter a RELEASE MLOAD statement
- Resubmitting the paused job as though it were a new one

If the job paused during the application phase, and the work and error tables have not been dropped, then you can restart the job. If the work and error tables have been dropped, you cannot restart the job and you will have to restore the target tables.

In either case, follow the After a Job Abort or Client System Failure instructions in “Restarting a Paused MultiLoad Job” on page 42.

**Work Table**

If you drop the work table, you will not be able to restart the paused job. The recovery procedure depends on whether the job entered the pause mode before or during the application phase. Also keep in mind:

- If the operations were only INSERT, then it may be possible to resubmit the paused job as a new job, although duplicate unique primary index or duplicate row errors may occur.
- If the operations were only DELETE, then it may be possible to resubmit the paused job as a new job, although missing row errors may occur.

**Note:** Be very careful with MULTISET tables. This procedure could produce the wrong number of duplicate rows.

- If the operations were only UPDATE, then it may or may not be possible to resubmit the paused job as a new job. If the update operation was setting the values of one or more columns without reference to original values in any columns, resubmitting the job will usually succeed. If, however, the update operation referenced any original column values (for example, to increment a column value, or multiply a column value by some constant), then resubmitting the job will fail.
- If the operations were a combination of INSERT, DELETE, UPDATE, then you must consider the collective effect of the insert, delete, and update operations.

**Acquisition Error Table**

If you drop the acquisition error table, you will not be able to restart the paused job without special procedures. The recovery procedure depends on whether the job entered the pause mode before or during the application phase.

- If the job paused before the application phase, then you can recover by:
  - Dropping the restart log table, the work table, and the application error table
  - Using BTEQ to enter a RELEASE MLOAD statement
  - Resubmitting the job as a new one
- If the job paused during the application phase, then you may be able to recover by creating a “dummy” acquisition error table. You will need technical assistance from Teradata technical support to do this, and even if the recovery succeeds, you will have lost any information about prior acquisition phase errors.
Chapter 2: Using MultiLoad

Restarting a Paused MultiLoad Job

**Application Error Table**

If you drop the application error table, you will not be able to restart the paused job without special procedures. The recovery procedure depends on whether the job entered the pause mode before or during the application phase.

- **If the job paused before** the application phase, then you can recover by:
  - Dropping the restart log table, the work table, and the acquisition error table
  - Using BTEQ to enter a RELEASE MLOAD statement
  - Resubmitting the job as a new one
    
    In some cases, you may be able to recover by creating a “dummy” application error table.

    You will need technical assistance from Teradata technical support to do this.

- **If the job paused during** the application phase, you may be able to recover by creating a “dummy” application error table. You will need technical assistance from Teradata technical support to do this, and even if the recovery succeeds, you will have lost any information about prior application phase errors.

**Restarting a Paused MultiLoad Job**

A paused MultiLoad job is one that was halted, before completing, during the acquisition phase of the MultiLoad operation. The paused condition can be intentional, or the result of a system failure or error condition.

You can pause a MultiLoad job intentionally by using a PAUSE ACQUISITION command between the BEGIN MLOAD command and the END MLOAD command in the MultiLoad job script.

Unintentional conditions that can pause a MultiLoad job include:

- A MultiLoad job script error
- An unrecoverable I/O error
- A down AMP recovery
- A Teradata Database restart
- A Teradata Database system reconfiguration
- A job abort or client system failure
- An application processor reset condition

The following topics describe the various pause conditions, the factors affecting MultiLoad restart operations, and, where appropriate, the procedure for restarting the MultiLoad job.

**Warning:** If you abort a MultiLoad job during the data acquisition phase and erroneously restart the job with different data, MultiLoad does not recognize that the new data is not valid. You must ensure that the same data is used to avoid corrupting the target table.
After Using the PAUSE ACQUISITION Command

To restart a MultiLoad job that was paused by a PAUSE ACQUISITION command in the MultiLoad job script:

1. Remove the PAUSE ACQUISITION command from the MultiLoad job script.
2. Invoke MultiLoad to restart the job. The MultiLoad utility:
   a. Reestablishes sessions with the Teradata Database
   b. Reads the restart log table to determine the restart point
   c. Resumes processing the MultiLoad job script

After a MultiLoad Job Script Error

When MultiLoad encounters an error in your job script, it generates a diagnostic error message and stops with a nonzero return code. At this point, you can modify the script to correct the error and resubmit the MultiLoad job. The utility resumes processing at the statement following the last one that completed successfully.

When correcting script errors, you can make changes at or after the indicated error. MultiLoad does not repeat the commands that executed successfully, but the job will fail, with additional error messages, if the utility detects changes before such errors.

**Note:** Any SQL statements you use outside the BEGIN MLOAD and END MLOAD commands are not part of a MultiLoad job. If one of these statements fails and generates an error, it is not treated as a terminating error or a user error. Instead, MultiLoad continues processing after setting the system variable SYSRC to the last error code. If the MultiLoad job is processed successfully, the restart log table is dropped at the end of the job.

After an Unrecoverable I/O Error

The MultiLoad utility automatically attempts a restart operation after encountering an unrecoverable I/O error while accessing a fallback type table.

Status messages on screen 7 of the database operator’s console reflect the progress of the restart operation, and some I/O error conditions can force an AMP to become a nonparticipant in the MultiLoad job. In this case, the MultiLoad job is stopped with error messages such as:

```
AMP XXX-X Starting rebuild of db.tbl
AMP XXX-X AMP marked as non-participant for - db.tbl
AMP XXX-X Completed rebuild of table db.tbl
```

Additional error messages indicate errors that may occur during the restart operation.

After a Down AMP is Recovered

When a down AMP with a fallback type MultiLoad worktable is recovered, status messages on screen 7 of the database operator’s console may indicate that special procedures might be required to recover the table. For example, this happens if the AMP goes down during the MultiLoad application phase, as indicated by messages such as:
Restarting a Paused MultiLoad Job

If the AMP goes down before the MultiLoad job enters the application phase, the recovery progress is indicated by messages such as:

AMP XXX-X Starting rebuild of db.wt_tbl
AMP XXX-X Table in MLOAD apply phase, check associated table - db.wt_tbl
AMP XXX-X Completed rebuild of table db.wt_tbl

After a Teradata Database Restart

If the Teradata Database restarts while a MultiLoad job is running, MultiLoad automatically resumes processing after the normal recovery operation. You do not have to intervene for a normal Teradata Database restart/recovery operation.

If, however, the work tables or error tables are missing, MultiLoad aborts the restart attempt with an error message.

After a Teradata Database System Reconfiguration

If you run the Reconfiguration program (a Teradata Database support utility) while a MultiLoad job is active, the associated target tables, work tables, and possibly the error tables, are not redistributed. The MultiLoad restart log table is redistributed.

When MultiLoad restarts after the reconfiguration completes, the BEGIN MLOAD command fails with the following error message:

UTY0805 DBC failure, 2563: MLOAD not allowed: DBC had been reconfigured.

At this point, the MultiLoad tables cannot be recovered and the target tables must be reloaded.

Caution: Do not reconfigure the Teradata Database during an active MultiLoad job.

After a Job Abort or Client System Failure

Restarting a MultiLoad job that was either aborted or stopped after a client system failure or restart depends on whether the MultiLoad job stopped during the application phase.

- If the job was stopped before or after the application phase, then you can restart the MultiLoad job as is, with no changes to the job script. MultiLoad uses the entries in the restart log table to determine its stopping point and resumes processing there.
  The BEGIN MLOAD command is the first step in a multiple-step job. If you re-execute commands that modify tables or databases involved in the MultiLoad job, you may corrupt them.

Caution: Do not modify or drop the MultiLoad restart log table or any of the work tables or error tables before restarting the MultiLoad job.

- If the job was stopped during the application phase, you must restart the job and let it run to completion. First, resolve the problem associated with the MultiLoad job abort or client system failure, and then restart the MultiLoad job.
Note, however, that the restart procedure depends on the severity of the situation that stopped the job:

- If the restart log table has been dropped, use the CREATE TABLE statement to create a new one, and use the OVERRIDE parameter to resubmit the MultiLoad job, as in the following MVS example:

  ```
  // EXEC MLOAD, PARM='OVERRIDE'
  ```

- If any of the target tables, work tables, or error tables are missing or corrupted, drop the target tables from the Teradata Database and restore them from your backup files. In this case, you must also drop the remaining work tables, error tables, and restart log tables, and resubmit the MultiLoad job as a new job.

- If the MultiLoad job script has been lost, use the following abbreviated script to restart the job:

  ```
  .LOGON tdpid/userid/password;
  .LOGTABLE logtablename;
  .BEGIN MLOAD . . . ;
  .END MLOAD;
  .LOGOFFs
  ```

The BEGIN MLOAD command must identify all of the target tables, work tables, and error tables of the original job, and they must all be intact and not corrupted. When you submit the abbreviated script, MultiLoad uses the checkpoint markers in the work tables to restart the task at the point it stopped in the application phase.

**After an Application Processor Reset Condition**

When your MultiLoad job is interrupted by a resetting application processor on the Teradata Database, the restart alternatives depend on the environment in which the utility is running:

- On the *resetting* application processor. In this case, your MultiLoad job is halted and must be manually restarted.

- On a *nonresetting* application processor. In this case, your MultiLoad job may or may not be halted, depending on whether it has sessions connected through the resetting AP. If your MultiLoad job has sessions connected through the resetting AP, the utility automatically:
  - Logs off all sessions.
  - Logs them back on.
  - Rolls back to the most recent checkpoint.
  - Resumes processing.

If the MultiLoad job does not have sessions connected through the resetting AP, the utility is not affected by the AP reset condition.

**Note:** The increased session loading caused by reconnecting other sessions through the resetting AP may degrade the system response time. This situation is similar to a database restart or network failure.
• On a channel-attached client system with AP reset containment enabled, the MultiLoad job is halted, but does not need to be manually restarted. MultiLoad automatically:
  • Logs off all sessions.
  • Logs them back on.
  • Rolls back to the most recent checkpoint.
  • Resumes processing.

**Programming Considerations**

This topic describes the things to consider when designing and coding MultiLoad job scripts.

**MultiLoad Configuration File**

You can use a MultiLoad configuration file to set the initial default values for the following operating parameters when you invoke MultiLoad:

• BRIEF
• CHARSET
• ERRLOG
• MATCHLEN
• MAXSESS
• MINSESS
• DATAENCRYPTION=ON or OFF

The values that you specify in the MultiLoad configuration file override the internal utility default values for these parameters.

The configuration file parameters can themselves be overridden by the corresponding runtime parameters, shown earlier in this chapter. (There is no MATCHLEN runtime parameter.) The order of preference for these parameters, from highest to lowest, is:

1—Runtime parameters
2—MultiLoad script
3—MultiLoad configuration file specifications
4—MultiLoad default values

**Configuration File Name and Location**

On network-attached systems, the MultiLoad configuration file must be named:

`mloadcfg.dat`

and it must be located in either:

• The directory from which you launched MultiLoad
  or
• The directory specified in the MLOADLIB environment variable
On channel-attached systems, the DD statement for the MultiLoad configuration file must be labeled:

MLOADCFG

**Configuration File Contents**

The MultiLoad configuration file can have up to eight entries, one for each parameter:

BRIEF=on/off
OFORMAT=outputencoding
IFORMAT=scriptencoding
CHARSET=character-set-name
ERRLOG=filename
MATCHLEN=on/off
MAXSESS=max-sessions
MINSESS=min-sessions

where

- *on/off* for BRIEF configures the reduced print output specification for the MultiLoad job.
- *outputencoding* for OFORMAT is the character set specification for the screen output. Valid values are UTF16, UTF16-LE, and UTF16-BE. The OFORMAT entry is before the IFORMAT or CHARSET parameters. This parameter can be used on network-attached systems only.
- *scriptencoding* for IFORMAT is the character set specification for the job script file. Valid values are UTF16, UTF16-LE, and UTF16-BE. This parameter can be used on network-attached systems only.
- *character-set-name* is the character set specification for the MultiLoad job.
- *filename* is the alternate file specification for MultiLoad error messages.
- *on/off* for MATCHLEN configures the record length validation option for the MultiLoad job. When you enable MATCHLEN, MultiLoad verifies that the record length of the import data is the same as the layout record length specified by the IMPORT command.
- *max-sessions* is the MAXSESS specification for the maximum number of MultiLoad sessions logged on to the Teradata Database.
- *min-sessions* is the MINSESS specification for the minimum number of MultiLoad sessions required to run the job.

Your MultiLoad configuration file can also have comment statements preceded by a number sign (#) character.

For detailed descriptions of the specifications listed here, see “Runtime Parameters” on page 31.

**Configuration File Processing**

MultiLoad automatically checks for a configuration file each time you enter the invocation command. Upon locating a configuration file, the utility sets the defaults as specified, produces the appropriate output messages, and begins processing the MultiLoad job.
If the configuration file cannot be opened, or if MultiLoad encounters syntax errors in the file, the utility produces an output message, disregards the error condition, and begins processing the MultiLoad job. An invalid configuration file entry will not abort the MultiLoad job.

If there is no configuration file, the utility begins processing the MultiLoad job without an error indication. The configuration file is an optional feature of MultiLoad, and its absence is not considered to be an error condition.

**MultiLoad Command Conventions**

The following describes the command conventions to observe in MultiLoad job scripts.

**Conditional Expressions**

Conditional expressions return a value of:

- 0 if the condition evaluates to FALSE
  or
- 1 if the condition evaluates to TRUE

With the following exceptions, MultiLoad handles conditional expressions as described in the Teradata SQL reference documentation:

- The field names of records from an external data source or a MultiLoad utility variable are equivalent to column names in Teradata SQL conditional expressions.
- The LIKE operator is not supported in logical expressions that make up a conditional expression. (The NOT IN operator *is* supported.)
- The following elements are not supported in arithmetic expressions that make up logical expressions:
  - The exponential operator
  - Aggregate operators
  - Arithmetic functions

**Operators**

Do not use words that are logical operators as keywords, such as:

- `AND`  `IN`  `MOD`
- `BETWEEN`  `IS`  `NE`
- `EQ`  `LE`  `NOT`
- `GE`  `LIKE`  `NULL`
- `GT`  `LT`  `OR`

**Record Mode Load Anomaly**

When loading data in Record Mode into a NULLABLE DATE field, if the source data is a binary integer of value zero, then the Teradata Database sets the field to NULL, not to zero.
Reserved Words
Commands that are supported by MultiLoad do not use reserved words, except:

- Those that are operators
- Where a specific expression is allowed

Though there is no specific restriction against doing so, avoid using the following as variable names:

- MultiLoad command keywords
- Teradata SQL reserved words

Table Column Names
The column names of the table specified by MultiLoad TABLE command must be standard Teradata SQL column names that are not enclosed in quotation marks.

Invalid column names produce errors:

- Embedded blanks cause a syntax error
- Invalid characters cause an invalid name error
- Reserved words cause a syntax error that identifies the reserved word

Data Conversion Factors
You can use the MultiLoad FIELD command to change the data type specification of source data before inserting it in the MultiLoad table on the Teradata Database. However, you are limited to one type conversion per column. For details about data types and data conversion, see SQL Reference: Data Types and Literals.

Note: You cannot use MultiLoad to define a column with an arithmetic expression. For example, MultiLoad will not calculate a monthly salary column from yearly salary data.

Variables
MultiLoad supports the types of variables described in the topics that follow, including predefined system variables, data and time variables, row count variables, and utility variables.

Predefined System Variables
The supported predefined system variables are listed in Table 8. Note that because they all begin with the &SYS prefix, avoid using this prefix in your user-defined utility variables. Note also that you can only reference the system variables; you cannot modify them.

Table 8: System Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;SYSAPPLYCNT [n]</td>
<td>Count of apply conditions satisfied for import file n. If n is not specified, the default is cumulative count of apply conditions satisfied (all import).</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&amp;SYSDATE</td>
<td>Eight-character date in \textit{yy/mm/dd} format.</td>
</tr>
<tr>
<td>&amp;SYSDATE4</td>
<td>Ten-character date in \textit{yyyy/mm/dd} format.</td>
</tr>
<tr>
<td>&amp;SYSDAY</td>
<td>Three-character uppercase day-of-week specification: MON, TUE, WED, THU, FRI, SAT, or SUN.</td>
</tr>
<tr>
<td>&amp;SYSDELCNT ([n])</td>
<td>Number of rows deleted from the (n)th target table. If (n) is not specified, the default is the first target table. The maximum value of (n) is 5.</td>
</tr>
<tr>
<td>&amp;SYSETCNT ([n])</td>
<td>Number of rows in the ET table for the (n)th target table. MultiLoad uses the ET table, also called the Acquisition Phase error table, to store data errors found during the acquisition phase of a MultiLoad import task. If (n) is not specified, the default is the first target table. The maximum value of (n) is 5.</td>
</tr>
<tr>
<td>&amp;SYSINSCNT ([n])</td>
<td>Number of rows inserted in the (n)th target table. If (n) is not specified, the default is the first target table. The maximum value of (n) is 5.</td>
</tr>
<tr>
<td>&amp;SYSNOAPLYCNT ([n])</td>
<td>Count of records not applied for import file (n). If (n) is not specified, the default is cumulative count of records not applied (all import files).</td>
</tr>
<tr>
<td>&amp;SYSOS</td>
<td>Client operating systems: \begin{itemize} \item For MVS: MVS, MVS/SP, MVS/ESA \item For VM: VM/SP, VM/XA SP, VM/HPO, VM/XA, VM/ESA, VS1 \item UNIX \item Win32 \end{itemize}</td>
</tr>
<tr>
<td>&amp;SYSRC</td>
<td>Completion code of the last response from the Teradata Database.</td>
</tr>
<tr>
<td>&amp;SYSRCDCNT ([n])</td>
<td>Count of records read from import file (n). If (n) is not specified, the default is the cumulative count of records read (all import files).</td>
</tr>
<tr>
<td>&amp;SYSRJCTCNT ([n])</td>
<td>Count of candidate records rejected from import file (n). If (n) is not specified, the default is cumulative count of candidate records rejected (all import).</td>
</tr>
<tr>
<td>&amp;SYSTIME</td>
<td>Eight-character time in \textit{hh:mm:ss} format.</td>
</tr>
<tr>
<td>&amp;SYSUPDCNT ([n])</td>
<td>Number of rows updated in the (n)th target table. If (n) is not specified, the default is the first target table. The maximum value of (n) is 5.</td>
</tr>
<tr>
<td>&amp;SYSUSER</td>
<td>Client system dependent: \begin{itemize} \item VM user\textit{id} \item MVS batch user\textit{id}. (MVS batch returns user\textit{id} only when a security package such as RACF, ACF2, or Top Secret has been installed.) \end{itemize}</td>
</tr>
</tbody>
</table>
Chapter 2: Using MultiLoad
Programming Considerations

Date and Time Variables
The four date and time variables reflect the time when MultiLoad begins execution:

- &SYSDAY
- &SYSDATE
- &SYSDATE4
- &SYSTIME

The original values are not maintained after a MultiLoad restart operation.

Note: Do not reference the values in numeric operations since the values are all character data types.

Row Count Variables
The five row count variables, which are updated for each MultiLoad task, allow you to query the insert, update, and delete row counts and the error table counts for each target table:

- &SYSDELCNT[n]
- &SYSETLCNT[n]
- &SYSINSCNT[n]
- &SYSUPDCNT[n]
- &SYSUVCNT[n]

The values are stored in MultiLoad restart log table and are restored after a client system or Teradata Database restart.

You must reference these variables after the END MLOAD command and before any subsequent BEGIN MLOAD command in the MultiLoad job script.

Utility Variables
MultiLoad supports utility variables that you set with either the SET command or the ACCEPT command.

MultiLoad also predefines some utility variables that provide information about the MultiLoad support environment at execution time.
Utility variable names must:

- Begin with the ampersand character (&) if you want to use variable substitution
- Comply with the rules for Teradata SQL column names, including the 30-character length limitation. (The variable name, then, without the ampersand character, is limited to 29 characters.)

**Variable Substitution**

MultiLoad performs variable substitution on any statement. When MultiLoad encounters a variable name with an ampersand character (&) prefix, it replaces the variable name with the current value of that variable.

The following table describes rules that apply to variable substitution.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric Variable Values</td>
<td>Are permitted, but MultiLoad converts them to character data types for the replacement.</td>
</tr>
<tr>
<td>Value Substitution</td>
<td>Takes place one time:</td>
</tr>
<tr>
<td></td>
<td>- On the initial scan</td>
</tr>
<tr>
<td></td>
<td>- Before the statement is analyzed</td>
</tr>
<tr>
<td></td>
<td>If your replacement produces a variable name with an ampersand character, that variable name is not replaced.</td>
</tr>
<tr>
<td>An Ampersand Character</td>
<td>Is always interpreted as the first character of a utility variable, unless it is immediately followed by another ampersand character. This is true, even when it appears in a quoted string. (MultiLoad converts two consecutive ampersand characters to a single ampersand text character.)</td>
</tr>
<tr>
<td>References to a Utility Variable</td>
<td>When followed by a non-blank character or character string that could appear in a variable, the reference must include a period between the variable and the non-blank characters. MultiLoad discards the period in this context. For example, if a utility variable called &amp;x has the value xy and is to be immediately followed by the characters .ab in some context, to produce xy.ab as the result, the sequence of variable and characters must appear as: &amp;x..ab</td>
</tr>
<tr>
<td></td>
<td>MultiLoad converts the two periods to a single period text character concatenated with the value of the utility variable.</td>
</tr>
</tbody>
</table>
ANSI/SQL DateTime Specifications

You can use the ANSI/SQL DATE, TIME, TIMESTAMP and INTERVAL DateTime data types in Teradata SQL CREATE TABLE statements, and you can specify them as column/field modifiers in INSERT statements. However, note the following restrictions:

- You cannot use ANSI/SQL DateTime data types when specifying the column/field names in a MultiLoad FIELD command.
- You must convert ANSI/SQL DateTime data types to fixed-length CHAR data types when specifying column/field names in the MultiLoad FIELD command.

For a description of the fixed-length CHAR representations for each DATE, TIME, TIMESTAMP, and INTERVAL data type specification, see command usage notes for “FIELD” on page 119.

Comments

MultiLoad supports C language style comments, as described in Table 9.

<table>
<thead>
<tr>
<th>Comment Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beginning and Ending Delimiters</td>
<td>A comment begins with a slash-asterisk (/<em>) character sequence and ends with an asterisk-slash (</em>/ sequence. All intervening text is treated as a comment.</td>
</tr>
<tr>
<td>Comment Destinations</td>
<td>Comments are always written to the message destination and they may or may not be sent to the Teradata Database. Comments that are followed by a semicolon character are considered to be stand alone:</td>
</tr>
<tr>
<td></td>
<td>/<em>Comment text</em>/; CREATE TABLE table1(f1 INTEGER); In this case, the comment is not associated with the SELECT statement and is not sent to the Teradata Database.</td>
</tr>
<tr>
<td></td>
<td>Comments that are not followed by a semicolon character are considered to be part of the following command or statement:</td>
</tr>
<tr>
<td></td>
<td>/<em>Comment text</em>/; CREATE TABLE table1(f1 INTEGER); In this case, the comment is associated with the SELECT statement and is sent to the Teradata Database.</td>
</tr>
<tr>
<td>Invalid Within String or Character Literals</td>
<td>Comments cannot occur within string or character literals. A /* within a quoted string is not treated as the beginning of a comment.</td>
</tr>
<tr>
<td>Nested Comments</td>
<td>MultiLoad supports nested comments, but the Teradata Database does not. Always delimit nested comments with a semicolon character. If you do not use a semicolon to delimit a nested comment, it is considered to be part of the current command or statement. If that happens to be a Teradata SQL statement, it is sent to the Teradata Database, producing a syntax error.</td>
</tr>
</tbody>
</table>
Character Set Specification

Teradata Database for UNIX and Teradata for Windows allow you to establish a character set when invoking MultiLoad. MultiLoad supports the character sets listed in Table 10. Notice that character sets containing “EBCDIC” as part of the name are for channel-attached clients; all others are for network-attached clients.

Table 9: Notes About MultiLoad Comments (continued)

<table>
<thead>
<tr>
<th>Comment Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Substitution</td>
<td>Substitution of values for variable names continues within comments. Use two ampersand characters (&amp;&amp;) when the variable name is required.</td>
</tr>
<tr>
<td>Using Comments With Teradata SQL Statements</td>
<td>If you use a comment with a Teradata SQL statement, add a semicolon to the end of the comment if you do not want the comment sent to the Teradata Database. If you do not use a semicolon, MultiLoad sends the comment to the Teradata Database along with the Teradata SQL statement.</td>
</tr>
</tbody>
</table>

Table 10: Character Sets Supported by MultiLoad

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>Description</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Latin</td>
<td>Network-attached</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Latin</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>UTF8</td>
<td>Unicode</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>UTF16</td>
<td>Unicode</td>
<td>Network-attached</td>
</tr>
<tr>
<td>HANGULEBCDIC933_1II</td>
<td>Korean</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>HANGULKSC5601_2R4</td>
<td>Korean</td>
<td>Network-attached</td>
</tr>
<tr>
<td>SCHEBCDIC935_2II</td>
<td>Simplified Chinese</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>SCHGB2312_1T0</td>
<td>Simplified Chinese</td>
<td>Network-attached</td>
</tr>
<tr>
<td>TCHBIG5_1R0</td>
<td>Traditional Chinese</td>
<td>Network-attached</td>
</tr>
<tr>
<td>TCHEBCDIC937_31B</td>
<td>Traditional Chinese</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>KATAKANAEBCDIC</td>
<td>Japanese</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>KANJIEBCDIC5026_01</td>
<td>Japanese</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>KANJIEBCDIC5035_01</td>
<td>Japanese</td>
<td>Channel-attached</td>
</tr>
<tr>
<td>KANJIEUC_0U</td>
<td>Japanese</td>
<td>Network-attached</td>
</tr>
<tr>
<td>KANJISJIS_0S</td>
<td>Japanese</td>
<td>Network-attached</td>
</tr>
</tbody>
</table>
Table 11 describes five ways to either specify the character set or accept a default specification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File Specification</td>
<td>One of the best ways to specify the character set is with the character set specification in the MultiLoad configuration file, as described earlier in this chapter: [CHARSET=\text{character-set-name}] This allows a standard default character set for several or all MultiLoad runs, without having to specify the character set explicitly for each run.</td>
</tr>
<tr>
<td>Runtime Parameter Specification</td>
<td>Another good way to specify the character set is with the character set runtime parameter when you invoke MultiLoad, as described in “Runtime Parameters” on page 31: [CHARSET=\text{character-set-name}] for channel-attached VM and MVS client systems [-c \text{character-set-name}] for network-attached UNIX and Windows client systems</td>
</tr>
<tr>
<td>Client System Specification</td>
<td>Another way is to specify the character set for your client system before invoking MultiLoad by configuring the: [HSHSPB] parameter for channel-attached VM and MVS client systems [clispb.dat] file for network-attached UNIX and Windows client systems Note: The character-set-name specification used when you invoke MultiLoad always takes precedence over your current client system specification.</td>
</tr>
<tr>
<td>Teradata Database Default</td>
<td>If you do not use a character-set-name specification when you invoke MultiLoad, and there is no character set specification for your client system, the utility uses the default specification in the Teradata Database system table DBC.Hosts. Note: If you rely on the DBC.Hosts table specification for the default character set, make sure that the initial logon is in the default character set: [EBCDIC] for channel-attached VM and MVS client systems [ASCII] for network-attached UNIX and Windows client systems</td>
</tr>
<tr>
<td>MultiLoad Utility Default</td>
<td>If there is no character set specification in DBC.Hosts, then MultiLoad defaults to: [EBCDIC] for channel-attached VM and MVS client systems [ASCII] for network-attached UNIX and Windows client systems</td>
</tr>
</tbody>
</table>

When an AXSMOD is used, MultiLoad will pass the session character set as an attribute to the AXSMOD for its possible use (most AXSMODs will not make any use of this information).
The attribute name will be CHARSET_NAME and the attribute value will be a variable length character string consisting of the character set name.

**Rules for Using Chinese and Korean Character Sets**

You need to follow these rules when using Chinese and Korean character sets on channel-attached and network-attached platforms.

- **Object Names.** Object names are limited to A-Z, a-z, 0-9, and special characters such as $ and _.

- **Maximum String Length.** The Teradata Database requires 2 bytes to process each of the Chinese or Korean characters. This limits both request size and record size. For example, if a record consists of one string, the length of that string is limited to a maximum of 32,000 characters or 64,000 bytes.

For more information about Chinese or Korean character set restrictions for the Teradata Database, see *International Character Set Support*.

For more information about alternate character sets, see *SQL Reference: Data Definition Statements*.

**UTF8 and UTF16 Character Sets**

Unicode character sets UTF8 and UTF16 are two of the standard ways of encoding Unicode character data.

The UTF8 client character set supports UTF8 encoding and UTF16 client character set supports UTF16 encoding.

Beginning with Teradata Database 12.0, the Teradata Database supports multi-byte characters in object names when UTF8 and UTF16 client character sets are used. Use double-quotes around object names in MultiLoad scripts when using multi-byte characters.

Do not use the TABLE command when using UTF8 and UTF16 client character sets. Instead, specify the layout of the input record.

There are restrictions imposed by Teradata Database on using the UTF8 or UTF16 character set. See *International Character Set Support* for restriction details.

**UTF8 Character Sets**

Teradata MultiLoad supports the UTF8 character set on network-attached platforms and IBM MVS. When using UTF8 client character set on IBM MVS, the job script must be in Teradata EBCDIC. Teradata MultiLoad translates commands in the job script from Teradata EBCDIC to UTF8 during the load.

Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See *International Character Set Support* for details.
For more information on using the UTF8 client character set on mainframe platforms, see:

- `nullexpr` and `fieldexpr` command parameters for the FIELD command in “FIELD” on page 119
- VARTEXT format delimiter and WHERE condition for the IMPORT command in “IMPORT” on page 135
- CONTINUEIF condition for the LAYOUT command in “LAYOUT” on page 148

**UTF16 Character Sets**

Teradata MultiLoad supports the UTF16 character set on network-attached platforms. In general, the command language and the job output are the same as the client character set used by the job. However, the command language and the job output are not required to be the same as the client character set when using a UTF16 character set. When using a UTF16 character set, the job script and the job output can be either UTF8 or UTF16 character set, which is specified by the runtime parameters "-i" and "-u" when the job is invoked.

For more information on the runtime parameters, see the parameters `-i scriptencoding` and `-u outputencoding` Table 7 on page 33. For more information on using the UTF16 client character set, see:

- `nullexpr` and `fieldexpr` command parameters for the FIELD command in “FIELD” on page 119
- WHERE condition for the IMPORT command in “IMPORT” on page 135
- CONTINUEIF condition for the LAYOUT command in “LAYOUT” on page 148

**User-Defined Session Character Sets**

MultiLoad also supports user-defined session character sets when the character sets defined for the Teradata Database are not appropriate for your site. Use a session character set that matches your data, such as:

- EBCDIC037_0E for channel-attached clients (for the United States or Canada)
- LATIN1_0A, LATIN9_0A (for Western European languages) or UTF8 for UNIX clients
- LATIN1252_0A for Western European Windows clients

Do not use ASCII or EBCDIC session character sets.

For information on defining your own character set, see SQL Reference: Data Definition Statements.

**Backwards Compatibility Issues with Character Sets**

Support for certain character sets depends on the version of the Teradata Database in use. Chinese and Korean character sets are available for channel-attached and network-attached client systems.

**Caution:** If you have been using ASCII and EBCDIC for loading in the past, do not make changes in the character sets you implement without first considering possible issues with respect to compatibility with earlier versions and the associated character-set support.
Considerations for Multibyte Character Sets

Multibyte character sets impact the operation of certain MultiLoad commands, as well as object names in Teradata SQL statements, as shown in Table 12.

Table 12: Commands Impacting Multibyte Character Sets

<table>
<thead>
<tr>
<th>MultiLoad Command</th>
<th>Affected Elements</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>Utility variables</td>
<td>The utility variables can have multibyte characters. If the client does not allow multibyte character set names, then the file name must be in uppercase English.</td>
</tr>
<tr>
<td>BEGIN MLOAD</td>
<td>Table names:</td>
<td>Target table names, work table names, and error table names can have multibyte characters.</td>
</tr>
<tr>
<td></td>
<td>• Target tables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Work tables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Error tables</td>
<td></td>
</tr>
<tr>
<td>DML LABEL</td>
<td>DML label name</td>
<td>The label name in a DML LABEL command can have multibyte characters. The label name may be referenced in the APPLY clause of an IMPORT command.</td>
</tr>
<tr>
<td>FIELD</td>
<td>Field name</td>
<td>The field name specified can have multibyte characters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name can be referenced in other FIELD commands, in NULLIF and field concatenation expressions, and in APPLY WHERE conditions in IMPORT commands.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The FIELD command can also contain a NULLIF expression, which may use multibyte characters.</td>
</tr>
<tr>
<td>FILLER</td>
<td>Filler name</td>
<td>The name specified in a FILLER command can have multibyte characters.</td>
</tr>
<tr>
<td>IF</td>
<td>IF condition</td>
<td>The condition in an IF statement can compare multibyte character strings.</td>
</tr>
<tr>
<td>LAYOUT</td>
<td>Layout name</td>
<td>The layout name can:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Have multibyte characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Be used in the LAYOUT clause of an IMPORT command</td>
</tr>
<tr>
<td></td>
<td>CONTINUEIF condition</td>
<td>The CONTINUEIF condition can specify multibyte character set character comparisons.</td>
</tr>
<tr>
<td>LOGON</td>
<td>User name and password</td>
<td>The user name and password can have multibyte characters.</td>
</tr>
<tr>
<td>LOGTABLE</td>
<td>Table and database names</td>
<td>The restart log table name and database name can have multibyte characters.</td>
</tr>
</tbody>
</table>
### Graphic Data Types

MultiLoad supports the following 2-byte graphic data types in both the input data set or file, and the file containing the MultiLoad job script:

- **GRAPHIC**
- **VARGRAPHIC**
- **LONG VARGRAPHIC**

Use the `datadesc` parameter of the `FIELD` command and `FILLER` command to define graphic data types in the MultiLoad job script.

### Graphic Constants

MultiLoad supports two forms of graphic constants:

- The graphic literal or string constant, which is allowed in the KANJI EBBCDIC character set on channel-attached VM and MVS client systems. This type of constant must have an even number of bytes within the quoted string to represent double-byte characters.
- The hexadecimal representation of graphic data used on both network-attached UNIX client systems and channel-attached VM and MVS client systems

For more information about graphic constants and hexadecimal representations of them, see *SQL Reference: Data Definition Statements*.

### Import Tasks

An import task quickly applies large amounts of client data to one or more tables or views on the Teradata Database.

If a Teradata Database restart/recovery operation interrupts a MultiLoad import task, the utility resumes processing from the point of interruption. It does not roll back and repeat processing that took place before the interruption. This effectively eliminates the processing penalty that would be incurred by transaction roll back during a restart/recovery operation.

Table 13 describes the things to consider when programming import tasks in the MultiLoad job script.
### Table 13: Import Task Considerations for MultiLoad

<table>
<thead>
<tr>
<th>Topic</th>
<th>Import Task Considerations</th>
</tr>
</thead>
</table>
| Statements Supported| MultiLoad supports any combination of the following DML statements when importing data from one or more sources:  
  - INSERT  
  - UPDATE  
  - DELETE  
  Each group of DML statements must be preceded by a DML LABEL command that defines the label and error treatment options.  
  Each data source:  
  - Contains a record for each table row to which one or more DML statements apply  
  - Is identified by an IMPORT command that references the DML statements and a LAYOUT command to match the records of the data source to the DML statements.  
  Each LAYOUT command requires a sequence of FIELD commands, FILLER commands, and TABLE commands to define the record layout of a data source. |
| The Upsert Function | The MultiLoad *upsert* function applies consecutive UPDATE and INSERT statements to the same row.  
  If the UPDATE statement fails because the target table row does not exist, MultiLoad automatically executes the INSERT statement, completing the operation in a single pass instead of two.  
  If the Update statement fails due to an invalid length of data in the statement's WHERE clause, modify the data length and resubmit the statement. This might occur when the length of the data field in the UPDATE statement is larger than the Primary Index column length. Possible workarounds are using FILLER or an explicit cast of the data field to match the primary index column length. |
| Multiple Operations | Each import task can include multiple INSERT, UPDATE, and DELETE statements, and the multiple DML operations can be conditionally applied to as many as five tables with a single pass of the client file.  
  When associated with a client system file, each DML operation must provide a value for each nonnullable or nondefaultable column making up the primary index of the target table or the underlaying table of a target view.  
  For each DML operation:  
  1. The client system sends a model DML statement, accompanied by a unique identifier, called a match tag, to the Teradata Database.  
  2. When processing the client data, MultiLoad evaluates each record according to the WHERE condition specifications of your IMPORT command. Each condition signifies a set of DML operations.  
  3. When a client record satisfies a condition, MultiLoad combines the record with the match tag for that set of DML operations and sends it to the Teradata Database.  
  4. The Teradata Database then performs the DML operation specified by the model statement.  
  If a single client record satisfies more than one condition, MultiLoad sends multiple copies of the record to the Teradata Database, each with a different match tag. |
Chapter 2: Using MultiLoad
Programming Considerations

Target Tables

Each import task can access up to five target tables on the Teradata Database. To perform an import task, you must have the appropriate access permission (for INSERT, UPDATE, and DELETE) on each target table.

Each target table:
- Can, but need not:
  - Be empty
  - Have NUSIs
- Cannot have unique secondary indexes

Each import task command need not access the same target table.

Locks

MultiLoad uses the access lock to allow concurrent select access to all affected target tables throughout the entire import task.

Concurrent DML statements that require read or write locks are permitted during the acquisition phase:
- Until MultiLoad finishes acquiring data from the client system
- While conducting a sort of the acquired data

Except for select access, concurrent DML statements are not permitted during the application and cleanup phases of an import task.

DROP, and Other DDL Statements

MultiLoad rejects all DDL statements except DROP during an import task. During the application phase, however, while it is actively applying the source data to the target tables, MultiLoad delays both processing of the DROP statements and rejection of the others. After the application phase completes, MultiLoad processes any waiting DROP statements.

**Note:** Though this is the usual treatment for DROP statements, some cases may require that MultiLoad be stopped. At all other times, MultiLoad processes DROP statements immediately.

Data Source Restriction

All of the source data for a MultiLoad import task must originate from a data source on the client system. It cannot originate from existing tables in the Teradata Database.

Additionally, no statement within an import task can even reference any table or any row in the Teradata Database that is not a specified import task table.

Column Value Specifications

When considered in conjunction with each applicable imported record, column values must be explicitly specified for all columns:
- Of the primary index of update and delete target tables
- Of the partitioning columns of update and delete target tables if the target table has Partitioned Primary Indexes (PPI).
- Except those for which a default value is specified, including null, of insert target tables

To fulfill the requirement for update and delete target tables, use the AND logical operator with a series of terms in either of the following forms:
- \( column\_reference = colon\_variable\_reference \)
- \( column+reference = constant \)

**Note:** MultiLoad cannot process UPDATE or DELETE statements with OR logical operators. Instead, use two separate DML statements and apply them conditionally.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Import Task Considerations</th>
</tr>
</thead>
</table>
| Target Tables | Each import task can access up to five target tables on the Teradata Database. To perform an import task, you must have the appropriate access permission (for INSERT, UPDATE, and DELETE) on each target table. Each target table:  
- Can, but need not:  
  - Be empty  
  - Have NUSIs  
- Cannot have unique secondary indexes  
Each import task command need not access the same target table. |
| Locks | MultiLoad uses the access lock to allow concurrent select access to all affected target tables throughout the entire import task. Concurrent DML statements that require read or write locks are permitted during the acquisition phase:  
- Until MultiLoad finishes acquiring data from the client system  
- While conducting a sort of the acquired data  
Except for select access, concurrent DML statements are not permitted during the application and cleanup phases of an import task. |
| DROP, and Other DDL Statements | MultiLoad rejects all DDL statements except DROP during an import task. During the application phase, however, while it is actively applying the source data to the target tables, MultiLoad delays both processing of the DROP statements and rejection of the others. After the application phase completes, MultiLoad processes any waiting DROP statements.  
**Note:** Though this is the usual treatment for DROP statements, some cases may require that MultiLoad be stopped. At all other times, MultiLoad processes DROP statements immediately. |
| Data Source Restriction | All of the source data for a MultiLoad import task must originate from a data source on the client system. It cannot originate from existing tables in the Teradata Database. Additionally, no statement within an import task can even reference any table or any row in the Teradata Database that is not a specified import task table. |
| Column Value Specifications | When considered in conjunction with each applicable imported record, column values must be explicitly specified for all columns:  
- Of the primary index of update and delete target tables  
- Of the partitioning columns of update and delete target tables if the target table has Partitioned Primary Indexes (PPI).  
- Except those for which a default value is specified, including null, of insert target tables  
To fulfill the requirement for update and delete target tables, use the AND logical operator with a series of terms in either of the following forms:  
- \( column\_reference = colon\_variable\_reference \)  
- \( column+reference = constant \)  
**Note:** MultiLoad cannot process UPDATE or DELETE statements with OR logical operators. Instead, use two separate DML statements and apply them conditionally. |
Chapter 2: Using MultiLoad
Programming Considerations

Table 13: Import Task Considerations for MultiLoad (continued)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Import Task Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive/Recovery Utility</td>
<td>These statements include:</td>
</tr>
<tr>
<td>Statements</td>
<td>• ROLLBACK</td>
</tr>
<tr>
<td></td>
<td>• ROLLFORWARD</td>
</tr>
<tr>
<td></td>
<td>• RESTORE</td>
</tr>
<tr>
<td></td>
<td>• DUMP</td>
</tr>
<tr>
<td>When associated with a MultiLoad target table, the Teradata Database:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Rejects all ROLLBACK and ROLLFORWARD statements, regardless of operational phase, and whether MultiLoad is active or not.</td>
</tr>
<tr>
<td></td>
<td>• Executes all RESTORE statements, without regard for MultiLoad locks. If a MultiLoad utility task is active in the application phase, however, the Teradata Database delays execution of a RESTORE statement until the application phase completes and releases the write locks.</td>
</tr>
<tr>
<td></td>
<td>• Processes DUMP statements differently, depending on the operational state of MultiLoad:</td>
</tr>
<tr>
<td></td>
<td>• If MultiLoad is not in the application phase, then the Teradata Database processes the DUMP statement normally.</td>
</tr>
<tr>
<td></td>
<td>• If MultiLoad is in the application phase, but inactive, then the Teradata Database processes the DUMP statement, but returns only the table header with field 4 cleared.</td>
</tr>
<tr>
<td></td>
<td>• If MultiLoad is in the application phase and active, the Teradata Database delays processing of the DUMP statement until the application phase completes. If the application phase ends normally, the Teradata Database processes the DUMP normally. If the application phase was interrupted before a normal completion, the Teradata Database processes the DUMP statement but returns only the table header.</td>
</tr>
<tr>
<td>The application phase can prevent normal processing of a DUMP statement. Take the following precautions to protect the target tables before starting a MultiLoad job:</td>
<td></td>
</tr>
<tr>
<td>1 First, dump the target tables.</td>
<td></td>
</tr>
<tr>
<td>2 Then, execute a CHECKPOINT statement against the associated permanent journal.</td>
<td></td>
</tr>
<tr>
<td>Now, if a table becomes corrupted because of bad data or an operational error, you can restore the table from the dump and roll it forward to the checkpoint.</td>
<td></td>
</tr>
</tbody>
</table>

Delete Tasks

The delete task uses a full file scan to remove a large number of rows from a single Teradata Database table. (You cannot use a delete task on a view.)

If a Teradata Database restart/recovery operation interrupts a MultiLoad delete task, MultiLoad resumes processing from the point of interruption. It does not roll back and repeat processing that took place before the interruption. This effectively eliminates the processing penalty that would be incurred by transaction rollback during a restart/recovery operation.

Table 14 describes the things to consider when programming delete tasks in the MultiLoad job script.
Dealing with AMPS

This section describes dealing with down and nonparticipant AMPs.

Down AMPs

The impact of down AMPs on MultiLoad tasks depends on:

- The number of AMPs that are down, either logically or physically, in a cluster.
- The operational phase of the MultiLoad task when the down AMP condition occurs.
- Whether the target tables are fallback or nonfallback.

If all of the target tables are *fallback* and *not more than one* AMP is down, then MultiLoad tasks continue to run as long as there is not more than one AMP down, either logically or physically, in a cluster. The down AMP does not participate in the application phase if:

### Table 14: Delete Task Considerations for MultiLoad

<table>
<thead>
<tr>
<th>Topic</th>
<th>Delete Task Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Specifications</td>
<td>A delete task can specify a unique primary index (UPI), a non-unique primary index (NUPI), or a non-unique secondary index (NUSI). A unique secondary index (USI) is not allowed.</td>
</tr>
<tr>
<td>Archive/Recovery Utility</td>
<td>These statements include:</td>
</tr>
<tr>
<td>Statements</td>
<td>- ROLLBACK</td>
</tr>
<tr>
<td></td>
<td>- ROLLFORWARD</td>
</tr>
<tr>
<td></td>
<td>- RESTORE</td>
</tr>
<tr>
<td></td>
<td>- DUMP</td>
</tr>
<tr>
<td></td>
<td>MultiLoad treats these the same during the application phase of a delete task as during the application phase of an import task, as described in the preceding topic.</td>
</tr>
<tr>
<td>Locks</td>
<td>The locks protocol for a MultiLoad delete task is the same as for an import task, as described in the preceding topic.</td>
</tr>
<tr>
<td>Invalid Command Parameters</td>
<td>The following command parameters are not allowed in a MultiLoad delete task:</td>
</tr>
<tr>
<td></td>
<td>- The CHECKPOINT and SESSIONS parameters of the BEGIN MLOAD command are not allowed because there is no row distribution in a MultiLoad delete task.</td>
</tr>
<tr>
<td></td>
<td>- The ERRLIMIT parameter of the BEGIN MLOAD command is not allowed because the error types that can be limited are not recognized in a MultiLoad delete task.</td>
</tr>
<tr>
<td></td>
<td>- The FOR and THRU parameters of the IMPORT command are not allowed because only one data record can be imported for a MultiLoad delete task.</td>
</tr>
<tr>
<td></td>
<td>- The DML LABEL command and the APPLY clause of the IMPORT command are not allowed because the single imported data record is unconditionally applied by the single DELETE statement in a MultiLoad delete task.</td>
</tr>
</tbody>
</table>
The AMP goes down before the MultiLoad task enters the application phase, and the AMPCHECK parameter is set to NONE.

Certain I/O errors occur during the application phase.

If all of the target tables are *fallback* and *two or more* AMPs are down, then MultiLoad tasks do not run, or terminate if two or more AMPs are down, either logically or physically, in a cluster.

**Note:** In the application phase, if AMPs are down to the extent that data on the DSU is corrupted, then you must restore the affected tables.

If one or more of the target tables is *nonfallback* and one or more AMPs are down, then MultiLoad tasks terminate and you cannot restart them until all of the AMPs are back up.

**Note:** MultiLoad also terminates if I/O errors corrupt the target tables in the application phase.

### Nonparticipant AMPs

There are three ways for an AMP to become nonparticipant for a MultiLoad task.

- When any AMP is down at the end of the acquisition phase/beginning of the application phase, then the associated AMP becomes a nonparticipant if the AMPCHECK NONE option is specified in the BEGIN MLOAD command.
  
  An AMP can only become a nonparticipant if the target table is defined as having fallback protection because MultiLoad does not run after the acquisition phase. The AMPCHECK APPLY and ALL options would prevent the occurrence of nonparticipant AMPs in this situation.

- When I/O errors occur in certain MultiLoad tables during the application phase, then the associated AMP becomes a nonparticipant when the I/O recovery operation stops the MultiLoad task.

- When a head/disk assembly (HDA) fails during the application phase, then the associated AMP becomes a nonparticipant, but it returns after the disk is replaced and the Disk Copy and Table Rebuild utilities are run.

In effect, MultiLoad treats a nonparticipant AMP as if it were a down AMP. Thus, MultiLoad does not run if a cluster has any combination of more than one AMP that is:

- Down
- Offline
- Nonparticipant

If more than one AMP in a cluster becomes a nonparticipant during the application phase, the MultiLoad task cannot continue. The target tables are considered unusable, and must be recovered from archives.
Using Locks

MultiLoad uses two kinds of long-term locks on the tables involved in a MultiLoad task:

- The set of transaction locks acquired against the tables involved in the tasks
- Flags in the table headers that reject certain concurrent access

The locks placed on target tables by MultiLoad are write locks, or less, in most situations. The locking levels are imposed by the utility for:

- MultiLoad import or delete tasks, including all MultiLoad utility commands except RELEASE MLOAD
- End phase target table rebuilds in AMPs marked as down for MultiLoad tasks
- Rebuild logic triggered by target table I/O errors

The most restrictive exclusive lock is used only when a RELEASE MLOAD statement is executed after a MultiLoad task has been suspended or aborted.

*Table 15* describes the different types of locks imposed on tables by MultiLoad tasks.

<table>
<thead>
<tr>
<th>Lock Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>An access lock allows selection of data from a table that may be locked for write access. MultiLoad maintains access locks against the target tables during the acquisition phase.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> There is concurrent access lock select access to all affected target tables in both import and delete tasks.</td>
</tr>
<tr>
<td>Write</td>
<td>A write lock enables a single user to modify the table. MultiLoad maintains write locks against each:</td>
</tr>
<tr>
<td></td>
<td>• Target table during the application phase</td>
</tr>
<tr>
<td></td>
<td>• Work tables and error tables for each task transaction</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Use access lock select access when querying the MultiLoad error tables to prevent lock time-outs that could cause MultiLoad task aborts.</td>
</tr>
<tr>
<td>MultiLoad Acquisition</td>
<td>A MultiLoad acquisition lock is a flag in the table header that effectively rejects certain types of Teradata SQL access statements. An acquisition lock:</td>
</tr>
<tr>
<td></td>
<td>• Allows all concurrent DML access and the DROP DDL statement</td>
</tr>
<tr>
<td></td>
<td>• Rejects DDL statements other than DROP</td>
</tr>
<tr>
<td></td>
<td>MultiLoad maintains acquisition locks against each:</td>
</tr>
<tr>
<td></td>
<td>• Target table from the beginning of the preliminary phase through the sort at the end of the acquisition phase</td>
</tr>
<tr>
<td></td>
<td>• Error table throughout the entire MultiLoad job</td>
</tr>
</tbody>
</table>
Chapter 2: Using MultiLoad
Programming Considerations

Table 15: Types of Table Locks Imposed by MultiLoad (continued)

<table>
<thead>
<tr>
<th>Lock Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiLoad Application</td>
<td>A MultiLoad application lock is a flag in the table header that effectively</td>
</tr>
<tr>
<td></td>
<td>rejects certain types of Teradata SQL access statements.</td>
</tr>
<tr>
<td></td>
<td>An application lock:</td>
</tr>
<tr>
<td></td>
<td>• Allows all concurrent access lock select access and the DROP DDL</td>
</tr>
<tr>
<td></td>
<td>statement</td>
</tr>
<tr>
<td></td>
<td>• Rejects all other DML and DDL statements</td>
</tr>
<tr>
<td></td>
<td>MultiLoad maintains application locks against each:</td>
</tr>
<tr>
<td></td>
<td>• Target table through the application phase</td>
</tr>
<tr>
<td></td>
<td>• Work table throughout the entire MultiLoad job</td>
</tr>
<tr>
<td>Exclusive</td>
<td>MultiLoad uses an exclusive lock to support the manual recovery procedure</td>
</tr>
<tr>
<td></td>
<td>when a RELEASE MLOAD statement is executed after a MultiLoad task has</td>
</tr>
<tr>
<td></td>
<td>been suspended or aborted.</td>
</tr>
<tr>
<td></td>
<td>The RELEASE MLOAD function:</td>
</tr>
<tr>
<td></td>
<td>• Cannot run until all access locked queries are either completed or</td>
</tr>
<tr>
<td></td>
<td>aborted</td>
</tr>
<tr>
<td></td>
<td>• Does not complete until it obtains an exclusive lock on the target</td>
</tr>
<tr>
<td></td>
<td>tables</td>
</tr>
</tbody>
</table>

Restrictions and Limitations

Table 16 describes the MultiLoad restrictions and limitations on operational features and functions.

Table 16: Feature Restrictions

<table>
<thead>
<tr>
<th>Operational Feature/Function</th>
<th>Restriction/Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum file size</td>
<td>On UNIX MP-RAS operating system, the maximum file size that is supported by MultiLoad is 2 gigabytes.</td>
</tr>
<tr>
<td></td>
<td>On Windows; Sun Solaris SPARC, Solaris Opteron; IBM AIX; and HP-UX operating systems, there is no file size restriction.</td>
</tr>
<tr>
<td>Maximum row size</td>
<td>The maximum row size for a MultiLoad job, data plus indicators, is approximately 64,000 bytes. This limit is a function of:</td>
</tr>
<tr>
<td></td>
<td>• The row size limit of the Teradata Database</td>
</tr>
<tr>
<td></td>
<td>• The MultiLoad work tables and error tables</td>
</tr>
<tr>
<td></td>
<td>MultiLoad cannot accommodate a row size that is near the row size limit of the Teradata Database because of the operational overhead associated with MultiLoad work tables and error tables. If your MultiLoad job exceeds this limit, the Teradata Database returns Error Message 3577 (row size or sort key size overflow), and terminates the job.</td>
</tr>
</tbody>
</table>
Concurrent load utility tasks

The maximum number of concurrent MultiLoad tasks that can run is variable; the limit can be controlled by the system administrator. MaxLoadTasks may be overridden if TASM is active.

Note:

For the most up-to-date information on concurrent task limits, see the description of the MaxLoadTask parameter of the DBSControl utility in Utilities Volume 1. Additional information is available in the Teradata Dynamic Workload Manager User Guide.

If your MultiLoad job exceeds the recommended limits, the Teradata Database returns a 2633 error message indicating that too many loads are running, and the utility retries until:

- It can execute the task
- It reaches the TENACITY hours time limit specified by the BEGIN MLOAD command

<table>
<thead>
<tr>
<th>Operational Feature/Function</th>
<th>Restriction/Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent load utility tasks</td>
<td>The maximum number of concurrent MultiLoad tasks that can run is variable; the limit can be controlled by the system administrator. MaxLoadTasks may be overridden if TASM is active.</td>
</tr>
<tr>
<td>Maximum number of columns</td>
<td>2048</td>
</tr>
<tr>
<td>Maximum statement text length</td>
<td>1 MB</td>
</tr>
<tr>
<td>Aggregate operators, exponential operators, arithmetic functions</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Concatenation of data files</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Data retrieval from the Teradata Database with the SELECT statements</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Expressions</td>
<td>Are evaluated from left to right, using the Teradata Database order of preference, but can be overridden by parentheses.</td>
</tr>
<tr>
<td>Hexadecimal form</td>
<td>MultiLoad does not accept and will not display object names specified in internal Teradata Database hexadecimal form.</td>
</tr>
<tr>
<td>Foreign key references</td>
<td>Not allowed for a MultiLoad target table. Attempting a MultiLoad task or any other action against a target table defined with a foreign key constraint produces an error condition.</td>
</tr>
<tr>
<td>Unique secondary indexes</td>
<td>Not allowed for a MultiLoad target table. Target tables for a MultiLoad task may contain NUSIs, but not unique secondary indexes.</td>
</tr>
<tr>
<td>Hash indexes</td>
<td>Not supported</td>
</tr>
<tr>
<td>Join index restrictions</td>
<td>MultiLoad does not support or maintain Join Indexes.</td>
</tr>
<tr>
<td>Journaling options</td>
<td>There are no restrictions on journaling options. MultiLoad recognizes target tables with all types of currently supported journaling.</td>
</tr>
</tbody>
</table>
Chapter 2: Using MultiLoad
Programming Considerations

**PPI Table Considerations**
Load utilities, including MultiLoad, are supported on PPI tables with the following restrictions and advisories:

- MultiLoad does not support tables with Unique Secondary Indexes (USI). Many PPI tables have USIs.
- MultiLoad IMPORT tasks require you to supply all values of the primary index column set and all values of the partitioning column set for deletes and updates.
- MultiLoad IMPORT tasks do not support updates of the partitioning column set.
- MultiLoad IMPORT tasks do not support primary index updates.
- MultiLoad does not support hash and join indexes.

**Termination Return Codes**
When a MultiLoad job terminates, the utility returns a termination return code to the client system:

- 0 = Normal completion
- 4 = Warning
- 8 = User error
- 12 = Severe internal error
- 16 = No message destination available

To avoid ambiguous or conflicting results, always use values greater than 20 when you specify a return code with your LOGOFF command.

**UNIX Signals**
If you are running MultiLoad in a UNIX operating system, you need to be aware of the UNIX signals used by MultiLoad. You cannot use the MultiLoad UNIX signals in any module or routine you program for use with MultiLoad. Doing so causes an error in MultiLoad.

MultiLoad uses the following UNIX signals:

- SIGALARM (alarm signal)
- SIGFPE
- SIGINT (interrupt signal)
- SIGQUIT (quit signal)
- SIGTERM (terminate signal)

**Note:** Signals are predefined messages sent between two UNIX processes to communicate the occurrence of unexpected external events, or exceptions. Aborting a MultiLoad session while MultiLoad is in the middle of processing a job is an example of an exception. In this scenario, MultiLoad uses the UNIX signals to trap the abort command, disconnect all sessions, perform any necessary cleanup, and then terminate in an orderly manner.
SIGINT in Windows Environment
The SIGINT signal is not supported in Windows environments. When a \texttt{CTRL+C} interrupt occurs, a new thread is generated to handle that interrupt. Because MultiLoad is not multi-threaded, errors can occur.

Using INMOD and Notify Exit Routines
This section provides information about how to use input modification (INMOD) and notify exit routines.

Overview
This section describes the different types of routines and when you might use them.

INMOD Routines
The term INMOD is an acronym for input modification routines. These are user-written routines that MultiLoad and other load/export utilities can use to provide enhanced processing functions on input records before they are sent to the Teradata Database.

You can use an INMOD routine to supply input records or to perform preprocessing tasks on the input records before passing them to MultiLoad. For example, such tasks could:

- Generate records to be passed to MultiLoad
- Read data directly from one or more database systems
- Validate data records before passing them to MultiLoad
- Convert fields in a data record before passing it to MultiLoad

Notify Exit Routines
An exit routine specifies a predefined action to be performed whenever certain significant events occur during a MultiLoad job.

Notify exit routines are especially useful in operator-free environments, where job scheduling relies heavily on automation to optimize system performance.

For example, by writing an exit in C (without using CLIv2) and using the NOTIFY...EXIT option of the BEGIN MLOAD command, you can provide a routine to detect whether a MultiLoad job succeeds or fails, how many records were loaded, what the return code was for a failed job, and so on.

Programming Considerations for Using Routines
This section describes programming languages supported for each type of routine, as well as other related considerations.
Programming Languages

MultiLoad is written in:

- SAS/C for channel-attached VM and MVS client systems
- C for network-attached UNIX and Windows client systems

You can write INMOD routines and notify exit routines in the programming languages listed in Table 17, depending on the platform that runs MultiLoad.

Table 17: Languages Supported by Platform and Type of User-Developed Routine

<table>
<thead>
<tr>
<th>Platform</th>
<th>INMOD Routine</th>
<th>Notify Exit Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM, MVS</td>
<td>Assembler, COBOL, PL/I, or SAS/C</td>
<td>SAS/C</td>
</tr>
<tr>
<td>UNIX, Windows</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Programming Structure

Table 18 defines the structure by programming language for communicating between MultiLoad and INMOD or notify exit routines.

Table 18: Programming Routines by Language

<table>
<thead>
<tr>
<th>For routines written in</th>
<th>the programming structure for each parameter is</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>First parameter:</td>
</tr>
<tr>
<td></td>
<td>RRECORD DSECT</td>
</tr>
<tr>
<td></td>
<td>RTNCODE DS F</td>
</tr>
<tr>
<td></td>
<td>RLENGTH DS F</td>
</tr>
<tr>
<td></td>
<td>RBODY DS CLxxxxx</td>
</tr>
<tr>
<td>Note:</td>
<td>In the RBODY specification, the body length xxxxx is:</td>
</tr>
<tr>
<td></td>
<td>• 32004 for Teradata for Windows</td>
</tr>
<tr>
<td></td>
<td>• 64004 for Teradata Database for UNIX</td>
</tr>
<tr>
<td></td>
<td>Second parameter:</td>
</tr>
<tr>
<td></td>
<td>IPARM DSECT</td>
</tr>
<tr>
<td></td>
<td>RSEQ DS F</td>
</tr>
<tr>
<td></td>
<td>PLEN DS H</td>
</tr>
<tr>
<td></td>
<td>PBODY DS CL100</td>
</tr>
</tbody>
</table>
Table 19 shows the entry points for notify exit routines.

In each structure, the records must be constructed so that the left-to-right order of the data field corresponds to the order of the field names specified in the MultiLoad `LAYOUT` command and subsequent `FIELD`, `FILLER`, and `TABLE` commands.

Table 18: Programming Routines by Language (continued)

<table>
<thead>
<tr>
<th>For routines written in</th>
<th>the programming structure for each parameter is</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td></td>
</tr>
<tr>
<td>First parameter:</td>
<td></td>
</tr>
<tr>
<td>struct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>long Status;</td>
</tr>
<tr>
<td></td>
<td>long RecordLength;</td>
</tr>
<tr>
<td></td>
<td>char buffer[xxxxx];</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
</tr>
<tr>
<td>In the char buffer specification, the buffer length <code>xxxxx</code> is:</td>
<td></td>
</tr>
<tr>
<td>• 32004 for Teradata for Windows</td>
<td></td>
</tr>
<tr>
<td>• 64004 for Teradata Database for UNIX</td>
<td></td>
</tr>
<tr>
<td>Second parameter:</td>
<td></td>
</tr>
<tr>
<td>struct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>long seqnum;</td>
</tr>
<tr>
<td></td>
<td>char parm[80];</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td><strong>COBOL</strong></td>
<td></td>
</tr>
<tr>
<td>First parameter:</td>
<td></td>
</tr>
<tr>
<td>01 INMOD-RECORD.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>03 RETURN-CODE PIC S9(9) COMP.</td>
</tr>
<tr>
<td></td>
<td>03 RECORD-LENGTH PIC 9(9) COMP.</td>
</tr>
<tr>
<td></td>
<td>03 RECORD-BODY PIC X(<code>xxxxx</code>)</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
</tr>
<tr>
<td>In the RECORD-BODY specification, the body length <code>xxxxx</code> is:</td>
<td></td>
</tr>
<tr>
<td>• 320004 for Teradata for Windows</td>
<td></td>
</tr>
<tr>
<td>• 64004 for Teradata Database for UNIX</td>
<td></td>
</tr>
<tr>
<td>Second parameter:</td>
<td></td>
</tr>
<tr>
<td>01 PARM-STRUCT.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>03 SEQ-NUM PIC 9(9) COMP.</td>
</tr>
<tr>
<td></td>
<td>03 PARM-LEN PIC 9(4) COMP.</td>
</tr>
<tr>
<td></td>
<td>03 PARM-BODY PIC X(80).</td>
</tr>
<tr>
<td><strong>PL/I</strong></td>
<td></td>
</tr>
<tr>
<td>First parameter:</td>
<td></td>
</tr>
<tr>
<td>DCL 1 PARMLIST,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 STATUS FIXED BINARY(31,0)</td>
</tr>
<tr>
<td></td>
<td>10 RLENGTH FIXED BINARY(31,0)</td>
</tr>
<tr>
<td></td>
<td>10 REC CHAR(<code>xxxxx</code>)</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
</tr>
<tr>
<td>In the REC CHAR specification, the length <code>xxxxx</code> is:</td>
<td></td>
</tr>
<tr>
<td>• 32004 for Teradata for Windows</td>
<td></td>
</tr>
<tr>
<td>• 64004 for Teradata Database for UNIX</td>
<td></td>
</tr>
<tr>
<td>Second parameter:</td>
<td></td>
</tr>
<tr>
<td>DCL 1 PARMLIST,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 SEQNUM FIXED BINARY(31,0)</td>
</tr>
<tr>
<td></td>
<td>10 PLENGTH FIXED BINARY(15,0)</td>
</tr>
<tr>
<td></td>
<td>10 PBODY CHAR(80)</td>
</tr>
</tbody>
</table>
**MultiLoad/INMOD Routine Interface**

MultiLoad exchanges information with an INMOD routine by using the conventional parameter register to point to a parameter list of two 32-bit addresses.

**Note:** Be sure to compile your INMOD and notify exit routines in 32-bit mode so they are compatible with Teradata MultiLoad.

The first 32-bit address points to a three-value structure consisting of StatusCode, Length, and Body. The second 32-bit address points to a data structure containing a sequence number and a parameter list.

**StatusCode**

StatusCode is a 32-bit signed binary value that carries information in both directions. Table 20 defines the eight status codes used by the MultiLoad-to-INMOD interface.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MultiLoad is calling for the first time. MultiLoad expects the INMOD routine to return a record. At this point the INMOD routine performs its initialization tasks before sending a data record to MultiLoad.</td>
</tr>
<tr>
<td>1</td>
<td>MultiLoad is calling, not for the first time. MultiLoad expects the INMOD routine to return a record.</td>
</tr>
<tr>
<td>2</td>
<td>The client system has been restarted. The INMOD routine repositions to the last checkpoint. MultiLoad is not expecting the INMOD routine to return a data record. Repositioning information, provided by the INMOD after a code 3, is read from the restart log table and returned in the buffer normally used for the data record. <strong>Note:</strong> If the client system restarts before the first checkpoint, MultiLoad sends entry code 0 to re-initialize.</td>
</tr>
</tbody>
</table>

---

**Table 19: Entry Points for Notify Exit Routines**

<table>
<thead>
<tr>
<th>Routine Language</th>
<th>Entry Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS/C on VM and MVS platforms</td>
<td>_dynamn</td>
</tr>
<tr>
<td>COBOL and PL/I on VM and MVS platforms</td>
<td>DYNAMN</td>
</tr>
<tr>
<td>C on UNIX and Windows platforms</td>
<td>_dynamn</td>
</tr>
</tbody>
</table>

**Table 20: MultiLoad-to-INMOD Status Codes**
Table 21 explains the two status codes used by the INMOD-to-MultiLoad interface.

**Table 21: INMOD-to-MultiLoad Interface Status Codes**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A record is being returned as the body value for a read call (code 1). For calls other than read, a value of 0 indicates successful completion.</td>
</tr>
<tr>
<td>Any nonzero value</td>
<td>The INMOD routine is at an end-of-file condition for a read call (code 1). For calls other than read, a nonzero value indicates a processing error that terminates MultiLoad.</td>
</tr>
</tbody>
</table>

**Length**

Length is a 32-bit signed binary value that the INMOD routine uses to specify the length, in bytes, of the data record.
The INMOD routine can use a length value of zero to indicate an end-of-file condition.

**Body**

Body represents the area where the INMOD routine places the data record. The maximum record length is:

- 31 K or 31,744 bytes for Teradata for Windows
- 62 K or 63,488 bytes for Teradata Database for UNIX

**Sequence Number**

Sequence number is a 4-byte integer, the integer record counter portion of the source sequence number.

**Parameter List**

The parameter list in the second 32-bit address consists of the following:

- VARCHAR specification
- A 2-byte length specification, \( m \)
- The \( m \)-byte `parms` string, as parsed and presented by MultiLoad

**Caution:** To prevent data corruption, INMOD routines that cannot comply with these protocols terminate if they encounter a restart code 2, 3, or 4. To support proper MultiLoad restart operations, INMOD routines must save and restore checkpoint information as described here. If the INMOD saves checkpoint information in some other manner, a subsequent restart/recovery operation could result in data loss or corruption.

**MultiLoad/Notify Exit Routine Interface**

MultiLoad accumulates operational information about specific events that occur during a MultiLoad job. If the BEGIN MLOAD command or the BEGIN DELETE MLOAD command includes a NOTIFY option with an EXIT specification, then, when the specific events occur, MultiLoad calls the named notify exit routine and passes to it:

- An event code to identify the event
- Specific information about the event

Table 22 lists the event codes and describes the data that MultiLoad passes to the notify exit routine for each event. For a description of the events associated with each level of notification (low, medium, and high), see the description of the NOTIFY command option for “BEGIN MLOAD and BEGIN DELETE MLOAD” on page 95.

**Note:** To support future enhancements, always make sure that your notify exit routines ignore invalid or undefined event codes, and that they do not cause MultiLoad to terminate abnormally.
<table>
<thead>
<tr>
<th>Event</th>
<th>Event Code</th>
<th>Event Description</th>
<th>Data Passed to the Notify Exit Routine</th>
</tr>
</thead>
</table>
| Initialize          | 0          | Successful processing of the NOTIFY option of the BEGIN MLOAD or BEGIN DELETE MLOAD command: | • Version ID length—4-byte unsigned integer  
                               • Version ID string—32-character (maximum) array  
                               • Utility ID—4-byte unsigned integer  
                               • Utility name length—4-byte unsigned integer  
                               • Utility name string—32-character (maximum) array  
                               • User name length—4-byte unsigned integer  
                               • User name string—64-character (maximum) array  
                               • Optional string length—4-byte unsigned integer  
                               • Optional string—80-character (maximum) array |
| File or INMOD open  | 1          | Successful processing of the IMPORT command that specifies the file or INMOD routine name | • File name length—4-byte unsigned integer  
                               • File name—256-character (maximum) array  
                               • Import number—4-byte unsigned integer |
| Phase 1 begin       | 2          | The beginning of the acquisition phase. It is called one time for each table specified in the BEGIN MLOAD or BEGIN DELETE MLOAD commands | • Table name length—4-byte unsigned integer  
                               • Table name—128-character (maximum) array  
                               • Table number—4-byte unsigned integer |
| Checkpoint          | 3          | Checkpoint information has been written to the restart log table                   | Record number—4-byte unsigned integer                                                                |
| Phase 1 end         | 4          | The CHECKPOINT LOADING END request has successfully completed after the end of the acquisition phase | • Records read—4-byte unsigned integer  
                               • Records skipped—4-byte unsigned integer  
                               • Records rejected—4-byte unsigned integer  
                               • Records sent to the Teradata Database—4-byte unsigned integer |
| Phase 2 begin       | 5          | The EXEC MLOAD request is about to be sent to the Teradata Database                | No data accompanies the phase 2 begin event code                                                      |
| Phase 2 end         | 6          | Processing of the EXEC MLOAD request completed successfully. For each table in the request | • Records inserted—4-byte unsigned integer  
                               • Records updated—4-byte unsigned integer  
                               • Records deleted—4-byte unsigned integer  
                               • Table number—4-byte unsigned integer |
| Error table 1       | 7          | Processing of the SEL COUNT(*) request completed successfully for the first error table. For each table specified in the BEGIN MLOAD command: | • Number of rows—4-byte unsigned integer  
                               • Table number—4-byte unsigned integer |
Table 22: Events Passed to the Notify Exit Routines (continued)

<table>
<thead>
<tr>
<th>Event</th>
<th>Event Code</th>
<th>Event Description</th>
<th>Data Passed to the Notify Exit Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error table 2</td>
<td>8</td>
<td>Processing of the SEL COUNT(*) request completed successfully for the second error</td>
<td>• Number of rows—4-byte unsigned integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>table. For each table specified in the BEGIN MLOAD command:</td>
<td>• Table number—4-byte unsigned integer</td>
</tr>
<tr>
<td>Teradata Database</td>
<td>9</td>
<td>MultiLoad received a crash message from the Teradata Database or from the CLIv2</td>
<td>No data accompanies the Teradata Database restart event code</td>
</tr>
<tr>
<td>restart</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIv2 error</td>
<td>10</td>
<td>MultiLoad received a CLIv2 error</td>
<td>Error code—4-byte unsigned integer</td>
</tr>
<tr>
<td>Teradata Database</td>
<td>11</td>
<td>MultiLoad received a Teradata Database error that will produce an exit code of 12</td>
<td>Error code—4-byte unsigned integer</td>
</tr>
<tr>
<td>error</td>
<td></td>
<td></td>
<td>Note: Not all Teradata Database errors cause this event.</td>
</tr>
<tr>
<td>Exit</td>
<td>12</td>
<td>MultiLoad is terminating</td>
<td>Exit code—4-byte unsigned integer</td>
</tr>
<tr>
<td>AMPs down</td>
<td>21</td>
<td>The Teradata Database has one or more down AMPs, just prior to the EXEC MLOAD</td>
<td>No data accompanies the AMPs down event code</td>
</tr>
<tr>
<td>Import begin</td>
<td>22</td>
<td>The first record is about to be read for each import task</td>
<td>Import number—4-byte unsigned integer</td>
</tr>
<tr>
<td>Import end</td>
<td>23</td>
<td>The last record has been read for each import task. The returned data is the</td>
<td>• Records read—4-byte unsigned integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>record statistics for the import task</td>
<td>• Records skipped—4-byte unsigned integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Records rejected—4-byte unsigned integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Records sent to the Teradata Database—4-byte unsigned integer</td>
</tr>
<tr>
<td>Delete init.</td>
<td>24</td>
<td>Successful processing of a BEGIN DELETE MLOAD command</td>
<td>No data accompanies the init. event code.</td>
</tr>
<tr>
<td>Delete begin</td>
<td>25</td>
<td>A DELETE statement is about to be sent to the Teradata Database</td>
<td>• Table name length—4-byte unsigned integer</td>
</tr>
<tr>
<td>Delete end</td>
<td>26</td>
<td>Successful processing of the EXEC MLOAD request for a delete task</td>
<td>• Table name—128-character (maximum) array</td>
</tr>
<tr>
<td>Delete exit</td>
<td>27</td>
<td>The end of a delete task</td>
<td>Exit code—4-byte unsigned integer</td>
</tr>
</tbody>
</table>

**Rules and Restrictions for Using Routines**

The following sections describe operational rules and restrictions for using INMOD and notify exit routines in MultiLoad jobs.
Specifying Routines
INMOD and notify exit routine names must be unique within the system.

A MultiLoad job can specify one INMOD routine with each IMPORT command. These specifications can be to the same or different INMOD routines.

Each MultiLoad job can specify an exit routine with the NOTIFY… EXIT option of the BEGIN MLOAD command.

Compiling and Linking Routines
The methods for compiling and linking routines vary with the operating system. The following sections describe the methods for VM, MVS, UNIX, and Windows.

Using VM
On channel-attached VM client systems, INMOD and notify exit routines must be compiled under SAS/C and passed to CLINK with the following options:

- CLINK <filename>
- LKED
- LIBE
- DYNAMC
- NAME <modulename>

The resulting module, which can be loaded by SAS/C at run time, is placed in a load library called DYNAMC LOADLIB. (The first name must be DYNAMC because this is the only place that SAS/C looks for user load modules.)

Multiple load modules can exist in the local library as long as each module has a unique name.

Using MVS
The procedure on MVS platforms is similar to the procedure on VM platforms, with one exception: user load modules can be located anywhere, as long as the location is identified by one of the DD name STEPLIB specifications in the JCL.

Using UNIX
On network-attached UNIX client systems, INMOD and notify exit routines must:

- Be compiled with the MetaWare High C compiler
- Be linked into a shared object module
- Use an entry point named _dynamn

Using Windows
On network-attached Windows client systems, INMOD and notify exit routines must:

- Be written in C
- Have a dynamn entry point that is a __declspec
- Be saved as a dynamic-link library (DLL) file
For sample programs and procedures that compile and link INMOD and notify exit routines for your operating system environment, see the examples in Appendix C: “INMOD and Notify Exit Routine Examples.”

### Addressing Mode on VM and MVS Systems

On MultiLoad 07.00.00 and later, you can use either 31-bit or 24-bit addressing for INMOD routines on channel-attached systems. The 31-bit mode provides access to more memory, which enhances performance for MultiLoad jobs with a large number of sessions.

Use the following linkage parameters to specify the addressing mode when building INMOD routines for VM and MVS systems:

- For 31-bit addressing: AMODE(31) RMODE(24)
- For 24-bit addressing: AMODE(24) RMODE(24)

### INMOD Routine Compatibility with Other Load Utilities

You can use FDL-compatible INMOD routines that were created for FastLoad by including the FDLINMOD parameter as the USING (parms) option of your IMPORT command. Using this parameter provides compatible support operations, except for the way checkpointing is performed:

- If a MultiLoad job uses the FROM, FOR, or THRU options to request a range of records from an FDL-compatible INMOD routine, then MultiLoad bypasses any default record checkpointing.

  If the Teradata Database experiences a restart/recovery operation, MultiLoad starts over and gets the records again from the beginning of the range.

  Under these same circumstances, if your BEGIN MLOAD command included a CHECKPOINT rate other than 0, MultiLoad terminates with an error condition.

- If a MultiLoad job does not request a range of records, then MultiLoad performs checkpointing either by default or per your job specifications.

  If the Teradata Database experiences a restart/recovery operation, and the INMOD routine supports recovery, MultiLoad continues the data acquisition activity from the last recorded checkpoint.

  Note, however, that the source sequence numbers generated by MultiLoad may not correctly identify the sequence in which the INMOD routine supplied the records. The data is still applied correctly, however, despite this discrepancy.

You cannot specify an FDL-compatible INMOD routine with the INFILE specification of a MultiLoad IMPORT command.

When you specify an INMOD routine with the INFILE specification:

- MultiLoad performs the file read operation
- The INMOD routine acts as a pass-through filter

The combination of an FDL-compatible INMOD routine with a MultiLoad INFILE specification is not valid because an FDL-compatible INMOD routine must always perform the file read operation.
Checkpoints

To support MultiLoad restart operations, your INMOD routine must support checkpoint operations, as described in “MultiLoad/INMOD Routine Interface” on page 72.

If you use an INMOD routine that does not support the checkpoint function, your job may encounter problems when MultiLoad takes a checkpoint.

By default, MultiLoad takes a checkpoint every 15 minutes. You can bypass the MultiLoad checkpoint function by specifying a CHECKPOINT rate of zero in your BEGIN MLOAD command; that way, the job completes without taking a checkpoint.

When the CHECKPOINT rate is greater than 60, the specified number refers to the number of records, not the number of minutes. Though this would nullify the MultiLoad restart/reload capability, it would allow you to use an INMOD routine that does not support the checkpoint function.

Writing a MultiLoad Job Script

This section describes the contents of a MultiLoad job script and explains how to write one.

Definition

A MultiLoad job script, or program, is a set of MultiLoad commands and Teradata SQL statements that makes the following changes to specified target tables and views in the Teradata Database:

- Insert new rows
- Update the contents of existing rows
- Delete existing rows

Each MultiLoad job includes a number of support commands that establish and maintain the MultiLoad support environment, and a number of task commands that perform the actual Teradata Database insert, update, or delete operations.

Note: Though a single MultiLoad job can include a number of different tasks, limiting your jobs to a single task for each invocation of MultiLoad provides the highest assurance of a successful restart/recovery operation if a system failure interrupts your job.

Procedure

A complete MultiLoad job includes:

- Invoking MultiLoad
- Logging on to the Teradata Database and establishing the MultiLoad support environment
- Specifying the MultiLoad tasks
- Logging off from the Teradata Database and terminating MultiLoad.

Use the following procedure as a guide in writing MultiLoad job scripts.
1 Invoke MultiLoad, specifying your runtime options:
   - Normal or abbreviated (brief) printout
   - Character set
   - Alternate error message file
   - Standard input file/device
   - Standard output file/device
   - Alternate run file

2 Establish the MultiLoad support environment as shown in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>Specify an external data source file and character fields for setting utility variables for a MultiLoad job.</td>
</tr>
<tr>
<td>DATEFORM</td>
<td>Specify the form of the DATE data type specifications for the MultiLoad job.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Write a specified text message to a specified file.</td>
</tr>
<tr>
<td>IF, ELSE, and ENDIF</td>
<td>Provide conditional control of the execution process.</td>
</tr>
<tr>
<td>LOGON</td>
<td>Establish a session with the Teradata Database</td>
</tr>
<tr>
<td>LOGTABLE</td>
<td>Specify the restart log table that will maintain checkpoint information for the MultiLoad job.</td>
</tr>
<tr>
<td>ROUTE MESSAGES</td>
<td>Specify an alternate destination for MultiLoad utility output messages.</td>
</tr>
<tr>
<td>RUN FILE</td>
<td>Transfer processing control to an alternate run file.</td>
</tr>
<tr>
<td>SET</td>
<td>Assign a value to a specific utility variable for the MultiLoad job.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Submit an operating system command to your client system.</td>
</tr>
<tr>
<td>Teradata SQL statements</td>
<td>Submit a supported Teradata SQL statement to the Teradata Database.</td>
</tr>
</tbody>
</table>

**Note:** As a minimum, this part of the MultiLoad job must include:

- A LOGTABLE command to specify the restart log table.
- A LOGON command to provide a logon string that is used to connect all Teradata SQL and MultiLoad utility sessions with the Teradata Database.

3 Specify the MultiLoad task as shown in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN MLOAD and BEGIN DELETE MLOAD</td>
<td>Signify the beginning of a MultiLoad import or delete task.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove rows from the target tables or views.</td>
</tr>
</tbody>
</table>
Estimating Space Requirements

MultiLoad supports a maximum of 2048 columns. The statement text limit is 1 MB. This section describes space requirements for running MultiLoad jobs and explains how to determine the amount of space needed to ensure that your jobs can run to completion.

**Definition**

Always estimate the final size of your MultiLoad target tables and make sure that the destination database has enough space to complete your MultiLoad job.
If the database that owns the MultiLoad tables runs out of space, the Teradata Database returns an error message and MultiLoad stops the job. When this happens, you need to allocate more space to the database before you restart the MultiLoad job.

Your target tables might need extra space and each MultiLoad job needs permanent space for:

- Work tables
- Error tables
- The restart log table

**Note:** You cannot use spool space for MultiLoad work tables, error tables, or the restart log table. Spool space is freed at each restart. Using permanent space for the MultiLoad tables preserves the data for restart operations after a system failure.

Note that work tables, especially, require a large amount of extra permanent space.

**Procedure**

Estimating the amount of permanent space that a MultiLoad job requires is a three-step procedure:

1. Prepare a preliminary estimate for the work table and error table requirements, using a set of simplifying assumptions:
   - There is only one target table.
   - It has no fallback protection, no journals, and no NUSIs.
   - The job does not write to the error tables.
   - The Teradata Database does not experience a restart condition while the job is running.

2. Adjust the preliminary estimate according to the actual job requirements and expectations:
   - Fallback protection
   - Journals
   - Non-unique indexes
   - Multiple target tables
   - Error table entries
   - Teradata Database restarts

3. Compute and add the space requirements for the restart log table.

The following topics describe each step of the estimation procedure.

**Preparing the Preliminary Estimate**

Use the following formula to prepare the preliminary estimate for one target table, assuming no fallback protection, no journals, and no NUSIs:

\[
\text{PERM} = (\text{using data size} + 38) \times (\text{number of rows processed}) \times (\text{number of apply conditions satisfied}) \times (\text{number of Teradata SQL statements within the applied DML})
\]
For example, in the following MultiLoad job the preliminary estimate would be 84 * (number of rows processed):

```
.LOGLTABLE LOGTABLE001;
.LOGON TDP0/xxxxx,xxxxx;
.BEGIN MLOAD TABLES A;
.LAYOUT TRANSACTION;
.FILLER TRANSCODE * CHAR(4);
.FIELD PIDX * CHAR(2);
.FIELD VAL * CHAR(2);
.DML LABEL DEL;
DELETE FROM A WHERE UPI = :PIDX AND VAL = :VAL (SMALLINT);
.DML LABEL INS;
INSERT INTO A (UPI, VAL) VALUES (:PIDX, :VAL (SMALLINT));
.IMPORT INFILE DAT01 LAYOUT TRANSACTION APPLY DEL APPLY INS;
.END MLOAD;
.LOGOFF;
```

where

- using data size = 4 (2 for FIELD PIDX and FIELD VAL. FILLER fields are not counted.)
- number of apply conditions satisfied = 2, since the APPLY statements APPLY and APPLY INS are unconditional.
- number of Teradata SQL statements within the applied DML = 1, since the DML LABEL DEL and DML LABEL INS commands have one SQL statement each.

**Adjusting the Preliminary Estimate**

To your preliminary estimate for one target table (84 times number of rows processed per the example in the preceding topic), make adjustments according to the requirements and expectations of the MultiLoad job.

<table>
<thead>
<tr>
<th>If your MultiLoad job</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>uses fallback protection for the target table,</td>
<td>multiply your estimated permanent space requirement by 2.</td>
</tr>
<tr>
<td>uses journaling,</td>
<td>add space for journals.</td>
</tr>
<tr>
<td></td>
<td>Permanent space for journaling is proportional to the number of rows updated.</td>
</tr>
<tr>
<td></td>
<td>To estimate the amount of space to add for journaling, multiply your estimated permanent space requirement by a factor of:</td>
</tr>
<tr>
<td></td>
<td>• 1 for SN journaling (single before images, no after images)</td>
</tr>
<tr>
<td></td>
<td>• 2 for SS journaling (single before images, single after images)</td>
</tr>
<tr>
<td></td>
<td>• 2 for DN journaling (dual before images, no after images)</td>
</tr>
<tr>
<td></td>
<td>• 2 for ND journaling (no before images, dual after images)</td>
</tr>
<tr>
<td></td>
<td>• 3 for DS journaling (dual before images, single after images)</td>
</tr>
<tr>
<td></td>
<td>• 3 for SD journaling (single before images, dual after images)</td>
</tr>
<tr>
<td></td>
<td>• 4 for DD journaling (dual before images, dual after images)</td>
</tr>
</tbody>
</table>
### Computing the Restart Log Table Requirements

The space requirements of a MultiLoad restart log table are highly dependent on the specifics of the job. Although there are mandatory inserts for every MultiLoad job, others occur on a job-dependent basis.

Following is an example of the table definition for a MultiLoad restart log table, which is automatically created by MultiLoad:

```sql
CREATE TABLE SFD.samplelog ,FALLBACK ,
   NO BEFORE JOURNAL,
   NO AFTER JOURNAL
(
   LogType INTEGER,
   Seq INTEGER,
   ReqRC INTEGER,
   ReqType INTEGER,
   ReqLen INTEGER,
   ReqMsg VARCHAR(255),
   SysInfo VARBYTE(255),
   MiscInt1 INTEGER,
   MiscInt2 INTEGER,
   MiscInt3 INTEGER,
   MiscInt4 INTEGER,
   MiscInt5 INTEGER,
   MiscInt6 INTEGER,
   MiscInt7 INTEGER,
   MiscInt8 INTEGER,
   MLoadSeq INTEGER DEFAULT 0 ,
   MLoadImpSeq INTEGER,
   MLoadSrcSeq INTEGER,
   MLoadCkpt VARBYTE(255),
   RunDate DATE DEFAULT DATE ,
   RunTime FLOAT DEFAULT TIME )
UNIQUE PRIMARY INDEX( LogType ,Seq ,MLoadSeq );
```

<table>
<thead>
<tr>
<th>If your MultiLoad job</th>
<th>Then</th>
</tr>
</thead>
</table>
| uses NUSIs,           | use the following formula to estimate the space required for changes to the NUSIs:
|                       | $NUSIPERM = (index\ length + 66) \ast (number\ of\ changes\ to\ indexes) \ast ((1\ for\ insert\ or\ delete) OR (2\ for\ update))$ |
|                       | This formula does not account for fallback tables. To account for fallback tables, multiply the $NUSIPERM$ amount by 2. |
|                       | The $NUSIPERM$ space estimation would be further impacted by conditional APPLY expressions and by multiple NUSIs with different index lengths. |
| has multiple target tables, | repeat the preliminary estimate for each target table. |
| writes to the error tables, | add space requirements for the error tables—two per target table. |
|                       | Note that MultiLoad error tables have fallback protection and their size varies with the using data size specification. |
| is interrupted by a Teradata Database restart, | remember that there may be duplicate rows in MultiLoad work tables if the data placed there is not rolled back upon failure. (These are later removed by the sort operation.) |

If your MultiLoad job uses NUSIs, use the following formula to estimate the space required for changes to the NUSIs:

$$NUSIPERM = (index\ length + 66) \ast (number\ of\ changes\ to\ indexes) \ast ((1\ for\ insert\ or\ delete) OR (2\ for\ update))$$

This formula does not account for fallback tables. To account for fallback tables, multiply the $NUSIPERM$ amount by 2.

The $NUSIPERM$ space estimation would be further impacted by conditional APPLY expressions and by multiple NUSIs with different index lengths.

If your MultiLoad job is interrupted by a Teradata Database restart, remember that there may be duplicate rows in MultiLoad work tables if the data placed there is not rolled back upon failure. (These are later removed by the sort operation.)
From the table definition, the size of each row depends on the VARCHAR length, which can range from 0 to a maximum of 255:

- If the VARCHAR length is 0, then the size of each row is 82 bytes.
- If the VARCHAR length is 255, then the size of each row is 847 bytes.

The possible inserts for this table are as follows:

- Seven mandatory inserts:
  - Three inserts, one each, upon entry to the DML phase, the acquisition phase, and the application phase
  - One insert to mark the end of a MultiLoad task
  - Three inserts, one each, to mark the dropping of the ET and UT error tables, and the WT work tables
  - One insert for each Teradata SQL statement processed (for example, CREATE TABLE, DROP TABLE)
  - One insert for each input data checkpoint

Using these guidelines provides an estimate of 260 KB of space required for the example restart log table:

- 574 bytes for mandatory entries: 82 bytes * 7 = 574 bytes
- 2,541 bytes for Teradata SQL requests—maximum of 847 bytes multiplied by an average of 3: 847 * 3 = 2,541 bytes
- 127,050 bytes for checkpoints—assuming a 15 million row MultiLoad job, with checkpoints every 100,000 rows, for an average of 150 checkpoints: 847 bytes * 150 = 127,050 bytes
- 130,165 bytes for the fallback table—duplicating the size of the Restart log table itself: 574 bytes + 2,541 bytes + 127,050 bytes = 130,165 bytes
- 260,330 bytes total estimated restart log table space requirement

These calculations estimate that 260 K of space would probably suffice for the restart log table of the example job. Use a similar computation to estimate the requirements for your MultiLoad job.

**Handling MultiLoad Errors**

This section provides information and instructions for handling MultiLoad errors.

**Error Recording**

When MultiLoad encounters a data row that cannot be processed properly, it creates a row in one of the two error tables that are created for each target table in the MultiLoad job:

- Acquisition error table
- Application error table
Each error table row includes eight columns of information that you can use to determine the cause of the problem. Additionally, the acquisition error table includes the faulty record, and the application error table includes a mirror image of the target table.

The target tables for your MultiLoad job cannot have column names that are the same as the error table column names because the application error table includes a mirror image of the target table, preceded by the error information. If the column names are duplicated, the Teradata Database terminates your MultiLoad job and returns an Error 3861 message.

Notes on recording errors:

- The error limit that you set in the BEGIN MLOAD command pertains to the error threshold that limits the number of rejected records that can be written to the error tables during the *acquisition* phase of a MultiLoad import task.
- When there are no expressions involved and at most one conversion required before loading the data into the target table, the data conversion takes place in the *acquisition* phase and the converted data is loaded into the worktable. This means that any errors with the data or with the conversion will be detected in the *acquisition* phase.
- But, when expression(s) are involved or more than one level of conversion is required before inserting the data into the target table, expression evaluations or conversions on primary index fields are done in the *acquisition* phase and conversions on the rest of the fields are deferred until the *application* phase. Therefore, any errors with the primary index fields will be detected in the *acquisition* phase, and any errors with the rest of the fields will be detected in the *application* phase.

**Acquisition Error Table**

The first error table, called the acquisition error table, is the one specified as tname3 (default ET_tname1) in your BEGIN MLOAD command. It provides information about:

- All errors that occur during the acquisition phase of the MultiLoad job
- Some errors that occur during the application phase if the Teradata Database cannot build a valid primary index

*Table 23* shows the format of the acquisition error table.

**Table 23: Acquisition Error Table Format**

<table>
<thead>
<tr>
<th>Column</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImportSeq</td>
<td>Sequence number assigned to the IMPORT command in which the error occurred</td>
</tr>
<tr>
<td>DMLSeq</td>
<td>Sequence number assigned to the DML statement in which the error occurred</td>
</tr>
<tr>
<td>SMTSeq</td>
<td>Sequence number of the statement that was being executed when the error occurred</td>
</tr>
<tr>
<td>ApplySeq</td>
<td>Sequence number of the APPLY clause that was executing when the error occurred</td>
</tr>
<tr>
<td>SourceSeq</td>
<td>Data row number in the client file that the Teradata Database was building when the error occurred</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Code for the error</td>
</tr>
</tbody>
</table>
The second error table, called the application error table, is the one specified as `tname4` (default `UV_tname1`) in your BEGIN MLOAD command. It provides information about:

- Uniqueness violations
- Field overflow on columns other than primary index fields
- Constraint errors

Table 24 shows the format of the application error table.

**Note:** A copy (or mirror) of the target table columns precedes the ImportSeq column in the application error table. If the target table is a VIEW, then only those eight fields specified Table 24 will be added into the Application Error Table (UV_table).

### Application Error Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorField</td>
<td>Field name of the target table in which the error occurred</td>
</tr>
<tr>
<td>HostData</td>
<td>Client data that was being processed when the error occurred</td>
</tr>
</tbody>
</table>

**Column**

<table>
<thead>
<tr>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field name of the target table in which the error occurred</td>
</tr>
<tr>
<td>This field may be blank if the system cannot determine which field caused the problem. Error 2677 (stack overflow) is an example of such a condition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client data that was being processed when the error occurred</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number assigned to the IMPORT command in which the error occurred</td>
</tr>
<tr>
<td>Sequence number assigned to the DML statement in which the error occurred</td>
</tr>
<tr>
<td>Sequence number of the statement that was being executed when the error occurred</td>
</tr>
<tr>
<td>Sequence number of the APPLY clause that was executing when the error occurred</td>
</tr>
<tr>
<td>Data row number in the client file that the Teradata Database was building when the error occurred</td>
</tr>
<tr>
<td>Value that prevents duplicate row errors in the error table</td>
</tr>
<tr>
<td>It can be ignored in error handling.</td>
</tr>
<tr>
<td>Code for the error</td>
</tr>
<tr>
<td>Field name of the target table in which the error occurred</td>
</tr>
<tr>
<td>This field may be blank if the system cannot determine which field caused the problem. Error 2677 (stack overflow) is an example of such a condition.</td>
</tr>
</tbody>
</table>
Determining the Cause of Errors

The following topics provide an abbreviated MultiLoad task example, an error table listing, and a procedure for determining the cause of the error.

Note that, for simplicity, the task example includes only DML and IMPORT commands for a complete MultiLoad job.

The example procedure uses all of the error information from the error table. In most cases, you can determine the error cause by evaluating only one or two columns of the error table entry.

MultiLoad Task Example

The following task example shows the Sequence Type and Sequence Number columns that are the type and number assignments for each statement, and the Statement column shows the actual MultiLoad job statements.

<table>
<thead>
<tr>
<th>Sequence Type</th>
<th>Sequence Number</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>DML</td>
<td>001</td>
<td>.DML LABEL FIRSTDML;</td>
</tr>
<tr>
<td>STMT</td>
<td>001</td>
<td>INSERT INTO table1 VALUES( :FIELD1, :FIELD2 );</td>
</tr>
<tr>
<td>STMT</td>
<td>002</td>
<td>UPDATE table2 SET field3 = :FIELD3 WHERE field4 = :FIELD4;</td>
</tr>
<tr>
<td>DML</td>
<td>002</td>
<td>.DML LABEL SECONDDML;</td>
</tr>
<tr>
<td>STMT</td>
<td>001</td>
<td>DELETE FROM table3 WHERE field3 = :FIELD3;</td>
</tr>
<tr>
<td>IMPORT</td>
<td>001</td>
<td>.IMPORT INFILE file1 LAYOUT layout1</td>
</tr>
<tr>
<td>APPLY</td>
<td>001</td>
<td>APPLY FIRSTDML</td>
</tr>
<tr>
<td>IMPORT</td>
<td>002</td>
<td>.IMPORT INFILE file2 LAYOUT layout2</td>
</tr>
<tr>
<td>APPLY</td>
<td>001</td>
<td>APPLY FIRSTDML</td>
</tr>
<tr>
<td>APPLY</td>
<td>002</td>
<td>APPLY SECONDDML;</td>
</tr>
</tbody>
</table>

Error Table Listing

The following information from the first error table (the acquisition or ET_tname1 error table) indicates a problem with the example task.

<table>
<thead>
<tr>
<th>Import Seq</th>
<th>DML Seq</th>
<th>SMT Seq</th>
<th>Apply Seq</th>
<th>Source Seq</th>
<th>Error Code</th>
<th>Error Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>001</td>
<td>002</td>
<td>001</td>
<td>20456</td>
<td>2679</td>
<td>field3</td>
</tr>
</tbody>
</table>
Error Isolation Procedure

The following procedure shows how you can use the information from the error table to isolate the problem:

1. Check the DMLSeq field to find the statement being executed. It contains the sequence number 001.
2. Check the SMTSeq field. The sequence number 002 in this field means that the error occurred while executing the second statement after the first DML LABEL command, which is the UPDATE command in the example task.
3. Verify that the MultiLoad job script uses two DML LABEL commands—one for each IMPORT command.
4. Check the ImportSeq field. The value of 002 indicates that the error occurred while processing the second IMPORT command.
5. Check the ApplySeq field. The value of 001 indicates that the error occurred while processing the first APPLY function.
6. Check the meaning of the ErrorCode field. Error 2679, “The format or data contains a bad character” indicates a problem with the data from your client system.
7. Check the ErrorField field. The field3 indicates that the error occurred while building field3 of the target table.
8. Check the MultiLoad job script. It shows that field3 of the target table was being built from FIELD3 of the client data.
9. Note that the LAYOUT specification of the IMPORT command shows where the problem data is positioned within the row of the client data.
   Since the script shows that the IMPORT command was loading file2, you now know:
   • What error occurred
   • Which statement detected the error
   • Which file has the error
10. Check the SourceSeq field. The value of 20456 indicates that the problem is with the 20,456th record of the file.
    The problem is isolated, and can now be fixed.

Monitoring MultiLoad Performance

This section describes performance considerations when using MultiLoad and discusses how to troubleshoot performance problems.

Performance Considerations

The majority of client processing during a MultiLoad job occurs when it is processing its input rows. The most efficient means of sending the row to the Teradata Database would be a bulk move of the input row to the output row.
However, there are many cases where fields need to be evaluated and field data may need to be individually moved from the input to the output row. Note, however, that performance is affected whenever a field needs to be evaluated or individually moved.

The need for moving individual field data from the input to the output row occurs for any of the following scenarios:

- DROP syntax on FIELD statements
- FILLER fields
- Concatenated fields
- Complex layout (first field is variable-length field, redefinition of field positions)

Variable length fields, NULLIF in the layout, and APPLY WHERE clauses might require additional CPU consumption.

**Troubleshooting Performance Problems**

Use the following procedure to determine the cause of MultiLoad performance problems that do not produce a specific error indication.

1. Determine which phase of the MultiLoad job is causing the poor performance.
   - If it is during the acquisition phase, when data is acquired from the client system, the problem may be with the client system. If it is during the application phase, when data is applied to the target tables, the problem is not likely to be with the client system.
   - The MultiLoad job output lists the job phases and other useful information. Save these listings for evaluation later.

2. Use the Teradata Database QrySessn utility to monitor the progress of the MultiLoad job.

3. Check for locks on the MultiLoad target tables:
   - Use the Teradata Database Show Locks utility to check for Hut locks.
   - Use the Query Session utility or other utilities that use the performance monitor feature of the Teradata Database (Teradata Manager, Amon, or Pmon) to check for the “blocked” status, indicating transaction locks.

4. Check the DBC.Resusage table for problem areas, such as data bus or CPU capacities at or near 100 percent for one or more processors.

5. Determine whether the target tables have NUSIs. NUSIs degrade MultiLoad performance because the utility builds a separate NUSI change row to be applied to each NUSI subtable after all of the rows have been applied to the primary table.

6. Check the size of the error tables. Write operations to the fallback error tables are performed at normal SQL speed, which is much slower than normal MultiLoad tasks.

7. Verify that the primary index is unique. NUPIs can cause severe MultiLoad performance problems.
CHAPTER 3

MultiLoad Commands

This chapter describes the MultiLoad commands and Teradata SQL statements that can be executed from Teradata MultiLoad, plus the RELEASE MLOAD statement that can be entered from Multiload or BTEQ.

Experienced MultiLoad users can also refer to the simplified command descriptions in the MultiLoad chapter of the Teradata Tools and Utilities Command Summary. This book provides the syntax diagrams for each Teradata client utility.

Syntax Notes

Each MultiLoad command must:

- Begin on a new line
- Start with a period (.) character
- End with a semicolon (;) character

**Note:** Start or end delimiter characters are not recognized within string literals or comments.

Each command may continue for as many lines as necessary, as long as it satisfies the beginning and ending requirements.

See Appendix A: “How to Read Syntax Diagrams” for a description of how to read the syntax diagrams used in this book.
**ACCEPT**

**Purpose**

The ACCEPT command sets MultiLoad variables to the value of a specified:

- External data source and valid character fields
- Internal environment variable

**Syntax**

```
.ACCEPT [var] FROM FILE fileid ENVIRONMENT VARIABLE env_var
   IGNORE charpos1 THRU charpos2
   THRU charpos1 THRU charpos2
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charpos1 and charpos2</td>
<td>Start and end character positions of a field in each input record that contain extraneous information</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>• Use charpos1 to ignore only the single specified character.</td>
</tr>
<tr>
<td></td>
<td>• Use charpos1/THRU to ignore all characters from charpos1 through the end of the record.</td>
</tr>
<tr>
<td></td>
<td>• Use THRU charpos2 to ignore all characters from the beginning of the record through charpos2.</td>
</tr>
<tr>
<td></td>
<td>• Use charpos1/THRU charpos2 to ignore all characters from charpos1 through charpos2.</td>
</tr>
<tr>
<td>env_var</td>
<td>Environment variable that provides the value for the specified utility variables (var)</td>
</tr>
</tbody>
</table>
Chapter 3: MultiLoad Commands

ACCEPT

The following table describes the things to consider when using the ACCEPT command.

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileid</td>
<td>Data source of the external system</td>
</tr>
<tr>
<td></td>
<td>The external system DD (or similar) statement specifies a file.</td>
</tr>
<tr>
<td></td>
<td>• In MVS, this is a DDNAME. (See the “MVS fileid Usage Rules” topic in Usage Notes below.)</td>
</tr>
<tr>
<td></td>
<td>• In UNIX and Windows, this is the path name for a file. If the path name has embedded white space characters, you must enclose the entire path name in single or double quotes.</td>
</tr>
<tr>
<td></td>
<td>• In VM, this is a FILEDEF name.</td>
</tr>
<tr>
<td>var</td>
<td>Name of the MultiLoad variable that is to be set with the value accepted from the designated source</td>
</tr>
<tr>
<td></td>
<td>Character string values appear as quoted strings in the data file.</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the ACCEPT command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying the System Console/Standard Input Device</td>
<td>You can use the asterisk (*) character as the fileid specification for the system console/standard input (stdin) device.</td>
</tr>
<tr>
<td></td>
<td>The system console is the:</td>
</tr>
<tr>
<td></td>
<td>• Keyboard in interactive mode</td>
</tr>
<tr>
<td></td>
<td>• Standard input device in batch mode</td>
</tr>
<tr>
<td></td>
<td>For more information about the keyboard and standard input devices, see “File Requirements” on page 29.</td>
</tr>
<tr>
<td>MVS fileid Usage Rules</td>
<td>If you specify a DDNAME, MultiLoad reads data records from the specified source.</td>
</tr>
<tr>
<td></td>
<td>A DDNAME must obey the same construction rules as Teradata SQL column names except that:</td>
</tr>
<tr>
<td></td>
<td>• The “at” character (@) is allowed as an alphabetic character.</td>
</tr>
<tr>
<td></td>
<td>• The underscore character (_) is not allowed.</td>
</tr>
<tr>
<td></td>
<td>The DDNAME must obey the applicable rules of the external system.</td>
</tr>
<tr>
<td></td>
<td>If the DDNAME represents a data source on magnetic tape as supported by the operating system, the tape may be either labeled or non-labeled.</td>
</tr>
<tr>
<td>Source File Record Restriction</td>
<td>A single record, row, or input line is accepted from the designated source. Always make sure that there is only one record in the file from which the ACCEPT command is getting the variables.</td>
</tr>
<tr>
<td>Coding Multiple Variables</td>
<td>When you code multiple variables, each is sequentially assigned input text up to the first space character encountered that is not within a quoted string.</td>
</tr>
</tbody>
</table>
### Delimiting Input Text

Input text for numeric values must be delimited only by space characters or record boundaries.

Character strings must be enclosed in apostrophes. For example, the data record provided to satisfy the following ACCEPT command includes two fields:

```
.Acept age, name from file info;
```

The following example shows two sample data records, where the first is correct but the second is not:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>'Tom'</td>
</tr>
<tr>
<td>32</td>
<td>Tom</td>
</tr>
</tbody>
</table>

### Number of Variables

When the number of variables listed is greater than the number of responses available, unused variables remain undefined (null). If there are not enough variables to hold all responses, MultiLoad issues a warning message.
**BEGIN MLOAD and BEGIN DELETE MLOAD**

**Purpose**

The BEGIN MLOAD and BEGIN DELETE MLOAD commands initiate or restart MultiLoad import or delete tasks.

The syntax for the BEGIN MLOAD command is similar but different for each task.

**Syntax for Import Task**
### Syntax for Delete Task

![Diagram of .BEGIN DELETE MLOAD TABLES syntax]

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPCHECK</td>
<td>The MultiLoad response to a down AMP condition:</td>
</tr>
<tr>
<td></td>
<td>- AMPCHECK NONE allows the MultiLoad job to start, restart, or continue as long as no more than one AMP is down in a cluster.</td>
</tr>
<tr>
<td></td>
<td>- AMPCHECK APPLY inhibits the MultiLoad job from entering or exiting the application phase when an AMP is down.</td>
</tr>
<tr>
<td></td>
<td>- AMPCHECK ALL pauses the MultiLoad job when an AMP is down.</td>
</tr>
<tr>
<td></td>
<td>If you do not enter an AMPCHECK specification, the MultiLoad default is AMPCHECK APPLY.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>All of the target <code>tname1</code> tables in the MultiLoad job must be fallback tables for the job to start, restart, or continue with a down AMP. The job does not start or restart if any of the target tables are nonfallback.</td>
</tr>
</tbody>
</table>
### Chapter 3: MultiLoad Commands

BEGIN MLOAD and BEGIN DELETE MLOAD

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| CHECKPOINT rate        | The interval between checkpoint operations during the acquisition phase of a MultiLoad import task, expressed as either:  
  - the number of rows read from your client system or sent to the Teradata Database, or  
  - an amount of time in minutes.  
  
  If you specify a CHECKPOINT rate of 60 or more, a checkpoint operation occurs after each multiple of that number of records is processed.  
  If you specify a CHECKPOINT rate of less than 60, a checkpoint operation occurs at the specified frequency, in minutes.  
  
  **Note:** Specifying a CHECKPOINT rate of zero inhibits the checkpoint function—MultiLoad does not perform any checkpoint operations during the import task.  
  If you do not use the CHECKPOINT rate specification, MultiLoad performs a checkpoint operation at the default rate—every 15 minutes.  
  Checkpoints are not set if you use an FDL-compatible INMOD routine with the FOR, FROM, or THRU options. If you use an FDL-compatible INMOD routine with the FOR, FROM, or THRU options and specify a CHECKPOINT rate other than zero, MultiLoad terminates and an error message appears. |
| DATAENCRYPTION value   | Keyword that enables data encryption for the MultiLoad job; available on network platforms only  
  The options for value are:  
  - ON = All sessions will be encrypted.  
  - OFF = Sessions will not be encrypted. This is the default.  
  This option will apply only to the BEGIN (DELETE) MLOAD request and the requests after the BEGIN (DELETE) MLOAD command.  
  Using this option overwrites the data encryption settings specified by both the runtime parameters and the mloadcfg.dat configuration file. |
| dbname                 | The database for the target tables, work tables, error tables, and uniqueness violation tables  
  Use the dbname specification when you want to:  
  - Specify a qualified database for tname2, tname3, and tname4 that is different from the database for the tname1 target table or view  
  - Create and drop tables without involving the production database. |
| DELETE                 | Specifies that a delete task is to be initiated by the following MultiLoad commands |

Teradata MultiLoad Reference
### Syntax Element Description

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRLIMIT <code>errcount</code></td>
<td>Error threshold that limits the number of rejected records that can be written to the <code>tname3</code> error tables during the acquisition phase of a MultiLoad import task. When used without the <code>errpercent</code> parameter, <code>errcount</code> specifies the total number of records, as an unsigned integer, that can be rejected and recorded in the <code>tname3</code> error tables. When used with the <code>errpercent</code> parameter, which is an approximation, <code>errcount</code> specifies the number of records that must be sent to the Teradata Database before the <code>errpercent</code> parameter is applied. The default is no limit. <strong>Note:</strong> Uniqueness violations do not count as rejected records. For more information, see “Error Recording” on page 85.</td>
</tr>
<tr>
<td>ERRORTABLES</td>
<td>The path for error tables</td>
</tr>
<tr>
<td><code>errpercent</code></td>
<td>Approximate percentage, expressed as an integer, of the total number of records sent to the Teradata Database so far, that may be rejected during the acquisition phase of the MultiLoad import task.</td>
</tr>
</tbody>
</table>
| EXIT `name` | The keyword phrase that calls a user-defined exit where `name` is the name of a user-supplied library with a member name of `_dynamn`. The default library names are:  
- `NOTFYEXT` for channel-attached VM and MVS client systems  
- `libnotfyext.so` for network-attached UNIX and Windows client systems  
  The exit must be written in C, or in a language with a runtime environment that is compatible with C.  
  **Note:** On some versions of UNIX, you may have to add `./` prefix characters to the EXIT `name` specification if the module is in the current directory. |
| IMPORT | Specifies that an import task is to be initiated by the following MultiLoad commands |
| MSG `'string'` | A user-supplied string of up to 16 characters that MultiLoad logs to:  
  - The operator’s console for channel-attached VM and MVS client systems  
  - The system log for network-attached UNIX and Windows client systems  
  The `string` specification must be enclosed in single quote characters (`'`). |
### NOTIFY
The MultiLoad implementation of the notify user exit option:

- **NOTIFY OFF** suppresses the notify user exit option.
- **NOTIFY LOW** enables the notify user exit option for those events signified by “Yes” in the Low Notification Level column of Table 26.
- **NOTIFY MEDIUM** enables the notify user exit option for the most significant events, as specified by “Yes” in the Medium Notification Level column of Table 26.
- **NOTIFY HIGH** enables the notify user exit option for every MultiLoad event that involves an operational decision point, as specified by “Yes” in the High Notification Level column of Table 26.

### QUEUE option
The keyword phrase that specifies queue management options (option) on channel-attached MVS client systems.

**Note:** This option is available only on MVS, and only for import tasks with a low notification specification.

This option invokes an ENQ when the BEGIN MLOAD command is processed, followed by a DEQ when the significant event occurs.

The option specification is one of the following:

- **RNAME**
  A parameter containing a quoted string of up to 255 characters.
  The default is TDUSER.

- **SCOPE**
  A parameter that is one of the following:
  - **JOB**—Specifies that the QUEUE is local to the job, including all of the job steps.
  - **SYSTEM**—Specifies that the QUEUE is global to the computer running it.
  - **SYSTEMS**—Specifies that the QUEUE is global to all computers in the complex.
  The default is SYSTEMS.

- **NOBLOCK**
  A parameter specifying that if the ENQ blocks for any reason, it must return an error instead. This is a fatal error for the job.
  The default, an implied BLOCK (there is no BLOCK keyword), means that the ENQ will wait for the QUEUE.

---

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTIFY</td>
<td>The MultiLoad implementation of the notify user exit option:</td>
</tr>
<tr>
<td></td>
<td>- <strong>NOTIFY OFF</strong> suppresses the notify user exit option.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NOTIFY LOW</strong> enables the notify user exit option for those events signified by “Yes” in the Low Notification Level column of Table 26.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NOTIFY MEDIUM</strong> enables the notify user exit option for the most significant events, as specified by “Yes” in the Medium Notification Level column of Table 26.</td>
</tr>
<tr>
<td></td>
<td>- <strong>NOTIFY HIGH</strong> enables the notify user exit option for every MultiLoad event that involves an operational decision point, as specified by “Yes” in the High Notification Level column of Table 26.</td>
</tr>
<tr>
<td>QUEUE option</td>
<td>The keyword phrase that specifies queue management options (option) on channel-attached MVS client systems.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This option is available only on MVS, and only for import tasks with a low notification specification.</td>
</tr>
<tr>
<td></td>
<td>This option invokes an ENQ when the BEGIN MLOAD command is processed, followed by a DEQ when the significant event occurs.</td>
</tr>
<tr>
<td></td>
<td>The option specification is one of the following:</td>
</tr>
</tbody>
</table>
|               |   - **RNAME**
|               |     A parameter containing a quoted string of up to 255 characters. |
|               |     The default is TDUSER. |
|               |   - **SCOPE**
|               |     A parameter that is one of the following:
|               |     - **JOB**—Specifies that the QUEUE is local to the job, including all of the job steps. |
|               |     - **SYSTEM**—Specifies that the QUEUE is global to the computer running it. |
|               |     - **SYSTEMS**—Specifies that the QUEUE is global to all computers in the complex. |
|               |     The default is SYSTEMS. |
|               |   - **NOBLOCK**
|               |     A parameter specifying that if the ENQ blocks for any reason, it must return an error instead. This is a fatal error for the job. |
|               |     The default, an implied BLOCK (there is no BLOCK keyword), means that the ENQ will wait for the QUEUE. |
Chapter 3: MultiLoad Commands
BEGIN MLOAD and BEGIN DELETE MLOAD

**SESSIONS**…

The maximum number of MultiLoad sessions that are logged on when you enter a LOGON command and, optionally, the minimum number of sessions required to run the job:

- The `max` parameter specifies the number of sessions to log on.
  - The `max` specification must be within the range of 1 to 32767, but if you specify a SESSIONS `max` value that is larger than the number of available AMPs, MultiLoad limits the sessions to one per working AMP.
  - The default maximum, if you do not use the SESSIONS option, is one session for each AMP.
- The `min` parameter specifies the minimum number of sessions required for the job to continue.
  - The `min` specification must be greater than zero.
  - The default minimum, if you do not use the SESSIONS option or specify a `min` value, is 1.
- The asterisk (*) parameter specifies the maximum and minimum number of sessions.
  - Using the asterisk character as the `max` specification logs on for the maximum number of sessions—one for each AMP.
  - Using the asterisk character as the `min` specification logs on for at least one session, but less than or equal to the `max` specification.
  - Specifying SESSIONS `* *` has the same effect as not using the SESSIONS option at all.

On large to very large Teradata Database configurations, the default of one session per AMP may be inappropriate. For more information about how to configure sessions, see “Usage Notes” on page 105.

**SLEEP minutes**

The SLEEP runtime option in which `minutes` is the length of time that MultiLoad waits before retrying a logon operation.

**Note:** The SLEEP specification must be greater than zero. If you specify zero, MultiLoad responds with an error message and uses the default value.

The default value is 6 minutes.

**TABLEWAIT hours**

The TABLEWAIT runtime option in which `hours` is the number of hours that Multiload continues trying to start Multiload when one of the target tables is being loaded by some other job.

When the Multiload utility tries to start MultiLoad, and the Teradata Database indicates that the target table is being loaded (Error 2574), the Multiload utility:

- Waits for 6 minutes, by default, or for the amount of time specified by the SLEEP runtime option.
- Then tries to start MultiLoad again.

Multiload repeats this process until it has either acquired the table or exceeded the TABLEWAIT (`hours`) time period.
The TENACITY runtime option in which hours is the number of hours that MultiLoad continues trying to log on when the maximum number of load jobs is already running on the Teradata Database.

The default tenacity value is 4 hours.

The Teradata Database either allows MultiLoad to log on for all of the required data loading sessions, or rejects the logon because load capacity is currently completely consumed.

When MultiLoad tries to log on for a new task, and the Teradata Database indicates that the maximum number of load sessions is already running, MultiLoad:

1. Waits for 6 minutes, by default, or for the amount of time specified by the SLEEP runtime option.
2. Tries again to log on to the Teradata Database.

MultiLoad repeats this process until it has either logged on for the required number of sessions or exceeded the TENACITY hours time period.

A user-supplied string of up to 80 characters that MultiLoad passes to the named exit routine.

The string specification must be enclosed in single quote characters (').

You can specify:

- The target table or view for an import task
- The target table for a delete task
  (The target cannot be a view for a delete task.)

You can specify:

- Up to five target tables or views for an import task
- One target table for a delete task

Each tname1 specification must identify an existing target table or view.

If you do not specify the database, MultiLoad uses the default database to qualify the table or view name.
Work tables are special un-hashed tables that Teradata MultiLoad uses when executing both import and delete tasks.

Each `tname2` specification must identify:
- A new (nonexisting) table for a non-restart task
- An existing table for a restart task

Otherwise, MultiLoad terminates the task with an error message.

If you do not use the WORKTABLES specification, MultiLoad creates the `tname2` work tables using a default name of `WT_tname1` for each `tname1` table or view that you specify.

MultiLoad automatically drops the work tables after completing the import or delete task.

**Caution:** In the case of a paused MultiLoad job, do not drop the work tables or error tables until MultiLoad completes the import or delete task.

For more information about dropping the work tables or error tables, see “Implications of Dropping Required MultiLoad-Created Tables” on page 40.

Do not share the work tables between two or more MultiLoad jobs. Do not share work tables between two or more target tables in a single MultiLoad job. Each table targeted by a MultiLoad job must have its own work table to ensure that the job runs correctly.

If you do not use a distinct work table for each table targeted by a MultiLoad job, the results are unexpected. You may not be able to restart one or more of the affected jobs.

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tname2</code></td>
<td>The work table for each <code>tname1</code> table or view</td>
</tr>
</tbody>
</table>
The fallback error table for each `tname1` table or view that receives information about errors detected during the acquisition phase of the MultiLoad import task.

Each `tname3` specification must identify:

- A new (nonexisting) table for a non-restart task
- An existing table for a restart task

Otherwise, MultiLoad terminates the task with an error message.

If you do not use the `ERRORTABLES` specification, MultiLoad creates the `tname3` error tables using a default name of `ET_tname1` for each target `tname1` table or view that you specify.

**Caution:** In the case of a paused MultiLoad job, do not drop the error tables or the work tables until MultiLoad completes the import or delete task. For more information about dropping the work tables or error tables, see “Implications of Dropping Required MultiLoad-Created Tables” on page 40.

Do not share the acquisition error tables between two or more MultiLoad jobs. Do not share acquisition error tables between two or more target tables in a single MultiLoad job. Each table targeted by a MultiLoad job must have its own acquisition error table to ensure that the job runs correctly.

If you do not use a distinct acquisition error table for each table targeted by a MultiLoad job, the results are unexpected. You may not be able to restart one or more of the affected jobs.

The fallback error table for each `tname1` table or view that receives information about errors detected during the application phase of the MultiLoad import or delete task.

Each `tname4` specification must identify:

- A new (nonexisting) table for a non-restart task
- An existing table for a restart task

Otherwise, MultiLoad terminates the task with an error message.

If you do not use the `ERRORTABLES` specification, MultiLoad creates the `tname4` error tables using a default name of `UV_tname1` for each target `tname1` table or view that you specify.

**Caution:** In the case of a paused MultiLoad job, do not drop the error tables or the work tables until MultiLoad completes the import or delete task. For more information about dropping the work tables or error tables, see “Implications of Dropping Required MultiLoad-Created Tables” on page 40.

Do not share the application error tables between two or more MultiLoad jobs. Do not share application error tables between two or more target tables in a single MultiLoad job. Each table targeted by a MultiLoad job must have its own application error table to ensure that the job runs correctly.

If you do not use a distinct application error table for each table targeted by a MultiLoad job, the results are unexpected. You may not be able to restart one or more of the affected jobs.
### Table 26: Events that Create Notifications

<table>
<thead>
<tr>
<th>Event</th>
<th>Import Task</th>
<th>Delete Task</th>
<th>Notification Level</th>
<th>Signifies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Initialize</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>File or INMOD open</td>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Phase 1 begin</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Checkpoint</td>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Phase 1 end</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Phase 2 begin</td>
<td>X</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Phase 2 end</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Error table 1</td>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Error table 2</td>
<td>X</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Teradata Database restart</td>
<td>X</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CLIV2 error</td>
<td>X</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Teradata Database error</td>
<td>X</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exit</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AMPs down</td>
<td>X</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Import begin</td>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Import end</td>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Usage Notes

The following table describes the things to consider when using the BEGIN MLOAD and BEGIN DELETE MLOAD commands.

<table>
<thead>
<tr>
<th>Event</th>
<th>Import Task</th>
<th>Delete Task</th>
<th>Notification Level</th>
<th>Signifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete init</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BEGIN DELETE MLOAD command processing.</td>
</tr>
<tr>
<td>Delete begin</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DELETE statement about to be sent to the Teradata Database.</td>
</tr>
<tr>
<td>Delete end</td>
<td>X</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Successful delete task processing.</td>
</tr>
<tr>
<td>Delete exit</td>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>End of delete task.</td>
</tr>
</tbody>
</table>

### Usage Notes

**Views as Import Target Objects**

To be a valid target object in a MultiLoad import task, a view must:
- Be a single-table view
- Permit updates to the underlying table
Also, a view must not:
- Specify either a join or self-join
- Convert primary index columns

**Unspecified Target Objects**

During processing after the BEGIN MLOAD command, MultiLoad terminates with an error message if it encounters a reference to:
- A target table or view not identified in the BEGIN MLOAD command
- An incompatible view
<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
</table>
| Error Table Column Names Not Allowed in Target Tables | MultiLoad posts the faulty client record information. MultiLoad also posts information that identifies the record and the type of error to the UV_ and ET_ error tables. This additional information is provided in eight columns that precede the faulty records from the client system. To maintain the integrity of the MultiLoad error table information, your target tables must not use any of the eight error table column names:  
  - ApplySeq  
  - DBCErrorCode  
  - DBCErrorField  
  - DMLSeq  
  - ImportSeq  
  - SMTSeq  
  - SourceSeq  
  - Uniqueness |
| Session Configurations                  | There is no general method to determine the optimal number of sessions, because this number is dependent on several factors, including, but not limited to:  
  - Teradata Database performance and workload  
  - Client platform type, performance, and workload  
  - Channel performance, for channel-attached systems  
  - Network topology and performance, for network-attached systems  
  - Volume of data to be processed by the application  
  Using too few sessions is likely to unnecessarily limit throughput. On the other hand, using too many sessions can increase session management overhead (and also reduce the number of sessions available to any other applications) and may, in some circumstances, degrade throughput.  
  Regardless of the size of the Teradata Database configuration, for large repetitive production applications, it will usually be appropriate to experiment with several different session configurations to determine the best trade-off between resource utilization and throughput performance.  
  For larger Teradata Database configurations, it is appropriate to establish an installation default for the maximum number of sessions that is less than one session per AMP. This can be done either with the installation configuration file (see “MultiLoad Configuration File” on page 46) or with a standard runtime parameter (see “Runtime Parameters” on page 31). An installation default for number of sessions, if specified in the configuration file, can be overridden in individual MultiLoad scripts, when necessary. |
### Topic: Database and Table Name Specifications

The database, target table or view, work table, and error table names that you specify with the BEGIN MLOAD command (*dbname* and *tname1* through *tname4*), can be up to 30 characters long. If you use more than 30 characters, MultiLoad terminates with an error message.

If you do not specify the work table or error table names, MultiLoad creates them using the target table or view names with a three-character prefix of WT_, ET_, or UV_, as in:

- WT_*tname1* for the *tname2* work tables
- ET_*tname1* for the *tname3* error tables
- UT_*tname1* for the *tname4* error tables

If your target table or view name specifications are longer than 27 characters, MultiLoad issues a warning message and truncates them, from the right, to create 30-character default names for the work tables and error tables.

**Caution:** Do not share the work and error tables between two or more MultiLoad jobs. Do not share work and error tables between two or more target tables in a single MultiLoad job. Each table targeted by a MultiLoad job must have its own work and error tables to ensure that the job runs correctly. If you do not use a distinct work and error table for each table targeted by a MultiLoad job, the results are unexpected. You may not be able to restart one or more of the affected jobs.

### Topic: Dropping MultiLoad Work and Error Tables

In the case of a paused MultiLoad job, do not drop the work tables or the error tables until MultiLoad completes the import or delete task.

For more information, see “Implications of Dropping Required MultiLoad-Created Tables” on page 40.
DATEFORM

Purpose

The DATEFORM command specifies the form of the DATE data type specifications for the MultiLoad job.

Syntax

```plaintext
.DATEFORM INTEGERDATE

where
```

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSIDATE</td>
<td>Keyword that specifies ANSI fixed-length CHAR(10) DATE data types for the MultiLoad job</td>
</tr>
<tr>
<td>INTEGERDATE</td>
<td>Keyword that specifies integer DATE data types for the MultiLoad job</td>
</tr>
<tr>
<td></td>
<td>This is the default Teradata DATE data type specification for MultiLoad jobs if you do not enter a DATEFORM command.</td>
</tr>
</tbody>
</table>

Usage Notes

The topics in the following table describe the things to consider when using the DATEFORM command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Frequency and Placement</td>
<td>You can only use one DATEFORM command.</td>
</tr>
<tr>
<td></td>
<td>You must enter the command before the LOGON command.</td>
</tr>
<tr>
<td>Data Type Conversions</td>
<td>When you use the ANSIDATE specification, you must convert ANSI/SQL DateTime data types to fixed-length CHAR data types when specifying the column/field names in the MultiLoad FIELD command.</td>
</tr>
<tr>
<td></td>
<td>See the command usage notes for “FIELD” on page 119 command description for a description of the fixed-length CHAR representations for each DATE, TIME, TIMESTAMP, and INTERVAL data type specification.</td>
</tr>
<tr>
<td>Release Applicability</td>
<td>The ANSIDATE specification is valid for MultiLoad jobs.</td>
</tr>
</tbody>
</table>
DELETE

Purpose

DELETE is a Teradata SQL statement that removes rows from a table or view that was previously identified as a target table or view in a BEGIN MLOAD command.

Syntax

```
DELETE [DEL] tname WHERE condition
FROM dbname.
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbname</td>
<td>Explicit qualification of the database for the target table or view. The default, if you do not enter a <code>dbname</code> specification, is the current default database for the MultiLoad sessions.</td>
</tr>
<tr>
<td>tname</td>
<td>Previously identified target table or view.</td>
</tr>
<tr>
<td>WHERE condition</td>
<td>Conditional clause identifying the rows to delete.</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the DELETE statement.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluating the Conditional Clause</td>
<td>The result of evaluating the conditional clause for a row can be either positive, negative, or indeterminate:</td>
</tr>
<tr>
<td></td>
<td>• If positive, then MultiLoad deletes the row.</td>
</tr>
<tr>
<td></td>
<td>• If negative, then MultiLoad does nothing.</td>
</tr>
<tr>
<td></td>
<td>• If indeterminate, then MultiLoad treats the result as an error condition and records the row and error number in the error table.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Indeterminate results are caused by an abnormal arithmetic condition, such as underflow, overflow, or division by zero.</td>
</tr>
</tbody>
</table>
### Using DELETE in Import Tasks

If you used a BEGIN MLOAD command to specify an import task:

- You can apply DELETE statements to either a table or a view, provided that the view does not specify a join.
- The number of input data records is unlimited.
- You must specify equality values for all the primary index columns in the WHERE clause.
- You must specify values for the partitioning columns if the target table has PPI.
- You must precede the DELETE statement with a DML LABEL command.
- You cannot use the OR construct in the WHERE clause of a DELETE statement. Instead, use two separate DELETE statements and apply them conditionally with the APPLY clause of the IMPORT command.

### Using DELETE in Delete Tasks

If you used a BEGIN DELETE MLOAD command to specify a delete task:

- You can apply the single DELETE statement to a table only, not to a view.
- The DELETE operation requires a full file scan.
- You must not specify equality values for all the primary index columns in the WHERE clause. You may, however, specify equality values for a lesser number of columns of a unique primary index. The columns specified in this clause need not be a part of any index, but can be one or more nonindexed columns.
- This clause may specify nonequality values for any combination of columns of unique indices, or any values for other columns.
- You must not precede the DELETE statement with a DML LABEL command. The single DELETE statement is always applied.
- You can optionally provide a single imported data record, properly defined by a LAYOUT command and its associated FIELD commands (and, optionally, FILLER commands), which is then specified in the single IMPORT command. The data record provides values for the WHERE clause of the DELETE statement.

In the absence of an IMPORT command, the WHERE clause of the DELETE statement must statically specify the rows to be deleted, without reference to any dynamic data from an imported data record.

### Joins

MultiLoad operates only on single-table statements. DELETE statements cannot contain joins.

### Required Access Privilege

To use the DELETE statement, you must have the DELETE privilege on the specified table or view.
Example

Using DELETE in an Import Task

In the following example, an input data source contains a series of one-field, four-byte records. Each record contains the value (EmpNum) of the primary index column (EmpNo) of a row to be deleted from the Employee table. The example assumes that the Employee table is in the current default database.

```
.LAYOUT Layoutname;
.FIELD EmpNum 1 INTEGER;
.DML LABEL DMLlabelname;
DELETE FROM Employee WHERE EmpNo = :EmpNum;
```
Chapter 3: MultiLoad Commands

DISPLAY

Purpose

The DISPLAY command writes messages to a specified destination.

Syntax

```
.DISPLAY 'text' TO FILE fileid
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fileid</code></td>
<td>Data source of the external system</td>
</tr>
<tr>
<td></td>
<td>The external system DD (or similar) statement specifies a file.</td>
</tr>
<tr>
<td></td>
<td>• In MVS, this is a DDNAME. (See the “MVS fileid Usage Rules” topic in “Usage Notes” on page 115.)</td>
</tr>
<tr>
<td></td>
<td>• In UNIX and Windows, this is the path name for a file. If the path name has embedded white space characters, you must enclose the entire pathname in single or double quotes.</td>
</tr>
<tr>
<td></td>
<td>• In VM, this is a FILEDEF name.</td>
</tr>
<tr>
<td><code>'text'</code></td>
<td>Text to be written to the specified output destination</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the DISPLAY command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying the System Console/Standard Output Device</td>
<td>You can use the asterisk (*) character as the <code>fileid</code> specification to direct the display messages to the system console/standard output (stdout) device.</td>
</tr>
<tr>
<td></td>
<td>The system console is the:</td>
</tr>
<tr>
<td></td>
<td>• Display screen in interactive mode</td>
</tr>
<tr>
<td></td>
<td>• Standard output device in batch mode</td>
</tr>
<tr>
<td></td>
<td>For more information about the display screen and standard output devices, see “File Requirements” on page 29.</td>
</tr>
<tr>
<td>Topic</td>
<td>Usage Notes</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Utility Variables</td>
<td>Utility variables are replaced by their values before text is displayed. This is done by preceding the variable name with an ampersand character (&amp;). To display the name of a utility variable, use two ampersand characters (&amp;&amp;) instead of one.</td>
</tr>
<tr>
<td>Displaying Apostrophes in the Text String</td>
<td>To display an apostrophe within the text string, use two consecutive apostrophes (single quotes) to distinguish it from both the single quotes enclosing the string and a regular double-quote character.</td>
</tr>
<tr>
<td>MVS fileid Usage Rules</td>
<td>A DDNAME must obey the same construction rules as Teradata SQL column names except that:</td>
</tr>
<tr>
<td></td>
<td>• The “at” character (@) is allowed as an alphabetic character.</td>
</tr>
<tr>
<td></td>
<td>• The underscore character (_) is not allowed.</td>
</tr>
<tr>
<td></td>
<td>The DDNAME must obey the applicable rules of the external system.</td>
</tr>
<tr>
<td></td>
<td>If the DDNAME represents a data source on magnetic tape, the tape may be either labeled or non-labeled, as supported by the operating system.</td>
</tr>
<tr>
<td>Conflicting Write Operations on Network-attached Systems</td>
<td>On network-attached client systems, if you specify the same file to redirect stdout and as the file in a DISPLAY command, the results may be incomplete due to conflicting write operations to the same file.</td>
</tr>
</tbody>
</table>
DML LABEL

Purpose

The DML LABEL command defines a label and error-treatment options for one or more immediately following INSERT, UPDATE, and DELETE statements.

**Note:** When using both UPDATE and INSERT statements, the resulting operation is referred to as an upsert.

Syntax

```
.DML LABEL label
  DO INSERT FOR
  MARK
  DUPLICATE
  ROWS

  IGNORE
  MISSING
  INSERT
  UPDATE
  MISSING UPDATE

  UPDATE
  DELETE
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DO INSERT FOR</code></td>
<td>An upsert may be implemented by subsequent UPDATE and INSERT statements for:</td>
</tr>
<tr>
<td></td>
<td>• ROWS</td>
</tr>
<tr>
<td></td>
<td>• MISSING UPDATE ROWS</td>
</tr>
<tr>
<td><code>label</code></td>
<td>The unique name of the label that is used for the immediately following set of one or more INSERT, UPDATE, or DELETE statements. The <code>label</code> name must obey the same construction rules as Teradata SQL column names.</td>
</tr>
<tr>
<td></td>
<td>You can reference the <code>label</code> name in the APPLY clause of an IMPORT command.</td>
</tr>
</tbody>
</table>
### Syntax Element

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARK or IGNORE</td>
<td>MultiLoad either places (MARK) or does not place (IGNORE) rows in the <code>tname4</code> error table for the type of entry you specify:</td>
</tr>
<tr>
<td></td>
<td>• DUPLICATE</td>
</tr>
<tr>
<td></td>
<td>• DUPLICATE INSERT</td>
</tr>
<tr>
<td></td>
<td>• DUPLICATE UPDATE</td>
</tr>
<tr>
<td></td>
<td>• MISSING</td>
</tr>
<tr>
<td></td>
<td>• MISSING UPDATE</td>
</tr>
<tr>
<td></td>
<td>• MISSING DELETE</td>
</tr>
</tbody>
</table>

If a uniqueness violation occurs with MARK specified, the duplicate rows go to the uniqueness violation table.

IGNORE DUPLICATE ROWS does not apply if there are any unique indexes in the table.

In the case of an upsert operation, both the insert and update portions must fail for an error to be recorded. In this case, the mark rows for the missing update operations then have nulls for the target table columns.

If you do not specify either INSERT or UPDATE with DUPLICATE, then the MARK or IGNORE specification applies to both insert and update operations.

Similarly, if you do not specify either UPDATE or DELETE with MISSING, then the MARK or IGNORE specification applies to both update and delete operations.

**Note:** MARK is the default for all actions except MISSING UPDATE for an upsert operation.

---

## Usage Notes

The following table describes the things to consider when using the DML LABEL command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bypassing the Duplicate Row Check</td>
<td>If you are updating or inserting a row into a table with a NUPI, and the table allows duplicate rows, MultiLoad automatically bypasses the duplicate-row check.</td>
</tr>
<tr>
<td></td>
<td>If the table is a multiset table, MultiLoad will not check for duplicate rows. In this case, specifying IGNORE DUPLICATE ROWS will have no effect.</td>
</tr>
</tbody>
</table>
### Usage Notes

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upsert Feature</strong></td>
<td>When using the <em>upsert</em> feature:</td>
</tr>
<tr>
<td></td>
<td>• There must be exactly two DML statements in the DML group.</td>
</tr>
<tr>
<td></td>
<td>• The first DML statement must be an UPDATE statement that follows all of the MultiLoad task rules.</td>
</tr>
<tr>
<td></td>
<td>• The second DML statement must be an INSERT statement, and it must reflect the same primary index specified in the WHERE clause of the UPDATE statement. This is true for both a single-column primary index and a compound primary index.</td>
</tr>
<tr>
<td></td>
<td>• The value for the partitioning column for an update must be specified if the target table has PPI.</td>
</tr>
<tr>
<td></td>
<td>• Both DML statements must refer to the same table.</td>
</tr>
<tr>
<td></td>
<td>• The data length you define for any character fields in the WHERE condition of an UPDATE statement for an upsert must be less than or equal to the corresponding field length. Otherwise, the upsert is not processed. For example:</td>
</tr>
<tr>
<td></td>
<td><code>UPDATE T1 SET c1=:f2 WHERE P1=:f1</code></td>
</tr>
<tr>
<td></td>
<td>If P1 is char(5) and f1 is char(6), the upsert is canceled.</td>
</tr>
<tr>
<td><strong>DO INSERT FOR ROWS</strong></td>
<td>By following the rules for <em>upsert</em> operations, you will find a number of uses for the DO INSERT ROWS option.</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td>With an <em>upsert</em> operation, MultiLoad needs only one pass of the data to both:</td>
</tr>
<tr>
<td></td>
<td>• Update the rows that need to be updated.</td>
</tr>
<tr>
<td></td>
<td>• Insert the rows that need to be inserted.</td>
</tr>
<tr>
<td></td>
<td>The alternative would be to either:</td>
</tr>
<tr>
<td></td>
<td>• Presort the data for the update and insert operations.</td>
</tr>
<tr>
<td></td>
<td>• First use an UPDATE statement with all of the data, and then use an INSERT statement with the data that failed the update operation.</td>
</tr>
<tr>
<td><strong>Import Tasks</strong></td>
<td>For import tasks, you can specify as many as five distinct error treatment options with one DML LABEL command. For example:</td>
</tr>
<tr>
<td></td>
<td><code>.DML LABEL COMPLEX IGNORE DUPLICATE INSERT ROWS MARK DUPLICATE UPDATE ROWS IGNORE MISSING UPDATE ROWS MARK MISSING DELETE ROWS DO INSERT FOR MISSING UPDATE ROWS;</code></td>
</tr>
<tr>
<td><strong>MARK MISSING UPDATE ROWS</strong></td>
<td>If you specify MARK MISSING UPDATE ROWS while using the DO INSERT ROWS option, MultiLoad records any update record that fails in the tname4 application error table, along with an error code indicating that the INSERT statement was then executed.</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td>Then, if the insert operation fails, the insert record is also recorded in the application error table.</td>
</tr>
<tr>
<td></td>
<td>The default for an <em>upsert</em> operation, however, is to not mark missing update rows. This is because when you use the <em>upsert</em> feature, you expect the insert operation to occur when the update operation fails.</td>
</tr>
<tr>
<td></td>
<td>Failure of the update portion of an <em>upsert</em> operation does not, in itself, constitute an error and it is not to be treated as one.</td>
</tr>
</tbody>
</table>
Chapter 3: MultiLoad Commands

DML LABEL

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARK MISSING DELETE ROWS Option</td>
<td>The MARK MISSING DELETE ROWS option has no meaning when used with the DO INSERT FOR ROWS option.</td>
</tr>
</tbody>
</table>

Import Tasks

For import tasks, you can specify as many as five distinct error treatment options with one DML LABEL command. For example:

```
.DML LABEL COMPLEX
IGNORE DUPLICATE INSERT ROWS
MARK DUPLICATE UPDATE ROWS
IGNORE MISSING UPDATE ROWS
MARK MISSING DELETE ROWS
DO INSERT FOR MISSING UPDATE ROWS;
```

**Upsert Example**

Each record in the following *upsert* example contains the value of the primary index column (EmpNo) of a row of the Employee table whose PhoneNo column is to be assigned a new phone number from field Fone.

The example assumes that the current default database is Personnel—the database containing Employee.

```
.BEGIN IMPORT MLOAD TABLES Employee
.LAYOUT Layoutname;
.FIELD EmpNum 1 INTEGER;
.FIELD Fone * (CHAR (10));
.DML LABEL DMLlabelname
DO INSERT FOR MISSING UPDATE ROWS;
UPDATE Employee SET PhoneNo = :Fone WHERE EmpNo = :EmpNum;
INSERT Employee (EmpNo, PhoneNo) VALUES (:EmpNum, :Fone);
```

When the update operation fails, the INSERT statement executes, per the *upsert* feature. In this case, each record contains the primary key value (EmpNum) of a row that is to be inserted successively into the Employee table whose columns are EmpNo and PhoneNo.
Purpose

The END MLOAD command, which must be the last command of a MultiLoad task, signifies the end of the task script and initiates task processing by the Teradata Database.

Syntax

```
.END MLOAD   ;
```
**FIELD**

**Purpose**

The FIELD command specifies a field of the input record to be sent to the Teradata Database. It can also specify a NULLIF expression.

**Syntax**

```
.FIELD fieldname1 startpos datadesc
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>datadesc</code></td>
<td>Type and length of data in the field</td>
</tr>
<tr>
<td></td>
<td>MultiLoad generates the USING phrase accordingly, with the user-assigned field name, for subsequent insert, update, and delete operations.</td>
</tr>
<tr>
<td></td>
<td>For details on data types and data conversions, see the SQL Reference: Data Definition Statements for details.</td>
</tr>
<tr>
<td><code>DROP...</code></td>
<td>Character positions to be dropped from the specified <code>fieldname1</code></td>
</tr>
<tr>
<td></td>
<td>These must be of a character data type.</td>
</tr>
<tr>
<td></td>
<td>MultiLoad drops the specified characters and presents the field to the Teradata Database as VARCHAR data type.</td>
</tr>
</tbody>
</table>
Chapter 3: MultiLoad Commands

FIELD

### Syntax Element

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| `fieldexpr`    | Concatenation of two or more items, either:  
|                | - fields  
|                | - character constants  
|                | - string constants  
|                | or a combination of these, as in:  
|                | `fieldname2||fieldname2||fieldname2...`  
| The field names within a layout must be unique. Nested concatenations are not supported.  
| Each `fieldname2` that is actually a field name must be specified in its own FIELD or FILLER command with a `startpos` parameter and a `datadesc` parameter that specifies the character data type.  
| The `fieldname1` parameter in other FIELD commands can be referenced in `fieldexpr` expressions.  
| When the character set of the job script is different from the client character set used for the job, Teradata MultiLoad translates the character and the string constants specified in the expression from the script character encoding to the client character encoding before concatenating the constants with the specified fields.  
| For example, the job script must be in Teradata EBCDIC when using the UTF8 client character set on MVS; the job script can be in UTF8 when using the UTF16 client character set on network-attached systems.  
| **Caution:** Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See *International Character Set Support* for details. |

### `fieldname1`

Name of an input record field

The `fieldname1` specification can be referenced by:

- Subsequent INSERT, UPDATE, or DELETE statements
- The NULLIF `nullexpr` expression of another FIELD command
- The conditional expression of a LAYOUT command or the APPLY clause of an IMPORT command

A `fieldname1` specification must obey the same construction rules as Teradata SQL column names.
Chapter 3: MultiLoad Commands

FIELD

nullexpr

Condition used for selectively inserting a null value into the affected column. The condition is specified as a conditional expression involving any number of fields, each represented by its fieldname, and constants.

Each fieldname appearing in the conditional expression must be defined by either:

- The startpos and datadesc parameters of the FIELD command
- A FILLER command
- A TABLE command

The fieldname specifications in a nullexpr condition cannot be defined by a fieldexpr of a FIELD command.

Note: The fieldname parameter in other FIELD commands can be referenced in nullexpr conditions. You cannot use a FastLoad-type value specification (NULLIF=value) as the nullexpr specification. The nullexpr specification must be a complete conditional expression that includes a logical operator.

When the character set of the job script is different from the client character set used for the job, Teradata MultiLoad translates the string constants and the import data referenced in the expression to the same character set before evaluating the expression.

For example, the job script must be in Teradata EBCDIC when using the UTF8 client character set on MVS; the job script can be in UTF8 when using the UTF16 client character set on network-attached systems.

Network Example: If the client character set is UTF16 and the script character set is UTF8, and the following commands are given, MLOAD translates the data in the C1 field to the UTF8 form and compares it with the UTF8 form of 'DELETED' to obtain the evaluation result.

```plaintext
.field C1 * varchar(20);
.field C2 * varchar(40) nullif c1 = 'DELETED';
```

Mainframe Example: If the client character set is UTF8 and the script character set is Teradata EBCDIC, and the following commands are given, Teradata MultiLoad translates the data in the C1 field from the UTF8 form to the Teradata EBCDIC form and compares it to the Teradata EBCDIC form of 'removed' to obtain the valuation result.

```plaintext
.field C1 * char(20);
.field C2 * char(40) nullif c1 = 'removed';
```

Caution: Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See International Character Set Support for details.
Chapter 3: MultiLoad Commands

FIELD

Usage Notes

The following table describes the things to consider when using the FIELD command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>startpos Specification</td>
<td>The specified starting position:</td>
</tr>
<tr>
<td></td>
<td>• Reflects removal of any columns per the CONTINUEIF clause of a LAYOUT command</td>
</tr>
<tr>
<td></td>
<td>• Ignores any indicator bytes per the INDICATORS specification of a LAYOUT command</td>
</tr>
<tr>
<td></td>
<td>• Is relative to character position 1</td>
</tr>
<tr>
<td></td>
<td>Therefore, the first data position of the input record is position 1. For more information about the CONTINUEIF and INDICATORS specifications, see the “LAYOUT” command description.</td>
</tr>
<tr>
<td>Changing the Data Type is Not Allowed</td>
<td>You cannot use the datadesc parameter to change the data type from character to decimal when you redefine an input record field in fieldname1. Doing so causes the MultiLoad job to abort with an error message.</td>
</tr>
<tr>
<td>Intermixing Commands</td>
<td>You can intermix one or more FIELD commands with TABLE and FILLER commands. All of these commands must follow a LAYOUT command.</td>
</tr>
</tbody>
</table>
NULLIF Performance

You could use the following:

```
... 
.FIELD fc * CHAR(5);
.FIELD fi * INTEGER;
...

.DML LABEL ins;
INSERT INTO tbl1 VALUES (...,
NULLIF(:fc,'empty'),
NULLIF(:fi,0),...);
```

And, in the more complex situation:

```
...
.FIELD fs * CHAR(1) ;
.FIELD fc * CHAR(5) NULLIF (fs <> 'M') AND (fs <> 'F');
.FIELD fi * INTEGER NULLIF fi < 0;
...

.DML LABEL ins; INSERT INTO tbl2 VALUES(...,:fs,:fc,:fi,...);
```

you could instead use:

```
...
.FIELD fs * CHAR(1) ;
.FIELD fc * CHAR(5);
.FIELD fi * INTEGER;
...

.DML LABEL ins;
INSERT INTO tbl2 VALUES(...,:fs,
CASE WHEN (:fs = 'M')
    OR (:fs = 'F')
    THEN :fc ELSE NULL END,
CASE WHEN (:fi >= 0)
    THEN :fi ELSE NULL END,...);
```

Parameter Evaluation Sequence

If you specify both NULLIF and DROP LEADING/TRAILING BLANKS/NULLS in the same FIELD command, MultiLoad evaluates the DROP clause after the NULLIF clause.

For example, if the input for field1 is 'x' in the following FIELD command, the NULLIF expression would evaluate to false because the leading blanks are not dropped before the NULLIF evaluation:

```
.FIELD FIELD1 * CHAR (5) NULLIF
FIELD1 = 'x'
DROP LEADING BLANKS;
```
### Specifying Graphic Data Types

Following is the input length and field description for the graphic data type specifications you can make in the `datadesc` parameter:

**GRAPHIC\( (n) \)**
- Length: \( n \times 2 \) bytes, if \( n \) is specified; otherwise 2 bytes, as \( n=1 \) is assumed.
- Description: \( n \) double-byte characters (\( n \) is the length of the input stream in terms of double-byte characters).

**VARGRAPHIC\( (n) \)**
- Length: \( m + 2 \) bytes where \( m/2 \leq 1600 \).
- Description: 2-byte integer followed by \( m/2 \) double-byte characters.

**LONG VARGRAPHIC**
- Length: \( m + 2 \) bytes where \( m/2 \leq 16000 \).
- Description: 2-byte integer followed by \( m/2 \) double-byte characters.

**Note:** LONG VARGRAPHIC also implies VARGRAPHIC \((16000)\). Range is 0 to 16000 in a 32,000-byte field. For both VARGRAPHIC and LONG VARGRAPHIC, \( m \), a value occupying the first 2 bytes of the input data, is the length of the input in bytes, not characters. Each multibyte character set character is 2 bytes.

### Specifying Decimal Data Types

Following is the input length and field description for the decimal data type specifications you can make in the `datadesc` parameter:

**DECIMAL\( (x) \)** and **DECIMAL\( (x, y) \)**
- Length: 1, 2, 4, or 8 bytes for network; packed decimal for mainframe
- Description: 64-bit double precision, floating point

For more information on the DECIMAL data type, see the *SQL Reference: Data Types and Literals* book.

### Specifying Relevant Fields

Because all fields specified by FIELD commands are sent to the Teradata Database, only specify those fields that are relevant to the MultiLoad task.

### Using ANSI/SQL DateTime Data Types

When you use the `DATEFORM` command to specify ANSIDATE as the DATE data type, MultiLoad internally converts each DATE field to a `CHAR(10)` field. You must convert all ANSI/SQL DateTime `TIME`, `TIMESTAMP`, and `INTERVAL` data types to fixed-length `CHAR` data types to specify column/field names in a MultiLoad FIELD command.

*Table 27* provides the conversion specifications and format examples for each ANSI/SQL DateTime specification.
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable Definition</th>
<th>Conversion Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td></td>
<td>CHAR(8 + n + (1 if n &gt; 0, otherwise 0))</td>
</tr>
<tr>
<td>TIME ((n))</td>
<td>(n = \text{number of digits after decimal point})</td>
<td>Format ((n = 0)): (hh:mm:ss)</td>
</tr>
<tr>
<td></td>
<td>(\text{Valid values: 0–6})</td>
<td>Example: 11:37:58</td>
</tr>
<tr>
<td></td>
<td>(\text{Default} = 6)</td>
<td>Format ((n = 4)): (hh:mm:ss.ssss)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 11:37:58.1234</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td>CHAR(19 + n + (1 if n &gt; 0, otherwise 0))</td>
</tr>
<tr>
<td>TIMESTAMP ((n))</td>
<td>(n = \text{number of digits after decimal point})</td>
<td>Format ((n = 0)): (yyyy-mm-dd\ hh:mm:ss)</td>
</tr>
<tr>
<td></td>
<td>(\text{Valid values: 0–6})</td>
<td>Example: 1998-09-04 11:37:58</td>
</tr>
<tr>
<td></td>
<td>(\text{Default} = 6)</td>
<td>Format ((n = 4)): (yyyy-mm-dd\ hh:mm:ss.ssss)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 1998-09-04 11:37:58.1234</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td></td>
<td>CHAR(14 + n + (1 if n &gt; 0, otherwise 0))</td>
</tr>
<tr>
<td>TIME ((n)) WITH TIME ZONE</td>
<td>(n = \text{number of digits after decimal point})</td>
<td>Format ((n = 0)): (hh:mm:ss(±)hh:mm)</td>
</tr>
<tr>
<td></td>
<td>(\text{Valid values: 0–6})</td>
<td>Example: 11:37:58-08:00</td>
</tr>
<tr>
<td></td>
<td>(\text{Default} = 6)</td>
<td>Format ((n = 4)): (hh:mm:ss.ssss(±)hh:mm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 11:37:58.1234-08:00</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td></td>
<td>CHAR(25 + n + (1 if n &gt; 0, otherwise 0))</td>
</tr>
<tr>
<td>TIMESTAMP ((n)) WITH TIME ZONE</td>
<td>(n = \text{number of digits after decimal point})</td>
<td>Format ((n = 0)): (yyyy-mm-dd\ hh:mm:ss(±)hh:mm)</td>
</tr>
<tr>
<td></td>
<td>(\text{Valid values: 0-6})</td>
<td>Example: 1998-09-24 11:37:58+07:00</td>
</tr>
<tr>
<td></td>
<td>(\text{Default} = 6)</td>
<td>Format ((n = 4)): (yyyy-mm-dd\ hh:mm:ss.ssss(±)hh:mm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 1998-09-24 11:37:58.1234+07:00</td>
</tr>
<tr>
<td>INTERVAL YEAR</td>
<td>(n = \text{number of digits})</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>INTERVAL YEAR ((n))</td>
<td>(\text{Valid values: 1-4})</td>
<td>Format ((n = 2)): (yy)</td>
</tr>
<tr>
<td></td>
<td>(\text{Default} = 2)</td>
<td>Example: 98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format ((n = 4)): (yyyy)</td>
</tr>
</tbody>
</table>
|                        |                     | Example: 1998
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable Definition</th>
<th>Conversion Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>n = number of digits</td>
<td>CHAR(n + 3)</td>
</tr>
<tr>
<td>INTERVAL YEAR (n) TO MONTH</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): yyyy-mm</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 98-12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): yyyy-mm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 1998-12</td>
</tr>
<tr>
<td>INTERVAL MONTH</td>
<td>n = number of digits</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>INTERVAL MONTH (n)</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): mm</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): mmmm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 0012</td>
</tr>
<tr>
<td>INTERVAL DAY</td>
<td>n = number of digits</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>INTERVAL DAY (n)</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): dd</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): dddd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 0031</td>
</tr>
<tr>
<td>INTERVAL DAY TO HOUR</td>
<td>n = number of digits</td>
<td>CHAR(n + 3)</td>
</tr>
<tr>
<td>INTERVAL DAY (n) TO HOUR</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): hh</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): hhhh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 0012</td>
</tr>
<tr>
<td>INTERVAL DAY TO MINUTE</td>
<td>n = number of digits</td>
<td>CHAR(n + 6)</td>
</tr>
<tr>
<td>INTERVAL DAY (n) TO MINUTE</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): hh:mm</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 31 12:59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): dddd hh:mm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 0031 12:59</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>n = number of digits</td>
<td>CHAR(n + 9 + m + (1 if m &gt; 0, otherwise 0))</td>
</tr>
<tr>
<td>INTERVAL DAY (n) TO SECOND</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2, m = 0): hh:mm:ss</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND (m)</td>
<td>Default = 2</td>
<td>Example: 12:59:59</td>
</tr>
<tr>
<td>INTERVAL DAY (n) TO SECOND</td>
<td>m = number of digits</td>
<td>Format (n = 4, m = 4): hhhh:mm:ss.sss</td>
</tr>
<tr>
<td></td>
<td>after decimal point</td>
<td>Example: 0012:59:59.1234</td>
</tr>
<tr>
<td>INTERVAL HOUR</td>
<td>n = number of digits</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>INTERVAL HOUR (n)</td>
<td>Valid values: 1-4</td>
<td>Format (n = 2): hh</td>
</tr>
<tr>
<td></td>
<td>Default = 2</td>
<td>Example: 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Format (n = 4): hhhh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 0012</td>
</tr>
</tbody>
</table>
Example 1

Concatenating Special Characters

Use the following command structure to concatenate a character string such as 'A'||'%'||'@':

```plaintext
.LAYOUT LAY1A;
.FIELD FIELD1 * INTEGER;
.FIELD FIELD2 * INTEGER;
```
Example 2

Multiple References to the Same Field

Consider a set of input records that contains a 25-character address field in positions 15 through 39, with the street number in the first seven positions and the street name in the last 18 positions.

The following example shows how to refer both to the address field as a whole and separately to the street name:

```plaintext
.FIELD ADDRESS 15 CHAR(25);
.FIELD STREET 22 CHAR(18);
```

Example 3

Invalid Concatenation

You cannot concatenate a field that has been defined by a concatenation of fields.

The following example shows the error message generated for this condition:

```plaintext
0009 .LAYOUT LAY3S;
0010 .FIELD FIELD1 * INTEGER;
0011 .FIELD FIELD2 * INTEGER;
0012 .FIELD FIELD3 * CHAR (1);
0013 .FIELD FIELD4 * CHAR (2);
0014 .FIELD FIELD5 FIELD3||FIELD4;
0015 .FIELD FIELD6 FIELD3||FIELD4||FIELD5;
0016 .FIELD FIELD7 FIELD3||FIELD4||FIELD5||FIELD6;
0017 .DML LABEL LABELA;
0018 INSERT TBL3S (FIELD1,FIELD2,FIELD3,FIELD4,FIELD5,FIELD6,FIELD7) VALUES (:FIELD1,:FIELD2,:FIELD3,:FIELD4,:FIELD5,:FIELD6,:FIELD7);
0019 .IMPORT INFILE INPUT FREE LAYOUT LAY3S APPLY LABELA;
0020 .END MLOAD;
```

Example 4

Using the GRAPHIC Data Types

The following example shows how to use the GRAPHIC data types in support of kanji or multibyte character data. The FIELD description of the data set or file can contain GRAPHIC data types.
.LAYOUT KANJIDATA;
.FIELD EMPNO    * SMALLINT;
.FIELD LASTNAME* GRAPHIC(30);
.FILLER FIRSTNAME* GRAPHIC(30);
.FIELD JOBTITLE* VARGRAPHIC(30);
The FILLER command describes a named or unnamed field as filler, which is not sent to the Teradata Database.

Syntax

```
.FILLER [fieldname] startpos datadesc ;
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>datadesc</code></td>
<td>type and length of data in the field</td>
</tr>
<tr>
<td><code>fieldname</code></td>
<td>optional name of an input record field</td>
</tr>
<tr>
<td><code>startpos</code></td>
<td>starting position of the specified field in the data records of an external data source</td>
</tr>
</tbody>
</table>

The `fieldname` specification can be referenced by:
- The NULLIF `nullexpr` expression of a FIELD command
- The condition expression of the APPLY clause of an IMPORT command

A `fieldname` specification must obey the same construction rules as Teradata SQL column names.

You can specify the `startpos` as:
- An unsigned integer, which is a character position starting with 1
- An asterisk, which means the next available character position beyond the preceding field
Usage Notes

The following table describes the things to consider when using the FILLER command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermixing Commands</td>
<td>You can intermix one or more FILLER commands with TABLE and FIELD commands.</td>
</tr>
<tr>
<td></td>
<td>All of these commands must follow a LAYOUT command.</td>
</tr>
<tr>
<td>startpos Specification</td>
<td>The specified starting position:</td>
</tr>
<tr>
<td></td>
<td>• Reflects removal of any columns per the CONTINUEIF clause of a LAYOUT command</td>
</tr>
<tr>
<td></td>
<td>• Ignores any indicator bytes per the INDICATORS specification of a LAYOUT command</td>
</tr>
<tr>
<td></td>
<td>• Is relative to character position 1</td>
</tr>
<tr>
<td></td>
<td>Therefore, the first data position of the input record is position 1.</td>
</tr>
<tr>
<td></td>
<td>For more information about the CONTINUEIF and INDICATORS specifications, see the LAYOUT command description.</td>
</tr>
<tr>
<td>Specifying Graphic Data Types</td>
<td>Following is the input length and field description for the graphic data type specifications you can make in the datadesc parameter:</td>
</tr>
<tr>
<td></td>
<td><strong>GRAPHIC</strong>(n)</td>
</tr>
<tr>
<td></td>
<td>• Length: (n \times 2) bytes, if (n) is specified; otherwise 2 bytes, as (n=1) is assumed.</td>
</tr>
<tr>
<td></td>
<td>• Description: (n) double-byte characters ((1n) is the length of the input stream in terms of double-byte characters).</td>
</tr>
<tr>
<td></td>
<td><strong>LONG VARGRAPHIC</strong></td>
</tr>
<tr>
<td></td>
<td>• Length: (m + 2) bytes where (m/2 \leq 16000).</td>
</tr>
<tr>
<td></td>
<td>• Description: 2-byte integer followed by (m/2) double-byte characters.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: LONG VARGRAPHIC also implies VARGRAPHIC (16000). Range is 0 to 16000 in a 32,000-byte field.</td>
</tr>
</tbody>
</table>

Example 1

**Concatenating a Field with a Filler**

Use the following command structure to concatenate a field with a filler:

```
.LAYOUT LAY1A;
.FIELD F1 * CHAR(2);
.FILLER F2 * CHAR(1);
.FIELD F3 F1||F2
```
Example 2

Using the GRAPHIC Data Types

The following example shows how to use the GRAPHIC data types in support of kanji or multibyte character data. The FILLER statement describing the input data set or file can contain GRAPHIC data types.

```
.LAYOUT KANJIDATA;
.FIELD EMPNO * SMALLINT;
.FIELD LASTNAME * GRAPHIC(30);
.FILLER FIRSTNAME * GRAPHIC(30);
.FIELD JOBTITLE * VARGRAPHIC(30);
```
IF, ELSE, and ENDF

Purpose

The IF, ELSE, and ENDIF commands provide conditional control of execution processes.

Syntax

```
IF conditional expression THEN
  statements to execute if true
ENDIF;
ELSE
  statements to execute if false
ENDIF;
```

Usage Notes

The following table describes the things to consider when using the IF, ELSE, and ENDIF commands.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables in the IF Expression</td>
<td>The <em>conditional expression</em> can be either user-defined variables or predefined system variables.</td>
</tr>
</tbody>
</table>
| Numeric Results of the *Conditional Expression* | The *conditional expression* is an expression that can be evaluated as either true or false. When evaluation of the expression returns a numeric result:  
  - Zero is interpreted as false  
  - Nonzero results are interpreted as true |
| ELSE Clause                          | Use the optional ELSE clause to execute commands when the condition is evaluated as false.                                               |
Example 1

MultiLoad is case sensitive when comparing &SYS system variables. In this example, the RUN FILE command does not execute because the substituted values returned are all capitals:

```
0003 .IF '&SYSDAY' = 'Fri' THEN;
14:10:28 - FRI MAY 09, 1993
UTY2402 Previous statement modified to:
0004 .IF 'FRI' = 'Fri' THEN;
0005 .RUN FILE UTNTS38;
0006 .ENDIF;
```

Always consider this factor when creating a script to force the execution of a predetermined sequence of events. If you substituted 'FRI' in line 0003, the compare would work and the RUN FILE command would execute.

Example 2

In the following example, the user has created the table named &TABLE and a variable named CREATERC, into which is set the system return code resulting from the execution of the CREATE TABLE statement:

```
.SET CREATERC TO &SYSRC;
.IF &CREATERC = 3803 /* Table &TABLE exists */ THEN;
.RUN FILE RUN01;
.ELSE;
.IF &CREATERC <> 0 THEN;
.LOGOFF &CREATRC;
.ENDIF;
.ENDIF;
```

If the table name has not already been used, and the return code is not zero, the return code evaluates to an error condition and the job logs off with the error code displayed.
**IMPORT**

**Purpose**

The IMPORT command specifies a source for data input.

**Syntax**

The IMPORT command syntax depends on whether MultiLoad is running on a channel-attached or network-attached client system. Several of the syntax elements are common to both configurations, while others are specific to each.

**For Channel-Attached Client Systems**

```
.IMPORT INFILE ddname AXSMOD name 'init-string' 

HOLD FREE INMOD modulename USING (parms) 

FROM m FOR n THRU k 

FORMAT VARTEXT 'c' DISPLAY ERRORS NOSTOP 

LAYOUT layoutname 

APPLY label WHERE condition 
```
Chapter 3: MultiLoad Commands

For Network-Attached Client Systems

```
.IMPORT

INFILE 'filename'
  AXSMOD 'name'
  'init-string'

FROM m
  FOR n
  THRU k

INMOD 'module-name'
  USING (parms)

FORMAT
  FASTLOAD
    BINARY
    TEXT
    UNFORMATTED
    VARTEXT
      'c'
      DISPLAY ERRORS
      NOSTOP

LAYOUT 'layout-name'
  APPLY 'label'
    WHERE 'condition'

```

24098022
where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY label</td>
<td>Error treatment options specified by a previous DML LABEL command for subsequent INSERT, UPDATE, or DELETE statements</td>
</tr>
<tr>
<td>AXSMOD name</td>
<td>Name of the access module file to be used to import data</td>
</tr>
</tbody>
</table>

The names of the access module files are:

- **OLE DB Access Module**: `oledb_axsmod.dll` on Windows platforms
- **Named Pipes Access Module**:
  - `np_axsmod.sl` on HP-UX platforms
  - `np_axsmod.so` on MP-RAS, IBM AIX, Sun Solaris SPARC and Sun Solaris Opteron platforms
  - `np_axsmod.dll` on Windows platforms
- **WebSphere® MQ Access Module for Teradata (client version)**:
  - `libmqsc.sl` on HP-UX platforms
  - `libmqsc.so` on MP-RAS, IBM AIX, and Sun Solaris SPARC platforms
  - `libmqsc.dll` on Windows platforms
- **WebSphere® MQ Access Module for Teradata (server version)**:
  - `libmqs.sl` on HP-UX platforms
  - `libmqs` on IBM MVS/ESA platforms
  - `libmqs.so` on IBM AIX and Sun Solaris SPARC platforms
  - `libmqs.dll` on Windows platforms

You may use your own shared library file name if you have a custom access module.

**Note:** Large File Access Module is no longer available because the Data Connector API supports file sizes greater than 2 gigabytes on Windows, HP-UX, IBM AIX, and Solaris SPARC platforms.

The AXSMOD option is not required for importing disk files on either network-attached or channel-attached client systems, or magnetic tape files on channel-attached client systems. It is required for importing magnetic tape and other types of files on network-attached client systems.

For more information about specific access modules, see the Teradata Tools and Utilities Access Module Reference.
Chapter 3: MultiLoad Commands

IMPORT

138 Teradata MultiLoad Reference

The import keyword is the first keyword in the Data Source Definition section of the MultiLoad command.

### Syntax Element: 'c'

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| 'c'            | Optional specification of the delimiter character that separates fields in the variable-length text records of the input data source. If you do not use a 'c' specification, the default is the pipe character (|). Except for the UTF16 client character set, the delimiter character must be a single byte in the client character set. When the character set of the job script is different from the client character set used for the job, Teradata MultiLoad translates the effective delimiter from the script character encoding to the client character encoding before separating fields with it. For example, the job script must be in Teradata EBCDIC when using the UTF8 client character set on MVS; the job script can be in UTF8 when using the UTF16 client character set on network-attached systems. **Network Example:** If the client character set is UTF16, the script character set is UTF8, and the following command is given:

```
...FORMAT VARTEXT '-'...
```

Teradata MultiLoad translates '-' from UTF8 to UTF16, and then separates the fields in the record according to the UTF16 form of '-'.

**Mainframe Example:** If the client character set is UTF8, the script character set is Teradata EBCDIC, and the following command is given:

```
...FORMAT VARTEXT '6A'xc...
```

Teradata MultiLoad interprets x'6A' according to Teradata EBCDIC, translates it to the corresponding Unicode code point (U+007C "VERTICAL LINE"), and uses the UTF8 encoding scheme of U+007C, 0x7C (which is '|' in 7-bit ASCII) as the delimiter character for the record.

**Caution:** Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See [International Character Set Support](InternationalCharacterSetSupport) for details.

For example, the code point of '|' is x'4F' in most IBM EBCDIC code pages. If you specify '|' as the delimiter in the script or leave the delimiter to default in a system environment using that type of IBM EBCDIC code page and your UTF8 data uses x'7C' (which is '|' in Unicode) as the delimiter, the job will run into errors because:

- The code point of x'4F' in Teradata EBCDIC maps to U+008D, not U+007C.
- The delimiter must be a single-byte character when it is in the client character encoding.

### Syntax Element: DISPLAY ERRORS

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY ERRORS</td>
<td>Optional keyword specification that writes input data records that produce errors to the standard error file.</td>
</tr>
</tbody>
</table>

### Syntax Element: FOR n

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR n</td>
<td>Number of records, as an integer, starting at record m, to be processed. If you do not use a FOR n or a THRU k specification, MultiLoad continues processing through the last record obtained from the data source.</td>
</tr>
<tr>
<td>Syntax Element</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FORMAT…</td>
<td>Format of the input record, where:</td>
</tr>
<tr>
<td></td>
<td>• FASTLOAD specifies that each input record is a 2-byte integer, $n$, followed by $n$ bytes of data, followed by an end-of-record marker (either X'0A' or X'0D').</td>
</tr>
<tr>
<td></td>
<td>• BINARY specifies that each input record is a 2-byte integer, $n$, followed by $n$ bytes of data.</td>
</tr>
<tr>
<td></td>
<td>• TEXT specifies that each input record is an arbitrary number of bytes, followed by an end-of-record marker, which is a:</td>
</tr>
<tr>
<td></td>
<td>• Linefeed (X'0A') on UNIX platforms</td>
</tr>
<tr>
<td></td>
<td>• Carriage-return/linefeed pair (X'0D0A') on Windows platforms</td>
</tr>
<tr>
<td></td>
<td>• UNFORMAT specifies that each input record is defined by FIELD, FILLER, and TABLE commands of the specified layout.</td>
</tr>
<tr>
<td></td>
<td>Note: When using UNFORMAT formatting in MVS, ensure that the data stream and data source are consistent with the layout defined in the utility script. Discrepancies in the length of the data stream could result in data corruption.</td>
</tr>
<tr>
<td></td>
<td>• VARTEXT specifies that each input record is in variable-length text record format, with each field separated by delimiter characters, which:</td>
</tr>
<tr>
<td></td>
<td>• can total 10 characters in length</td>
</tr>
<tr>
<td></td>
<td>• cannot be characters that appear in the data</td>
</tr>
<tr>
<td></td>
<td>• cannot be control characters, other than a TAB</td>
</tr>
<tr>
<td>FREE</td>
<td>Deallocation of the tape input device specified by $ddname$ when the import operation completes on channel-attached client systems.</td>
</tr>
<tr>
<td></td>
<td>When de-allocated, any attempt to open the input device, either in the same MultiLoad task or in another task within the same script, produces an undefined $ddname$ error.</td>
</tr>
<tr>
<td></td>
<td>The default is to not deallocate the device.</td>
</tr>
<tr>
<td>FROM $m$</td>
<td>Logical record number, as an integer, of the record in the identified data source where processing is to begin.</td>
</tr>
<tr>
<td></td>
<td>If you do not use a FROM $m$ specification, MultiLoad begins processing with the first record received from the data source.</td>
</tr>
<tr>
<td>HOLD</td>
<td>Default condition to not deallocate the input tape device specified by $ddname$ when the import operation completes on channel-attached client systems. Instead, the HOLD specification deallocates the device when the entire MultiLoad operation completes.</td>
</tr>
<tr>
<td>INFILE $ddname$</td>
<td>External data source that contains the input records on channel-attached client systems</td>
</tr>
<tr>
<td></td>
<td>In MVS, this is a DDNAME. In VM, it is a FILEDEF name.</td>
</tr>
<tr>
<td></td>
<td>If $ddname$ is specified, MultiLoad reads data records from the specified source. If $modulename$ is also specified, MultiLoad passes the records it reads to the specified module.</td>
</tr>
<tr>
<td></td>
<td>The DDNAME must obey the applicable rules of the external system.</td>
</tr>
<tr>
<td></td>
<td>A DDNAME must obey the same construction rules as Teradata SQL column names except that:</td>
</tr>
<tr>
<td></td>
<td>• The “at” character (@) is allowed as an alphabetic character</td>
</tr>
<tr>
<td></td>
<td>• The underscore character (_) is not allowed</td>
</tr>
<tr>
<td></td>
<td>If the DDNAME represents a data source on magnetic tape, the tape may be either labeled or non-labeled, as supported by the operating system.</td>
</tr>
</tbody>
</table>
# Chapter 3: MultiLoad Commands

## IMPORT

### Syntax Element Description

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INFILE filename</strong></td>
<td>Fully qualified UNIX or Windows path name for an input file on network-attached client systems. If the path name has embedded white space characters, you must enclose the entire path name in single or double quotes. If <strong>INFILE filename</strong> is specified, MultiLoad reads data from the specified source. If <strong>INMOD modulename</strong> is also specified, MultiLoad passes the data it reads to the specified module.</td>
</tr>
<tr>
<td><strong>init-string</strong></td>
<td>Optional initialization string for the access module.</td>
</tr>
</tbody>
</table>
| **INMOD modulename** | - User exit routine that optionally reads, and always preprocesses, each record before passing the record to MultiLoad for processing on channel-attached client systems.  
- Fully qualified UNIX or Windows pathname of the INMOD executable code on network-attached client systems.  
  The **modulename** specification must obey the same construction rules as Teradata SQL column names except that on channel-attached client systems:  
  - The "at" character (@) is allowed as an alphabetic character.  
  - The underscore character (_) is not allowed.  
  The **modulename** specification must obey the applicable rules of the external system.  
  **Note:** On some versions of UNIX, you may have to add ./ prefix characters to the **modulename** specification if the module is in the current directory. |
| **LAYOUT layoutname** | Layout of the input record, as specified by a previous **LAYOUT** command. |
| **NOSTOP** | Optional keyword specification that inhibits the MultiLoad termination in response to an error condition associated with a variable-length text record. |
| **THRU k** | Logical record number, as an integer, of the record in the identified data source where processing is to end. If you do not use a **THRU k** or a **FOR n** specification, MultiLoad continues processing through the last record obtained from the data source. |
| **USING (parms)** | Character string containing whatever parameters are to be passed to the corresponding user exit routine:  
- The **parms** string can include one or more character strings, each delimited on either end by an apostrophe or quotation mark.  
- The maximum size of the **parms** string is 1 KB.  
- Parentheses within delimited character strings or comments have the same syntactical significance as alphabetic characters.  
- Before passing the **parms** string to the user exit routine, MultiLoad replaces the following with a single blank character:  
  - Each comment  
  - Each consecutive sequence of white-space characters, such as blank, tab and so on, that appears outside of delimited strings  
  - The **parms** string, as a whole, must be enclosed in parentheses. On channel-attached client systems, the parentheses are included in the string passed to the user exit routine.  
  - When the user exit routine is an old FastLoad INMOD, the **parms** string must be FDLINMOD. |
### Syntax Element

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| WHERE condition | Condition that determines whether the indicated label options are applied to the records and sent to the Teradata Database per subsequent INSERT, UPDATE, or DELETE statements where:  
  - condition true = yes  
  - condition false = no  
  Your condition specification can reference:  
    - Any combination of fields defined in the currently active layout  
    - System and user-defined constants and variables  
    - The `fieldname1` specified in FIELD commands  
  If you do not use a WHERE condition specification, the default condition is true/yes. |

When the character set of the job script is different from the client character set used for the job, Teradata MultiLoad translates the string constants specified and the import data referenced in the condition to the same character set before evaluating the condition.

For example, the job script must be in Teradata EBCDIC when using the UTF8 client character set on MVS; the job script can be in UTF8 when using the UTF16 client character set on network-attached systems.

**Network Example:** If the client character set is UTF16, the script character set is UTF8, and the following command is given:

```
...APPLY label1 WHERE C1 = 'INSERT';
```

Teradata MultiLoad translates the data in the C1 field to the UTF8 form and compares it with the UTF8 form of 'INSERT' to obtain the evaluation result.

**Mainframe Example:** If the client character set is UTF8, the script character set is Teradata EBCDIC, and the following command is given:

```
...APPLY label2 WHERE C2 = 'DELETE';
```

Teradata MultiLoad translates the data in the C2 field from the UTF8 form to the Teradata EBCDIC form and compares it with the Teradata EBCDIC form of 'DELETE' to obtain the evaluation result.

**Caution:** Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See *International Character Set Support* for details.

---

**Usage Notes**

The following table describes the things to consider when using the IMPORT command.
### Chapter 3: MultiLoad Commands

#### IMPORT

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
</table>
| **Import Task Command Restrictions** | The combined number of Teradata SQL statements under the DML commands cannot exceed 100 within a single MultiLoad import task.  
Sending an excessive number of statements to the Teradata Database produces an error message indicating that there are too many DML steps for one MultiLoad import task.  
For an import task, a candidate statement or group of statements is applied if no condition is specified, or if the specified condition is true.  
Note, however, that for an import task, the only DML statements that are candidates for application by an IMPORT command are those within the scope of DML commands whose labels appear in one or more of the IMPORT command APPLY clauses. (The referenced DML commands and their following DML statements must appear between the BEGIN MLOAD command that defines the import task and the referencing IMPORT commands.) |
| **VARTEXT Records** | When you specify VARTEXT, MultiLoad assumes that the input data is variable-length text fields separated by up to 10 field delimiter characters. The utility parses each input data record on a field-by-field basis, and creates a VARCHAR field for each input text field. |
| **Data Type Specifications** | When using the VARTEXT specification, VARCHAR, VARBYTE and LONG VARCHAR are the only valid data type specifications you can use in the MultiLoad layout FIELD and FILLER commands. |
| **Null Fields** | Two consecutive delimiter characters direct MultiLoad to null the field corresponding to the one right after the first delimiter character.  
Also, if the last character in a record is a delimiter character, and yet there was at least one more field to be processed, then MultiLoad nulls the field corresponding to the next one to be processed, as defined in the layout FIELD and FILLER commands. |
| **Input Record Requirements** | The total number of fields in each input record must be equal to or greater than the number of fields described in the MultiLoad layout FIELD and FILLER commands.  
If it is less, MultiLoad generates an error message. If it is more, the Teradata Database ignores the extra fields.  
Note that a delimiter character in the last field of a record is optional. |
| **Error Record Handling** | When MultiLoad encounters an error condition in an input record, it normally discards the record and terminates. When loading variable-length text records, you can inhibit either or both of these functions by specifying the error-handling options:  
• DISPLAY ERRORS  
• NOSTOP  
By specifying both options and redirecting STDERR to a file location instead of your terminal screen, your MultiLoad job runs to completion and saves all the error records. Then you can manually modify them and load them into the table. |
| **FREE/HOLD Option When Running Under MVS** | The disposition of the output device specified in the JCL must be KEEP, not PASS, for the FREE/HOLD option to work in MultiLoad tasks running under MVS. |
### Multiple APPLY Clauses

In an import task, you can apply multiple APPLY clauses to the same data record in either of two ways. This feature allows you to apply the same data record to different tables under the same or different conditions.

First, if an APPLY clause refers to a label whose scope includes multiple DML statements, each of these statements is applied to the same data record under the same condition specified in the clause.

Second, if you use multiple APPLY clauses, each can refer to the label specification of a different DML statement or group of statements. Each label specification is applied to the same data record under the condition specified in the respective clause.

### Record Length Validation

By default, MultiLoad does not compare the actual record length of the import data with the record length indicated by the layout specifications for the job.

If they are not the same, the default behavior of MultiLoad depends on whether the actual import data record length is less than or greater than the record length indicated by the layout specifications:

- If the actual import data record length is less than the length indicated by the layout specifications, then MultiLoad terminates with an error indication.
- If the actual import data record length is greater than the length indicated by the layout specifications, then MultiLoad ignores the extra fields and continues with the import task.

To change the default behavior and enforce a record-length validation check, use a MATCHLEN=on entry in the MultiLoad configuration file before you invoke MultiLoad. In this case, MultiLoad terminates with an error message whenever the actual and specified record lengths are different.

For information about using the MultiLoad configuration file, see “MultiLoad Configuration File” on page 46.

### Primary Indexes and Partitioning Column Sets

IMPORT tasks require you to supply all values of the primary index column set and all values of the partitioning column set for deletes and updates. IMPORT tasks do not support updates of the partitioning column set. IMPORT tasks do not support primary index updates.
**Purpose**

INSERT is a Teradata SQL statement that adds new rows to a table or view.

**Syntax**

```
INSERT INTO tname (cname, :fieldname) VALUES (.*, :fieldname).
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Default VALUES clause for the insert operation</td>
</tr>
<tr>
<td>When you use the * characters as a suffix with the tname specification, MultiLoad replaces these characters with a default VALUES clause before executing the command.</td>
<td></td>
</tr>
<tr>
<td>Note: When using the * default, tname must specify a table name and not a view name. The default option is most useful when used with the TABLE command, which builds the layout corresponding to the same table.</td>
<td></td>
</tr>
<tr>
<td><strong>cname</strong></td>
<td>Column of the specified table that is to receive the value from a field of matching input records</td>
</tr>
<tr>
<td>If you do not enter the cname specifications, MultiLoad uses the column identifiers as they were defined by the CREATE TABLE statement.</td>
<td></td>
</tr>
<tr>
<td>The value is identified by the corresponding entry in the fieldname list.</td>
<td></td>
</tr>
<tr>
<td><strong>fieldname</strong></td>
<td>Field of an input record, whose value is given to a column of the tname table that is identified by the corresponding cname specification of this command</td>
</tr>
<tr>
<td>Alternatively, each fieldname clause may instead be an expression that includes one or more actual term.</td>
<td></td>
</tr>
<tr>
<td><strong>tname</strong></td>
<td>Table or view that is to receive rows from MultiLoad input data records</td>
</tr>
<tr>
<td>The tname specification must have been previously identified as tname1 in the BEGIN MLOAD command.</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Element

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| VALUES         | The VALUES clause is of the form:  
VALUES (:column1, :column2,...)  
where :column1, :column2,... are the names of the columns from tname in the order in which they were defined by the CREATE TABLE statement. |

### Usage Notes

The following table describes the things to consider when using the INSERT statement.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
</table>
| Object Restriction                         | You can specify only one tname object for an INSERT statement, and it must have been previously identified as a target object in a BEGIN MLOAD command.  
If the tname object is a view, it must not specify a join. MultiLoad operates only on single-table commands, so your INSERT statements must not contain joins. |
| Specifying the Applicable DML Statements   | One way of specifying the applicable DML statements is to relate each field name to the name of the column to which the field's data is applied.  
Another way tells MultiLoad to apply the first nonfiller field of a record that is sent to the Teradata Database to the first defined column of the affected table, the second nonfiller field to the second column, and so on.  
In either case, you must specify a value for every column, either explicitly or by default. |
| Required Access Privilege                  | To use the INSERT statement, you must have the INSERT privilege on the tname table or view.                                                                                                                    |
| ANSI/SQL DateTime Specifications           | You can use the ANSI/SQL DATE, TIME, TIMESTAMP, and INTERVAL DateTime data types in Teradata SQL CREATE TABLE statements, and you can specify them as column/field modifiers in INSERT statements. You must convert them to fixed-length CHAR data types when specifying the column/field names in the FIELD command. |
Example 1

The following examples show three ways to specify the relationship between the fields of input data records and the columns of the target table, using `targetable` as the target object name.

- **Using the `tname` Specification**

  ```
  .LAYOUT lname;
  .TABLE targetable;  
  .DML LABEL label;  
  INSERT INTO targetable .*;
  ```

- **Using the `cname` Specification**

  ```
  .LAYOUT lname;
  .FIELD first 1 somedatatype;  
  .FIELD f2nd * anydatatype;  
  .FIELD flast * datatype;  
  .DML LABEL label;  
  INSERT INTO targetable (col1, col2, ... colast) 
  VALUES (:f2nd, :first, ... :flast);  
  ```

- **Using the VALUES `fieldname` Specification**

  ```
  .LAYOUT lname;  
  .FIELD first 1 somedatatype;  
  .FIELD f2nd * anydatatype;  
  .FIELD flast * datatype;  
  .DML LABEL label;  
  INSERT INTO targetable VALUES (:first, :f2nd, ... :flast);
  ```
Example 2

The following example supposes an input data source that contains a series of 10- to 40-byte records. Each record contains the primary key value (EmpNum) of a row that is to be inserted successively into the Employee table whose columns are EmpNo, Name, and Salary. The example assumes that the current default database is Personnel, the database containing the Employee table.

```
.LAYOUT Layoutname;
.FIELD EmpNum 1 INTEGER;
.FIELD Name * (VARCHAR (30));
.FIELD Sal * (DECIMAL (7,2));
.DML LABEL DMLlabelname;
INSERT Employee (EmpNo, Name, Salary) VALUES (:EmpNum, :Name, :Sal);
```
LAYOUT

**Purpose**

The LAYOUT command, used with an immediately following sequence of FIELD, FILLER, and TABLE commands, specifies the layout of the input data records.

**Syntax**

```
.LAYOUT layoutname
  CONTINUEIF condition
  INDICATORS
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| CONTINUEIF      | Conditional phrase in which `condition` is of the form: `position = value`
| `condition`    | where
|                | - `position` is an unsigned integer (never an asterisk) that specifies the starting character position of the field of every input record that contains the continuation indicator. The position is relative to the first character position of the input record or input record fragment, which is always position 1.
|                | - `value` is the continuation indicator specified as a character constant or a string constant. MultiLoad uses the length of the constant as the length of the continuation indicator field.
|                | The `condition` specified as `position = value` is case sensitive—always specify the correct character case for this parameter.
|                | If the `condition` phrase is true, then Multiload forms a single record to be sent to the Teradata Database by concatenating the next input record at the end of the current record. (The current record is the one most recently obtained from the external data source.)
<p>|                | If the <code>condition</code> is false, then Multiload sends the current input record to the Teradata Database either by itself or as the last of a sequence of concatenated records. Regardless of whether the <code>condition</code> evaluates to true or false, MultiLoad removes the tested string (the continuation indicator field) from each record. All CONTINUEIF processing necessary to construct a complete record is done before any other processing of the record. |</p>
<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUEIF condition cont’d.</td>
<td>When the character set of the job script is different from the client character set used for the job, Teradata MultiLoad translates the specified value, which is either a character constant or a string constant, from the script character encoding to the client character encoding before evaluating the condition. Teradata MultiLoad uses the length of the constant in the client character encoding as the length of the continuation indicator field. For example, the job script must be in Teradata EBCDIC when using the UTF8 client character set on MVS; the job script can be in UTF8 when using the UTF16 client character set on network-attached systems. <strong>Caution:</strong> Before using the UTF8 client character set on a mainframe platform, check the character set definition to determine the code points and the Teradata EBCDIC and Unicode character mapping. Different versions of EBCDIC do not always agree as to the placement of any special characters you might require in the job script. See <em>International Character Set Support</em> for details.</td>
</tr>
<tr>
<td>INDICATORS</td>
<td>Condition that the data is in the indicator mode When you use the INDICATORS specification, MultiLoad sends all of the FIELD commands, including redefines, to the Teradata Database. <strong>Caution:</strong> Inappropriate INDICATORS specifications can corrupt the target table on the Teradata Database. If INDICATORS is specified in the LAYOUT command and the data file does not contain indicator bytes in each record, the target table is loaded with spurious data. Conversely, if INDICATORS is not specified and the data file contains indicator bytes in each record, the target table also is corrupted. Always make sure that your INDICATORS specifications match the mode of the data you are sending to the Teradata Database. <strong>Note:</strong> INDICATORS processing is done only after any CONTINUEIF processing is completed for a record.</td>
</tr>
<tr>
<td>layoutname</td>
<td>Name assigned to the layout for reference by one or more subsequent IMPORT commands A <em>layoutname</em> must obey the same construction rules as Teradata SQL column names.</td>
</tr>
</tbody>
</table>
## Usage Notes

The following table describes the things to consider when using the `LAYOUT` command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Following Commands</td>
<td>A <code>LAYOUT</code> command must be immediately followed by a combination of <code>FIELD</code>, <code>FILLER</code>, or <code>TABLE</code> commands. This sequence of commands, referenced by the <code>layoutname</code>, may describe one or more record formats contained in one or more client data sources. (See the redefinition options for <code>FIELD</code>, <code>FILLER</code>, and <code>TABLE</code> commands). The <code>LAYOUT</code> command sequence is terminated by the first subsequent command that is not a <code>FIELD</code>, <code>FILLER</code>, or <code>TABLE</code> command.</td>
</tr>
</tbody>
</table>
| Using the Same LAYOUT Command in More Than One MultiLoad Task | You can reference the same `layoutname` specification in more than one MultiLoad task, provided that:  
  - Each task is delimited by `BEGIN MLOAD` and `END MLOAD` commands in a single job step  
  - The `LAYOUT` command appears before any `IMPORT` commands that reference it |
| CONTINUEIF and INDICATORS Processing | When you specify both `CONTINUEIF` and `INDICATORS`:  
  - MultiLoad processes all of the `CONTINUEIF` specifications before performing the `INDICATORS` processing  
  - If the `CONTINUEIF` columns start in position 1, then any indicator bytes will follow the `CONTINUEIF` columns |
LOGDATA

Purpose
Supplies parameters to the LOGMECH command beyond those needed by the logon mechanism, such as user ID and password, to successfully authenticate the user. The LOGDATA command is optional. Whether or not parameters are supplied and the values and types of parameters depend on the selected logon method.

LOGDATA is only available on network-based platforms.

Syntax

```
.logdata_string
.`logdata_string`
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>logdata_string</code></td>
<td>Parameters for the logon mechanism specified using “LOGMECH” on page 152. For information about the logon parameters for supported mechanisms, see the Security Administration guide. The string is limited to 64 KB and must be in the session character set. To specify a string containing white space or other special characters, enclose the data string in single quotes.</td>
</tr>
</tbody>
</table>

Usage Notes
For more information about logon security, see Security Administration.

Example
If used, the LOGDATA and LOGMECH commands must precede the LOGON command. The commands themselves may occur in any order.

The following example demonstrates using the LOGDATA, LOGMECH, and LOGON commands in combination to specify the Kerberos logon authentication method and associated parameters:

```
.logmech KRB5;
.logdata joe@domain1@mypassword;
.logon cs4400s3;
```
LOGMECH

Purpose

Identifies the appropriate logon mechanism by name. If the mechanism specified requires parameters other than user ID and password for authentication, the LOGDATA command provides these parameters. The LOGMECH command is optional and available only on network-attached systems.

Syntax

where

```
.logmech logmech_name;
```

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>logmech_name</code></td>
<td>Logon mechanism</td>
</tr>
<tr>
<td></td>
<td>For a discussion of supported mechanisms, see Security Administration. The name is limited to 8 bytes; it is not case-sensitive.</td>
</tr>
</tbody>
</table>

Usage Notes

Every session to be connected requires a mechanism name. If none is supplied, a default mechanism can be used instead, as defined on either the server or client system in an XML-based configuration file.

For more information about logon security, see Security Administration.

Example

If used, the LOGDATA and LOGMECH commands must precede the LOGON command. The commands themselves may occur in any order.

The following example demonstrates using the LOGDATA, LOGMECH, and LOGON commands in combination to specify the Windows logon authentication method and associated parameters:

```
.logmech NTLM;
.logdata joe@domain1@mypassword;
.logon cs4400s3;
```
LOGOFF

Purpose

The LOGOFF command disconnects all active sessions and terminates MultiLoad on the client system.

Syntax

```
LOGOFF retcode
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retcode</td>
<td>Completion code returned to the client operating system</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a value for retcode, MultiLoad returns the appropriate terminating return code.</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the LOGOFF command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional Completion Code</td>
<td>You can specify the optional completion code value, retcode, as a conditional or an arithmetic expression, evaluated to a single integer. The LOGOFF command is processed when the highest return code reached prior to the LOGOFF command is no more than 4 (warning). Any higher return code would have already terminated the MultiLoad job. If the LOGOFF command is processed, MultiLoad returns the higher of:</td>
</tr>
<tr>
<td></td>
<td>• The retcode value specified as a LOGOFF command option</td>
</tr>
<tr>
<td></td>
<td>• The highest return code reached prior to the LOGOFF command For example, if the retcode value was specified as 2, but a warning had occurred earlier in the run, MultiLoad would return 4, not 2.</td>
</tr>
<tr>
<td></td>
<td>If a serious error terminates the program before the LOGOFF command is processed, the return code output is the value generated by the error condition rather than the retcode value specified as a LOGOFF command option.</td>
</tr>
</tbody>
</table>
Example

Assume that:

- Your MultiLoad job requires successful execution of a Teradata SQL statement, such as a CREATE TABLE statement.
- The statement fails with an unacceptable completion code.

If BADRC is set to &SYSRC after the failed Teradata SQL statement, you can use the following command to terminate the MultiLoad utility and return the unacceptable code to the client system:

```plaintext
.LOGOFF &BADRC;
```

Executing this command also drops the restart log table. If execution is terminated before the LOGOFF command is encountered, the restart log table is not dropped, so as to support a restart at a later time.
LOGON

Purpose

The LOGON command establishes an Teradata SQL session with the Teradata Database.

Syntax

Standard LOGON Syntax

```
LOGON [tdpid /] username, password, acctid ;
```

On MVS/VM, with the use of the User Logon Exit routine in TDP, the user name is not required. For more information, see the Teradata Director Program Reference.

Single Sign-On LOGON Syntax

```
LOGON [tdpid /] username, password, acctid ;
```

Note: When logon encryption is enabled on the gateway, single sign-on is disabled on the client and standard logon syntax is used instead.

Note: Use of the period preceding the LOGON command is optional.

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acctid</td>
<td>Account identifier of up to 30 characters associated with the user name. If you do not specify an acctid, MultiLoad uses the default identifier defined when the user was created.</td>
</tr>
<tr>
<td>password</td>
<td>Password of up to 30 characters associated with the user name.</td>
</tr>
</tbody>
</table>
Chapter 3: MultiLoad Commands

LOGON

The following table describes the things to consider when using the LOGON command.

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tdpid</code></td>
<td>Optional character string that identifies the name of a TDP</td>
</tr>
<tr>
<td></td>
<td>If you do not specify the <code>tdpid</code>, MultiLoad uses the default TDP established by the system administrator.</td>
</tr>
<tr>
<td></td>
<td>For channel-attached systems, the <code>tdpid</code> string must be in the form:</td>
</tr>
<tr>
<td></td>
<td>TDPn</td>
</tr>
<tr>
<td></td>
<td>where n is the TDP identifier.</td>
</tr>
<tr>
<td><code>username</code></td>
<td>User identifier of up to 30 characters</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the LOGON command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logon</td>
<td>For standard logon, the parameters (<code>tdpid</code>, <code>username</code>, <code>password</code>, and <code>acctid</code>) are used in all sessions established with the Teradata Database. The LOGON command may occur only once.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Multi-byte character support limits the logon string to 30 bytes. This limit is applied to the logon object name, which includes the <code>username</code>, <code>password</code>, and <code>acctid</code>. Therefore, when using multi-byte character sets, create logon strings that are less than 30 bytes. Otherwise, the logon could fail.</td>
</tr>
<tr>
<td></td>
<td>For single sign-on, if your Gateway to Teradata Database is configured to use single sign-on (SSO), and you are already logged on to your Teradata client machine, then the machine name, user name, and password are not required in the LOGON command. The user name and password combination specified when you logged on to your Teradata client machine are authenticated with network security for a single sign-on such that valid Teradata users will be permitted to log on to the Teradata Database. The use of SSO is strictly optional, unless the Gateway has been configured to accept only SSO-style logons.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> On MVS/VM, with the use of User Logon Exit routine in TDP, <code>userid</code> is not required (see Teradata Director Program Reference for more information), otherwise, <code>userid</code> is required.</td>
</tr>
<tr>
<td></td>
<td>If you want to connect to a Teradata Database other than the one you are currently logged onto, the <code>tdpid</code> must be included in the LOGON command.</td>
</tr>
<tr>
<td></td>
<td>If the <code>tdpid</code> is not specified, the default contained in <code>clispb.dat</code> will be used.</td>
</tr>
<tr>
<td></td>
<td>Refer to Teradata Call-Level Interface Version 2 Reference for Network-Attached Systems for information about setting defaults.</td>
</tr>
<tr>
<td></td>
<td>To be interpreted correctly, the <code>tdpid</code> must be followed by the slash separator (<code>/</code>), to distinguish the <code>tdpid</code> from a Teradata Database user name. For example, to connect to <code>serverxyz</code>, you would enter one of the following:</td>
</tr>
<tr>
<td></td>
<td><code>.LOGON serverxyz/</code>;</td>
</tr>
<tr>
<td></td>
<td>`.LOGON serverxyz/,,acctinfo``;</td>
</tr>
<tr>
<td></td>
<td>If an account ID is to be used, the optional account ID must be specified in the LOGON command.</td>
</tr>
</tbody>
</table>
Chapter 3: MultiLoad Commands
LOGON

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using LOGON with the LOGTABLE Command</td>
<td>Both the LOGON and LOGTABLE commands are required. LOGON and LOGTABLE commands may appear in any order, but must precede other commands except RUN commands used to identify the file containing the LOGON command. If you enter the LOGON command first, MultiLoad warns you that the LOGTABLE command also is required.</td>
</tr>
</tbody>
</table>

**Example**

The following example presents both the LOGON command and LOGTABLE command as they typically occur:

```
.logtable logtable001;
.logon tdpx/me,paswd;
```
Chapter 3: MultiLoad Commands

LOGTABLE

Purpose

The LOGTABLE command specifies a restart log table for the MultiLoad checkpoint information. MultiLoad uses the information in the restart log table to restart jobs that are halted because of a Teradata Database or client system failure.

Syntax

```
LOGTABLE dbname tname ;
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbname</code></td>
<td>Name of the database under which the log table exists. The default is the database name associated with the user name specified in the LOGON command. MultiLoad searches for the <code>tname</code> table in that database unless another database name is specified in this option.</td>
</tr>
<tr>
<td><code>tname</code></td>
<td>Name of the restart log table</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the LOGTABLE command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using LOGTABLE with the LOGON command</td>
<td>Both the LOGTABLE and LOGON command are required. LOGTABLE and LOGON commands may appear in any order, but must precede other commands except RUN commands used to identify the file containing the LOGON command. If you enter the LOGON command first, MultiLoad warns you that the LOGTABLE command is also required.</td>
</tr>
<tr>
<td>The Restart Log Table</td>
<td>The table that you specify as the MultiLoad restart log table does not have to be fully qualified.</td>
</tr>
</tbody>
</table>
### Usage Notes

The following example presents both the LOGTABLE and LOGON commands as they typically occur.

```
.logtable Mine.Logtable001;
.logon tdpx/me,paswd;
```
PAUSE ACQUISITION

Purpose

The PAUSE ACQUISITION command pauses the MultiLoad job during the acquisition phase of an import task. (You cannot use the PAUSE ACQUISITION command in a MultiLoad delete task.)

Syntax

.PAUSE ACQUISITION ;

Usage Notes

The following table describes the things to consider when using the PAUSE ACQUISITION command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational Requirements</td>
<td>You can use PAUSE ACQUISITION commands to separate a MultiLoad job into two or more distinct operations provided that it has either:</td>
</tr>
<tr>
<td></td>
<td>• One or more acquisition phase operations that load import data into the MultiLoad work tables.</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>• An application phase operation that inserts the work table data into the target tables or views on the Teradata Database.</td>
</tr>
</tbody>
</table>
Your MultiLoad job can use more than one PAUSE ACQUISITION command, as long as each one is associated with a different import task—you can only use one PAUSE ACQUISITION command in each import task.

In your MultiLoad job script, PAUSE ACQUISITION commands must appear:
- After the BEGIN MLOAD command
- Before the END MLOAD command

If a PAUSE ACQUISITION command appears before the first IMPORT command, then MultiLoad pauses the job before starting the acquisition phase.
If a PAUSE ACQUISITION command appears after the last IMPORT command, then MultiLoad pauses the job before starting the application phase.
If a PAUSE ACQUISITION command appears between two IMPORT commands, then MultiLoad begins the acquisition phase, processes the first IMPORT command, and pauses the job before processing the second IMPORT command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Frequency and Placement</td>
<td>Your MultiLoad job can use more than one PAUSE ACQUISITION command, as long as each one is associated with a different import task—you can only use one PAUSE ACQUISITION command in each import task.</td>
</tr>
<tr>
<td>Pause Conditions</td>
<td>When a MultiLoad job pauses, the utility:</td>
</tr>
<tr>
<td></td>
<td>• Displays a message indicating that the job was paused by user request.</td>
</tr>
<tr>
<td></td>
<td>• Terminates the job with a return code of 4.</td>
</tr>
<tr>
<td>Restarting the Job</td>
<td>To restart a paused MultiLoad job, remove the associated PAUSE ACQUISITION statement from the MultiLoad job script and resubmit the job. MultiLoad then resumes processing acquisition phase IMPORT commands, and either:</td>
</tr>
<tr>
<td></td>
<td>• Continues into the application phase</td>
</tr>
<tr>
<td></td>
<td>• Pauses again in response to another PAUSE ACQUISITION command</td>
</tr>
</tbody>
</table>
**Purpose**

When a MultiLoad task has been suspended or aborted before the end of the application phase, the RELEASE MLOAD statement removes the access locks from the target tables in the Teradata Database and inhibits any attempts to restart the MultiLoad utility.

**Note:** RELEASE MLOAD is a Teradata SQL statement that is recognized by BTEQ and Multload. RELEASE MLOAD may be used in your MultiLoad job script or in an interactive MultiLoad session. It may also be used in BTEQ. Log on to BTEQ to enter the RELEASE MLOAD statement. Do not use a leading period (.) when using the RELEASE MLOAD statement in BTEQ.

**Syntax**

```
RELEASE MLOAD dbname tablename IN APPLY
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbname</code></td>
<td>Name of the database under which the target table that you want to release exists. The default is the database name associated with the user name specified in the BTEQ LOGON command. BTEQ searches for the <code>tablename</code> table in that database unless this option specifies another database name.</td>
</tr>
<tr>
<td><code>tablename</code></td>
<td>Name of the MultiLoad target table</td>
</tr>
<tr>
<td><code>IN APPLY</code></td>
<td>MultiLoad task that was in the application phase when it was suspended</td>
</tr>
</tbody>
</table>

**Note:** You cannot use the IN APPLY option to release the locks that were placed on the MultiLoad target tables during the acquisition phase. In this case, use the RELEASE MLOAD statement without the IN APPLY option.

**Note:** The RELEASE MLOAD statement frees the target tables, but does not delete the error tables, the work tables, or the restart log table. They remain in the database, and you must drop them manually to free up the space and avoid conflicts when you resubmit the MultiLoad job.
### Usage Notes

The following table describes the things to consider when using the RELEASE MLOAD statement.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Releasing Application Locks</td>
<td>The IN APPLY option of the RELEASE MLOAD statement releases the application locks on the target tables that satisfy one of the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• No changes have been made to the table</td>
</tr>
<tr>
<td></td>
<td>• The table is empty and has no permanent journals</td>
</tr>
<tr>
<td></td>
<td>• The table has no fallback, no NUSIs, and no permanent journals</td>
</tr>
<tr>
<td></td>
<td>If none of these conditions is true, and the table has no permanent journals, then MultiLoad changes the application lock to a restoration lock that prevents MultiLoad from restarting and allows the following table accesses:</td>
</tr>
<tr>
<td></td>
<td>• DELETE ALL</td>
</tr>
<tr>
<td></td>
<td>• DROP FALBACK</td>
</tr>
<tr>
<td></td>
<td>• DROP INDEX</td>
</tr>
<tr>
<td></td>
<td>• DROP TABLE</td>
</tr>
<tr>
<td></td>
<td>• SELECT with access lock</td>
</tr>
<tr>
<td></td>
<td>If permanent journals are defined on the table, the only option is to drop the target table. If the target table belongs to a replication group and change data capture is active (for example, the replication group status is not Defined nor Terminated), then the following steps must be completed to drop the table:</td>
</tr>
<tr>
<td></td>
<td>1 Put the replication group in Suspended status.</td>
</tr>
<tr>
<td></td>
<td>2 Use the ALTER REPLICATION GROUP statement to remove the table from the replication group.</td>
</tr>
<tr>
<td></td>
<td>3 Drop the table.</td>
</tr>
<tr>
<td>Table Requirements</td>
<td>All of the tables that you specify must have been involved in a MultiLoad task. If a specified table is not involved in a MultiLoad task, the Teradata Database rejects the RELEASE MLOAD statement.</td>
</tr>
<tr>
<td>Required Access Privilege</td>
<td>To use the RELEASE MLOAD statement, you must either be the owner of the database or have at least one of the following privileges on the specified tables:</td>
</tr>
<tr>
<td></td>
<td>• Insert</td>
</tr>
<tr>
<td></td>
<td>• Update</td>
</tr>
<tr>
<td></td>
<td>• Delete</td>
</tr>
</tbody>
</table>
Locks

To release the target tables, the MultiLoad release function must first obtain an exclusive lock on each specified table. This is not possible, and the RELEASE MLOAD statement will fail if either a database lock or a MultiLoad write lock has been placed on any of the specified tables.

MultiLoad normally places write locks on the target tables:
- Near the end of the acquisition phase of an import task, when data acquisition is complete and the data sort operation is about to begin.
- During the preliminary phase of a delete task when the DELETE statement is sent to the Teradata Database.

If you execute the RELEASE MLOAD statement before the MultiLoad task reaches these critical points, the release function completes before the utility can place write locks on the target tables.

Messages Returned

A release completed message indicates that all of the specified tables were releasable and the release function completed without error.

If the release function encounters a table that cannot be released, it terminates with an error message identifying the table and the reason it could not be released.

Acquisition phase error conditions include:
- Table does not exist
- Table has some other lock
- Table is not a MultiLoad target table
- The requesting user does not have the required access privilege

Application phase error conditions include all of the acquisition phase error conditions plus:
- Table has fallback
- Table has a NUSI
- Table has permanent journals
- Table is not in the application phase

In this case, you could either:
- Execute another RELEASE MLOAD statement, specifying only the tables that were releasable
- Take corrective action, based on the error condition, and then execute another RELEASE MLOAD statement

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locks</td>
<td>To release the target tables, the MultiLoad release function must first obtain an exclusive lock on each specified table. This is not possible, and the RELEASE MLOAD statement will fail if either a database lock or a MultiLoad write lock has been placed on any of the specified tables.</td>
</tr>
<tr>
<td>Messages Returned</td>
<td>A release completed message indicates that all of the specified tables were releasable and the release function completed without error. If the release function encounters a table that cannot be released, it terminates with an error message identifying the table and the reason it could not be released.</td>
</tr>
</tbody>
</table>

Acquisition phase error conditions include:
- Table does not exist
- Table has some other lock
- Table is not a MultiLoad target table
- The requesting user does not have the required access privilege

Application phase error conditions include all of the acquisition phase error conditions plus:
- Table has fallback
- Table has a NUSI
- Table has permanent journals
- Table is not in the application phase

In this case, you could either:
- Execute another RELEASE MLOAD statement, specifying only the tables that were releasable
- Take corrective action, based on the error condition, and then execute another RELEASE MLOAD statement
ROUTE MESSAGES

Purpose

The ROUTE MESSAGES command specifies alternate destinations for MultiLoad utility output messages.

Syntax

```
ROUTE MESSAGES FILE fileid1 TO echo options
        WITH ECHO TO FILE fileid2
OFF
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO</td>
<td>Additional destination, with a fileid specification</td>
</tr>
<tr>
<td></td>
<td>For example, use the ECHO keyword to specify that messages be captured in a file (fileid2) while still being written to your terminal.</td>
</tr>
<tr>
<td></td>
<td>Note: The ECHO OFF specification cancels the additional file specification of a previously established ECHO destination.</td>
</tr>
<tr>
<td>fileid1 and fileid2</td>
<td>Alternate message destinations in the external system</td>
</tr>
<tr>
<td></td>
<td>• In MVS, this is a DDNAME. (See the “MVS fileid Usage Rules” topic in “Usage Notes” on page 166.)</td>
</tr>
<tr>
<td></td>
<td>• In UNIX and Windows, this is the path name for a file. If the path name has embedded white space characters, you must enclose the entire pathname in single or double quotes.</td>
</tr>
<tr>
<td></td>
<td>• In VM, this is a FILEDEF name.</td>
</tr>
<tr>
<td></td>
<td>If you specify the same destination with both fileid1 and fileid2 parameters, MultiLoad duplicates the messages at each destination.</td>
</tr>
<tr>
<td></td>
<td>For more information, see the “ACCEPT” or “RUN FILE” command descriptions.</td>
</tr>
</tbody>
</table>
Usage Notes

The following table describes the things to consider when using the ROUTE MESSAGES command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
</table>
| Specifying the System Console/Standard Output Device | You can use the asterisk character (*) as the fileid1 or fileid2 specifications to route messages to the system console/standard output (stdout) device.   
   The system console is the:                                                           |
   • Display screen in interactive mode  
   • Standard output device in batch mode  
   For more information about the display screen and standard output devices, see “File Requirements” on page 29. | |
| Default Message Destinations              | If you do not use the ROUTE MESSAGES command, MultiLoad writes output messages to:                                                         |
   • DDNAME SYSPRINT in VM and MVS  
   • stdout in UNIX and Windows                                                         | |
| MVS fileid Usage Rules                    | If you specify a DDNAME, MultiLoad writes messages to the specified source.                                                                   |
   A DDNAME must obey the same construction rules as Teradata SQL column names, except that:  
   • The “at” character (@) is allowed as an alphabetic character.                         
   • The underscore character (_) is not allowed.                                           
   The DDNAME must obey the applicable rules of the external system.                       
   The DDNAME represents a data source on magnetic tape, the tape may be either labeled or non-labeled, as supported by the operating system. | |

Example

.ROUTE MESSAGES FILE OUTPUT;

The messages are written to the file designated by OUTPUT from this point unless redirected by another ROUTE MESSAGES command.

Note: On network-attached systems, if outfilename is used both to redirect stdout and as the fileid in a ROUTE MESSAGES WITH ECHO command, the results written to outfilename may be incomplete due to conflicting writes to the same file.
RUN FILE

Purpose

The RUN FILE command invokes the specified external source as the current source of commands and statements.

Syntax

```
.RUN FILE fileid

  IGNORE charpos1
  THRU

  THRU charpos2

; 
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileid</td>
<td>Data source of the external system. The external system DD (or similar) statement specifies a file.</td>
</tr>
<tr>
<td></td>
<td>- In MVS, this is a DDNAME. (See the “MVS fileid Usage Rules” topic in “Usage Notes” on page 168.)</td>
</tr>
<tr>
<td></td>
<td>- In UNIX and Windows, this is the path name for a file. If the path name has embedded white space characters, you must enclose the entire pathname in single or double quotes.</td>
</tr>
<tr>
<td></td>
<td>- In VM, this is a FILEDEF name.</td>
</tr>
<tr>
<td>IGNORE charpos1 and charpos2</td>
<td>Start and end character positions of a field in each input record that contains extraneous information.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>- Use charpos1 to ignore only the single specified character.</td>
</tr>
<tr>
<td></td>
<td>- Use charpos1 THRU to ignore all characters from charpos1 through the end of the record.</td>
</tr>
<tr>
<td></td>
<td>- Use THRU charpos2 to ignore all characters from the beginning of the record through charpos2.</td>
</tr>
<tr>
<td></td>
<td>- Use charpos1 THRU charpos2 to ignore all characters from charpos1 through charpos2.</td>
</tr>
</tbody>
</table>
### Usage Notes

The following table describes the things to consider when using the RUN FILE command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying the System Console/Standard Input Device</td>
<td>You can use the asterisk character (*) as the fileid specification for the system console/standard input (stdin) device. The system console is the: • Keyboard in interactive mode • Standard input device in batch mode For more information about the keyboard and standard input devices, see “File Requirements” on page 29.</td>
</tr>
<tr>
<td>MVS fileid Usage Rules</td>
<td>If you specify a DDNAME, MultiLoad reads data records from the specified source. A DDNAME must obey the same construction rules as Teradata SQL column names except that: • The “at” character (@) is allowed as an alphabetic character. • The underscore character (_) is not allowed. The DDNAME must obey the applicable rules of the external system. If the DDNAME represents a data source on magnetic tape, the tape may be either labeled or non-labeled, as supported by the operating system.</td>
</tr>
<tr>
<td>Executing the RUN FILE Command</td>
<td>After MultiLoad executes the RUN FILE command, it reads additional commands from the specified source until a LOGOFF command or end of file condition is encountered, whichever occurs first. An end of file condition automatically causes MultiLoad to resume reading its commands and DML statements from the previously active source: • SYSIN for VM and MVS • stdin (normal or redirected) for UNIX and Windows Note: SYSIN(stdin remains the active input source after MultiLoad processes any user-provided invocation parameters.</td>
</tr>
<tr>
<td>Nested RUN Commands</td>
<td>The source specified by a RUN FILE command can have up to 16 levels of nested RUN commands.</td>
</tr>
<tr>
<td>-i scriptencoding parameter</td>
<td>When the -i scriptencoding parameter is used, the specified encoding form and byte order apply to all the command files specified by the .RUN FILE command and any nested RUN FILE commands.</td>
</tr>
</tbody>
</table>
SET

Purpose

The SET command assigns a data type and a value to a MultiLoad variable.

Syntax

\[
\text{SET} \quad \text{var} \quad \text{TO} \quad \text{expression} \quad ;
\]

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>New value for the utility variable var</td>
</tr>
<tr>
<td>var</td>
<td>Name of the MultiLoad variable to be set to the evaluated expression</td>
</tr>
</tbody>
</table>

Usage Notes

The following table describes the things to consider when using the SET command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaring Variables</td>
<td>Variables need not be declared in advance to be the object of the SET command. If a variable does not already exist, MultiLoad creates it. Variables used to the right of TO in the expression must be declared in advance.</td>
</tr>
<tr>
<td>Changing the Data Type</td>
<td>The SET command also dynamically changes the data type to that of the assigned value if it had already been defined. If the expression evaluates to a numeric value, the symbol is assigned an integer value, as in: (.\text{SET F00NUM TO -151 ;}) If the expression is a quoted string, the symbol is assigned a string value, as in: (.\text{SET F00CHAR TO '151' ;}) The minimum and maximum limits for floating point data types are: (4.0\text{E-75} \leq \text{abs(float variable)} &lt; 7.0\text{E75})</td>
</tr>
<tr>
<td>Variable Substitution</td>
<td>MultiLoad variable can be substituted wherever substitution is allowed.</td>
</tr>
</tbody>
</table>
Examples

MultiLoad supports concatenation of variables, using the SET command, such as:

```
.SET C TO 1;
.SET D TO 2;
.SET X TO &C.&D;
```

In this example, X evaluates to 12.

If a decimal point is added to the concatenated variables, then X evaluates to 1.2, as in:

```
.SET C TO 1;
.SET D TO 2;
.SET X TO &C..&D;
```
Purpose

The SYSTEM command submits an operating system command to your client environment during a MultiLoad operation.

Syntax

```
.SYSTEM 'oscommand' ;
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oscommand</td>
<td>Any legal command in your client operating system</td>
</tr>
</tbody>
</table>

Usage Notes

The SYSTEM command suspends the current MultiLoad operation to execute the client operating system command.

When the client operating system command completes, MultiLoad displays the return code from the invoked command and updates the &SYSRC variable.
## TABLE

### Purpose

The TABLE command identifies a table whose column names and data descriptions are used as the names and data descriptions of fields of the input records. These are assigned in the order defined.

### Syntax

```plaintext
.TABLE tableref ;
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tableref</code></td>
<td>Existing table whose column names and data descriptions are assigned, in the order defined, to fields of the input data records</td>
</tr>
</tbody>
</table>

### Usage Notes

The following table describes the things to consider when using the TABLE command.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Names in the Referenced Table</td>
<td>The column names of the specified table must be Teradata SQL column names that need not be enclosed in quotation marks. Tables cannot be created with invalid column names, and any nonstandard column name produces an error condition, depending on the nature of the divergence from the standard. The errors are:</td>
</tr>
<tr>
<td></td>
<td>• Embedded blanks cause a syntax error, depending on the non-blank contents of the name.</td>
</tr>
<tr>
<td></td>
<td>• Invalid characters cause an invalid name error.</td>
</tr>
<tr>
<td></td>
<td>• Reserved words cause a syntax error that mentions invalid use of the reserved word.</td>
</tr>
<tr>
<td>TABLE command and UDT type</td>
<td>When the TABLE command is used and the table contains a structured UDT type, MultiLoad returns an external representation of the UDT and that requires the user to transform. The term &quot;external type&quot; means the data type of the external opaque container for a structured UDT and is the type returned by the from-sql transform method.</td>
</tr>
</tbody>
</table>
You can intermix one or more TABLE commands with the FIELD or FILLER following a LAYOUT command.

This method of specifying record layout fields assumes each field, as defined by the data description of the corresponding column of tableres, is contiguous with the previous one, beginning at the next-available character position beyond any previous field specifications for the input records. The fields must appear in the order defined for the columns of the table.

The object identified by the tableres parameter must be a table. It need not appear as a parameter of the BEGIN MLOAD or BEGIN DELETE MLOAD command, but you must either be the owner of the object or have at least one privilege on it.

If specified as an unqualified table name, the current default database qualifies it.
The UPDATE command is a version of the Teradata SQL UPDATE statement that changes field values in existing rows of a table.

Syntax

```
UPDATE tname SET cname = expr WHERE condition ;
```

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cname</code></td>
<td>Column whose value is to be replaced by the value of <code>expr</code>&lt;br&gt;The column named must not be a column of the primary index.</td>
</tr>
<tr>
<td><code>expr</code></td>
<td>Expression whose resulting value is to replace the current value of the identified column&lt;br&gt;The expression can contain any combination of:&lt;br&gt;• Constants&lt;br&gt;• Current values of columns of the referenced row&lt;br&gt;• Values from fields of input data records</td>
</tr>
<tr>
<td><code>tname</code></td>
<td>Table or view to be updated&lt;br&gt;This table was previously identified as <code>tname1</code> in the BEGIN MLOAD command.&lt;br&gt;If <code>tname</code> is not explicitly qualified by database name, the current default database qualifies it.</td>
</tr>
<tr>
<td><code>condition</code></td>
<td>Conditional clause that specifies the row or rows to be updated&lt;br&gt;The conditional clause can use values from fields of input data records by referring to their field names.</td>
</tr>
</tbody>
</table>
### Usage Notes

The following table describes the things to consider when using the UPDATE statement.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Usage Notes</th>
</tr>
</thead>
</table>
| Referencing Field Names | Make references to fields of the input data records for the expr and WHERE conditional as follows:  
:fieldname  
where *fieldname* is defined by a FIELD or TABLE command of the layout referenced by an IMPORT command using this UPDATE statement.  
For the WHERE *condition* clause, the equality values for all the columns of the primary index must be explicitly specified. |
| Specifying Multiple UPI Columns in an Import Task | If you specify multiple UPI columns in an import task, specify them all in the WHERE clause of the UPDATE statement.  
In this case, the WHERE clause is fully qualified, thereby allowing MultiLoad to optimize the processing. |
| Update Object Restrictions | If the object of the UPDATE statement is a view, it must not specify a join. MultiLoad operates only on single tables, so UPDATE statements must not contain any joins.  
Only one object may be identified and that must be a target object as specified in the BEGIN MLOAD command. |
| OR Construct | You cannot use the OR construct in an UPDATE statement.  
To accomplish the result normally achieved with the OR construct, use two separate UPDATE statements and use the APPLY clause of the IMPORT command to apply them conditionally. |
| Required Access Privileges | To use the UPDATE statement, you must have the UPDATE privilege on the *tname* table or view. |

### Example

The following example depicts an input data source that contains a series of 14-byte records. Each record contains the value of the primary index column (EmpNo) of a row of the Employee table whose PhoneNo column is to be assigned a new phone number from field Fone. The example assumes that the current default database is Personnel, the database containing the Employee table.

```plaintext
.LAYOUT Layoutname;  
.FIELD EmpNum 1 INTEGER;  
.FIELD Fone * (CHAR (10));  
.DML LABEL DMLlabelname;  
UPDATE Employee SET PhoneNo = :Fone WHERE EmpNo = :EmpNum;
```
This appendix describes the conventions that apply to reading the syntax diagrams used in this book.

## Syntax Diagram Conventions

### Notation Conventions

The following table defines the notation used in this section.

<table>
<thead>
<tr>
<th>Item</th>
<th>Definition / Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>An uppercase or lowercase alphabetic character ranging from A through Z.</td>
</tr>
<tr>
<td>Number</td>
<td>A digit ranging from 0 through 9. Do not use commas when entering a number with more than three digits.</td>
</tr>
<tr>
<td>Word</td>
<td>Variables and reserved words:</td>
</tr>
<tr>
<td></td>
<td><strong>UPPERCASE LETTERS</strong></td>
</tr>
<tr>
<td></td>
<td>Represents a keyword.</td>
</tr>
<tr>
<td></td>
<td>Syntax diagrams show all keywords in uppercase, unless operating system restrictions require them to be in lowercase.</td>
</tr>
<tr>
<td></td>
<td>If a keyword is shown in uppercase, you may enter it in uppercase or mixed case.</td>
</tr>
<tr>
<td></td>
<td><strong>LOWERCASE LETTERS</strong></td>
</tr>
<tr>
<td></td>
<td>Represents a keyword that you must enter in lowercase, such as a UNIX command.</td>
</tr>
<tr>
<td></td>
<td><strong>LOWERCASE ITALIC LETTERS</strong></td>
</tr>
<tr>
<td></td>
<td>Represents a variable such as a column or table name. You must substitute a proper value.</td>
</tr>
<tr>
<td>Spaces</td>
<td>Use one space between items, such as keywords or variables.</td>
</tr>
<tr>
<td>Punctuation</td>
<td>Enter all punctuation exactly as it appears in the diagram.</td>
</tr>
</tbody>
</table>

### Paths

The main path along the syntax diagram begins at the left, and proceeds, left to right, to the vertical bar, which marks the end of the diagram. Paths that do not have an arrow or a vertical bar only show portions of the syntax.
Note that the only part of a path that reads from right to left is a loop.
Paths that are too long for one line use continuation links. Continuation links are small circles with letters indicating the beginning and ending of a link:

When you see a circled letter in a syntax diagram, go to the corresponding circled letter and continue.

**Required Items**

Required items appear on the main path.

If you can choose from more than one item, the choices appear vertically, in a stack. The first item appears on the main path.

**Optional Items**

Optional items appear below the main path.

If choosing one of the items is optional, all the choices appear below the main path.
You can choose one of the options, or you can disregard all of the options.

**Abbreviations**

If a keyword or a reserved word has a valid abbreviation, the unabbreviated form always appears on the main path. The shortest valid abbreviation appears beneath.

In the above syntax, the following formats are valid:

- SHOW CONTROLS
- SHOW CONTROL

**Loops**

A loop is an entry or a group of entries that you can repeat one or more times. Syntax diagrams show loops as a return path above the main path, over the item or items that you can repeat.

The following rules apply to loops.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>there is a maximum number of entries allowed</td>
<td>the number appears in a circle on the return path.</td>
</tr>
<tr>
<td></td>
<td>In the example, you may specify <code>cname</code> a maximum of four times.</td>
</tr>
<tr>
<td>there is a minimum number of entries required</td>
<td>the number appears in a square on the return path.</td>
</tr>
<tr>
<td></td>
<td>In the example, you must enter at least three groups of column names.</td>
</tr>
<tr>
<td>a separator character is required between entries</td>
<td>the character appears on the return path.</td>
</tr>
<tr>
<td></td>
<td>If the diagram does not show a separator character, use one blank space.</td>
</tr>
<tr>
<td></td>
<td>In the example, the separator character is a comma.</td>
</tr>
<tr>
<td>a delimiter character is required around entries</td>
<td>the beginning and ending characters appear outside the return path.</td>
</tr>
<tr>
<td></td>
<td>Generally, a space is not needed between delimiter characters and entries.</td>
</tr>
<tr>
<td></td>
<td>In the example, the delimiter characters are the left and right parentheses.</td>
</tr>
</tbody>
</table>
Sometimes a piece of a syntax phrase is too large to fit into the diagram. Such a phrase is indicated by a break in the path, marked by | terminators on either side of the break. A name for the excerpted piece appears between the break marks in boldface type.

The named phrase appears immediately after the complete diagram, as illustrated by the following example.
This appendix provides program JCL and command examples for invoking MultiLoad on the following systems:

- VM
- MVS
- UNIX and Windows

**VM**

This section provides program JCL and command examples for invoking MultiLoad on VM systems.

**Note:** The host application environment for these examples is H.3.0.

**Using the Reduced Print Output (BRIEF) Parameter**

**Sample Command:**

```bash
mload stress1 brief
```

**Note:** `stress1` is a sample input script.

**Sample Output:**

```plaintext
**** 10:15:36 UTY2414 BRIEF option is enabled.
========================================================================
=          MultiLoad Utility    Release MLOD.12.00.00.00                =
=          Platform VM                                                   =
========================================================================
= Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
========================================================================
**** 10:15:36 UTY2411 Processing start date: THU NOVEMBER 30, 2006        =
========================================================================
= Logon/Connection                                                        =
========================================================================
0001 .logtable mkbtest1;
0002 .logon tdp9/mkb,;
**** 10:15:52 UTY8400 Teradata Database Release: 12.00.00.00
**** 10:15:52 UTY8400 Teradata Database Version: 12.00.00.00
**** 10:15:52 UTY8400 Default character set: EBCDIC
**** 10:15:52 UTY6217 A successful connect was made to the DBS.
**** 10:15:52 UTY6211 Logtable 'MKB.MKBTEST1' has been created.
```
Using the Character Set (CHARSET) Selection Parameter

Sample Command:

```sql
mload stress1 charset=ebcdic
```

Note: stress1 is a sample input script.

Sample Output:

```sql
*** 10:27:56 UTY2407 Run time parameters in effect are: CHARSET=EBCDIC.
========================================================================
| = MultiLoad Utility       Release MLOD.12.00.00.000 =
| = Platform VM =
|========================================================================
| = Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
|========================================================================
| **** 10:27:56 UTY2411 Processing start date: THU NOVEMBER 30, 2006 |
|========================================================================
| | Logon/Connection |
|========================================================================

| 0001 .logtable mkbtest1;
| 0002 .logon tdp9/mkb,;

**** 10:28:10 UTY8400 Teradata Database Release: 12.00.00.00
**** 10:28:10 UTY8400 Teradata Database Version: 12.00.00.00
**** 10:28:10 UTY8400 Default character set: EBCDIC
**** 10:28:10 UTY6211 A successful connect was made to the DBS.
**** 10:28:10 UTY6217 Logtable 'MKB.MKBTEST1' has been created.
========================================================================
| = Processing Control Statements |
|========================================================================

| 0003 /*************************************************************************/
|/* */
|/***************************************************************************/
```
/* Test handling multiple Mload tasks. */
/*
*********************************************************************/
create table foo1 ( f1 char(1),
                   f2 char(2),
                   f3 char(3) );
.
.

Using the Error Logging (ERRLOG) Parameter

Sample Command:

mload stress1 errlog=foo

Note: stress1 is a sample input script. foo must be defined. In this example foo was defined by:

cfiledef foo disk error file a

Sample Output:

**** 10:31:54 UTY2413 Error Logging is enabled: FOO
========================================================================
= Multiloading Utility                   Release MLOD.12.00.00.000 =
= Platform VM                           =
========================================================================
= Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
========================================================================
**** 10:31:54 UTY2411 Processing start date: THU NOVEMBER 30, 2006
========================================================================
== Logon/Connection =
========================================================================
0001 .logtable mkbtest1;
0002 .logon tdp9/mkb,;
**** 10:32:26 UTY8400 Teradata Database Release: 12.00.00.00
**** 10:32:26 UTY8400 Teradata Database Version: 12.00.00.00
**** 10:32:26 UTY8400 Default character set: EBCDIC
/*********************************************************************/
/*                      Test handling multiple Mload tasks.          */
/*********************************************************************/
create table foo1 ( f1 char(1),
                   f2 char(2),
                   f3 char(3) );
.
.

Teradata MultiLoad Reference 183
Appendix B: Invocation Examples

MVS

Specifying Multiple Parameters

**Sample Command:**

mload stress1 charset=ebcdic brief

**Note:** stress1 is a sample input script.

**Sample Output:**

```plaintext
*** 10:43:18 UTY2407 Run time parameters in effect are: CHARSET=EBCDIC.
*** 10:43:18 UTY2414 BRIEF option is enabled.

= Multiloading Utility  Release MLOD.12.00.00.000 =
= Platform VM =

= Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =

= Logon/Connection =

0001 .logtable mkbtest1;
0002 .logon tdp9/mkb,;
*** 10:43:32 UTY8400 Teradata Database Release: 12.00.00.00
*** 10:43:32 UTY8400 Teradata Database Version: 12.00.00.00
*** 10:43:32 UTY6211 A successful connect was made to the DBS.
*** 10:43:32 UTY6217 Logtable 'MKB.MKBTEST1' has been created.

0003 /*
/==================================================================================*/
/*
/*   Test handling multiple Mload tasks.
/*
/==================================================================================*/
create table foo1 ( f1 char(1),
   f2 char(2),
   f3 char(3) );
```

**MVS**

This section provides program JCL and command examples for invoking MultiLoad on MVS systems. The MVS JCL for MultiLoad is in <pfx>.PROCLIB (TDSMLOAD).

**Using the Reduced Print Output (BRIEF) Parameter**

**Sample JCL:**

```plaintext
//MKBMLD05 JOB (78030000), 'FOO',
//    REGION=4096K
//JOBLIB DD DISP=SHR, DSN=STV.TH30APP.APP.L
```
Appendix B: Invocation Examples

MVS

//         DD DISP=SHR,DSN=TER2.SASC550C.LINKLIB
//MLOADRUN     EXEC PGM=MLOAD,PARM='BRIEF'
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  DATA,DLM=##
  .LOGTABLE MKB_TABLE5;
  .LOGON TDP9/MKB,
  .VERSION;
  LOGOFF;
##

Sample Output:

000081 ************************************************************
000082
000083 **** 14:23:13 UTY2414 BRIEF option is enabled.
000084 **************************************************************
000085 MultiLoad Utility    Release MLOD.12.00.00.000  =
000086 Platform MVS                                                =
000087 **************************************************************
000088 Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
000089 **************************************************************
000090 14:23:13 UTY2411 Processing start date: THU NOVEMBER 30, 2006
000091 **************************************************************
000092 = Logon/Connection =
000093 =----------------------------------------------------------------------
000094 0001 .LOGTABLE MKB_TABLE5;
000095 0002 .LOGON TDP9/MKB,;
000097 **** 14:23:46 UTY8400 Teradata Database Release: 12.00.00.00
000098 **** 14:23:46 UTY8400 Teradata Database Version: 12.00.00.00
000096 **** 14:23:46 UTY8400 Default character set: EBCDIC
000099 **** 14:23:46 UTY6211 A successful connect was made to the DBS.
000110 **** 14:23:46 UTY6217 Logtable 'MKB.MKB_TABLE5' has been created.
000113 0003 .VERSION;


Using the Character Set Selection (CHARSET) Parameter

Sample JCL:

//MKBMLD15 JOB (78030000), 'FOO',
//             REGION=4096K
//JOBLIB      DD DISP=SHR,DSN=STV.TH30APP.APP.L
//MLOADRUN    EXEC PGM=MLOAD,PARM='CHARSET=EBCDIC'
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  DATA,DLM=##
  .LOGTABLE MKB_TABLE5;
  .LOGON TDP9/MKB,
  .VERSION;
  LOGOFF;
##

Sample Output:

000083 **** 15:43:31 UTY2407 Run time parameters in effect are: CHARSET=EBCDIC.
000084 ===============================================================
000085
Using the Error Logging (ERRLOG) Parameter

Sample JCL:

```
//MKBMLD16 JOB (78030000),'FOO',
//   REGION=4096K
//JOBLIB DD DISP=SHR,DSN=STV.TH30APP.APP.L
//   DD DISP=SHR,DSN=TER2.SASC550C.LINKLIB
//MLOADRUN EXEC PGM=MLOAD,PARM='ERRLOG=FOO'
//FOO DD DSN=MKB.FOO.OUTPUT,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD DATA,DLM=##
  .LOGTABLE MKB_TABLE5;
  .LOGON TDP9/MKB,MKB;
  .VERSION;
  .LOGOFF;
##
```

Sample Output:

```
000085  ***************************************************************
000086 000087 **** 15:19:47 UTY2413 Error Logging is enabled: FOO
000088 000089 = =
000090 = MultiLoad Utility Release MLOD.12.00.00.000 =
000091 = Platform MVS =
```
Specifying Multiple Parameters

Sample JCL:

```plaintext
//MKBMLD17 JOB (78030000), 'FOO',
REGION=4096K
//JOBLIB DD DISP=SHR, DSN=STV.TH30APP.APP.L
// DD DISP=SHR, DSN=TER2.SASC550C.LINKLIB
//MLOADRUN EXEC PGM=MLOAD, PARM='BRIEF, CHARSET=EBCDIC'
//SYSPRINT DD SYSOUT=A
//SYSIN DD DATA, DLM=##
.LOGTABLE MKB_TABLE5;
.LOGON TDP9/MKB,MKB;
.VERSION;
.LOGOFF;
##
```

Sample Output:

```plaintext
00083 **** 15:49:40 UTY2414 BRIEF option is enabled.
00084 **** 15:49:40 UTY2407 Run time parameters in effect are: CHARSET= EBCDIC.
00085 ==============================================================
00086 = Multiloading Utility Release MLOD.12.00.00.000 =
00087 = Platform MVS =
00088 ==============================================================
00089 = Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
00090 ==============================================================
00091 **** 15:49:40 UTY2411 Processing start date: THU NOVEMBER 30, 2006
```
Appendix B: Invocation Examples

UNIX and Windows

This section provides program JCL and command examples for invoking MultiLoad on UNIX and Windows systems.

Using the Reduced Print Output (-b) Parameter

**Sample Command:**

mload -b < foo2

**Sample Output:**

```plaintext
**** 15:13:01 UTY2414 BRIEF option is enabled.
========================================================================
=          MultiLoad Utility    Release MLOD.12.00.00.000  =
=          Platform MP-RAS                                             =
=          Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED.  =
========================================================================
**** 15:13:01 UTY2411 Processing start date: THU NOVEMBER 30, 2006
========================================================================
= Logon/Connection =
========================================================================
0001  .run file
      logon;
0002  .logon shogun/mkb,;
**** 15:13:01 UTY6214 Reminder: A .Logtable statement must be entered for a
successful logon.
0003  .logtable mkbtest2;
**** 15:13:46 UTY8400 Teradata Database Release: 12.00.00.00
**** 15:13:46 UTY8400 Teradata Database Version: 12.00.00.00
**** 15:13:46 UTY8400 Default character set: KANJIEUC_0U
001001 0003 .VERSION;
**** 15:13:08 UTY6211 A successful connect was made to the DBS.
**** 15:13:08 UTY6217 Logtable 'MKB.mkbtest2' has been created.
001001 0004 drop table wt_foo;
**** 15:13:08 UTY1008 DBS failure: 3807, Table/view 'wt_foo' does not exist.
```
Appendix B: Invocation Examples
UNIX and Windows

0005 drop table et_foo;
**** 15:13:08 UTY1008 DBS failure: 3807, Table/view 'et_foo' does not exist.
0006 drop table uv_foo;
**** 15:13:09 UTY1008 DBS failure: 3807, Table/view 'uv_foo' does not exist.
0007 drop table foo;
**** 15:13:10 UTY1016 'DROP' request successful.

0008 create table foo,fallback as
    f1 (char(3)),
    f2 (char(3)),
    f3 (varchar(3))
;
**** 15:13:10 UTY1016 'CREATE' request successful.

0009 .begin mload tables foo sessions 4;
====================================================================
= Processing MultiLoad Statements =
====================================================================

0010 .layout lay1;
0011 .field f1 * char(3);
0012 .field f2 * char(3);
0013 .field f3 * varchar(3);
0014 .dml label dml1;
0015 insert into foo values (:f1,:f2,:f3);
0016 .import infile foo.sun
    layout lay1
    apply dml1;

0017 .end mload;
====================================================================
= MultiLoad Initial Phase =
====================================================================

**** 15:13:11 UTY0829 Options in effect for this MultiLoad import task:
.       Sessions:    4 session(s).
.       Checkpoint:  15 minute(s).
.       Tenacity:    4 hour limit to successfully connect load sessions.
.       Errlimit:    No limit in effect.
.       AmpCheck:    In effect for apply phase transitions.

**** 15:13:20 UTY0812 MLOAD session(s) requested: 4.
**** 15:13:20 UTY0815 MLOAD session(s) connected: 4.
====================================================================
= MultiLoad Acquisition Phase =
====================================================================

**** 15:13:25 UTY0826 A checkpoint has been taken, recording that end of file has
been reached for IMPORT 1 of this MultiLoad Import task.
**** 15:13:25 UTY1803 Import processing statistics
.       IMPORT 1  Total thus far
.       =========  ==============
Candidate records considered:........ 2....... 2
Apply conditions satisfied:......... 2....... 2
Candidate records not applied:...... 0....... 0
Candidate records rejected:......... 0....... 0
====================================================================
= MultiLoad Application Phase =
====================================================================

**** 15:13:26 UTY0818 Statistics for table foo:
    Inserts:    2
    Updates:    0
Using the Character Set (-c) Selection Parameter

Sample Command:

mload -c ascii < foo2

Sample Output:

**** 15:27:35 UTY2407 Run time parameters in effect are: ASCII.
========================================================================
= =
= MultiLoad Utility    Release MLOD.12.00.00.000 =
= Platform MP-RAS =
= =
= Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
= =
========================================================================
**** 10:27:35 UTY2411 Processing start date: THU NOVEMBER 30, 2006
========================================================================
= =
= Logon/Connection =
= =
========================================================================
0001 .run file
logon;
0002 .logon shogun/mkb,;
**** 15:27:35 UTY6214 Reminder: A .Logtable statement must be entered for a
successful logon.
========================================================================
= = Processing Control Statements = =
========================================================================
0003 .logtable mkbtest2;
**** 15:27:39 UTY6211 A successful connect was made to the DBS.
**** 15:27:39 UTY6217 Logtable 'MKB.mkbtest2' has been created.
.
.

Using the Error Logging (-e) Parameter

Sample Command:

mload -e errfile < foo2

Sample Output:

**** 15:33:10 UTY2413 Error Logging is enabled: errfile
========================================================================
= = MultiLoad Utility Release MLOD.12.00.00.000 =
= = Platform MP-RAS = =
========================================================================
= = Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED. =
= =
========================================================================
**** 10:33:10 UTY2411 Processing start date: THU NOVEMBER 30, 2006
========================================================================
= = Logon/Connection = =
========================================================================
0001 .run file
  logon;
0002 .logon shogun/mkb,;
**** 15:33:10 UTY6214 Reminder: A .Logtable statement must be entered for a
successful logon.
========================================================================
= = Processing Control Statements = =
========================================================================
0003 .logtable mkbtest2;
.
.
Using the Runfile (-r) Parameter

The typical option selection is:

```
mload -r '.RUN FILE mld.startup;' < infilename > outfilename
```

**Sample Logon Contents:**

```
.logtable mloadlog;
.logon 15440/sfd,sfd;
```

**Sample MLOAD Job:**

```
drop table wt_foo;
drop table et_foo;
drop table uv_foo;
drop table foo;
create table foo (  
    f1 char(3),  
    f2 char(3),  
    f3 varchar(3)  
)
;
.begin mload tables foo sessions 4;
.layout lay1;
.field f1 * char(3);
.field f2 * char(3);
.field f3 * varchar(3);
.dml label dml1;
.insert foo (f1) values (:f1);
.import infile foo.data
.layout lay1
.apply dml1;
.end mload;
.logoff;
```

**Sample Command:**

```
$ mload -r ".run file logon;" < job.mload
```

**Sample Output:**

```
========================================================================
==            MultiLoad Utility    Release MLOD.12.00.00.000    ==
==            Platform MP-RAS                                              ==
========================================================================
==
==          Copyright 1990-2007, NCR Corporation. ALL RIGHTS RESERVED.  ==
==
========================================================================
**** 10:14:12 UTY2411 Processing start date: THU NOVEMBER 30, 2006
========================================================================
==
==          Logon/Connection                                              ==
==
========================================================================
0001 .run file logon;
0002 .logtable mloadlog;
```
Appendix B: Invocation Examples

UNIX and Windows

Specifying Multiple Parameters

Sample Command:

mload -c ascii -b < foo2

Sample Output:

**** 15:38:31 UTY2407 Run time parameters in effect are: ASCII.
**** 15:38:31 UTY2414 BRIEF option is enabled.
Appendix B: Invocation Examples
UNIX and Windows

*** 15:38:34 UTY6217 Logtable 'MKB.mkbtest2' has been created.
0004 drop table wt_foo;
*** 15:38:34 UTY1008 DBS failure: 3807, Table/view 'wt_foo' does not exist.
0005 drop table et_foo;
*** 15:38:35 UTY1008 DBS failure: 3807, Table/view 'et_foo' does not exist.
0006 drop table uv_foo;
*** 15:38:35 UTY1008 DBS failure: 3807, Table/view 'uv_foo' does not exist.
0007 drop table foo;
*** 15:38:36 UTY1016 'DROP' request successful.
0008 create table foo,fallback as
    f1 (char(3)),
    f2 (char(3)),
    f3 (varchar(3))

. . .
INMOD and Notify Exit Routine Examples

This appendix provides program listings of sample INMOD and notify exit routines on the following client platforms:

- VM and MVS
- UNIX
- Windows

Listings are provided of sample INMOD routines and notify exit routines written in:

- Assembler
- C
- COBOL
- PL/I

Channel-attached VM and MVS client systems support INMOD and notify exit routines written in any of the languages listed. Network-attached UNIX and Windows client systems support routines written in C.

VM and MVS

Note: The listings in this topic have MVS control statements. To use them on VM systems, change the control statements to reflect the proper operating system.

Assembler INMOD Example

This INMOD example obtains records from the Teradata Database that can be used to insert, update, or delete rows of a target table.

```
//JCKAS1 JOB 1,'JAMES KIM',MSGCLASS=A,NOTIFY=JCK,CLASS=B,
REGION=4096K
//*******************************************************************************
//**                                                                             *
//** IDENTIFY NECESSARY LOAD LIBRARIES FOR RELEASE 4.1                          *
//**                                                                             *
//*******************************************************************************
//JOBLIB   DD DISP=SHR,DSN=STV.GG10.APP.L
```
// DD DISP=SHR, DSN=STV.GG00.APP.L
// DD DISP=SHR, DSN=STV.TG00.APP.L
// DD DISP=SHR, DSN=STV.RG00.APP.L
// DD DISP=SHR, DSN=TER2.SASC301H.LINKLIB
// ASMFCL EXEC ASMFCL
// ASMSYSIN DD *

DYNAMN TITLE 'CONCATENATE INPUT RECORDS FOR INPUT TO MULTILOAD'
DYNAMN CSECT

******************************************************************
* THIS PROGRAM IS CALLED BY THE TERADATA MULTILOAD PROGRAM *
* TO OBTAIN A RECORD TO BE USED TO INSERT, UPDATE, OR DELETE *
* ROWS OF A TARGET TABLE *
* *
* THIS PROGRAM IS NOT REENTRANT *
* *
* FUNCTION: *
* READ AN INPUT RECORD AND ADD A FOUR-BYTE INTEGER FIELD *
* TO THE FRONT OF THE RECORD. THE NEW FIELD WILL *
* CONTAIN A SEQUENCE NUMBER THAT RANGES FROM 1 TO ... *
* NUMBER-OF-INPUT-RECRODS. *
* *
* RETURN TO THE CALLER (MULTILOAD) INDICATING *
* THAT MORE RECORDS ARE AVAILABLE OR NO MORE RECORDS *
* ARE TO BE PROCESSED. *
* *
* THIS INMOD PROGRAM CAN BE USED TO ENSURE UNIQUE RECORDS *
* IN CERTAIN APPLICATIONS, THE SEQUENCE FIELD *
* CAN BE USED FOR "DATA SAMPLING". *
* *
* DDNAME OF THE INPUT DATA SET: "INDATA"
******************************************************************
B STOREGS BRANCH AROUND EP
DC AL1(31) DEFINE EP LENGTH
DC CL9'DYNAMN' DEFINE
DC CL9'&SYSDATE' ENTRY
DC CL8' VM' POINT
DC CL5'&SYSTIME' IDENTIFIER
******************************************************************
* SAVE REGISTERS *
******************************************************************
STOREGS DS 0H DEFINE AND ALIGN SYMBOL
STM R14,R12,12(R13) STORE OFF CALLER'S REGISTERS
LR R12,R15 COPY BASE ADDRESS
DROP R15 DROP VOLATILE BASE REGISTER
USING DYNAMN,R12 ESTAB PERM CSECT ADDRBLTY
LA R14,SSAVEAREA POINT AT LOCAL SAVE WORK
ST R14,8,(R13) STORE FWD LINK IN SA CHAIN
ST R13,4,(R14) STORE BWD LINK IN SA CHAIN
LR R13,R14 COPY LOCAL SAVE/WORK AREA ADDR
L R11,0,(R1) POINT TO PARM
SPACE 1
******************************************************************
* OPEN "DATA" DATA SET *
* (ONLY THE FIRST TIME) *
******************************************************************
USING PREBUF,R11 COVER PRE-PROC AREA
LA R9,PREREC POINT TO START OF PREPROC. DATA
Appendix C: INMOD and Notify Exit Routine Examples

VM and MVS

OC PRECODE,PRECODE FIRST ENTRY ?
   (0=FIRST ENTRY)
BNZ NOOPEN NO, SKIP OPEN
USING IHADCB,R10 YES,COVER DCB FOR OPEN
LA R10,INDATA POINT TO DATA DCB
OPEN INDATA OPEN INPUT DATA SET
TM DCBFLGS,X'10' DID IT OPEN ?
BO OPENOK YES,
WTO 'UNABLE TO OPEN INDATA DATA SET',ROUTCDE=11
B BADRET RETURN WITH ERROR CODE
******************************************************************
* CHECK MULTILOAD STATUS CODES *
* 0 = FIRST ENTRY (MULTILOAD EXPECTS TO RECEIVE A RECORD) *
* 1 = GET NEXT RECORD MULTILOAD EXPECTS TO RECEIVE A RECORD) *
* 2 = CLIENT RESTART CALL (MULTILOAD DOES NOT EXPECT A RECORD) *
* 3 = CHECKPOINT CALL (MULTILOAD DOES NOT EXPECT A RECORD) *
* 4 = RESTART CALL (MULTILOAD DOES NOT EXPECT A RECORD) *
* 5 = CLOSE INMOD (MULTILOAD DOES NOT EXPECT A RECORD) *
* *
* NOTE: CODES 2, 3 AND 4 ARE NOT HANDLED BY THIS PROGRAM *
******************************************************************
OPENOK DS 0H
NOOPEN L R15,PACODE CHECK ON CODE FROM MULTILOAD
   C R15,=F'1' NEED RECORD ?
   BH NOREC NO , DO NOT "GET" A RECORD
   L R15,SAMPNUM GET CURRENT SAMPLE NUM.
   LA R15,1(R15) INCR BY 1
   ST R15,0(R9) STORE AT FRONT OF RECORD
   ST R15,SAMPNUM RESET COUNTER
   LA R9,4(R9) ADVANCE FOR READ ADDR.
   LA R10,INDATA COVER INDATA DCB
GETNEXT GET INDATA,(R9) READ A RECORD
INCREC LH R9,DCBLRECL GET RECORD LENGTH
   AH R9,=H'4' ADD 4 FOR NEW FIELD
   SR R15,R15 SET RETURN CODE VALUE
RETURN ST R9,PRELEN SET LENGTH (ZERO AFTER EOF)
   ST R15,PACODE
   L R13,4(R13)
RETURN (14,12),RC=0
RETURN
SPACE 5
******************************************************************
* EOF ENTERED AT END-OF-FILE *
******************************************************************
EOF CLOSE INDATA, CLOSE INPUT DATA SET
******************************************************************
* CONSTANTS *
******************************************************************
* REGEQU
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
EJECT
*
* DATA STRUCTURES AND VARIABLES
*
SPACE 1
SAVEAREA DC 9D'0' SAVE AREA
SAMPNUM DC F'0'
SPACE 10
INDATA DCB DDNAME=INDATA,MACRF=(GM),DSORG=PS,EODAD=EOF
PREBUF DSECT
PRECODE DS F
PRELEN DS F
PREREC DS 0XL31000
DCBD DEVD=DA,DSORG=PS
PREPRM DSECT
PRESEQ DS F
PREPRML DS H
PREPRMS DS CL80
END
//LKED.SYSLMOD DD DSN=JCK.INMOD.LOAD(INMODG1),DISP=MOD,UNIT=3380,
// VOLUME=SER=TSO805
//LKED.SYSIN DD *
ENTRY DYNAMN
NAME INMODG1(R)
/*
//MLOADDEL EXEC PGM=IEFBR14
//MLOADLOG DD DSN=JCK.INMOD.TDQ8.MLOADLOG,
// DISP=(MOD,DELETE),UNIT=SYSDA,SPACE=(TRK,0)
//MLOADCAT EXEC PGM=BLKMAIN
//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR
// DD DSN=STV.TG00.APP.L,DISP=SHR
// DD DSN=STV.RG00.APP.L,DISP=SHR
//SYSPRINT DD SYSOUT=* 
//MLOADLOG DD DSN=JCK.INMOD.TDQ8.MLOADLOG,DISP=(NEW,CATLG),
// UNIT=SYSDA,DCB=(RECFM=F,DSORG=PS,LRECL=8244),
// SPACE=(8244,(12,5))
//SYSDP DD *
//******************************************************************************
//** THIS STEP WILL ONLY DROP THE TABLES                                      *
//** IF MLOAD IS NOT IN APPLY PHASE                                         *
//******************************************************************************
//CREATE EXEC BTEQ
//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR
// DD DSN=STV.TG00.APP.L,DISP=SHR
Appendix C: INMOD and Notify Exit Routine Examples

VM and MVS

Teradata MultiLoad Reference 199

// DD DSN=STV.RG00.APP.L,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSENV DD SYSOUT=* 
//SYSIN DD DATA,DLM=## 
.LOGON TDQ8/DBC,DBC; 
RELEASE MLOAD XXXX.INMODAS1;
.DE ERRORCODE = 2572 THEN .GOTO NODROP;
DROP TABLE XXXX.LOGTABLE;
DROP TABLE XXXX.ET_INMODAS1;
DROP TABLE XXXX.UV_INMODAS1;
DROP TABLE XXXX.WT_INMODAS1;
.QUIT;
.LABEL NODROP;
.EXIT 4;
##
//********************************************************************
//* *
//* RUN MULTILOAD *
//* *
//********************************************************************
//LOADIT EXEC PGM=MLOAD
//STEPLIB DD DISP=SHR,DSN=STV.GG10.APP.L
// DD DISP=SHR,DSN=STV.GG00.APP.L
// DD DISP=SHR,DSN=STV.TG00.APP.L
// DD DISP=SHR,DSN=STV.RG00.APP.L
// DD DISP=SHR,DSN=TER2.SASC301H.LINKLIB
// DD DISP=SHR,DSN=JCK.INMOD.LOAD,VOLUME=SER=TSO805,UNIT=3380 
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//SYSOUT DD SYSOUT=* 
//SYSIN DD DATA,DLM=## 
.LOGON TDQ8/XXXX,XXXX; 
/* TEST DATAIN, DATALOC */
DROP TABLE XXXX.INMODAS1;
CREATE TABLE INMODAS1 (F1 CHAR(10), F2 CHAR(70)); 
.BEGIN IMPORT MLOAD TABLES INMODAS1; 
.Layout layname1;
.FIELD L1FLD0 1 CHAR(4); 
.FIELD L1FLD1 * CHAR(10); 
.Field L1Fld2 * Char(70); 
.DML Label DML1; 
INSERT INMODAS1(F1,F2) VALUES (:L1FLD1, :L1FLD2); 
.IMPORT INMOD INMODG1 USING ("AAA" "BBB") LAYOUT LAYNAME1 APPLY DML1; 
.End Mload; 
.LOGOFF;
##
//INDATA DD DATA,DLM=## 
01ASSEMBLEAAAAAAAAAAAAAAAA 
02ASSEMBLEBBBBBBBBBBBBBBBBB 
03ASSEMBLECCCCCCCCCCCCCCCCC 
04ASSEMBLEDDDDDDDDDDDDDDDD 
##
//SELECT EXEC BTEQ 
//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR 
// DD DSN=STV.TG00.APP.L,DISP=SHR 
// DD DSN=STV.RG00.APP.L,DISP=SHR 
//SYSPRINT DD SYSOUT=A 
//SYSENV DD SYSOUT=*
Appendix C: INMOD and Notify Exit Routine Examples
VM and MVS

C INMOD Example

This INMOD example reads the function code and executes different processing functions based on its value.

#include <stddef.h>

//SYSIN DD DATA,DLM=##
.LOGON TDQ8/XXXX,XXXX;
SELECT * FROM INMODAS1;
.LOGOFF;
##
//

This program is for release 4.1 MULTILOAD INMOD testing using C user exit routine.

When this routine is activated it looks at the content of the function code passed (a->code) and depending on its value, it
0) initializes, i.e., opens a file, etc...
1) reads a record
5) acknowledges "close inmod" request. The user exit routine
must return "return code"(a->code) and "length" (a->len). You
should send return code = zero when no errors occur and non-zero for
an error. MULTILOAD expects length = zero at the end of
file. Then it sends "CLOSE INMOD" request. THE USER EXIT routine
must explicitly return "return code" = ZERO to terminate the
conversation. */
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
typedef unsigned short    Int16;
typedef unsigned char    Int8;
typedef unsigned long int Int32;

/* PASSING parameter structures */

typedef struct {
    Int32 code;
    Int32 len;
    Int8  buf[80];
} inmodbuf;

typedef struct {
    Int32 seq;
    Int16 len;
    char  param[80];
} inmodpty;

static FILE *IN;
static int count=0;
char *memcpy();

void _dynamn(a,b)
inmodbuf *a;
inmodpty *b;
{int code=0;
 char tempbuf[80];

memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]='$';
printf("BEGIN--> %d %d %s
",a->code,a->len,tempbuf);
printf(" +++ %d %d %s
",b->seq ,b->len,b->param);

code= (int) a->code;
switch (code) {
    case 0:
        /* Here you open the file and read the first record */
        printf("## CODE=0, opening...
");
        IN=fopen("ddn:INDATA","rb");
        if (! ferror(IN)) {
            if (! readrecord(a))
                fclose(IN);
        };
        break;
    case 1:
        /* MultiLoad requested next record, read it */
        printf("## CODE=1, reading...
");
        if (! readrecord(a))
            fclose(IN);
        break;
    case 5:
        /* MultiLoad is closing INMOD routine */
        a->code=0;
        a->len=0;
        printf("## CODE=5, terminating...
");
        break;
    default:
```c
a->code=12; /* any number not = to zero */
a->len=0;
printf("##### UNKNOWN code ######\n");a->code=0;a->len=0;
}

memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]=\'\0\';
printf("END --> %d %d %s\n",a->code,a->len,tempbuf);
printf(" +++ %d %d %s\n",b->seq ,b->len,b->param);
}

int readrecord(a)
inmodbuf *a;
{
int rtn=0;
char tempbuf[80];

if (fread((char *)&(a->buf),sizeof(a->buf),1,IN)) {
    count++;
    memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]=\'\0\';
    printf(" %d %s \n",count,tempbuf);
a->len=80;
a->code=0;
rtn=1;
};
if ferror(IN) {
    printf("==== error ====
");
a->code=16; /* any non zero number */
a->len=0;
};
if feof(IN) { /* EOF, set length = zero */
    printf("=== EOF ===\n");
a->code=9;
a->len=9;
};
return(rtn);
}
```
Appendix C: INMOD and Notify Exit Routine Examples

Teradata MultiLoad Reference

VM and MVS

//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR
// DD DSN=STV.TG00.APP.L,DISP=SHR
// DD DSN=STV.RG00.APP.L,DISP=SHR

//SYSPRINT DD SYSOUT=*00171 //BDLLOG DD
DSN=JCK.INMOD.TDQ8.BDLLOG,DISP=(NEW,CATLG),
UNIT=SYSDA,DCB=(RECFM=F,DSORG=PS,LRECL=8244),
SPACE=(8244,(12,5))
//SYSIN DD *
 jihadists //************************************************************************* //**THIS STEP WILL ONLY DROP THE TABLES IF MLOAD NOT IN APPLY PHASE** //************************************************************************* //CREATE EXEC BTEQ
//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR
// DD DSN=STV.TG00.APP.L,DISP=SHR
// DD DSN=STV.RG00.APP.L,DISP=SHR
//SYSPRINT DD SYSOUT=* //SYSABEND DD SYSOUT=* //SYSIN DD DATA,DLM=##
.LOGON TDQ8/DBC,DBC;
DROP TABLE XXXX.LOGTABLE;
DROP TABLE XXXX.ET_INMODLC1;
DROP TABLE XXXX.UV_INMODLC1;
DROP TABLE XXXX.WT_INMODLC1;
.QUIT;
.LABEL NODROP;
.EXIT 4;
## //************************************************************************** //** RUN MULTILOAD ** //************************************************************************** //LOADIT EXEC PGM=MLOAD
//STEPLIB DD DISP=SHR,DSN=STV.GG10.APP.L
// DD DISP=SHR,DSN=STV.GG00.APP.L
// DD DISP=SHR,DSN=STV.TG00.APP.L
// DD DISP=SHR,DSN=STV.RG00.APP.L
// DD DISP=SHR,DSN=TER2.SASC301H.LINKLIB
// DD DISP=SHR,DSN=JCK.INMOD.LOAD,VOLUME=SER=TSO805,
// UNIT=338
//SYSPRINT DD SYSOUT=* //SYSTEM DD SYSOUT=* //SYSOUT DD SYSOUT=* //SYSIN DD DATA,DLM=##
.LOGTABLE XXXX.LOGTABLE;
.LOGON TDQ8/XXXX,XXXX; /* TEST DATAIN, DATALOC */
DROP TABLE XXXX.INMODLC1;
CREATE TABLE INMODLC1 (F1 CHAR(10), F2 CHAR(70));
.BEGIN IMPORT MLOAD TABLES INMODLC1;
.Layout layname1;
.Field L1Fld1 1 Char(10);
.Field L1Fld2 * Char(70);
.DML Label DML1;
INSERT INMODLC1(F1,F2) VALUES (:L1FLD1, :L1FLD2);
.IMPORT INMOD INMODG1 USING ("AAA" "BBB") LAYOUT LAYNAME1 APPLY DML1
; .End Mload;
COBOL INMOD Example

This INMOD example reads and returns an 80-byte record string.
**INMOD and Notify Exit Routine Examples**

**VM and MVS**

Teradata MultiLoad Reference 205

---

*MLOAD COMMUNICATES WITH INMOD VIA STRUCT-1 AND STRUCT-2.*

**01 STRUCT-1.**

- **02 RETURN-INDICATE** PIC S9(9) COMP.
- **02 RECORD-LEN** PIC S9(9) COMP.
- **02 RECORD-BODY**.
- **03 DATA-AREA1** PIC X(80).

**01 STRUCT-2.**

- **02 SEQ-NUMBER** PIC S9(9) COMP.
- **02 PARM-LIST**.
  - **05 PARM-LENTH** PIC X(2).
  - **05 PARM-STRING** PIC X(80).

**PROCEDURE DIVISION USING STRUCT-1, STRUCT-2.**

**BEGIN.**

**MAIN.**

- DISPLAY "==============================================" 
- DISPLAY STRUCT-1.
- DISPLAY STRUCT-2.

**IF RETURN-INDICATE = 0 THEN**

- *INMOD INITIALIZATION - OPEN FILE AND READ THE 1ST REC.*
- DISPLAY "INMOD CALLED - RETURN CODE 0"
- **PERFORM OPEN-FILES**
- **PERFORM READ-RECORDS**
- **GOBACK**

**ELSE**

**IF RETURN-INDICATE = 1 THEN**

- *READ A RECORD.*
- DISPLAY "INMOD CALLED - RETURN CODE 1"
- **PERFORM READ-RECORDS**
- **GOBACK**

**ELSE**

**IF RETURN-INDICATE = 5 THEN**

- *CLOSE INMOD - JUST SEND RETURN CODE = 0*
- DISPLAY "INMOD CALLED - RETURN CODE 5"
- MOVE 0 TO RECORD-LEN
- MOVE 0 TO RETURN-INDICATE
- **GOBACK**

**ELSE**

- **UNKNOWN CODE.**
- DISPLAY "INMOD CALLED - RETURN CODE X"
- MOVE 0 TO RECORD-LEN
- MOVE 16 TO RETURN-INDICATE
- **GOBACK.**

**OPEN-FILES.**

- OPEN INPUT INMOD-DATA-FILE.
- MOVE 0 TO RETURN-INDICATE.

**READ-RECORDS.**

- **READ INMOD-DATA-FILE INTO DATA-AREA1**
- **AT END GO TO END-DATA.**
- ADD 1 TO NUMIN.
- MOVE 80 TO RECORD-LEN.
- MOVE 0 TO RETURN-INDICATE.
- ADD 1 TO NUMOUT.

**END-DATA.**

- **CLOSE INMOD-DATA-FILE.**
- DISPLAY "NUMBER OF INPUT RECORDS = " NUMIN.
- DISPLAY "NUMBER OF OUTPUT RECORDS = " NUMOUT.
- MOVE 0 TO RECORD-LEN.
- MOVE 0 TO RETURN-INDICATE.
- **GOBACK.**
Appendix C: INMOD and Notify Exit Routine Examples
VM and MVS

/*
//LKED.SYSLMOD DD DSN=JCK.INMOD.LOAD(INMODG1),DISP=MOD
//LKED.SYSIN DD *
ENTRY DYNAMN
NAME INMODG1(R)
*/

//*****************************************************************
//* NEXT 3 STEPS PREPARE TERADATA DBS FOR THE MULTILOAD'S INMOD *
//* TEST. *
//*****************************************************************
//MLOADDEL EXEC PGM=IEFBR14
//MLOADLOG DD DSN=JCK.INMOD.TDQ8.MLOADLOG,
// Disp=(MOD,DELETE),UNIT=SYSDA,SPACE=(TRK,0)
//MLOADCAT EXEC PGM=BLKMAIN
//SYSPRINT DD SYSOUT=* 
//MLOADLOG DD DSN=JCK.INMOD.TDQ8.MLOADLOG,Disp=(NEW,CATLG),
// UNIT=SYSDA,Dcb=(RECFM=F,DSORG=PS,LRECL=8244),
// SPACE=(8244,(12,5))
//SYSLIB DD *

//*****************************************************************
//* THIS STEP WILL ONLY DROP THE TABLES IF MLOAD NOT IN APPLY *
//* PHASE. *
//*****************************************************************
//CREATE EXEC BTEQ
//STEPLIB DD DSN=STV.GG00.APP.L,Disp=SHR
// DD DSN=STV.TG00.APP.L,Disp=SHR
// DD DSN=STV.RG00.APP.L,Disp=SHR
//SYSPRINT DD SYSOUT=* 
//SYSABEND DD SYSOUT=* 
//SYSLIB DD DATA,DLM=##
..LOGON TDQ8/DBC,DBC;
RELEASE MLOAD XXXX.INMODCB1;
..IF ERRORCODE = 2572 THEN .GOTO NODROP;
DROP TABLE XXXX.LOGTABLE;
DROP TABLE XXXX.ET_INMODCB1;
DROP TABLE XXXX.UV_INMODCB1;
DROP TABLE XXXX.WT_INMODCB1
..QUIT;
..LABEL NODROP;
..EXIT 4;;
DROP USER XXXX;
##

//*****************************************************************
//* RUN MULTILOAD *
//*****************************************************************
//LOADIT EXEC PGM=MLOAD
//STEPLIB DD Disp=SHR,DSN=JCK.INMOD.LOAD
//SYSPRINT DD SYSOUT=* 
//SYSTEM DD SYSOUT=* 
//SYSOUT DD SYSOUT=* 
//SYSLIB DD DATA,DLM=##
..LOGTABLE XXXX.LOGTABLE;
..LOGON TDQ8/XXXX,XXXX;
/* TEST DATAIN, DATALOC */
DROP TABLE XXXX.INMODCB1;
CREATE TABLE INMODCB1 (F1 CHAR(10), F2 CHAR(70));
Appendix C: INMOD and Notify Exit Routine Examples

VM and MVS

BEGIN IMPORT MLOAD TABLES INMODCB1;
.Layout layname1;
.Field L1Fld1 1 Char(10);
.Field L1Fld2 * Char(70);
.DML Label DML1;
INSERT INMODCB1(F1,F2) VALUES (:L1FLD1, :L1FLD2);
.IIMPORT INMOD INMODG1 USING ("AAA" "BBB") LAYOUT LAYNAME1 APPLY DML1;
.End Mload;
.LOGOFF;
#
//INDATA DD DATA,DLM=##
01COBOL1 AAAAAAAAAAAAAAAA
02COBOL1 BBBBBBBBBBBBBBBB
03COBOL1 CCCCCCCCCCCCCCCC
04COBOL1 DDDDDDDDDDDDDDD
#
//SELECT EXEC BTEQ
//STEPLIB DD DSN=STV.GG00.APP.L,DISP=SHR
// DD DSN=STV.TG00.APP.L,DISP=SHR
// DD DSN=STV.RG00.APP.L,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSIN DD DATA,DLM=##
.LOGON TDQ8/XXXX,XXXX;
SELECT * FROM INMODCB1;
.LOGOFF;
#
/

COBOL Pass Thru INMOD Example

This INMOD example receives a record from MultiLoad and then either modifies or rejects it.

IDENTIFICATION DIVISION.
PROGRAM-ID. INMOD2.
AUTHOR. STV.
INSTALLATION. TERADATA.
DATE-WRITTEN. 18 SEPTEMBER 1989.
DATE-COMPILED.
SECURITY. OPEN.
REMARKS.

THIS PROGRAM IS AN EXAMPLE OF A COBOL INMOD ROUTINE
WHICH RECEIVES A RECORD FROM MLOAD THEN MODIFIES OR
REJECTS IT.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STOREAGE SECTION.
 01 COUNTRows PICTURE S9(4) COMP VALUE +0.
 01 REJROWS PICTURE S9(4) COMP VALUE +0.
 01 INSROWS PICTURE S9(4) COMP VALUE +0.
 01 I PICTURE S9(4) COMP.
 01 MATCHFLAG PIC 9.
     88 NOTMATCH VALUE 0.
88 MATCH VALUE 1.

LINKAGE SECTION.

01 STRUCT-1.
   02 RETURN-INDICATE PIC S9(9) COMP.
   02 RECORD-LEN PIC S9(9) COMP.
   02 RECORD-BODY OCCURS 80 TIMES.
   03 DATA-AREA1 PIC X.

01 STRUCT-2.
   02 SEQ-NUMBER PIC S9(9) COMP.
   02 PARM-LIST.
      05 PARM-LENGTH PIC S9(4) COMP.
      05 PARM-STRING OCCURS 80 TIMES.
      07 PARM-DATA PIC X.

PROCEDURE DIVISION USING STRUCT-1, STRUCT-2.
BEGIN.
MAIN.
   DISPLAY "=============================================
   IF RETURN-INDICATE = 6 THEN
      DISPLAY "INMOD2 CALLED - RETURN CODE 6 "
      PERFORM INITIALIZE
      GOBACK
   ELSE
      IF RETURN-INDICATE = 7 THEN
         DISPLAY "INMOD2 CALLED - RETURN CODE 7 "
         PERFORM PROCESS-RECORD
         GOBACK
      ELSE
         IF RETURN-INDICATE = 5 THEN
            DISPLAY "INMOD2 CALLED - RETURN CODE 5 "
            PERFORM FINALIZE
            GOBACK
         ELSE
            DISPLAY "BLKEXIT CALLED - RETURN CODE X "
            MOVE 0 TO RETURN-INDICATE.
            GOBACK.
      INITIALIZE.
         MOVE 0 TO COUNTROWS INSROWS REJROWS.
         MOVE 0 TO RETURN-INDICATE.
      PROCESS-RECORD.
         ADD 1 TO COUNTROWS.
         MOVE 0 TO RETURN-INDICATE. MOVE 1 TO I.
         MOVE 1 TO MATCHFLAG.
         PERFORM COMPARE UNTIL (I > PARM-LENGTH) OR (NOTMATCH).
         IF NOTMATCH THEN
            DISPLAY "REJECTED"
            ADD 1 TO REJROWS
            MOVE 0 TO RECORD-LEN
         ELSE
            DISPLAY "ACCEPTED"
            ADD 1 TO INSROWS.
      COMPARE.
         IF (RECORD-BODY(I) = PARM-STRING(I)) THEN
            NEXT SENTENCE
         ELSE
            MOVE 0 TO MATCHFLAG.
            ADD 1 TO I.
      FINALIZE.
         MOVE 0 TO RETURN-INDICATE.
      DISPLAY "NUMBER OF TOTAL RECORDS = " COUNTROWS.
DISPLAY "NUMBER OF REJECTED RECORDS = " REJROWS.
DISPLAY "NUMBER OF ACCEPTED RECORDS = " INSROWS.
GOBACK.

**PL/I INMOD Example**

```
//MXM049A  JOB (22150000), 'MXM', MSGCLASS=A, NOTIFY=MXM
//STEP1  EXEC PGM=IEV90, REGION=1024K,
//   PARM='OBJ, NODECK, XREF(FULL)'
// SYSLIB   DD   DSN=SYS1.MACLIB, DISP=SHR
// DD   DSN=SYS1.AMODGEN, DISP=SHR
// SYSPRINT DD   SYSOUT=A
// SYSTERM  DD   SYSOUT=* 
// SYSSUT1 DD   UNIT=VIO, SPACE=(CYL, (9, 5))
// SYSSUT2 DD   UNIT=VIO, SPACE=(CYL, (9, 5))
// SYSSUT3 DD   UNIT=VIO, SPACE=(CYL, (9, 5))
// SYSSLIN DD   DSN=&PL1A, DISP=(, PASS), UNIT=SCR,
//   SPACE=(CYL, (1, 1)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=3200)
// SYSSIN DD   * 
// TITLE 'DYNAMN'
DYNAMN  CSECT
EXTRN PL1START
B     START-*(, R15)            BRANCH AROUND CONSTANTS
DC    AL1(L'PL1AFLAG)          LENGTH OF CONSTANTS
PL1AFLAG DC    C'ASSEMBLED AT &SYSTIME ON &SYSDATE.. PL1A'
DC    C' COPYRIGHT (C) 1999 NCR CORPORATION,'
DC    C' ALL RIGHTS RESERVED.'

*================================================================
*   ENTRY POINT
*================================================================
START    SAVE  (14, 12)
   LR    R12, R15            -> PROGRAM ENTRY POINT
   USING DYNAMN, R12
*   LA    R10, SAVAREA
   ST    R10, 8(R13)         FORWARD CHAIN
   ST    R13, 4(R10)         BACK CHAIN
   LR    R13, R10
*   LR    R4, R1            SAVE PARM LIST ADDRESS
   L     R3, 0(, R1)         -> COMMAND WORD
   L     R3, 0(, R3)         COMMAND WORD
   CH    R3, =H'0'           INITIAL CALL?
   BE    DO_INIT            YES, DO INITIAL CODE
   CH    R3, =H'6'           INITIAL CALL?
   BE    DO_INIT            YES, DO INITIAL CODE
   CH    R3, =H'2'           INITIAL CALL?
   BNE   DO_CALL            NO, JUST GO CALL PROGRAM

*================================================================
*   SETUP PL/I ENVIRONMENT
*================================================================
DO_INIT  DS   0H
*   WTO   'PRIOR TO INIT REQUEST'
```
MVC PRP_REQUEST,INIT     INDICATE THE INIT REQUEST

LA R1,EXEC_ADDR         GET THE PARM ADDR LIST
ST R1,EPL_EXEC_OPTS    SAVE IN EPL

LA R1,PARM_EPL         R1 --> POINTER --> REQUEST LIST
L R15,PSTART          PL/I ENTRY ADDR
BALR R14,R15           INVOKE PL/I

WTO 'AFTER INIT REQUEST'

CALL "OPTIONS( MAIN )" INMOD

DO_CALL DS 0H

WTO 'PRIOR TO CALL REQUEST'

MVC PRP_REQUEST,CALL    INDICATE THE CALL REQUEST

ST R4,EPL_PROG_PARMS   SAVE PARM ADDR IN EPL
LA R1,PARM_EPL         R1 --> POINTER --> REQUEST LIST
L R15,PSTART          PL/I ENTRY ADDR
BALR R14,R15           INVOKE PL/I

CH R3,=H'5'             FINAL CALL?
BNE DO_RTN             NO, JUST RETURN TO CALLER

TERMINATE THE PL/I ENVIRONMENT

DO_TERM DS 0H

ST R15,RETCODE         SAVE PL/I RETURN CODE

WTO 'PRIOR TO TERM REQUEST'

MVC PRP_REQUEST,TERM   INDICATE A TERM COMMAND

LA R1,0                 NO PARM LIST IS PRESENT
ST R1,EPL_PROG_PARMS   SAVE IN EPL

LA R1,PARM_EPL         R1 --> POINTER --> REQUEST LIST
L R15,PSTART          PL/I ENTRY ADDR
BALR R14,R15           INVOKE PL/I

WTO 'AFTER TERM REQUEST'

RETURN TO CALLER

DO_RTN DS 0H

L R13,SAVAREA+4
L R14,12(R13)
L R15,RETCODE
LM R0,R12,20(R13)
BR R14          RETURN TO YOUR CALLER
EJECT
EJECT

================================================================
* CONSTANTS AND WORKAREAS
*================================================================
SAVAREA DS 20F
RETCODE DC F'0'
PARAM_EPL DC A(X'80000000'+IBMBZPRP) PARAMETER ADDR LIST
PSTART DC A(PL1START)
*================================================================
* REQUEST STRINGS ALLOWED IN THE INTERFACE
*================================================================
INIT DC CL8'INIT' INITIALIZE THE PROGRAM ENVIR
CALL DC CL8'CALL' INVOKE THE APPL - LEAVE ENVIR UP
TERM DC CL8'TERM' TERMINATE ENVIRONMENT
EXEC DC CL8'EXECUTE' INIT, CALL, TERM - ALL IN ONE
*================================================================
* PARAMETER LIST PASSED BY A PRE-INITIALIZED PROGRAM
* ADDRESSED BY REG 1 = A(IBMBZPRP)
* SEE IBMZEPL DSECT.
*================================================================
IBMBZPRP DS 0F
PRP_LENGTH DC H'16' LEN OF THIS PRP PASSED (16)
PRP_ZERO DC H'0' MUST BE ZERO
PRP_REQUEST DC CL8'' 'INIT' - INITIALIZE PL/I
* 'CALL' - INVOKE APPLICATION
* 'TERM' - TERMINATE PL/I
* 'EXECUTE' - INIT, CALL, TERM
* PRP_EPL_PTR DC A(IBMBZEPL) A(EPL) - EXTENDED PARM LIST
*================================================================
* PARAMETER LIST FOR THE PRE-INITIALIZED PROGRAM
*================================================================
IBMBZEPL DS 0F
EPL_LENGTH DC A(EPL_SIZE) LENGTH OF THIS EPL PASSED
EPL_TOKEN1 DC F'0' FIRST ENV TOKEN
EPL_TOKEN2 DC F'0' SECOND ENV TOKEN
EPL_PROG_PARMS DC A(EPL_PROG_ADDR) A(PARM ADDRESS LIST) ...
EPL_EXEC_OPTS DC A(EXEC_ADDR) A(EXECUTION TIME OPTNS) ...
EPL_ALTMAIN DC F'0' A(ALTERNATE MAIN)
EPL_SERVICE_VEC DC A(IBMBZSRV) A(SERVICE ROUTINES VECTOR)
EPL_SIZE EQU *-IBMBZEPL THE SIZE OF THIS BLOCK
*================================================================
* SERVICE ROUTINE VECTOR
*================================================================
IBMBZSRV DS 0F
SRV_SLOTS DC F'2' COUNT OF SLOTS DEFINED
SRV_USERWORD DC A(SRV_UA) USER WORD
SRV_WORKAREA DC A(SRV_WA) A(WORKAREA)
SRV_LOAD DC F'0' A(LOAD ROUTINE)
SRV_DELETE DC F'0' A(DELETE ROUTINE)
SRV_GETSTOR DC F'0' A(GET STORAGE ROUTINE)
SRV_FREESTOR DC F'0' A(FREE STORAGE ROUTINE)
SRV_EXCEP_RTR DC F'0' A(EXCEPTION ROUTER SERVICE)
SRV_ATTN_RTR DC F'0' A(ATTENTION ROUTER SERVICE)
SRV_MSG_RTR DC F'0' A(MESSAGE ROUTER SERVICE)
SRV_END DS 0F
*================================================================
* SERVICE ROUTINE USERAREA
*================================================================
SRV_UA DS 8F
* SERVICE ROUTINE WORKAREA
*=====================================================================
SRV_WA DS 0D
DC '256' LENGTH OF WORKAREA
DS 63F ACTUAL WORKAREA
*=====================================================================
* EXECUTION TIME PARAMETERS
*=====================================================================
EXEC_ADDR DC A(X'80000000'+EXEC_LEN)
EXEC_LEN DC AL2(EXEC_OLEN)
EXEC_OPTS DC C'NATLANG(ENU),NOSTAE'
EXEC_OLEN EQU *-EXEC_OPTS
* LTORG
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
END

INMODPL2: PROCEDURE (X,Y) OPTIONS (MAIN);
DCL X FIXED, Y FIXED;
DCL 1 PARM_LIST ALIGNED BASED(P),
  10 STATUS FIXED BINARY (31,0),
  10 RLENGTH FIXED BINARY (31,0),
  10 BUFFER CHAR (80);
DCL 1 PARM_PARM2 ALIGNED BASED(Q),
  10 SEQ FIXED BINARY (31,0),
  10 LEN FIXED BINARY (15,0),
  10 PARAMETER CHAR (80);
DCL COUNT STATIC FIXED BINARY (31,0),
  INSROWS STATIC FIXED BINARY (31,0),
  REJROWS STATIC FIXED BINARY (31,0);
DCL I, NOTMATCH FIXED BINARY (31,0);
DCL ADDR BUILTIN, SUBSTR BUILTIN;
DCL P POINTER, Q POINTER;
DCL SYSPRINT FILE OUTPUT;
P = ADDR(X);
Q = ADDR(Y);
OPEN FILE(SYSPRINT);
Appendix C: INMOD and Notify Exit Routine Examples
VM and MVS

```plaintext
PUT SKIP LIST('### INSIDE PL/I INMOD ROUTINE...');
PUT SKIP LIST('STATUS =');
PUT LIST(P->STATUS);
PUT SKIP LIST('LENGTH =');
PUT LIST(P->RLENGTH);
PUT SKIP LIST('BUFFER =');
PUT LIST(SUBSTR(P->BUFFER,1,30));
PUT SKIP LIST('SEQ =');
PUT LIST(Q->SEQ);
PUT SKIP LIST('FARM =');
PUT LIST(SUBSTR(Q->PARAMETER,1,Q->LEN));

SELECT (P->STATUS);

WHEN (6) DO; /* INITIALIZE */
  COUNT = 0;
  REJROWS = 0;
  INSROWS = 0;
  P->STATUS = 0;
END;

WHEN (7) DO; /* PROCESS */
  COUNT = COUNT + 1;
  NOTMATCH = 0;
  P->STATUS = 0;
  DO I = 1 TO Q -> LEN;
    IF SUBSTR(P->BUFFER,I,1) ^= SUBSTR(Q->PARAMETER,I,1)
      THEN DO;
        NOTMATCH= 1;
        LEAVE;
      END;
    END;
  IF NOTMATCH = 1
    THEN DO;
      PUT SKIP LIST('------> REJECTED <------');
      REJROWS = REJROWS + 1;
      P->RLENGTH = 0;
    END;
  ELSE
    DO;
      PUT SKIP LIST('------> ACCEPTED <------');
      INSROWS = INSROWS + 1;
    END;
END;

WHEN (5) DO; /* FINALIZE */
  P->STATUS = 0;
END;

OTHERWISE DO;
  PUT SKIP LIST ('UNKNOWN CODE...');
  P->STATUS = 99;
END;
END;

PUT SKIP LIST('STATUS =');
PUT LIST(P->STATUS);
PUT SKIP LIST('LENGTH =');
PUT LIST(P->RLENGTH);
```

Appendix C: INMOD and Notify Exit Routine Examples
VM and MVS

```
PUT SKIP LIST('TOTAL =');
PUT LIST(COUNT);
PUT SKIP LIST('INSERTS =');
PUT LIST(INSROWS);
PUT SKIP LIST('REJROWS =');
PUT LIST(REJROWS);
PUT SKIP LIST('---------------------------------------------------');
CLOSE FILE(SYSPRINT);

END INMDPL2;
//LKED.SYSPRINT DD SYSOUT=* 
//LKED.PL1A DD DISP=(OLD,DELETE),DSN=&&PL1A 
//LKED.SYSIN DD *
ENTRY DYNAMN 
NAME INMDPL2(R) 
//COPY EXEC PGM=IEBGENER  
//SYSIN DD DUMMY 
//SYSPRINT DD SYSOUT=* 
//SYSUT2 DD DISP=(NEW,PASS),DSN=&&TEMP,UNIT=SYSDA,  
// DCB=(LRECL=80,BLKSIZE=1760,RECFM=FB),  
// SPACE=(CYL,(1,1),RLSE) 
//SYSUT1 DD DATA,DLM=@@ 
("SASC") A0000000000000000000000000000A 
("PASC") A0000000000000000000000000000A 
("COBOL") A0000000000000000000000000000A 
("ASSEM") A0000000000000000000000000000A 
("SASC") B1111111111111111111111111111B 
("PASC") B1111111111111111111111111111B 
("COBOL") B1111111111111111111111111111B 
("ASSEM") B1111111111111111111111111111B 
("SASC") C2222222222222222222222222222C 
("PASC") C2222222222222222222222222222C 
("COBOL") C2222222222222222222222222222C 
("ASSEM") C2222222222222222222222222222C 
("PL/I") C2222222222222222222222222222C 
("SASC") D3333333333333333333333333333D 
("PASC") D3333333333333333333333333333D 
("PL/I") D3333333333333333333333333333D 
("SASC") E4444444444444444444444444444E 
("PASC") E4444444444444444444444444444E 
("PL/I") E4444444444444444444444444444E 
("SASC") F5555555555555555555555555555F 
("PASC") F5555555555555555555555555555F 
("PL/I") F5555555555555555555555555555F 
@@ 

/********************************************************************
/* THIS STEP WILL ONLY DROP THE TABLES IF MLOAD IS NOT IN APPLY */
/* PHASE */
********************************************************************/

CREATE EXEC BTEQ  
.LOGON TDP5/DMD,DMD;  
/* INMOD TEST CASE II - PL/I */  
.RELEASE MLOAD DMD.INMODPL2;  
.IF ERRORCODE = 2572 THEN .GOTO NODROP;  
.DROP TABLE DMD.LOGTABLE;  
.DROP TABLE DMD.ET_INMODPL2;  
.DROP TABLE DMD.UV_INMODPL2;  
.DROP TABLE DMD.WT_INMODPL2;  
```
DROP TABLE DMD.INMODPL2;
QUIT;
LABEL NODROP;
EXIT 4;
CREATE TABLE INMODPL2 (F1 CHAR(10), F2 CHAR(70));
#
//******************************************************************************
//**
//**  RUN MULTILOAD
//**
//******************************************************************************
//LOADIT EXEC PGM=MLOAD,TIME=(,3)
//STEPLIB DD DSN=STV.RG20.APPLOAD,DISP=SHR
// DD DSN=STV.EG14MLL1.APP.L,DISP=SHR
// DD DSN=STV.TG13BLD.APP.L,DISP=SHR
// DD DSN=TER2.SASC450F.LINKLIB,DISP=SHR
// DD DSN=*.STEP2.LKED.SYSLMOD,DISP=(OLD,PASS),
// VOL=REF=*.STEP2.LKED.SYSLMOD
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//SYSPUR DD SYSOUT=* 
//INDATA DD DISP=OLD,DSN=*.COPY.SYSUT2,DCB=(LRECL=80,RECFM=F),
// VOL=REF=*.COPY.SYSUT2
//SYSIN DD DATA,DLM=##
.LOGON TDP5/DMD,DMD;
.LOGTABLE DMD.LOGTABLE_SFD;
BEGIN IMPORT MLOAD TABLES INMODPL2;
.Layout layname1;
.Field L1Fld1 1 Char(10);
.Field L1Fld2 * Char(30);
.Field L1Fld3 * Char(40);
.DML Label DML1;
INSERT INMODPL2(F1,F2) VALUES (:L1FLD1, :L1FLD2);
.IMPORT INFILE INDATA
.INMOD INMDPL2 USING ("PL/I") LAYOUT LAYNAME1 APPLY DML1;
.End Mload;
.LOGOFF;
##

UNIX

C INMOD Example

This INMOD example reads the function code and executes different processing functions based on its value.

/* This program is for release 4.1 MULTILOAD INMOD testing using C user exit routine.  
When this routine is activated it looks at the content of the function code passed (a->code) and depending on its value, it 0) initializes, i.e., opens a file, etc... 1) reads a record 5) acknowledges "close inmod" request. The user exit routine must return "return code" (a->code) and "length" (a->len). You
should send return code = zero when no errors occur and non-zero for an error. MULTILOAD expects length = zero at the end of file. Then it sends "CLOSE INMOD" request. THE USER EXIT routine must explicitly return "return code" = ZERO to terminate the conversation. /*
The file includes <stddef.h>, <stdlib.h>, and <stdio.h>. The typedefs for Int16, Int8, and Int32 are defined.

typedef unsigned short   Int16;
typedef unsigned char    Int8;
typedef unsigned long int Int32;

/* PASSING parameter structures */

typedef struct {
    Int32 code;
    Int32 len;
    Int8 buf[80];
} inmodbuf;

typedef struct {
    Int32 seq;
    Int16 len;
    char param[80];
} inmodpty;

/* "IN" is a static FILE pointer. */
static FILE *IN;
static int count=0;
char *memcpy();

void _dynamn(a,b)
inmodbuf *a;
inmodpty *b;
{ int code=0;
char tempbuf[80];
memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]='$0';
printf("BEGIN---> %d %d %s
",a->code,a->len,tempbuf);
printf(" +++ %d %d %s
",b->seq ,b->len,b->param);
    code= (int) a->code;
switch (code) {
    case 0:
        /* Here you open the file and read the first record */
    printf("## CODE=0, openinig...");
    IN=fopen("ddn:INDATA","rb");
    if (! ferror(IN)) {
        if (! readrecord(a))
            fclose(IN);
    }
    break;
    case 1:
        /* MultiLoad requested next record, read it */
    printf("## CODE=1, reading...");
    if (! readrecord(a))
        fclose(IN);
    break;
    case 5:
        /* MultiLoad is closing INMOD routine */
Appendix C: INMOD and Notify Exit Routine Examples

UNIX

a->code=0;
a->len=0;
printf("## CODE=5, terminating...
break;
default:
a->code=12; /* any number not = to zero */
a->len=0;
printf("#### Unknown code ####");a->code=0;a->len=0;
}

memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]=’\0’;
printf("END --> %d %d %s
",a->code,a->len,tempbuf);
printf(" +++ %d %d %s
",b->seq,b->len,b->param);
}

vint readrecord(a)
inmodbuf *a;
{
  int rtn=0;
  char tempbuf[80];

  if (fread((char *)&(a->buf),sizeof(a->buf),1,IN)) {
      count++;
      memcpy(tempbuf,a->buf,sizeof(a->buf));
tempbuf[79]=’\0’;
      printf(" %d %s 
",count,tempbuf);
a->len=80;
a->code=0;
      rtn=1;
  }
  if ferror(IN) {
      printf("==== error ====");
a->code=16; /* any non zero number */
a->len=0;
  }
  if feof(IN) {
      /* EOF, set length = zero */
      printf("=== EOF ===");
a->code=9;
a->len=9;
  }
  return(rtn);
}

C Notify Exit Parameters

Following is a C structure that describes the parameters passed to a notify exit routine. The pointer is a 32-bit value, and all long definitions are 32-bit unsigned.

typedef unsigned long UInt32;
typedef enum {
  NMEventInitialize      = 0,
  NMEventFileInmodOpen   = 1,
  NMEventPhaseIBegin     = 2,
  NMEventCheckPoint      = 3,
}
Appendix C: INMOD and Notify Exit Routine Examples

UNIX

NMEventPhaseIEnd = 4,
NMEventPhaseIIBegin = 5,
NMEventPhaseIIEnd = 6,
NMEventErrorTableI = 7,
NMEventErrorTableII = 8,
NMEventDBSRestart = 9,
NMEventCLIError = 10,
NMEventDBSError = 11,
NMEventExit = 12,
NMEventAmpsDown = 21,
NMEventImportBegin = 22,
NMEventImportEnd = 23,
NMEventDeleteInit = 24,
NMEventDeleteBegin = 25,
NMEventDeleteEnd = 26,
NMEventDeleteExit = 27
} NfyMLDEvent;

/**************************************/
/* Structure for User Exit Interface */
/* DR42570 - redesigned and rewritten */
/**************************************/
#define NOTIFYID_FASTLOAD 1
#define NOTIFYID_MULTILOAD 2
#define NOTIFYID_FASTEXPORT 3
#define NOTIFYID_BTEQ 4
#define NOTIFYID_F 5
#define MAXVERSIONIDLEN 32
#define MAXUTILITYNAMELEN 32
#define MAXUSERNAMELEN 64
#define MAXUSERSTRLEN 80
#define MAXTABLENAMELEN 128
#define MAXFILENAMELEN 256
typedef struct _MLNotifyExitParm {
    UInt32 Event; /* should be NfyMLDEvent values */
    union {
        struct {
            UInt32 VersionLen;
            char VersionId[MAXVERSIONIDLEN];
            UInt32 UtilityId;
            UInt32 UtilityLen;
            char UtilityName[MAXUTILITYNAMELEN];
            UInt32 UserNameLen;
            char UserName[MAXUSERNAMELEN];
            UInt32 UserStringLen;
            char UserString[MAXUSERSTRLEN];
        } Initialize;
        struct {
            UInt32 FileNameLen;
            char FileOrInmodName[MAXFILENAMELEN];
            UInt32 ImportNo;
        } FileInmodOpen;
        struct {
            UInt32 TableNameLen;
            char TableName[MAXTABLENAMELEN];
            UInt32 TableNo;
        } PhaseIBegin;
        struct {
            UInt32 RecordCount;
        } CheckPoint;
    } Initialize;
} _MLNotifyExitParm;
struct {
    UInt32 RecsRead;
    UInt32 RecsSkipped;
    UInt32 RecsRejected;
    UInt32 RecsSent;
} PhaseIEnd;
struct {
    UInt32 dummy;
} PhaseIIBegin;
struct {
    UInt32 Inserts;
    UInt32 Updates;
    UInt32 Deletes;
    UInt32 TableNo;
} PhaseIIEnd;
struct {
    UInt32 Rows;
    UInt32 TableNo;
} ErrorTableI;
struct {
    UInt32 Rows;
    UInt32 TableNo;
} ErrorTableII;
struct {
    UInt32 dummy;
} DBSRestart;
struct {
    UInt32 ErrorCode;
} CLIError;
struct {
    UInt32 ErrorCode;
} DBSError;
struct {
    UInt32 ReturnCode;
} Exit;
struct {
    UInt32 dummy;
} AmpsDown;
struct {
    UInt32 ImportNo;
} ImportBegin;
struct {
    UInt32 RecsRead;
    UInt32 RecsSkipped;
    UInt32 RecsRejected;
    UInt32 RecsSent;
    UInt32 ImportNo;
} ImportEnd;
struct {
    UInt32 dummy;
} DeleteInit;
struct {
    UInt32 TableNameLen;
    char TableName[MAXTABLENAMELEN];
    UInt32 TableNo;
} DeleteBegin;
struct {
    UInt32 Deletes;
    UInt32 TableNo;
{ } DeleteEnd;
struct {
    Uint32 ReturnCode;
} DeleteExit;
} Vals;
} MLNotifyExitParm;

#ifdef I370
#define MLNfyExit MLNfEx
#endif
extern long MLNfyExit(
#ifdef __STDC__
    MLNotifyExitParm *Parms
#endif
);

## Compiling and Linking Routines

**Note:** For a description of the syntax diagrams used in this book, see Appendix A: “How to Read Syntax Diagrams.”

### MP-RAS and Sun Solaris SPARC

To compile and link source files into a shared object module for INMOD or notify exit routines on MP-RAS and Sun Solaris SPARC client systems, use the following syntax:

**Compile Syntax**

```
cc -G -KPIC sourcefile.c -o shared-object-name
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>Call to the program that invokes the native UNIX C compiler</td>
</tr>
<tr>
<td>-G</td>
<td>Linker option that generates a shared object file</td>
</tr>
<tr>
<td>-KPIC</td>
<td>Compiler option that generates Position Independent Code for all user exit routines</td>
</tr>
<tr>
<td>-o</td>
<td>Switch to the linker</td>
</tr>
<tr>
<td>shared-object-name</td>
<td>Name of your shared object file</td>
</tr>
</tbody>
</table>

This is the name you specify as the:

- INMOD `modulename` parameter in the `IMPORT` of your MultiLoad job script
- EXIT `name` parameter of the NOTIFY option in the `BEGIN MLOAD` and `BEGIN DELETE MLOAD` of your MultiLoad job script

The `shared-object-name` can be any valid UNIX file name.
Appendix C: INMOD and Notify Exit Routine Examples

UNIX

To compile and link source files into a shared object module for INMOD or notify exit routines on Solaris Opteron client systems, use the following syntax:

**Compile Syntax**

\[
\text{cc} \ -\text{dy} \ -\text{G} \ \text{sourcefile.c} \ -\text{o} \ \text{shared-object-name}
\]

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sourcefile</code></td>
<td>UNIX file name of the source file for your INMOD or notify exit routine</td>
</tr>
<tr>
<td><code>cc</code></td>
<td>Call to the program that invokes the native UNIX C compiler</td>
</tr>
<tr>
<td><code>-dy</code></td>
<td>Specifies to use dynamic linking</td>
</tr>
<tr>
<td><code>-G</code></td>
<td>Produces a shared object-enabled for use with the runtime linker</td>
</tr>
<tr>
<td><code>shared-object-name</code></td>
<td>Name of your shared object file</td>
</tr>
<tr>
<td></td>
<td>This is the name you specify as the:</td>
</tr>
<tr>
<td></td>
<td>• INMOD <code>modulename</code> parameter in the <code>IMPORT</code> of your MultiLoad job script</td>
</tr>
<tr>
<td></td>
<td>• EXIT <code>name</code> parameter of the NOTIFY option in the <code>BEGIN MLOAD</code> and <code>BEGIN DELETE MLOAD</code> of your MultiLoad job script</td>
</tr>
<tr>
<td><code>-o</code></td>
<td>Switch to the linker</td>
</tr>
<tr>
<td><code>sourcefile</code></td>
<td>UNIX file name of the source file for your INMOD or notify exit routine</td>
</tr>
</tbody>
</table>

**Sun Solaris Opteron**

To compile and link source files into a shared object module for INMOD or notify exit routines on Solaris Opteron client systems, use the following syntax:

**Compile Syntax**

\[
\text{cc} \ +\text{z} \ +\text{ul} \ -\text{c} \ \text{sourcefile.c}
\]

**HP-UX PA RISC**

To compile and link source files into a shared object module for INMOD and notify exit routines on HP-UX PA RISC client systems, use the following syntax:

**Compile Syntax**

\[
\text{cc} \ +\text{z} \ +\text{ul} \ -\text{c} \ \text{sourcefile.c}
\]
### Link Syntax

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b</code></td>
<td>Linker option that generates a shared object file</td>
</tr>
<tr>
<td><code>-c</code></td>
<td>Compile-only option (does not link)</td>
</tr>
<tr>
<td><code>cc</code></td>
<td>Call to the program that invokes the native UNIX C compiler</td>
</tr>
<tr>
<td><code>ld</code></td>
<td>Call to the program that invokes the native UNIX linker</td>
</tr>
<tr>
<td><code>-o</code></td>
<td>Switch to the linker</td>
</tr>
<tr>
<td><code>objectfile</code></td>
<td>Compiler-generated file used by the linker to generate <code>shared-object-name</code></td>
</tr>
<tr>
<td><code>shared-object-name</code></td>
<td>Name of your shared object file</td>
</tr>
<tr>
<td><code>sourcefile</code></td>
<td>UNIX file name(s) of the source file(s) for your INMOD or notify exit routine</td>
</tr>
<tr>
<td><code>+ul</code></td>
<td>Compiler option that allows pointers to access non-natively aligned data</td>
</tr>
<tr>
<td><code>+z</code></td>
<td>Compiler option that generates Position Independent Code for all user exit routines</td>
</tr>
</tbody>
</table>

### HP-UX Itanium

To compile and link source files into a shared object module for INMOD and notify exit routines on HP-UX Itanium-based clients, use the following syntax:

### Compile Syntax

```
cc -D_REENTRANT -DD64 -c inmod.c
```
Use the following syntax example to link the object modules on HP-UX Itanium into the shared object.

**Link Syntax**

```
ld -n -b inmod.o -lc -o inmod.so
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
<td>Invokes the UNIX linker editor</td>
</tr>
<tr>
<td>-n</td>
<td>Generates an executable with file type SHARE_MAGIC. This option is ignored in 64-bit mode.</td>
</tr>
<tr>
<td>-b</td>
<td>Is a linker option specified to generate a shared object file</td>
</tr>
<tr>
<td>inmod.o</td>
<td>Is an object module derived from the compile step (see above)</td>
</tr>
<tr>
<td>-lc</td>
<td>Search a library lib.a, libc.so, or libc.sh</td>
</tr>
<tr>
<td>-o</td>
<td>Specifies the output filename; default is a.out</td>
</tr>
<tr>
<td>inmod.so</td>
<td>Specifies the resulting shared object module This is the user-specified name in the IMPORT command.</td>
</tr>
</tbody>
</table>
IBM AIX

To compile and link source files into a shared object module for INMOD and notify exit routines on IBM AIX client systems, use the following syntax:

**Compile Syntax**

```
cc       -c       -brtl       -fPIC       sourcefile.c
```

**Link Syntax**

```
ld        -G          -e_dynamn  -bE: export_dynamn.txt

objectfile.o       -o shared-object-name       -lm       -lc
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Compiler option specifying to not send object files to the linkage editor</td>
</tr>
<tr>
<td>cc</td>
<td>Call to the program that invokes the native UNIX C compiler</td>
</tr>
<tr>
<td>-bE:</td>
<td>The linker option that exports the symbol &quot;_dynamn&quot; explicitly and the file export-dynamn.txt contains the symbol</td>
</tr>
<tr>
<td>export_dynamn.txt</td>
<td></td>
</tr>
<tr>
<td>-brtl</td>
<td>Tells the linkage editor to accept both .sl and .a library file types</td>
</tr>
<tr>
<td>-e_dynamn</td>
<td>Sets the entry point of the exit routine to _dynamn</td>
</tr>
<tr>
<td>-fPIC</td>
<td>Compiler option that generates Position Independent Code for all user exit routines</td>
</tr>
<tr>
<td>-G</td>
<td>Produces a shared object-enabled for use with the runtime linker</td>
</tr>
<tr>
<td>-lc</td>
<td>Link with the /lib/libc.a library</td>
</tr>
<tr>
<td>ld</td>
<td>Call to the program that invokes the native UNIX linker</td>
</tr>
<tr>
<td>-lm</td>
<td>Link with the /lib/libm.a library</td>
</tr>
<tr>
<td>-o</td>
<td>Switch to the linker</td>
</tr>
<tr>
<td>objectfile</td>
<td>Compiler-generated file used by the linker to generate shared-object-name</td>
</tr>
</tbody>
</table>
To compile and link source files into a shared object module for INMOD or notify exit routines on LINUX client systems, use the following syntax.

**Note:** Be sure to compile your INMOD and notify exit routines in 32-bit mode so they are compatible with Teradata MultiLoad.

### Compile Syntax

```
gcc -I/usr/include -shared -fPIC sourcefile.c -o shared-object-name
```

where

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>Call to the program that invokes the native C compiler</td>
</tr>
<tr>
<td>-shared</td>
<td>Flag that produces a shared object that can then be linked with other objects to form an executable</td>
</tr>
<tr>
<td>-fPIC</td>
<td>Compiler option that generates Position Independent Code for all user exit routines</td>
</tr>
<tr>
<td>-o</td>
<td>Output file name</td>
</tr>
<tr>
<td>sourcefile</td>
<td>File name(s) of the source file(s) for your INMOD or notify exit routine</td>
</tr>
<tr>
<td>shared-object-name</td>
<td>Name of your shared object file</td>
</tr>
</tbody>
</table>

The *shared-object-name* can be any valid UNIX file name. This is the name you specify as:

- The INMOD *modulename* parameter of the IMPORT of your MultiLoad job script.
- The EXIT *name* parameter for the NOTIFY option of the BEGIN MLOAD and BEGIN DELETE MLOAD of your MultiLoad job script.

### Note:

For a description of the syntax diagrams used in this book, see Appendix A: “How to Read Syntax Diagrams.”
To generate and use an INMOD or notify exit routine on a Windows client system, the routine must:

- Be written in C
- Have a dynamic entry point that is a __declspec
- Be saved as a Dynamic-Link Library (DLL) file

**Generating Routines**

Two sample program files are provided with MultiLoad software to help you generate and use INMOD and notify exit routines in your MultiLoad job scripts on network-attached Windows client systems. The listings of these sample files are presented later in this appendix.

<table>
<thead>
<tr>
<th>Sample File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlimod.c</td>
<td>Source file for an INMOD routine</td>
</tr>
<tr>
<td>mlnotf.c</td>
<td>Source file for a notify exit routine</td>
</tr>
</tbody>
</table>

Refer to the referenced sample file listings and use the following procedure to generate and use an INMOD or notify exit routine in your MultiLoad job:

1. Edit your routine source file and make sure that the dynamic name is a __declspec.
   
   See the listing of the sample routine files later in this appendix:
   
   - mlimod.c
   - mlnotf.c

2. Use the following command to create a DLL:
   
   ```
   cl /DWIN32 /LD sourcefilename
   ```
   
   where `sourcefilename` is the name of your INMOD or notify exit routine source file.
   
   Successful command execution produces a file with the same name as your source file with the `.dll` file extension, as:
   
   `sourcefilename.dll`

3. Use the `sourcefilename.dll` file name in your MultiLoad job script as follows:

<table>
<thead>
<tr>
<th>Routine Type</th>
<th>Use the <code>sourcefilename.dll</code> File as the</th>
</tr>
</thead>
<tbody>
<tr>
<td>INMOD</td>
<td>INMOD <code>modulename</code> specification in the <code>IMPORT</code> of your MultiLoad job script.</td>
</tr>
<tr>
<td>Notify Exit</td>
<td>EXIT <code>name</code> specification of the NOTIFY option in the BEGIN MLOAD or BEGIN DELETE MLOAD command of your MultiLoad job script.</td>
</tr>
</tbody>
</table>
Sample INMOD Routine

Following is the listing of the `mlimod.c` sample INMOD routine that is provided with MultiLoad software:

```c
#include <stdio.h>
#include <stdlib.h>

/*====================================================================
/*
/* mlimod.c   - Sample Inmod for MultiLoad.
/*
/* Purpose    - This inmod generates two integers per record. The first is even and the second number is odd.
/*
/* Note       - The number of records per file is determined by the variable NUM_RECORDS
/*
/* Execute    - Build Inmod on a Unix system
/*   compile and link into shared object
/*      cc -G mlimod.c - o mlimod.so
/*
/*   - Build Inmod on a Win32 system
/*   compile and link into dynamic link library
/*      cl /DWIN32 /LD mlimod.c
/*
/*====================================================================

static int msg_cnt = 0;
typedef struct {
    long  ioseq;
    short len;
    char  param[2000];
} param_type;
/* This structure is used to pass an odd and an even integer back to multiload */
typedef struct {
    long  code;
    long  len;
    char  data[32768];
} data_type;
#ifdef WIN32                      /* Change for WIN32 */
    __declspec(dllexport) void _dynamn(data_type *data_buf , param_type *parm_buf)
#else
    void _dynamn(data_buf, parm_buf)
    data_type  *data_buf;
    param_type *parm_buf;
#endif
{
    char *myptr;
    int  i;
    static long RECNUM = 0;       /* number of records to load */
    static int odd_counter = 0;  /* odd integer counter */
    static int even_counter = 0; /* even integer counter */
    #ifdef DEBUG
        printf("\n");
        printf("jmod2: on input:\n");
        printf("jmod2:    message code:   %d\n", data_buf->code);
        printf("jmod2:      data bytes:   %d\n", data_buf->len);
        if (data_buf->len)
            printf("jmod2:            data:   *%s*\n", data_buf->data);
```

```
printf("jmod2:     param ioseq:   %d
", parm_buf->ioseq);
printf("jmod2:     param bytes:   %d
", parm_buf->len);
if (parm_buf->len)
printf("jmod2:       param str:   %s
", parm_buf->param);
printf("jmod2:     message cnt:   %d
", ++msg_cnt);
#endif
switch (data_buf->code)
{
    case 0: printf("jmod2: initializing and returning 1st record:\n");
    RECNUM = 6;
    printf("jmod2: Records requested = %ld\n", RECNUM);
    if ( RECNUM <= 0 ) {
        printf("jmod2: numbers of records is <= 0 [%ld]\n", RECNUM);
    }
    /* initialize the counters */
    odd_counter = 1;
    even_counter = 2;
    /* copy the counters to the data buffer */
    myptr = (char *) &odd_counter;
    for (i=0; i<4; ++i, ++myptr) {
        data_buf->data[i] = *myptr;
    }
    myptr = (char *) &even_counter;
    for (i=4; i<8; ++i, ++myptr) {
        data_buf->data[i] = *myptr;
    }
    /* go to next values and increment the counters */
    odd_counter += 2;
    even_counter +=2;
    --RECNUM;
    /* return the results */
    data_buf->code = 0;
    data_buf->len  = 8;
    break;
    case 1:
        #ifdef DEBUG
        printf("jmod2: returning a record:\n");
        #endif
        if (RECNUM) {
            /* copy the counters to the data buffer */
            myptr = (char *) &odd_counter;
            for (i=0; i<4; ++i, ++myptr) {
                data_buf->data[i] = *myptr;
            }
            myptr = (char *) &even_counter;
            for (i=4; i<8; ++i, ++myptr) {
                data_buf->data[i] = *myptr;
            }
            /* increment to next values and decrement record counter */
            odd_counter += 2;
            even_counter += 2;
            --RECNUM;
            /* return the results */
            data_buf->code = 0;
            data_buf->len  = 8;
            break;
        } else {
            /* done sending records, return non-zero result */
            printf("jmod2: all records sent\n");
data_buf->code = 1;
break;
}
#endif

Sample Notify Exit Routine

Following is the listing of the mlnotf.c sample notify exit routine that is provided with MultiLoad software:

/********************************************************************************
/* mlnotf.c - Sample Notify Exit for MultiLoad. */
/* Purpose - This is a sample notify exit for MultiLoad. */
/* Execute - Build Notify on a Unix system */
/* compile and link into shared object */
/* cc -G mlnotf.c -o mlnotf.so */
/* */
/* Execute - Build Notify on a Win32 system */
/* compile and link into dynamic link library */
/* cl /WIN32 /LD mlnotf.c */
/* */
/* History : Updated with new events. */
*********************************************************************************/
#include <stdio.h>
typedef unsigned long UInt32;
typedef enum {
    NMEventInitialize = 0,
    NMEventFileInmodOpen = 1,
    NMEventPhaseIBegin = 2,
    NMEventCheckPoint = 3,
    NMEventPhaseIEnd = 4,
    NMEventPhaseIIBegin = 5,
    NMEventPhaseIIEnd = 6,
    NMEventErrorTableI = 7,
    NMEventErrorTableII = 8,
    NMEventDBSRestart = 9,
    NMEventCLIError = 10,
    NMEventDBSError = 11,
    NMEventExit = 12,
    NMEventAmpsDown = 21,
    NMEventImportBegin = 22,
    NMEventImportEnd = 23,
    NMEventDeleteInit = 24,
    NMEventDeleteBegin = 25,
    NMEventDeleteEnd = 26,
    NMEventDeleteExit = 27
} NfyMLDEvent;

/**********************************************/
/* Structure for User Exit Interface   */
/* DR42570 - redesigned and rewritten */
/***********************************************/
#define NOTIFYID_FASTLOAD 1
#define NOTIFYID_MULTILOAD 2
#define NOTIFYID_FASTEXPORT 3
#define NOTIFYID_BTEQ 4
#define NOTIFYID_TPUMP 5
#define MAXVERSIONIDLEN 32
#define MAXUTILITYNAMELEN 32
#define MAXUSERNAMELEN 64
#define MAXUSERSTRLEN 80
#define MAXTABLENAMELEN 128
#define MAXFILENAMELEN 256
typedef struct _MLNotifyExitParm {
    UInt32 Event; /* should be NfyMLDEvent values */
    union {
        struct {
            UInt32 VersionLen;
            char VersionId[MAXVERSIONIDLEN];
            UInt32 UtilityId;
            char UtilityNameLen;
            char UtilityName[MAXUTILITYNAMELEN];
            UInt32 UserNameLen;
            char UserName[MAXUSERNAMELEN];
            UInt32 UserStringLen;
            char UserString[MAXUSERSTRLEN];
        } Initialize;
        struct {
            UInt32 FileNameLen;
            char FileOrInmodName[MAXFILENAMELEN];
            UInt32 ImportNo;
        } FileInmodOpen;
    } ...
}
UInt32 TableNameLen;
char TableName[MAXTABLENAMELEN];
UInt32 TableNo;
} PhaseIBegin;
struct {
    UInt32 RecordCount;
} CheckPoint;
struct {
    UInt32 RecsRead;
    UInt32 RecsSkipped;
    UInt32 RecsRejected;
    UInt32 RecsSent;
} PhaseIEnd;
struct {
    UInt32 dummy;
} PhaseIIBegin;
struct {
    UInt32 Inserts;
    UInt32 Updates;
    UInt32 Deletes;
    UInt32 TableNo;
} PhaseIIEnd;
struct {
    UInt32 Rows;
    UInt32 TableNo;
} ErrorTableI;
struct {
    UInt32 Rows;
    UInt32 TableNo;
} ErrorTableII;
struct {
    UInt32 dummy;
} DBSRestart;
struct {
    UInt32 ErrorCode;
} CLIError;
struct {
    UInt32 ErrorCode;
} DBSError;
struct {
   UInt32 ReturnCode;
} Exit;
struct {
    UInt32 dummy;
} AmpsDown;
struct {
    UInt32 ImportNo;
} ImportBegin;
struct {
    UInt32 RecsRead;
    UInt32 RecsSkipped;
    UInt32 RecsRejected;
    UInt32 RecsSent;
    UInt32 ImportNo;
} ImportEnd;
struct {
    UInt32 dummy;
} DeleteInit;
struct {
    
}
Appendix C: INMOD and Notify Exit Routine Examples

Windows

```c
UInt32 TableNameLen;
char TableName[MAXTABLENAMELEN];
UInt32 TableNo;
} DeleteBegin;
struct {
    UInt32 Deletes;
    UInt32 TableNo;
} DeleteEnd;
struct {
    UInt32 ReturnCode;
} DeleteExit;
} Vals;
} MLNotifyExitParm;

#ifdef I370
#define MLNotifyExit MLNotifyExit
#endif
extern long MLNotifyExit(
#ifdef __STDC__
    MLNotifyExitParm *Parms
#endif
);
#ifdef WIN32
__declspec(dllexport) long _dynamn(MLNotifyExitParm *P)
#else
long _dynamn( MLNotifyExitParm *P)
#endif
{
    FILE *fp;
    if (!(fp = fopen("NFYEXIT.OUT", "a")))
        return(1);
    switch(P->Event) {
    case NMEventInitialize :
        fprintf(fp, "exit called @ mload init.\n");
        fprintf(fp, "Version: %s\n", P->Vals.Initialize.VersionId);
        fprintf(fp, "Utility: %s\n", P->Vals.Initialize.UtilityName);
        fprintf(fp, "User: %s\n", P->Vals.Initialize.UserName);
        if (P->Vals.Initialize.UserStringLen)
            fprintf(fp, "UserString: %s\n", P->Vals.Initialize.UserString);
        break;
    case NMEventFileInmodOpen:
        fprintf(fp, "exit called @ file open: import[%d]: %s\n", P->Vals.FileInmodOpen.ImportNo, P->Vals.FileInmodOpen.FileOrInmodName);
        break;
    case NMEventPhaseIBegin :
        fprintf(fp, "exit called @ acquistion start: tablename[%d] : %s .\n", P->Vals.PhaseIBegin.TableNo, P->Vals.PhaseIBegin.TableName);
        break;
    case NMEventCheckPoint :
        fprintf(fp, "exit called @ checkpoint : %d records loaded.\n", P->Vals.CheckPoint.RecordCount);
        break;
    case NMEventPhaseIEnd :
        fprintf(fp, "exit called @ acquistion end.\n");
```
fprintf(fp, "Records Read: %d\n", P->Vals.PhaseIEnd.RecsRead);
fprintf(fp, "Records Skipped: %d\n", P->Vals.PhaseIEnd.RecsSkipped);
fprintf(fp, "Records Rejected: %d\n", P->Vals.PhaseIEnd.RecsRejected);
fprintf(fp, "Records Sent: %d\n", P->Vals.PhaseIEnd.RecsSent);
b prominent
break;

case NMEventPhaseIIBegin :
    fprintf(fp, "exit called @ application start\n");
    break;

case NMEventPhaseIIEnd :
    fprintf(fp, "exit called @ application complete for table %d.\n", P->Vals.PhaseIIEnd.TableNo);
    fprintf(fp, "%d updates, %d inserts, %d deletes\n", P->Vals.PhaseIIEnd.Updates,
            P->Vals.PhaseIIEnd.Inserts,
            P->Vals.PhaseIIEnd.Deletes);
    break;

case NMEventErrorTableI :
    fprintf(fp,
            "exit called @ ET Table[%d] Drop : %d records in table.\n", P->Vals.ErrorTableI.TableNo, P->Vals.ErrorTableI.Rows);
    break;

case NMEventErrorTableII :
    fprintf(fp,
            "exit called @ UV Table[%d] Drop : %d records in table.\n", P->Vals.ErrorTableII.TableNo, P->Vals.ErrorTableII.Rows);
    break;

case NMEventDBSRestart :
    fprintf(fp, "exit called @ RDBMS restarted\n");
    break;

case NMEventCLIError :
    fprintf(fp, "exit called @ CLI error %d\n", P->Vals.CLIError.ErrorCode);
    break;

case NMEventDBSError :
    fprintf(fp, "exit called @ DBS error %d\n", P->Vals.DBSError.ErrorCode);
    break;

case NMEventExit :
    fprintf(fp, "exit called @ mload notify out of scope: return code %d.\n", P->Vals.Exit.ReturnCode);
    break;

case NMEventAmpsDown :
    fprintf(fp, "exit called @ down amps have been detected\n");
    break;

case NMEventImportBegin :
    fprintf(fp, "exit called @ import %d starting\n", P->Vals.ImportBegin.ImportNo);
    break;

case NMEventImportEnd :
    fprintf(fp, "exit called @ import %d ending.\n", P->Vals.ImportEnd.ImportNo);
    fprintf(fp, "Records Read: %d\n", P->Vals.ImportEnd.RecsRead);
    fprintf(fp, "Records Skipped: %d\n", P->Vals.ImportEnd.RecsSkipped);
    fprintf(fp, "Records Rejected: %d\n", P->Vals.ImportEnd.RecsRejected);
    fprintf(fp, "Records Sent: %d\n", P->Vals.ImportEnd.RecsSent);
    break;
**Appendix C: INMOD and Notify Exit Routine Examples**

**Windows**

```c
case NMEventDeleteInit : /* nothing */
    fprintf(fp, "exit called @ mload delete init.
" );
    break;

case NMEventDeleteBegin :
    fprintf(fp, "exit called @ delete app start for table[%d]: %s.
", P->Vals.DeleteBegin.TableNo, P->Vals.DeleteBegin.TableName);
    break;

case NMEventDeleteEnd :
    fprintf(fp, "exit called @ delete app done for table[%d]: %d rows.
", P->Vals.DeleteEnd.TableNo, P->Vals.DeleteEnd.Deletes);
    break;

case NMEventDeleteExit :
    fprintf(fp, "exit called @ mload delete notify out of scope: return code %d.
", P->Vals.DeleteExit.ReturnCode);
    break;
}
fclose(fp);
return(0);
```
APPENDIX D
MultiLoad Job Script Examples

This appendix provides examples of program listings of MultiLoad job scripts performing the following tasks:

- **Import Task**
- **Delete Task**
- **Upsert Task**
- **DDL for PERSONNEL Database Tables**

### Import Task

```sql
/*******************************************************************/
/* This task will change the departments of some employees */
/* and delete others from the Employee table, according */
/* to a transaction code. All transaction records will */
/* be inserted into a history table. */
/*******************************************************************/
.Logtable Logtable002; /* Connect to the Teradata DBS */
.Logon tdpx/user,pwd; /* identify the restart table */
Create table History /* Create the history table */
    ( TransCode char(3),
      EmpNo smallint,
      DeptNo smallint )
    Unique Primary Index (EmpNo);
.Begin Import Mload /* Specify MLOAD IMPORT task */
    tables /* and identify the */
    Employee, /* target */
    History; /* tables. */
.Layout Transaction; /* Provide layout name and */
.Field TransCode * Char(3); /* define the */
.Field EmpNo * Smallint; /* fields of */
.Field DeptNo * Smallint; /* the client record. */
.DML Label Updates; /* This dml request */
.Update Employee /* transfers employees */
    set DeptNo = :DeptNo /* with an update */
    where EmpNo = :EmpNo; /* to change dept. number. */
.DML Label Deletes; /* This dml request gets rid */
.Delete from Employee /* of employees by deleting */
    where EmpNo = :EmpNo; /* the employee from the table. */
.DML Label Inserts; /* This dml request */
.Insert into History.*; /* Inserts the transaction */
    / * into the history table. */
.Import Infile INPUT /* Identify import file */
    Layout Transaction /* and record layout */
```
Delete Task

Apply Updates where TransCode = 'TRA' /* condition for transfer... */
Apply Deletes where TransCode = 'BYE' /* condition for getting rid of */
Apply Inserts; /* unconditional history */
.End Mload; /* Initiate MLOAD processing */
.Logoff; /* Logoff Teradata Database */

Delete Task

/***************************************************************************/
/* This task will delete all employees with less than 5 years experience from the employee table. */
/***************************************************************************/
.Logon tdpx/user,pwd; /* logon to Teradata Database */
.Delete Employee /* specify delete task and identify target table */
.Delete From Employee /* The delete statement drops employees with <5 yrs exp. */
.Where YrsExp < 5; /* Initiate MLOAD processing */
.End Mload; /* Logoff Teradata Database */

Upsert Task

/***************************************************************************/
/* This task will change the phone numbers of all employees by first trying to update each row of the employee table, and then inserting the data if the update fails */
/***************************************************************************/
.Logon tdpx/user,pwd; /* logon to Teradata Database */
.Import Employee /* specify import task and identify target table */
.Import From Employee /* provide layout name and define fields of the client record. */
.Do insert for missing update rows; /* calls for an upsert */
.Update Employee /* Try an update statement to change the phone number */
.set PhoneNo = :Fone /* for each employee. */
.where EmpNo = :EmpNum; /* Complete the upsert */
.Insert Employee /* with the insert part */
.(EmpNo, PhoneNo) values /* for when the update fails. */
.(EmpNum, :Fone); /* Identify import file. */
.Import Infile INPUT /* initiate MLOAD processing. */
.Layout LayoutName /* Logoff Teradata Database */
.Apply DMLLabelname;
.End Mload; /* Logoff Teradata Database */

236 Teradata MultiLoad Reference
### DDL for PERSONNEL Database Tables

```
CREATE TABLE charges ,FALLBACK ,
   NO BEFORE JOURNAL,
   NO AFTER JOURNAL
   (   EmpNo SMALLINT FORMAT '9(5)' TITLE 'Employee//Id'
       BETWEEN 10001 AND 32001 NOT NULL,
   Proj_Id CHAR(8) TITLE 'Project// Id' NOT NULL,
   WkEnd DATE TITLE 'Week//Ending',
   Hours DECIMAL(4,1) FORMAT 'ZZ9.9' BETWEEN 0.5 AND 999.5 )
PRIMARY INDEX( EmpNo ,Proj_Id )
INDEX( Proj_Id );
CREATE TABLE department ,FALLBACK ,
   NO BEFORE JOURNAL,
   NO AFTER JOURNAL
   (   DeptNo SMALLINT FORMAT '999' BETWEEN 100 AND 900
       NOT NULL,
   DeptName VARCHAR(14),
   Loc CHAR(3),
   MgrNo SMALLINT FORMAT '9(5)' BETWEEN 10001 AND 32001
       NOT NULL)
UNIQUE PRIMARY INDEX( DeptNo );
CREATE TABLE employee ,FALLBACK ,
   NO BEFORE JOURNAL,
   NO AFTER JOURNAL
   (   EmpNo SMALLINT FORMAT '9(5)' BETWEEN 10001 AND 32001
       NOT NULL,
   Name VARCHAR(12),
   DeptNo SMALLINT FORMAT '999' BETWEEN 100 AND 900 ,
   PhoneNo SMALLINT FORMAT '9999' BETWEEN 1000 AND 9999,
   JobTitle VARCHAR(12),
   Salary DECIMAL(8,2) FORMAT 'ZZZ,ZZ9.99' BETWEEN 1.00 AND 999000.00,
   YrsExp BYTEINT FORMAT 'Z9' BETWEEN -99 AND 99 ,
   DOB DATE FORMAT 'MMMbDDbYYYY',
   Sex CHAR(1) UPPERCASE,
   Race CHAR(1) UPPERCASE,
   MStat CHAR(1) UPPERCASE,
   EdLev BYTEINT FORMAT 'Z9' BETWEEN 0 AND 22,
   HCap BYTEINT FORMAT 'Z9' BETWEEN -99 AND 99 )
UNIQUE PRIMARY INDEX( EmpNo )
INDEX( Name );
CREATE TABLE project ,FALLBACK ,
   NO BEFORE JOURNAL,
   NO AFTER JOURNAL
   (   Proj_Id CHAR(8) TITLE 'Project// Id' NOT NULL,
   Description VARCHAR(25) TITLE ' Project Description',
   RecDate DATE TITLE 'Received//Date',
   DueDate DATE TITLE 'Due //Date',
   ComDate DATE TITLE 'Compl//Date')
UNIQUE PRIMARY INDEX( Proj_Id );
```
24x7 Lights Out Operations: The use of Systems Management tools to ensure the reliable movement and update of data from operational systems to analytical systems.

2PC: Two-Phase Commit

abend: Abnormal END of task. Termination of a task prior to its completion because of an error condition that cannot be resolved by the recovery facilities that operate during execution.

ABORT: In Teradata SQL, a statement that stops a transaction in progress and backs out changes to the database only if the conditional expression associated with the abort statement is true.

Access Lock: A lock that allows selection of data from a table that may be locked for write access. The Teradata MultiLoad utility maintains access locks against the target tables during the Acquisition Phase.

Access Module: A software component that provides a standard set of I/O functions to access data on a specific device.

Access Module Processor (AMP): A virtual processor that receives steps from a parsing engine (PE) and performs database functions to retrieve or update data. Each AMP is associated with one virtual disk, where the data is stored. An AMP manages only its own virtual disk and not the virtual disk of any other AMP.

access right: A user’s right to perform the Teradata SQL statements granted to him against a table, database, user, macro, or view. Also known as privilege.

account: The distinct account name portion of the system account strings, excluding the performance group designation. Accounts can be employed wherever a user object can be specified.

Acquisition Lock: A lock that is a flag in the table header that effectively rejects certain types of Teradata SQL access statements. An acquisition lock allows all concurrent DML access and the DROP DDL statement, and rejects DDL statements other than DROP.

Acquisition Phase: Responsible for populating the primary data subtables of the work tables. Data are received from the host, converted into internal format, and inserted into the work tables. The work tables will be sorted at the end of the Acquisition Phase and prior to the Application Phase.
action definition: A logical action consisting of a single physical action and related attributes.

active data warehouse (ADW): An active data warehouse provides information that enables decision-makers within an organization to manage customer relationships quickly, efficiently and proactively. Active data warehousing is about integrating advanced decision support with day-to-day, even minute-to-minute decision making that increases quality which encourages customer loyalty and thus secures an organization’s bottom line. The market is maturing as it progresses from first-generation "passive" decision-support systems to current- and next-generation "active" data warehouse implementations.

Active Database: Active database systems integrate event-based rule processing with traditional database functionality. The behavior of the database is achieved through a set of Event-Condition-Action rules associated with the database. When an event is detected the relevant rules fire. Firing of a rule implies evaluating a condition on the database and carrying out the corresponding action. An active database system derives its power from the variety of events it can respond to and the kind of actions it can perform in response.

Ad Hoc Query: Any query that cannot be determined prior to the moment the query is issued.

administrator: A special user responsible for allocating resources to a community of users.

Aggregation: Used in the broad sense to mean aggregating data horizontally, vertically, and chronologically.

all joins: In Teradata SQL, a join is a SELECT operation that allows you to combine columns and rows from two or more tables to produce a result. Join types restricted by DWM are: inner join, outer join, merge join, product join, and all joins.

All joins are a combination of the above types, depending on how the user selects the information to be returned. In addition to the four types listed above, selecting all joins may include an exclusion join, nested join, and RowID join.

allocation group: (AG) A set of parameters that determine the amount of resources available to the sessions assigned to a PG referencing a specific AG. Has an assigned weight that is compared to other AG weights. An AG can limit the total amount of CPU used by sessions under its control.

AMP: Access Module Processor (UNIX-based systems), a type of virtual processor (vproc) that controls the management of the Teradata Database and the disk subsystem, with each AMP being assigned to a virtual disk (vdisk). For more information, see the Introduction to Teradata Warehouse.

AMP worker task: (AWT) Processes (threads on some platforms) dedicated to servicing the Teradata Database work requests. For each AMP vproc, a fixed number of AWTs are pre-allocated during Teradata Database initialization. Each AWT looks for a work request to arrive in the Teradata Database, services the request, and then looks for another. An AWT can process requests of any work type. Each Teradata Database query is composed of a series of work requests that are performed by AWTs. Each work request is assigned a work type.
indicating when the request is to be executed relative to other work requests waiting to execute.

**Analytical Data Store:** Useful in making strategic decisions, this data storage area maintains summarized or historical data. This stored data is time variant, unlike operational systems which contain real-time data. Information contained in this data store is determined and collected based on the corporate business rules.

**ANSI:** American National Standards Institute. ANSI maintains a standard for SQL. For information about Teradata compliance with ANSI SQL, see the *SQL Reference: Fundamentals*.

**AP:** Application Processor

**APE:** Alert Policy Editor. Use this Teradata Manager component to define alert policies: create actions, set event thresholds, assign actions to events, and apply the policy to the Teradata Database.

**APH:** Alternate Parcel Header.

**Application Lock:** A flag set in the table header of a target table indicating that the Application Phase is in progress. An application lock allows all concurrent access lock select access and the DROP DDL statement, and rejects all other DML and DDL statements.

**Application Lifecycle:** Includes the following three stages:

- process and change management
- analysis and design
- construction and testing

**Application Phase:** Responsible for turning rows from a work table into updates, deletes, and inserts and applying them to a single target table.

**APRC:** Application Processor Reset Containment

**API:** Application Program Interface. An interface (calling conventions) by which an application program accesses an operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code.

An API can also provide an interface between a high level language and lower level utilities and services written without consideration for the calling conventions supported by compiled languages. In this case, the API may translate the parameter lists from one format to another and the interpret call-by-value and call-by-reference arguments in one or both directions.

**Architecture:** A definition and preliminary design which describes the components of a solution and their interactions. An architecture is the blueprint by which implementers construct a solution which meets the users’ needs.

**ARCMAIN:** ARC executable that extracts (or inserts) database headers and data rows from the HUT (Host UTility) archive interface.
**ASCII:** American Standard Code for Information Interchange, a character set used primarily on personal computers.

**Availability:** A measure of the percentage of time that a computer system is capable of supporting a user request. A system may be considered unavailable as a result of events such as system failures or unplanned application outages.

**B Tree:** An indexing technique in which pointers to data are kept in a structure such that all referenced data is equally accessible in an equal time frame.

**BAR:** Backup and restore; also referred to as Backup/Archive/Restore; a software and hardware product set.

**BLOB:** An acronym for binary large object. A BLOB is a large database object that can be anything that doesn’t require character set conversion. This includes MIDI, MP3, PDF, graphics and much more. BLOBs can be up to 2 GB in size.

**BTEQ:** Basic Teradata Query facility. A utility that allows users on a workstation to access data on a Teradata Database, and format reports for both print and screen output.

**Business-Driven:** An approach to identifying the data needed to support business activities, acquiring or capturing those data, and maintaining them in a data resource that is readily available.

**bypass objects:** Specific users, groups, and accounts can be set up to circumvent DWM query management by declaring them to be bypassed. Basically, this turns off the DWM query checking mechanism for all of the requests issued by those users and/or using those accounts.

**Call-Level Interface Version 2 (CLIv2):** A collection of callable service routines that provide an interface to the Teradata Database. Specifically, CLI is the interface between the application program and the Micro Teradata Directory Program (MTDP) for network-attached clients. CLI builds parcels that MTDP packages for sending to the Teradata Database using the Micro Operating System Interface (for network-attached clients), and provides the application with a pointer to each of the parcels returned from the Teradata Database.

**Capture:** The process of capturing a production data source.

**cardinality:** In set theory, cardinality refers to the number of members in the set. When specifically applied to database theory, the cardinality of a table refers to the number of rows contained in a table.

**Change Data Capture:** The process of capturing changes made to a production data source. Change data capture is typically performed by reading the source DBMS log. It consolidates units of work, ensures data is synchronized with the original source, and reduces data volume in a data warehousing environment.
**channel-attached:** A mainframe computer that communicates with a server (for example, a Teradata Database) through a channel driver.

**Character Set:** A grouping of alphanumeric and special characters used by computer systems to support different user languages and applications. Various character sets have been codified by the American National Standards Institute (ANSI).

**Checkpoint Rate:** The interval between checkpoint operations during the Acquisition Phase of a Teradata MultiLoad import task expressed as either the number of rows read from your client system or sent to the Teradata Database, or an amount of time, in minutes.

**CICS:** Customer Information Control System

**CLI:** Call-Level Interface. The interface between the application program and the MTDP (for network-attached clients) or TDP (for channel-attached clients). CLIv2 refers to version two of the interface.

**Client:** A computer that can access the Teradata Database.

**CLlv2:** Call-Level Interface Version 2. The interface between the application program and the MTDP (for network-attached clients) or TDP (for channel-attached clients).

**CLlv2so:** Call-Level Interface Version 2 Shared Object (CLlv2so); this program installs the CLI libraries required by other utilities. When the CLlv2so program submits a request to a Teradata Database, CLI Library components transform the request into Teradata Database formats. The CLI Library sends requests to, and receives responses from, the Teradata Database over a network.

**client-server environment** The distribution of work on a LAN in which the processing of an application is divided between a front-end client and a back-end server, resulting in faster, more efficient processing. The server performs shared functions such as managing communication and providing database services. The client performs individual user functions such as providing customized interfaces, performing screen-to-screen navigation, and offering help functions.

**CMS:** Conventional Monitor System

**CLOB:** An acronym for character large object. A CLOB is a pure character-based large object in a database. It can be a large text file, HTML, RTF or other character-based file. CLOBs can be up 2 GB in size. Also see BLOB and LOB.

**Cluster:** Logical, table-level archive whereby only those rows residing on specific AMPs, and which are members of the specified cluster, are archived onto a single tape data set. This allows multiple jobs to be applied for backup of large tables, to reduce the backup window. This method is used to affect a parallel archive/restore operation using a “divide and conquer” backup strategy.

**COBOL:** COmmon Business-Oriented Language

**Coexistence System** A Teradata system running on mixed platforms
**column:** In the relational model of Teradata SQL, databases consist of one or more tables. In turn, each table consists of fields, organized into one or more columns by zero or more rows. All of the fields of a given column share the same attributes.

**COP:** Communications Processor. One kind of interface processor (IFP) on the Teradata Database. A COP contains a gateway process for communicating with workstations using a network.

**COP Interface:** Workstation-resident software and hardware, and Teradata Database-resident software and hardware, that allows workstations and the Teradata Database to communicate over networks.

**CPU:** Central processing unit.

**DASD:** Direct access storage device (pronounced DAZ-dee). A general term for magnetic disk storage devices that has historically been used in the mainframe and minicomputer (mid-range computer) environments. When used, it may also include hard disk drives for personal computers. A recent form of DASD is the redundant array of independent disks (RAID).

The "direct access" means that all data can be accessed directly in about the same amount of time rather than having to progress sequentially through the data.

**database:** A related set of tables that share a common space allocation and owner. A collection of objects that provide a logical grouping for information. The objects include tables, views, macros, triggers, and stored procedures.

**Data Cardinality:** Cardinality is a property of data elements which indicates the number of allowable entries in the element. A data element such as gender only allows two entries (male or female) and is said to possess low cardinality. Data elements for which many allowable entries are possible, such as age or income, are said to have high cardinality.

**Data Definition Language (DDL):** In Teradata SQL, the statements and facilities that manipulate database structures (such as CREATE, MODIFY, DROP, GRANT, REVOKE, and GIVE) and the Data Dictionary information kept about those structures. In the typical, pre-relational data management system, data definition and data manipulation facilities are separated, and the data definition facilities are less flexible and more difficult to use than in a relational system.

**Data Dictionary:** In the Teradata Database, the information automatically maintained about all tables, views, macros, databases, and users known to the Teradata Database system, including information about ownership, space allocation, accounting, and access right relationships between those objects. Data Dictionary information is updated automatically during the processing of Teradata SQL data definition statements, and is used by the parser to obtain information needed to process all Teradata SQL statements.

**data loading:** The process of loading data from a client platform to a Teradata Database server. For Teradata MultiLoad, data loading includes any combination of INSERT, UPDATE, DELETE, and/or UPSERT operations.
data manipulation: In Teradata SQL, the statements and facilities that change the information content of the database. These statements include INSERT, UPDATE, and DELETE.

Data Mart: A type of data warehouse designed to meet the needs of a specific group of users such as a single department or part of an organization. Typically a data mart focuses on a single subject area such as sales data. Data marts may or may not be designed to fit into a broader enterprise data warehouse design.

Data Mining: A process of analyzing large amounts of data to identify hidden relationships, patterns, and associations.

Data Model: A logical map that represents the inherent properties of the data independent of software, hardware, or machine performance considerations. The model shows data elements grouped into records, as well as the association around those records.

Data Synchronization: The process of identifying active data replicates and ensuring that data concurrency is maintained. Also known as data version synchronization or data version concurrency because all replicated data values are consistent with the same version as the official data.

Data Scrubbing: The process of filtering, merging, decoding, and translating source data to create validated data for the data warehouse.

data streams: Buffers in memory for temporarily holding data. A data stream is not a physical file; instead, it is more like a pipe (in UNIX or Windows), or a batch pipe in MVS.

Data Warehouse: A subject oriented, integrated, time-variant, non-volatile collection of data in support of management’s decision making process. A repository of consistent historical data that can be easily accessed and manipulated for decision support.

DB2: IBM DATABASE 2

DBA: Database Administrator

DBQL: Database Query Log. DBQL are a series of system tables created in the DBC database during the Teradata Database installation process. They are used to track query processing. See Database Administration to learn more about the DBQL

DD: Data dictionary or data definition.

DDL: Data definition language, which supports manipulating database structures and the Data Dictionary information kept about these structures.

DDL operator: The DDL operator is a stand-alone operator that allows you to perform any necessary database routines prior to a load/apply job without having to use another utility such as BTEQ. For example, you can create tables or indexes, or drop tables, as needed, before starting a load/apply job. As a stand-alone operator, supporting only one instance, the DDL operator does not send or retrieve data to or from a Teradata MultiLoad operator interface.
**DEFINE Statement:** A statement preceding the INSERT statement that describes the fields in a record before the record is inserted in the table. This statement is similar to the SQL USING clause.

**Delete Task:** A task that uses a full file scan to remove a large number of rows from a single Teradata Database table. A delete task is composed of three major phases: Preliminary, Application, and End. The phases are a collection of one or more transactions that are processed in a predefined order according to the Teradata MultiLoad protocol.

**delimiter:** In *Teradata SQL*, a punctuation mark or other special symbol that separates one clause in a *Teradata SQL* statement from another, or that separates one *Teradata SQL* statement from another.

**DIT:** Directory Information Tree. A graphical display of an organization’s directory structure, sites, and servers, shown as a branching structure. The top-level (root) directory usually represents the organization level.

**DLL:** Dynamic-link library. A feature of the Windows family of operating systems that allows executable routines to be stored separately as files with *.dll* extensions and to be loaded only when needed by a program.

**DML:** Data manipulation language. In *Teradata SQL*, the statements and facilities that manipulate or change the information content of the database. These statements include SELECT, INSERT, UPDATE, and DELETE.

**domain name:** A group of computers whose host names (the unique name by which a computer is known on a network) share a common suffix, that is the domain name.

**Drill down:** A method of exploring detailed data that was used in creating a summary level of data.

**DSN:** Digital Switched Network. The completely digital version of the PSTN.

**Dual Active System:** A dual active system is comprised of two active database systems that operate in tandem and serve the needs of both the production and development environments. Dual active systems virtually eliminate all down time and provide seamless disaster recovery protection for critical users and applications.

**Duplicate Row Check:** A logic within the Teradata Database used to check for duplicate rows while processing each primary data row for INSERTs and UPDATEs.

**EBCDIC:** Extended binary coded decimal interchange code. An IBM code that uses 8 bits to represent 256 possible characters. It is used primarily in IBM mainframes, whereas personal computers use ASCII.

**E-CLI:** Extended Call-Level Interface
Error Tables: Tables created during the Preliminary Phase used to store errors detected while processing a Teradata MultiLoad job. There are two error tables, ET and UV, that contains errors found during the Acquisition Phase and Application Phase, respectively.

EOF: End of File

ETL: Extract, transform, and load

EUC: Extended UNIX Code. Extended UNIX Code (EUC) for Japanese and Traditional-Chinese defines a set of encoding rules that can support from 1 to 4 character sets.

exclusion join: In Teradata SQL, a product join or merge join where only the rows that do not satisfy (are NOT in) the conditional specified in the SELECT are joined.

Exclusive Lock: Supports the manual recovery procedure when a RELEASE MLOAD statement is executed after a Teradata MultiLoad task has been suspended or aborted.

Export operator: A producer-type operator that emulates some of the functions of the Teradata FastExport utility in the Teradata PT infrastructure.

Extract: The process of copying a subset of data from a source to a target environment.

Exit Routines: Specifies a predefined action to be performed whenever certain significant events occur during a Teradata MultiLoad job.

failure: Any condition that precludes complete processing of a Teradata SQL statement. Any failure will abort the current transaction.

FastExport: Teradata FastExport utility. A program that quickly transfers large amounts of data from tables and views of the Teradata Database to a client-based application.

FastExport OUTMOD Adapter operator: A consumer-type operator that acts as a “wrapper” for Teradata FastExport utility OUTMOD routines, allowing you to use them within the Teradata PT infrastructure.

FastLoad: Teradata FastLoad utility. A program that loads empty tables on the Teradata Database with data from a network-attached or channel-attached client.

FastLoad INMOD Adapter operator: A Teradata PT producer-type operator that acts as a “wrapper” for Teradata FastLoad utility INMOD routines, allowing you to use them within the Teradata PT infrastructure.

field: The basic unit of information stored in the Teradata Database. A field is either null, or has a single numeric or string value. See also column, database, row, table.

FIFO: First In first out queue.

FIPS: Federal Information Processing Standards

Flat File As a noun, an ASCII text file consisting of records of a single type, in which there is no embedded structure information governing relationships between records.
As an adjective, describes a flattened representation of a database as single file from which the structure could implicitly be rebuilt.

A particular type of database structure, as opposed to relational.

**Foreign Key:** The primary key of a parent data subject that is placed in a subordinate data subject. Its value identifies the data occurrence in the parent data subject that is the parent of the data occurrence in the subordinate data subject.

**Formatted Records:** See Records.

**Function:** User Defined Functions (UDF) are extensions to Teradata SQL. Users can write UDFs to analyze and transform data already stored in their data warehouse in ways that are beyond the functionality of Teradata's native functions.

**Gateway:** A device that connects networks having different protocols.

**global rule:** Object Access and Query Resource rules can be specified as being global, that is, they apply to all objects, and therefore to all requests. When a rule is specified as being global, no query objects need be (or can be) associated with the rule because all objects are implicitly included. Take care to define a global access rule, as it causes all requests to be rejected except those from the DBC user and any bypassed objects.

**Globally Distributed Objects (GDO):** A data structure that is shared by all of the virtual processors in the Teradata Database system configuration.

**graphical user interface (GUI):** The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under a Windows operating system. The GUI displays certain icons, buttons, dialog boxes in its windows on the screen and the user controls it by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse. This contrasts with a command line interface where communication is by exchange of strings of text.

**GSS:** Generic Security Services. An application level interface (API) to system security services. It provides a generic interface to services which may be provided by a variety of different security mechanisms. Vanilla GSS-API supports security contexts between two entities (known as "principals").

**heuristics:** Statistics recommendations, based on general rules of thumb.

**HOSI:** Acronym for hash-ordered secondary index.

**IPT:** I/Os Per Transaction
**import:** This refers to the process of pulling system information into a program. To add system information from an external source to another system. The system receiving the data must support the internal format or structure of the data.

**Import Task:** A task that quickly applies large amounts of client data to one or more tables or views on the Teradata Database. Composed of four major phases: Preliminary, Acquisition, Application, and End. The phases are a collection of one or more transactions that are processed in a predefined order according to the Teradata MultiLoad protocol. An import task references up to five target tables.

**In-Doubt:** A transaction that was in process on two or more independent computer processing systems when an interruption of service occurred on one or more of the systems. The transaction is said to be in doubt because it is not known whether the transaction was successfully processed on all of the systems.

**Information engineering:** The discipline for identifying information needs and developing information systems that produce messages that provide information to a recipient. Information engineering is a filtering process that reduces masses of data to a message that provides information.

**INMOD:** INput MODule, a program that administrators can develop to select, validate, and preprocess input data.

**INMOD Routine:** User-written routines that Teradata MultiLoad and other load/export utilities use to provide enhanced processing functions on input records before they are sent to the Teradata Database. Routines can be written in C language (for network-attached platforms), or SAS/S, COBOL, PL/I or Assembler (for channel-attached platforms). A routine can read and preprocess records from a file, generate data records, read data from other database systems, validate data records, and convert data record fields.

**inner join:** In Teradata SQL, a join operation on two or more tables, according to a join condition, that returns the qualifying rows from each table.

**instance:** In object-oriented programming, refers to the relationship between an object and its class. The object is an instance of the class.

**interface processor (IFP):** Used to manage the dialog between the Teradata Database and the host. Its components consist of session control, client interface, the parser, the dispatcher, and the BYNET. One type of IFP is a communications processor (COP). A COP contains a gateway process for communicating with workstations using a network.

**Intermediary:** A computer software process written by a third party which interfaces to one or more Teradata servers and initiates a change data capture operation with replication services.

**internet protocol (IP):** Data transmission standard; the standard that controls the routing and structure of data transmitted over the Internet.

**interval histogram:** Interval histograms are a form of synopsis data structure. A synopsis data structure is a data structure that is substantially smaller than the base data it represents. Interval histograms provide a useful statistical profile of attribute values that characterize the
properties of that raw data. The Teradata Database uses interval histograms to represent the cardinalities and certain other statistical values and demographics of columns and indexes for all-AMPs sampled statistics and for full-table statistics. Each histogram is composed of a maximum of 100 intervals.

I/O: Input/output.

ISO: International Standards Organization

J

JES: Job Entry Subsystem (JES) is an MVS subsystem of the OS/390 and z/OS mainframe operating systems that manages jobs (units of work) that execute on the system. Each job is described to the operating system by system administrators or other users in job control language (JCL). There are two versions, JES2 and JES3. JES3 allows central control of the processing of jobs using a common work queue. Both OS/390 and z/OS provide an interactive menu for initiating and managing jobs.

JCL: Job Control Language is a language for describing jobs (units of work) to the OS/390, z/OS, and VSE operating systems, which run on IBM’s OS/390 and z800/900 large server (mainframe) computers. These operating systems allocate their time and space resources among the total number of jobs that have been started in the computer. Jobs in turn break down into job steps. All the statements required to run a particular program constitute a job step. Jobs are background (sometimes called batch) units of work that run without requiring user interaction (for example, print jobs). The operating system manages interactive (foreground) user requests that initiate units of work. In general, foreground work is given priority over background work.

JIS: Japanese Industrial Standards specify the standards used for industrial activities in Japan. The standardization process is coordinated by Japanese Industrial Standards Committee and published through Japanese Standards Association.

Job Script: A job script, or program, is a set of Teradata MultiLoad commands and Teradata SQL statements that make changes to specified target tables and views in the Teradata Database. These changes can include inserting new rows, updating the contents of existing rows, and deleting existing rows.

join: A select operation that combines information from two or more tables to produce a result.

L

LAN: Local Area Network. LANs supported by Teradata products must conform to the IEEE 802.3 standard (Ethernet LAN).

Load operator: A Teradata PT consumer-type operator that emulates some of the functions of the Teradata FastLoad utility in the Teradata PT infrastructure.
LOB: An acronym for large object. A large object is a database object that is large in size. LOBs can be up to 2 gigabytes. There are two types of LOBs, CLOBs and BLOBs. CLOBs are character-based objects, BLOBs are binary-based objects.

Locks: Teradata FastLoad automatically locks any table being loaded and frees a lock only after an END LOADING statement is entered. Therefore, access to a table is available when FastLoad completes.

log: A record of events. A file that records events. Many programs produce log files. Often you will look at a log file to determine what is happening when problems occur. Log files have the extension “.log”.

log stream: A log stream is a series of log messages defined in one message catalog and initiated from one originator. One originator may initiate several log streams (for example, if there are multiple operators in one originator).

logical action: A named action that is defined on the Alert Policy Editor’s Actions tab. Logical actions can be assigned to events in the alert policy.

Logical Data Model: A data model that represents the normalized design of data needed to support an information system. Data are drawn from the common data model and normalized to support the design of a specific information system.

Actual implementation of a conceptual module in a database. It may take multiple logical data models to implement one conceptual data model.

loner value: A value that has a frequency greater than the total number of table rows divided by the maximum interval times 2.

MAPI: Messaging Application Programming Interface. A set of Microsoft-defined functions and interfaces that support Email capabilities.

macro: A file that is created and stored on the Teradata Database, and is executed in response to a Teradata SQL EXECUTE statement

merge join: In Teradata SQL, the type of join that occurs when the WHERE conditional of a SELECT statement causes the system first to sort the rows of two tables based on a join field (specified in the statement), then traverse the result while performing a merge/match process.

Metadata: Data about data. For example, information about where the data is stored, who is responsible for maintaining the data, and how often the data is refreshed.

methods: In object-oriented programming, methods are the programming routines by which objects are manipulated.

NFS: Network file system.

MIB: Management Information Base
**MOSI:** Micro Operating System Interface. A library of routines that implement operating system dependent and protocol dependent operations on the workstation.

**MTDP:** Micro Teradata Director Program. A library of routines that implement the session layer on the workstation. MTDP is the interface between CLI and the Teradata Database.

**MPP:** Massively Parallel Processing

**multi-threading:** An option that enables you to speed up your export and import operations with multiple connections.

**MultiLoad:** Teradata MultiLoad. A command-driven utility that performs fast, high-volume maintenance functions on multiple tables and views of the Teradata Database.

**Multiset Tables:** Tables that allow duplicate rows.

**MVS (Multiple Virtual Storage):** One of the primary operating systems for large IBM computers.

**name:** A word supplied by the user that refers to an object, such as a *column, database, macro, table, user, or view*.

**nested join:** In Teradata SQL, this join occurs when the user specifies a field that is a unique primary index on one table and which is in itself an index (unique/non-unique primary or secondary) to the second table.

**Network:** In the context of the Teradata Database, a LAN (see LAN).

**network attached:** A computer that communicates over the LAN with a server (for example, a Teradata Database).

**NIC:** Network Interface Card.

**NO REWIND:** A tape device definition that prevents a rewind operation at either file open or file close. NO REWIND allows a program to access multiple files on a tape by leaving the tape positioned at the end of the current file at close, thus allowing the subsequent file to be easily accessed by the next open.

**notify exit:** A user-defined exit routine that specifies a predefined action to be performed whenever certain significant events occur during a Teradata MultiLoad job. For example, by writing an exit in C (without using CLIv2) and using the NotifyExit attribute in an operator definition, you can provide a routine to detect whether a Teradata MultiLoad job succeeds or fails, how many records were loaded, what the return code is for a failed job, and so on.

**null:** The absence of a value for a field.

**Nullif Option:** This option allows the user to null a column in a table under certain conditions; it is only used in conjunction with DEFINE statements.
NUPI: Non-unique primary index; an NUPI is typically assigned to minor entities in the database.

NUSI: Non-unique secondary index; an NUSI is efficient for range query access, while a unique secondary index (USI) is efficient for accessing a single value.

Object: In object-oriented programming, a unique instance of a data structure defined according to the template provided by its class. Each object has its own values for the variables belonging to its class and can respond to the messages, or methods, defined by its class.

Object access rule: An Object Access filter allows you to define the criteria for limiting access to issuing objects and/or query objects. Queries that reference objects associated with the rule (either individually or in combination) during the specified dates and times are rejected. Global rules are not applicable for this type.

Object definition: The details of the structure and instances of the objects used by a given query. Object definitions are used to create the tables, views, and macros, triggers, join indexes, and stored procedures in a database.

ODBC: (Open Database Connectivity) Under ODBC, drivers are used to connect applications with databases. The ODBC driver processes ODBC calls from an application, but passes SQL requests to the Teradata Database for processing.

OLTP: (On-Line Transaction Processing) Processing that supports the daily business operations. Also known as operational processing.

Operator routine: In object-oriented programming, refers to a function that implements a method.

The terms operator routine and operator function may be used interchangeably.

OS/390 Operating System 390
OS/VS Operating System/Virtual Storage

Outer join: In Teradata SQL, an extension of an inner join operation. In addition to returning qualifying rows from tables joined according to a join condition (the inner join), an outer join returns non-matching rows from one or both of its tables. Multiple tables are joined two at a time.

Owner: In Teradata SQL, the user who has the ability to grant or revoke all access rights on a database to and from other users. By default, the creator of the database is the owner, but ownership can be transferred from one user to another by the GIVE statement.

Parameter: A variable name in a macro for which an argument value is substituted when the macro is executed.
**parser:** A program executing in a PE that translates Teradata SQL statements entered by a user into the steps that accomplish the user’s intentions.

**parsing engine (PE):** An instance (virtual processor) of the database management session control, parsing, and dispatching processes and their data context (caches).

**Paused MultiLoad Job:** A job that was halted, before completing, during the Acquisition Phase of the Teradata MultiLoad operation. The paused condition can be intentional, or the result of a system failure or error condition.

**PDE:** Parallel Database Extensions

**peak perm:** Highest amount of permanent disk space, in bytes, used by a table.

**performance groups:** A performance group is a collection of parameters used to control and prioritize resource allocation for a particular set of Teradata Database sessions within the Priority Scheduler. Every Teradata Database session is assigned to a performance group during the logon process. Performance groups are the primary consideration in partitioning the working capacity of the Teradata Database. To learn more about performance groups, see the Priority Scheduler section of Utilities.

**performance periods:** A threshold or limit value that determines when a session is under the control of that performance period. A performance period links PGs/Teradata Database sessions under its control to an AG that defines a scheduling strategy. A performance period allows you to change AG assignments based on time-of-day or resource usage.

**Physical Data Model:** A data model that represents the denormalized physical implementation of data that support an information system. The logical data model is denormalized to a physical data model according to specific criteria that do not compromise the logical data model but allow the database to operate efficiently in a specific operating environment.

**Primary server:** A Teradata server in which client applications execute transactions through use of Teradata SQL or utilities such as Teradata MultiLoad and update the tables of one or more replication groups. The changes are captured by replication services and given to an intermediary connected to the server.

**priority definition set:** A collection of data that includes the resource partition, performance group, allocation group, performance period type, and other definitions that control how the Priority Scheduler manages and schedules session execution.

**product join:** In Teradata SQL, the type of join that occurs when the WHERE conditional of a SELECT statement causes the Teradata Database system to compare all qualifying rows from one table to all qualifying rows from the other table. Because each row of one table is compared to each row of another table, this join can be costly in terms of system performance. Note that product joins without an overall WHERE constraint are considered unconstrained (Cartesian). If the tables to be joined are small, the effect of an unconstrained join on performance may be negligible, but if they are large, there may be a severe negative effect on system performance.
profiles: A profile is a set of parameters you assign to a user, group of users, or an account that determines what scheduling capabilities are available and how your Teradata Query Scheduler scheduled requests servers handles their scheduled requests.

physical action: A basic action type, such as <Send a Page>, <Send an EMail>, and so on. Physical actions must be encapsulated by logical actions to be used in the alert policy.

PIC: Position independent code

PL/I: Programming Language/1, a programming language supported for Teradata MultiLoad development.

PMPC: Performance Monitor and Production Controls

PP2: Preprocessor2

Primary Key: A set of one or more data characteristics whose value uniquely identifies each data occurrence in a data subject. A primary key is also known as a unique identifier.

privilege: A user’s right to perform the Teradata SQL statements granted to him against a table, database, user, macro, or view. Also known as access right.

procedure: Short name for Teradata stored procedure. Teradata provides Stored Procedural Language (SPL) to create stored procedures. A stored procedure contains SQL to access data from within Teradata and SPL to control the execution of the SQL.

producer: A type of operator that retrieves data from an external data store, such as a file, Teradata Database table, and so on, and provides it to other operators. A producer operator produces the data into the data stream’s buffer.

production system: A database used in a live environment. A system that is actively used for day-to-day business operations. This differs from a test or development system that is used to create new queries or test new features before using them on the production system.

Protocol: The rules for the format, sequence, and relative timing of messages exchanged on a network.

query analysis: A feature that estimates the answer set size (number of rows) and processing time of a SELECT type query.

Query Capture Database (QCD): A database of relational tables that store the steps of any query plan captured by the Query Capture Facility (QCF).

Query Capture Facility (QCF): Provides a method to capture and store the steps from any query plan in a set of predefined relational tables called the Query Capture Database (QCD).

query: A Teradata SQL statement, particularly a SELECT statement.

query management: The primary function of DWM is to manage logons and queries. This feature examines logon and query requests before they are dispatched for execution within the
Teradata Database, and may reject logons, and may reject or delay queries. It does this by comparing the objects referenced in the requests to the types of DBA-defined rules.

**Query Resource filter:** A Query Resource filter allows you to define the criteria for limiting resource usage associated with queries. You can define resource criteria such as:

- Row count
- Processing time
- No joins permitted
- No full table scans permitted

Queries that are estimated to meet or exceed the limits for the rule during the specified dates and times are rejected. You may define global rules for this type.

**Query Session Utility:** A separate utility program used to monitor the progress of your Teradata MultiLoad job. It reports different sets of status information for each phase of your job.

**random AMP sample (RAS):** An arbitrary sample from an Access Module Processor (AMP). These are samples of the tables in a query or all of the tables in a given database. Also known as RAS.

**RDBMS (Relational Database Management System):** A database management system in which complex data structures are represented as simple two-dimensional tables consisting of columns and rows.

**Records:** When using the Teradata MultiLoad utility, both formatted and unformatted records are accepted for loading. A formatted record, in the Teradata Database world, consists of a record created by a Teradata Database utility, such as BTEQ, where the record is packaged with begin- and end-record bytes specific to the Teradata Database. Unformatted records are any records not originating on a Teradata Database, such as Lotus 1-2-3 files. These files contain records that must be defined before loading onto the Teradata Database.

**recursive query:** A named query expression that is allowed to reference itself in its own definition, giving the user a simple way to specify a search of a table using iterative self-join and set operations. Use a recursive query to query hierarchies of data. Hierarchical data could be organizational structures such as department and sub-department, forums of discussions such as posting, response, and response to response, bill of materials, and document hierarchies.

**Replication Group:** A set of tables for which data changes are being captured on a primary server or applied on a subscriber server.

**Replication Services:** A set of software functions implemented in the Teradata server that interact with an intermediary to capture data on the tables of a replication group.

**request:** In host software, a message sent from an application program to the Teradata Database.
resource partition: A collection of prioritized PGs related by their users’ associations. Has an assigned weight that determines the proportion of resources available to that partition relative to the other partitions defined for that Teradata Database.

Restart Log Table: One of four restart tables the Teradata MultiLoad utility creates that are required for restarting a paused Teradata MultiLoad job.

Restoration Lock: A flag set in the table header of a target table indicating that the table was aborted during the Application Phase and is now ready to be restored. A limited set of operations can be done on the table: Delete All, Drop Fallback, Drop Index, Drop Table, and Select with access lock. No Teradata MultiLoad restart will be allowed on a table with a Restoration Lock.

result: The information returned to the user to satisfy a request made of the Teradata Database.

results table/file: In the Schedule Request environment, a results table or file is a database table or a Windows file into which result data for a schedule request that is not self-contained are stored.

results file storage: A symbolic name to a root directory where scheduled requests results are stored. You map a file storage location to a Windows root directory where results are stored.

RowID join: In Teradata SQL, this join occurs when one of the join tables has a non-unique primary index constant, and another column of that table matches weakly with a non-unique secondary index column of the second table.

rule: Rules are the name given to the method used by DWM to define what requests are prohibited from being immediately executed on the Teradata Database. That is, the rules enforced by DWM provide the Query Management capabilities.

Routing: A general term that describes how Teradata Query Director receives sessions and sends them to one system or another.

Routing Configuration File: The routing configuration file in Teradata Query Director allows administrators to associate specific userids and account strings to specific systems.

row: Whether null or not, that represent one entry under each column in a table. The row is the smallest unit of information operated on by data manipulation statements.

RowID join: In Teradata SQL, this join occurs when one of the join tables has a non-unique primary index constant, and another column of that table matches weakly with a non-unique secondary index column of the second table.

RSG: Relay Services Gateway. A virtual processor residing on a node in which the replication services software will execute.

RT: Response Time

RTF: Rich Text File
run file: A script that is not contained within the SYSIN file, but rather executed through use of the .RUN BTEQ command.

scheduled requests: The capability to store scripts of SQL requests and execute them at a scheduled later time.

schema: Schemas are used to identify the structure of the data. Producers have an output schema, to define what the source data will look like in the data stream. Consumers have an input schema, to define what will be read from the data stream. If the input and output schemas are the same, you only define the schema once.

script: A file that contains a set of commands and/or SQL statements.

Security token: A binary string generated by a server when a replication group is created or altered that must be inputted to secure a change data capture or apply operation.

self-contained statement: A query request that stores the result data that it generates, if any. For example, an INSERT/SELECT statement would be self-contained, whereas a SELECT statement would not.

separator: A character or group of characters that separates words and special symbols in Teradata SQL. Blanks and comments are the most common separators.

server: A computer system running the Teradata Database. Typically, a Teradata Database server has multiple nodes, which may include both TPA and non-TPA nodes. All nodes of the server are connected using the Teradata BYNET or other similar interconnect.

session: In client software, a logical connection between an application program on a host and the Teradata Database that permits the application program to send one request to and receive one response from the Teradata Database at a time.

skew: This value is calculated based on a single Database collection interval. If the Session Collection rate is 60, then the skew is calculated for a 60-second period.

The value is calculated using 'current' data values. For example, the Max CPU used during the past 60 seconds relative to the Average used over that same 60 seconds:

\[ \text{skew} = 100 \times (1 - \frac{\text{avg}}{\text{max}}) \]

SMP: Symmetric Multi-Processing


Source Database: The database from which data will be extracted or copied into the Data Warehouse.

SQL: Structured Query Language. An industry-standard language for creating, updating and, querying relational database management systems. SQL was developed by IBM in the 1970s for use in System R. It is the de facto standard as well as being an ISO and ANSI standard. It is often embedded in general purpose programming languages.
Programming language used to communicate with the Teradata Database.

**SQL Inserter operator:** A consumer-type operator that accepts data from other operators using the Terdata PT operator interface.

**SQL Selector operator:** A producer-type operator that submits a Teradata SQL SELECT statement to the Teradata Database.

**SSO:** Single sign-on, an authentication option that allows users of the Teradata Database on Windows 2000 systems to access the Teradata Database based on their authorized network usernames and passwords. This feature simplifies the procedure requiring users to enter an additional username and password when logging on to Teradata Database with client applications.

**stand-alone operator:** A type of operator that does not exchange data with other operators.

**Star Schema:** A modeling scheme that has a single object in the middle connected to a number of objects around it radially.

**statement:** A request for processing by the Teradata Database that consists of a keyword verb, optional phrases, operands and is processed as a single entity.

**statistics:** These are the details of the processes used to collect, analyze, and transform the database objects used by a given query.

**stored procedure:** A stored procedure is a combination of SQL statements and control and conditional handling statements that run using a single call statement.

**Stream operator:** A consumer-type operator that allows parallel inserts, updates, and deletes to new or preexisting Teradata tables.

**Subscriber server:** A Teradata server in which changes captured from a primary server by an intermediary are applied to tables that duplicate those of the primary. Replication services executing on the servers provide the capture and apply functions.

**supervisory user:** In Data Dictionary, a user who has been delegated authority by the administrator to further allocate Teradata Database resources such as space and the ability to create, drop, and modify users within the overall user community.

**T**

**table:** A set of one or more columns with zero or more rows that consist of fields of related information.

**Target Database:** The database in which data will be loaded or inserted.

**Target table:** A user table where changes are to be made by a Teradata MultiLoad task.

**TCP/IP:** Transmission Control Protocol/Internet Protocol.

**TDPID:** Teradata Director Program Identifier. The name of the Teradata Database being accessed.
**Teradata SQL:** The Teradata Database dialect of the relational language SQL, having data definition and data manipulation statements. A data definition statement would be a CREATE TABLE statement and a data manipulation statement would be a data retrieval statement (a SELECT statement).

**TDP:** Teradata Director Program; TDP provides a high-performance interface for messages communicated between the client and the Teradata system.

**Target Level Emulation (TLE):** Permits you to emulate a target environment (target system) by capturing system-level information from that environment. The captured information is stored in the relational tables `SystemFE.Opt_Cost_Table` and `SystemFE.Opt_RAS_Table`. The information in these tables can be used on a test system with the appropriate column and indexes to make the Optimizer generate query plans as if it were operating in the target system rather than the test system.

**test system:** A Teradata Database where you want to import Optimizer-specific information to emulate a target system and create new queries or test new features.

**title:** In Teradata SQL, a string used as a column heading in a report. By default, it is the column name, but a title can also be explicitly declared by a TITLE phrase.

**TPA:** Trusted Parallel Application.

**TOS:** Teradata Operating System

**TPM:** Transactions Per Minute

**Transport:** The process of extracting data from a source, interfacing with a destination environment, and then loading data to the destination.

**transaction:** A set of Teradata SQL statements that is performed as a unit. Either all of the statements are executed normally or else any changes made during the transaction are backed out and the remainder of the statements in the transaction are not executed. The Teradata Database supports both ANSI and Teradata transaction semantics.

**trigger:** One or more Teradata SQL statements associated with a table and executed when specified conditions are met.

**TSM:** Tivoli Storage Management; IBM’s storage management solution.

**TTU:** Teradata Tools and Utilities is a robust suite of tools and utilities that enables users and system administrators to enjoy optimal response time and system manageability with the Teradata system.

**tuple:** In a database table (relation), a set of related values one for each attribute (column). A tuple is stored as a row in a relational database management system. It is analogous to a record in a nonrelational file.

**Two Phase Commit:** Two Phase Commit is the process by which a relational database ensures that distributed transactions are performed in an orderly manner. In this system, transactions may be terminated by either committing them or rolling them back.
**type:** An attribute of a column that specifies the representation of data values for fields in that column. *Teradata SQL* data types include numerics and strings.

**UDF**  User-Defined Functions

**UDM**  User-Defined Methods. The database developer can create custom functions that are explicitly connected to UDTs; these are known as UDMs. Functionalities directly applicable to a UDT can be located within the UDMs associated with that UDT rather than being replicated to all of the applications that use that UDT, resulting in increased maintainability.

**UDT**  A custom data type, known as a user-defined type. By creating UDTs, a database developer can augment the Teradata Database with data types having capabilities not offered by Teradata predefined (built-in) data types. Use Teradata MultiLoad to import values into tables containing UDT columns in the same manner as is done for other tables. The input records to Teradata MultiLoad must have the column data for UDT columns in its external type format.

**Unformatted Records:**  See Records.

**Unicode:**  A fixed-width (16 bits) encoding of virtually all characters present in all languages in the world.

**unique secondary index (USI):**  One of two types of secondary indexes. A secondary index may be specified at table creation or at any time during the life of the table. It may consist of up to 16 columns. To get the benefit of the index, the query has to specify a value for all columns in the secondary index. A USI has two purposes: It can speed up access to a row which otherwise might require a full table scan without having to rely on the primary index, and it can be used to enforce uniqueness of a column or set of columns.

**user:**  In *Teradata SQL*, a database associated with a person who uses the Teradata Database. The database stores the person’s private information and accesses other Teradata Databases.

**Update operator:**  A consumer-type operator that emulates some of the functions of the Teradata MultiLoad utility in the infrastructure.

**UPI:**  Unique primary index; a UPI is required and is typically assigned to major entities in the database.

**user groups:**  A group of users can be specified as either as a collection of individual users, or as all user names which satisfy a character string pattern (such as SALE*). The Teradata concept of roles is not used to define user groups, as it applies to privileges. User groups can generally be employed wherever an issuing object can be specified, and any condition applied to a group implicitly applies to all users within that group.

**UTF8:**  In simple terms, UTF8 is an 8-bit encoding of 16-bit Unicode to achieve an international character representation.

In more technical terms, in UTF8, characters are encoded using sequences of 1 to 6 octets. The only octet of a sequence of one has the higher-order bit set to 0, the remaining 7 bits are used
to encode the character value. UTF8 uses all bits of an octet, but has the quality of preserving the full US-ASCII range. The UTF8 encoding of Unicode and UCS avoids the problems of fixed-length Unicode encodings because an ASCII file encoded in UTF is exactly same as the original ASCII file and all non-ASCII characters are guaranteed to have the most significant bit set (bit 0x80). This means that normal tools for text searching work as expected.

**UTF16** A 16-bit Unicode Translation Format.

**value-ordered secondary index (VOSI):** A non-unique secondary index (NUSI) can be value-ordered, which means the NUSI can be sorted on the key values themselves rather than on the corresponding hash codes. This is useful for range queries where only a portion of the index subtable will be accessed. With a value-ordered NUSI, only those blocks in the NUSI subtable that are within the range are scanned. It must be a number value, up to 4 bytes, versus a longer character column. DATE is the most commonly used data type. The actual data value is stored as part of the NUSI structure.

**Varbyte:** A data type that represents a variable-length binary string.

**Varchar:** A data type that represents a variable-length non-numeric character.

**Vargraphic:** A data type that represents a variable-length string of characters.

**view:** An alternate way of organizing and presenting information in a Teradata Database. A view, like a table, has rows and columns. However, the rows and columns of a view are not directly stored by the Teradata Database. They are derived from the rows and columns of tables (or other views) whenever the view is referenced.

**VM (Virtual Machine):** One of the primary operating systems for large IBM computers.

**VM/CMS** Virtual Machine/Conversational Monitor System

**workgroups:** Workgroups represent collections of related scheduled request work for users, user groups, or accounts. Each workgroup is assigned a maximum number of requests that can be executing from that workgroup simultaneously, thereby ensuring that requests for all workgroups get a fair share of their scheduled work done within the execution timeframes.

**workload limits rule** A workload limits rule allows you to limit the number of logon sessions and all-AMP queries, as well as reject or delay queries when workload limits are encountered. You can define which users, accounts, performance groups, or users within performance groups that are associated with this type of rule.

**Workstation:** A network-attached client.

**Work Table:** A table created during the Preliminary Phase used to store intermediate data acquired from the host during an MLOAD task. These data will eventually be applied to a target table.
**Write Lock:** A write lock enables a single user to modify a table. The MultiLoad utility maintains write locks against each target table during the Application Phase, and work tables and error tables for each task transaction.

**XML:** XML is the eXtensible Markup Language—a system created to define other markup languages. For this reason, it can also be referred to as a metalanguage. XML is commonly used on the Internet to create simple methods for the exchange of data among diverse clients.
Symbols

& (ampersand) character, in variable substitution 52
* (asterisk character) as the fileid specification
  ACCEPT command 93
  DISPLAY command 112
  ROUTE MESSAGES command 166
  RUN FILE command 168
  .* default VALUES clause, INSERT statement 144
  ./ prefix
    EXIT name specification, BEGIN MLOAD command 98, 140

A

abort termination 39
abort, defined 239
ACCEPT command
  defined 21
  syntax 92
  with multibyte character sets 58
access
  locks, programming considerations 65
  privileges, required
    DELETE statement 110
    INSERT statement 145
    RELEASE MLOAD command 163
    UPDATE statement 175
access module support 137
accounts
  defined 239
acctid specification, LOGON command 155
acquisition error table 86
  format 86
  implications of dropping 40, 41
  nonshareability 103
acquisition locks, programming considerations 65
acquisition phase, description 16
addressing mode, INMOD routines 78
administrator, defined 240
aggregate operators, programming considerations 67
all joins
  defined 240
ALL keyword, AMPCHECK specification, BEGIN MLOAD command 96
ALTER TABLE statement, Teradata SQL 22
alternate error file runtime parameter 32
AMPCHECK specification, BEGIN MLOAD command 96
ampersand ( & ) characters, in variable substitution 52
AMPS
  down 63
  nonparticipant 64
AND keyword, DROP specification, FIELD command 119
ANSI/SQL DateTime data types
  INSERT statement 145
  programming considerations 53
ANSIDATE specification, DATEFORM command 108
AP reset conditions, restarting after 45
API
  defined 241
apostrrophes, displaying in the text string, DISPLAY command 113
application error table 87
  format 87
  implications of dropping 40, 42
  nonshareability 103
application locks, programming considerations 66
application phase, description 16
APPLY
  clauses, multiple 143
  keyword, AMPCHECK specification, BEGIN MLOAD command 96
  label specification, IMPORT command 137
ApplySeq column
  acquisition error table 86
  application error table 87
arithmetic functions, programming considerations 67
Assembler INMOD
  MVS Assembler INMOD example 195
  programming structure 70
AXSMOD character set 55
AXSMOD option, IMPORT command 137

B

-b runtime parameter 32
  UNIX example 188
batch mode, syntax for invoking 30, 31
BEGIN DELETE MLOAD command
  defined 21
  syntax 95
BEGIN MLOAD command
  defined 21
  syntax 95
BINARY format specification, IMPORT command 139
Index

BLANKS keyword, DROP specification, FIELD command 119
body data record area, MultiLoad/INMOD interface 74
BRIEF runtime parameter 32, 46
  MVS example 184
  VM example 181
bypass objects
defined 242

C
- notify exit parameters 217
  -c character-set-name runtime parameter 32, 33, 47, 55
  UNIX example 190
C INMOD examples
  for MVS 200
  for UNIX 215
C language INMODs
  programming structure 71
character sets
AXSMOD 55
client system specifications 55
default 55
  programming considerations 54
runtime parameters 32, 33, 47, 55
  MVS example 185
  UNIX example 190
  VM example 182
site-defined 57
supported 17
Teradata Database default 55
character-to-date data conversions 20
character-to-numeric
data conversions 20
charpos specifications
ACCEPT command 92
RUN FILE command 167
CHARSET runtime parameter 46
CHARSET= character-set-name runtime parameter 32, 33, 47, 55
  MVS example 185
  VM example 182
CHECKPOINT
  rate specification, BEGIN MLOAD command 97
  statement, Teradata SQL 22
checkpoints
description 20
  INMOD rules and restrictions 78
cleanup phase, description 16
client system failure, restarting after 44
cname specification
INSERT statement 144
UPDATE statement 174

COBOL INMOD
  example 204, 207
  for MVS 207
  programming structure 71
COLLECT STATISTICS statement, Teradata SQL 23
  columns
    maximum number 81
    names in referenced tables, TABLE command 172
value specifications, import tasks 61
command conventions
  conditional expressions 48
  operators 48
reserved words 49
table column names 49
commands
  conventions 48
  functions 21
  intermixing 122, 131
  syntax, see individual commands
COMMENT statement, Teradata SQL 23
comments
  programming considerations 53
  using variable substitutions 54
  using with Teradata SQL statements 54
compiling routine
  HP-UX 221
  IBM-AIX 224
  LINUX 225
  MP-RAS 220, 221
  SOLARIS 220, 225
concatenating a field with a filler, example 131
concurrent load utility tasks, programming considerations 67
condition variable
  CONTINUEIF specification, LAYOUT command 148, 149
  IMPORT command 135, 137
WHERE specification, DELETE statement 109
WHERE specification, UPDATE statement 174
conditional expressions 48
  IF command 133
configuration file
  character set specification 55
  contents 47
  file name and location 46
  optional specification 29
  overriding internal utility defaults 46
parameters overridden by run-time parameters
  CHARSET 32
parameters overridden by runtime parameters 46
  BRIEF 32
  CHARSET 33, 47, 55
  ERRLOG 32
  MAXSESS 32
  MINSESS 32
processing by the MultiLoad utility 47
using 46
CONTINUEIF
  condition phrase, LAYOUT command 148, 149
conditions, with multibyte character sets 58
conversions
  character-to-numeric 20
  integer-to-decimal 20
  numeric-to-numeric 20
CREATE DATABASE statement, Teradata SQL 23
CREATE MACRO statement, Teradata SQL 23
CREATE TABLE statement, Teradata SQL 23
CREATE VIEW statement, Teradata SQL 23

D
data conversions
  capabilities 19
  capabilities, limiting factors 49
  character-to-date 20
  character-to-numeric 20
  date-to-character 20
  integer-to-decimal 20
  numeric-to-numeric 20
data dictionary, defined 244
data encryption
  runtime option specification 36
data file concatenation, programming considerations 67
data manipulation, defined 245
data source restrictions, import tasks 61
data type specification, changing
  not allowed with FIELD command 122
  SET command 169
database names
  with multibyte character sets 58, 59
DATABASE statement, Teradata SQL 23
data desc specifications
  FIELD command 59, 119
  FILLER command 130
DATABENCRIPTION keyword
  BEGIN MLOAD command 97
DATE system variables (&SYSDATE and &SYSDATE4) 51
DATEFORM command
  defined 21
  syntax 108
date-to-character data conversions 20
DAY system variable (&SYSDAY) 51
DBErrorCode column, application error table 87
DBCErrorField column, application error table 87
dbname specification
  FROM keyword, DELETE statement 109
dbname specifications
  BEGIN MLOAD command 97, 107
  LOGTABLE command 158, 159
  RELEASE MLOAD command 162
DBQL
  defined 245
DDL statements
  programming considerations 61, 65, 66
dbname specification
  IMPORT command 139
DECIMAL data type specification, DEFINE command
  input length and field description 124
decimal, zoned 20
DELCNT system variable (&SYSDELCNT) 51
DELETE
  keyword
    BEGIN DELETE MLOAD command 97
    DML LABEL command 114
  statement, Teradata SQL 23, 109
    using in delete tasks 110
    using in import tasks 110
DELETE DATABASE statement, Teradata SQL 23
delete task
  archive/recovery statements 63
  locks 63
  programming considerations 62
  specifying index 63
delete tasks
  description 15
  invalid command parameters 63
  MultiLoad job script example 236
  programming considerations 62
  utility statements 63
delimiter characters, in comments 53
delimiters
  defined 246
DISPLAY command
  defined 21
  syntax 112
DISPLAY ERRORS option, FORMAT specification,
  IMPORT command 138
DIT
  defined 246
DML
  commands, specifying, INSERT statement 145
  label names, with multibyte character sets 58
  transaction phase, description 16
DML LABEL command
  defined 21
  syntax 114
  with multibyte character sets 58
DML statements
  programming considerations 65, 66
  specification WARNING 24
DMLSeq column
  acquisition error table 86
  application error table 87
DO INSERT FOR option, DML LABEL command 114
ROWS specification 116
DROP
  keyword, FIELD command 119
  statements, import tasks 61
DROP DATABASE statement, Teradata SQL 23
DUMP statements
  delete tasks 63
  import tasks 62
DUPLICATE keyword, DML LABEL command 114
DYNAMN entry point
  for COBOL and PL/I notify exit routines 72
dynamn entry point
  for C notify exit routines 72
  for SAS/C notify exit routines 72

E
-e filename runtime parameter 32
  UNIX example 191
ECHO option, ROUTE MESSAGES command 165
ELSE command
  defined 21
  syntax 133
END MLOAD command
  defined 22
  syntax 118
ENDIF command
  defined 21
  syntax 133
env_var environment variable specification
  ACCEPT command 92
ENVIRONMENT keyword, ACCEPT command 92
errcount variable, ERRLIMIT specification, BEGIN MLOAD command 95
ERRLIMIT errcount specification, BEGIN MLOAD command 98
ERRLOG runtime parameter 46
ERRLOG=filename runtime parameter 32
  MVS example 186
  VM example 183
error handling 85
error limit not honored 86
error recording 85
error table
  acquisition, format 86
  application, format 87
  column names, not allowed in target tables 106
  format 86, 87
  implications of dropping 40, 41, 42
  nonshareability 103, 107
ErrorCode column, acquisition error table 86
ErrorField column, acquisition error table 87
errors
fieldexpr specifications, FIELD command 120
fieldname specifications
  FIELD command 120
  FILLER command 130
  INSERT statement 144
file
  requirements 29
  size, maximum 66
FILE keyword
  ACCEPT command 92
  DISPLAY command 112
  ROUTE MESSAGES command 165
fileid specifications
  ACCEPT command 93
  DISPLAY command 112
  ROUTE MESSAGES command 165
  RUN FILE command 167
  usage rules for MVS 93, 113, 166, 168
filename specifications, IMPORT command 140
FILLER command
  defined 22
  intermixing with TABLE or FIELD commands 173
  syntax 130
  with multibyte character sets 58
FOR n specification, IMPORT command 138
foreign key references, programming considerations 67
FORMAT . . . input record specification, IMPORT command 139
formats
  acquisition error table 86
  application error table 87
FREE option, IMPORT command 139
  when running under MVS 142
FROM keyword
  ACCEPT command 92
  DELETE statement 109
  IMPORT command 139
G
GIVE statement, Teradata SQL 23
global rule
  defined 248
GRANT statement, Teradata SQL 23
graphic constants, programming considerations 59
GRAPHIC data type specifications
  FIELD command 124
  example 128
  FILLER command 131
  example 132
  programming considerations 59
GSS
  defined 248
H
hexadecimal form, programming considerations 67
HIGH keyword, NOTIFY specification
  BEGIN DELETE MLOAD command 96
  BEGIN MLOAD command 96
HOLD option, IMPORT command 139
  when running under MVS 142
HostData column
  application error table 87
hours variable, TENACITY specification 101
  BEGIN DELETE MLOAD command 96
  BEGIN MLOAD command 95
HP-UX
  compiling and linking routines 221
I
IBM VM and MVS
  entry points for notify exit routines 72
IBM-AIX
  compiling and linking routines 224
IF command
  conditional expression numeric results 133
  defined 21
  nesting 134
  syntax 133
  with multibyte character sets 58
IGNORE keyword
  ACCEPT command 92
  DML LABEL command 115
  RUN FILE command 167
IMPORT command
  defined 22
  syntax 135
IMPORT keyword, BEGIN MLOAD command 98
import tasks
  command restrictions 142
  description 15
  MultiLoad job script example 235
programming considerations 59
  column value specifications 61
  data source restrictions 61
  DROP statements, import tasks 61
  DUMP statements 62, 63
  index specifications 63
  locks 61
  multiple operations 60
  RESTORE statements 62, 63
  ROLLBACK statements 62, 63
  ROLLFORWARD statements 62, 63
  statements supported 60
  target tables 61
  upsert function 60
utility statements 62
ImportSeq column
  acquisition error table 86
  application error table 87
IN APPLY option, RELEASE MLOAD statement 162, 163
index specifications, delete tasks 63
INDICATORS specification, LAYOUT command 149
INFILE keyword, IMPORT command 135, 137
infilename standard input file specification 36
init-string variable, AXSMOD specification, IMPORT command 140
INMOD
  compatibility 78
  modulename specification, IMPORT command 140
  programs, for MultiLoad import tasks 15
  routines
    checkpoint support 78
    compiling and linking 220
    definition 69
    generating for Windows 226
    MultiLoad interface 72
    restart operations--Caution 74
    rules and restrictions 76
    using 69
  sample programs
    Assembler 195
    C, for MVS 200
    C, for UNIX 215
    C, for Windows 227
    COBOL 204, 207
    COBOL, for MVS 207
    PL/I for MVS 209
inner join
  defined 249
input
  files, for MultiLoad import tasks 15
  text, delimiting 94
INSCNT system variable (&SYSINSCNT) 51
INSERT
  keyword, DML LABEL command 114
  statement, Teradata SQL 23, 144
INTEGERDATE specification, DATEFORM command 108
integer-to-decimal
  data conversions 20
INTO keyword, INSERT statement 144
invalid command parameters, delete tasks 63
invoking MultiLoad 29
  MVS example 37
  UNIX and Windows Example 37
  VM example 36

J
JIS
  defined 250
job scripts
  definition 79
  errors, restarting after 43
  procedure 79
  programming considerations
    record mode load anomaly 48
    programming considerations, data conversion factors 49
    writing 79
jobs
  aborts, restarting after 44
  restarting 42
  restarting after a PAUSE ACQUISITION command 43
  join indexes, programming considerations 67
  join, defined 250
  joins, not allowed, DELETE statement target tables 110
  journaling options, programming considerations 67

L
label variable
  APPLY specification, IMPORT command 135, 137
  DML LABEL command 114
LAYOUT command
  defined 22
  syntax 148
  using in more than one task 150
  with multibyte character sets 58
LAYOUT keyword, IMPORT command 135, 137
layoutname specifications
  IMPORT command 140
  LAYOUT command 149
  with multibyte character sets 58
LEADING keyword, DROP specification, FIELD command 119
length value, MultiLoad/INMOD interface 73
linking routine
  HP-US 221
  IBM-AIX 224
  LINUX 225
  MP-RAS 220, 221
  SOLARIS 220, 225
LINUX
  compiling and linking routines 225
load utility/INMOD compatibility
  rules and restrictions 78
locks
  delete tasks 63
  import tasks 61
  programming considerations 65, 66
    access 65
    exclusive 66
MultiLoad acquisition 65
write 65
RELEASE MLOAD command 164
log tables
nonshareability 159
LOGDATA command
syntax 151
LOGMECH command
syntax 152
LOGOFF command
defined 21
example 154
syntax 153
logoff/disconnect messages 17
description 18
LOGON command
defined 21
logon parameters 156
syntax 155
using with the LOGTABLE command 157, 158
example 157, 159
with multibyte character sets 58
LOGTABLE command
defined 21
syntax 158
using with the LOGON command 157, 158
example 157, 159
with multibyte character sets 58
LONG VARCHAR data type specifications
FIELD command 124
FILLER command 131
LOW keyword, NOTIFY specification, BEGIN MLOAD command 96

M
-M max-sessions runtime parameter 35
MARK option, DML LABEL command 115, 116
MATCHLEN runtime parameter 46
max variable, SESSIONS specification, BEGIN MLOAD command 95
maximum
file size, programming considerations 66
row size, programming considerations 66
sessions 35
MAXSESS runtime parameter 46
MEDIUM keyword, NOTIFY specification
BEGIN DELETE MLOAD command 96
BEGIN MLOAD command 96
merge join
defined 251
min variable, SESSIONS specification, BEGIN MLOAD command 95
minimum sessions 35
MINSESS runtime parameter 46
minutes variable, SLEEP specification 100
BEGIN DELETE MLOAD command 96
BEGIN MLOAD command 95
MISSING keyword, DML LABEL command 114
mlimod.c sample INMOD routine 227
mlnotf.c sample notify exit routine 229
MODIFY DATABASE statement, Teradata SQL 23
modulename variable, AXSMOD specification, IMPORT command 135, 137
monitoring MultiLoad performance 89
MP-RAS
compiling and linking routines 220, 221
MSG string specification, BEGIN MLOAD command 98
multibyte character sets, programming considerations 58
MultiLoad utility
data conversion capabilities 19
description 15
invoking 29
batch mode 30, 31
terminating 38
multiple
APPLY clauses, using 143
operations, import tasks 60
parameters
specifying, MVS example 187
specifying, UNIX example 193
specifying, VM example 184
references to the same field, example 128
UPI columns, UPDATE statement 175
variables, coding 93
MVS
C INMOD example 200
COBOL INMOD example 204, 207
COBOL Pass Thru INMOD example 207
PL/I INMOD example 209

N
-N min-sessions runtime parameter 35
name variable
AXSMOD specification, IMPORT command 135, 137
NOTIFY specification, BEGIN DELETE MLOAD command 96
NOTIFY specification, BEGIN MLOAD command 95
name, defined 252
Named Pipes Access Module 137
nested comments 53
nested join
defined 252
NOBLOCK 99
NONE keyword, AMPCHECK specification, BEGIN MLOAD command 96
nonparticipant AMPS, programming considerations 64
nonshareability
- acquisition error table 103
- application error table 103
- error tables 107
- log tables 159
- work tables 102, 107
- normal termination 38
- NOSTOP keyword, FORMAT specification, IMPORT command 140
- notify exit routines
  - compiling and linking 220
  - definition 69
  - entry points 71
  - event codes 104
  - generating for Windows 226
  - sample parameters 217
  - sample programs for Windows
    - source file 229
- NOTIFY option specification, BEGIN MLOAD command 99
- null, defined 252
- nulexpr specification, FIELD command 121
- NULLIF keyword, FIELD command 119
- NULLS keyword, DROP specification, FIELD command 119
- numeric values, in variable substitution 52
- numeric-to-numeric data conversions 20

O
- Object Access filter
  - defined 253
- OFF keyword
  - ECHO specification, ROUTE MESSAGES command 165
  - NOTIFY specification
    - BEGIN DELETE MLOAD command 96
    - BEGIN MLOAD command 96
  - OLE DB Access Module 137
  - operators, restrictions 48
  - option variable, NOTIFY specification, BEGIN MLOAD command 95
  - options messages, description 17
  - OR construct, accomplishing for an UPDATE statement 175
  - oscommand specification, SYSTEM command 171
- outer join
  - defined 253
- outfilename standard output file specification 36
- owner, defined 253

P
- parameter evaluation sequence, FIELD command 123
- parms variable, USING specification, IMPORT command 135, 137
- parser, defined 254
- parsing engine (PE), defined 254
- password specification, LOGON command 155
- passwords, with multibyte character sets 58
- PAUSE ACQUISITION command
  - defined 22
  - syntax 160
- performance group
  - defined 254
- performance monitoring 89
  - procedure 90
- PERSONNEL database tables, DDL MultiLoad job script example 237
- PL/I INMOD examples
  - for MVS 209
- PL/I language INMODs
  - programming structure 71
- preliminary phase, description 16
- procedures
  - defined 255
- product join
  - defined 254
- product version numbers 3
- profiles
  - defined 255
- programming considerations 46
  - character set specifications 54
  - command conventions 48
  - comments 53
  - date and time variables 51
  - down AMPs 63
  - graphic constants 59
  - graphic data types 59
  - import tasks 59, 62
  - locks 65
  - multibyte character sets 58
  - nonparticipant AMPs 64
  - restrictions and limitations 66
  - row count variables 51
  - system variables 49
  - termination control codes 68
  - utility variables 51
  - variable substitution 52

Q
- queries
  - defined 255
- query analysis
  - defined 255
- query management
  - defined 255
- Query Resource filter
  - defined 255
- Query Session utility
  - description 18
used for MultiLoad reports 17
using to monitor MultiLoad performance 90
QUEUE option specification, BEGIN MLOAD command 99

R
-r 'multiload command' command specification
UNIX example 192
-r multiload command specification
runtime variable 35
rate variable, CHECKPOINT specification, BEGIN MLOAD command 95
record mode load anomaly 48
recovered AMPs, restarting after 43
reduced print output runtime parameter 32
  MVS example 184
  UNIX example 188
  VM example 181
redundant conversions
  supported 20
RELEASE MLOAD command
  defined 23
  messages returned 164
  syntax 162
RENAME statement, Teradata SQL 23
REPLACE MACRO statement, Teradata SQL 23
REPLACE VIEW statement, Teradata SQL 23
request, defined 256
required access privileges
  DELETE statement 110
  INSERT statement 145
  RELEASE MLOAD command 163
  UPDATE statement 175
reserved words 49
restart log table 40
  checkpoint information 75, 80, 104
  computing space requirements 82
  dropping 40, 154
  estimating space requirements 82, 85
  implications of dropping 40
  INMOD repositioning information 72, 73
  listing in logoff/disconnect messages 18
  LOGON command 158
  maintaining 159
  nonshareability 159
  not dropped by RELEASE MLOAD 162
  row count variables 51
  specifying 159
  used for restarting paused jobs 44, 45
  using multibyte character sets 58
restarting jobs 42
  after a client system failure 44
  after a job abort 44
  after a PAUSE ACQUISITION command 43
after a Teradata Database restart 44
after a Teradata Database system reconfiguration 44
after an AP reset condition 45
after job script errors 43
after recovering a down AMP 43
after unrecoverable I/O errors 43
with delete task after pause 62
RESTORE statements
delete tasks 63
import tasks 62
restrictions and limitations
  programming considerations 66
    aggregate operators 67
    arithmetic functions 67
    column limit 67
    concurrent load tasks 67
    data file concatenation 67
    exponential operators 67
    expressions 67
    foreign key references 67
    hexadecimal form 67
    join indexes 67
    journaling options 67
    maximum file size 66
    maximum row size 66
    statement text length limit 67
    Teradata Database data retrieval 67
    unique secondary indexes 67
result, defined 257
results file storage
  defined 257
results files
  defined 257
results tables
  defined 257
retcode specification, LOGOFF command 153
return codes 38, 68
REVOKE statement, Teradata SQL 23
RNAME 99
ROLLBACK statements
delete tasks 63
import tasks 62
ROLLFORWARD statements
delete tasks 63
import tasks 62
ROUTE MESSAGES command
  default destinations 166
  defined 21
  syntax 165
row
  count variables, programming considerations 51
  size, maximum 66
row, defined 257
RowID join
space requirements
- estimating 81
- preliminary estimate 82
- preliminary estimate adjustments 83
- restart log table requirements 84
SQL
- defined 258
SSO
- LOGON command 155, 156
startpos specifications
- FIELD command 122
- FILLER command 130, 131
statement
- defined 259
- statements
  - execute if true/false variable
    - IF command 133
  - supported
    - import tasks 60
  - text limit 81
statistics messages, description 19
status reporting 17
StatusCode values, MultiLoad/INMOD interface 72
stored procedures
- defined 259
string variable, NOTIFY specification
- BEGIN DELETE MLOAD command 96
- BEGIN MLOAD command 95
supervisory user, defined 259
support commands 20
syntax, how to read 177
SYSTEM command
- defined 21
- syntax 171
system variables
- programming considerations 49

T
- table
  - column names 49
  - names, with multibyte character sets 58, 59
  - requirements, RELEASE MLOAD command 163
TABLE command
- defined 22
  - intermixing with FIELD or FILLER commands 173
  - syntax 172
    - with multibyte character sets 59
  - table locks 65
table, defined 259
tablename specifications
- RELEASE MLOAD command 162
tableref specification, TABLE command 172
TABLES keyword
BEGIN DELETE MLOAD command 96
BEGIN MLOAD command 96
target object restrictions
  INSERT statement 145
  UPDATE statement 175
target tables
  import tasks 61
  name specification, BEGIN MLOAD command 101
task
  commands 20
  status reporting 17
tdpid specification, LOGON command 156
TENACITY hours specification
  BEGIN DELETE MLOAD command 101
  BEGIN MLOAD command 101
Teradata Database
  data retrieval, programming considerations 67
  restart, restarting MultiLoad after 44
  system reconfiguration, restarting MultiLoad after 44
Teradata SQL statements
  DELETE 109
  INSERT 144
    supported by MultiLoad 22
  UPDATE 174
terminating a MultiLoad job 38, 39
termination codes 68
  programming considerations 68
termination return codes 38
TEXT keyword
  FORMAT specification, IMPORT command 139
  NOTIFY specification, BEGIN MLOAD command 101
text variable, DISPLAY command 112
THEN keyword, IF command 133
THRU keyword
  IGNORE specification, ACCEPT command 92
  IMPORT command 140
  RUN FILE command 168
TIME system variable (&SYSTIME) 51
title, defined 260
tname specifications
  BEGIN MLOAD command 107
  DELETE statement 109
  fallback error table
    BEGIN MLOAD command 103
  INSERT statement 144
  LOGTABLE command 158
  target table or view
    BEGIN MLOAD command 101
  UPDATE statement 174
  work table
    BEGIN MLOAD command 102
TO keyword
  expression specification, SET command 169
  FILE specification
DISPLAY command 112
ROUTE MESSAGES command 165
TRAILING keyword, DROP specification, FIELD command 119
transaction, defined 260
type, defined 261

U
UNFORMAT format specification, IMPORT command 139
unicode data, using 146
unique secondary indexes, programming considerations 67
uniqueness column, application error table 87
UNIX
  C INMOD example 215
  C notify exit parameters 217
    compiling and linking routines 220
    entry points for notify exit routines 72
unrecoverable I/O errors, restarting after 43
unspecified target objects 105
UPDATE
  keyword, DML LABEL command 114
  statement, Teradata SQL 23, 174
  UPDCNT system variable (&SYSUPDCNT) 51
UPI columns, UPDATE statement 175
upsert feature
  defined 114
  errors with 115
  import tasks 60
  MultiLoad job script example 236
  rules for using 116
  task example 117
  using INSERT 116
  using UPDATE 116
user 261
user groups
  defined 261
user names, with multibyte character sets 58
username specification, LOGON command 156
USING (parms) specification, IMPORT command 140
UTF-8
  defined 261
utility statements
  delete tasks 63
  import tasks 62
utility variables 113
  referenced in variable substitution 52
    with multibyte character sets 58, 59
UVCNT system variable (&SYSUVCNT) 51

V
-V runtime parameter 36
value substitution
  sequence 52
VALUES specification, INSERT statement 145

var utility variable specifications
   ACCEPT command 93
   SET command 169

VARGRAPHIC data type specifications
   FIELD command 124

VARIABLE keyword, ACCEPT command 92

variables
   declaring, SET command 169
   in the IF conditional expression 133
   number of, ACCEPT command 94
   programming considerations 51
   substitutions
      ELSE and ENDIF commands 134
      programming considerations 52
      SET command 169
      within comment strings 54
   system 49

VARTEXT format specification, IMPORT command 139

version numbers 3

views
   as import target objects 105
   delete task not allowed 62

VM and MVS
   entry points for notify exit routines 72

W

WebSphere MQ Access Module for Teradata
   client version 137
   server version 137

WHERE condition specifications
   DELETE statement 109
   IMPORT command 141
   UPDATE statement 174

Windows
   entry points for notify exit routines 72
   generating INMOD and notify exit routines 226
   sample INMOD routine 227
   sample notify exit routines 229

WITH option, ECHO specification, ROUTE MESSAGES command 165

work table
   implications of dropping 40, 41
   name specifications, BEGIN MLOAD command 102
   nonshareability 102, 107

workgroups
   defined 262

workload limits rule
   defined 262

WORKTABLES keyword
   BEGIN DELETE MLOAD command 96
   BEGIN MLOAD command 96

write

Z

zoned decimal format 20