TERADATA

Teradata Tools and Utilities Access Module

Reference

Release 13.00.00
B035-2425-088A
August 2008

Preface

Purpose

Teradata® Tools and Utilities is a group of products designed to work with Teradata Database. This reference details how to use the access modules that link the Teradata Tools and Utilities to external data sources and destination storage devices.

For information about other third-party access modules, refer to the third-party vendor documentation.

Audience

This book is intended for use by:

- Systems programmers
- Application programmers
- Systems administrators

Supported Releases

This book supports the following releases:

- Teradata Database 13.00.00
- Teradata Tools and Utilities 13.00.00
- Data Connector 13.00.00
- Named Pipes Access Module 13.00.00
- OLE DB Access Module 13.00.00
- WebSphere MQ Access Module 13.00.00
- Teradata Access Module for JMS 13.00.00

Each chapter contains support information that is specific to each access module; however, for the most current version information, do the following:

1. Go to http://www.info.teradata.com.
2. Under **Online Publications**, click **General Search**.
3. Type *3119* in the **Publication Product ID** box.
4. Under **Sort By**, select **Date**.
5. Click **Search**.

6 Open the version of the *Teradata Tools and Utilities xx.xx.xx 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:

- Familiarity with computer technology, database management systems, and utilities that load and retrieve data.
- Familiarity with SQL and the Teradata Database.

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 *Teradata Tools and Utilities Release Definition* associated with this release.

Date/Release	Description

August 2008
Teradata Tools and Utilities 13.00.00
	• Updated version numbers for all products.
	• Named Pipes Access Module: removed support for MP-RAS.
	• OLE DB Access Module:
	  • Added support of Teradata Parallel Transporter (PT). See "Table 1: Commands for Teradata Utilities" on page HIDDEN and "Overview" on page 21.
	  • Added support for duplicate rows (multiset tables). See "Step 1 - Select a Data Source and Target" on page 24.
	  • Added support for the PERIOD data type. See Table 2 on page 40
	• WebSphere MQ Access Module:
	  • Added a limitation regarding multi-volume checkpoint files. See "Checkpoint Processing" on page 89.
	  • Updated a JCL example. See "MVS JCL Requirements" on page 90.
	  • Added 64-bit HP-UX as a supported operating system.
	  • Removed support for MP-RAS.
	• Teradata Access Module for JMS:
	  • Added support for Teradata Parallel Transporter (PT).

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.

Type of Information	Description	Access to Information
Release overview Late information	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 http://www.info.teradata.com. 2 Under **Online Publications**, click **General Search** 3 In the **Publication Product ID** box, type *2029*. 4 Click **Search**. 5 Select the appropriate Release Definition from the search results.

Type of Information	Description	Access to 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 http://www.info.teradata.com. 2 Under the **Online Publications** subcategory, **Browse by Category**, click **Data Warehousing**. 3 Do one of the following: • For a list of Teradata Tools and Utilities documents, click **Teradata Tools and Utilities**, and then select an item under **Releases** or **Products**. • Select a link to any of the data warehousing publications categories listed. Specific books related to access modules are as follows: • *Basic Teradata Query Reference* B035-2414-*mmyx* • *Teradata FastExport Reference* B035-2410-*mmyx* • *Teradata FastLoad Reference* B035-2411-*mmyx* • *Teradata MultiLoad Reference* B035-2409-*mmyx* • *Teradata Parallel Data Pump Reference* B035-3021-*mmyx* • *Teradata Parallel Transporter Reference* B035-2436-*mmyx* • *Teradata Tools and Utilities Access Module Programmer Guide* B035-2424-*mmyx*
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 http://www.info.teradata.com/. 2 Under the **Online Publications** subcategory, **Browse by Category**, click **Data Warehousing**. 3 Click **CD-ROM List and Images**. 4 Follow the ordering instructions.
Ordering information for manuals	Use the Teradata Information Products Publishing Library site to order printed versions of manuals.	1 Go to http://www.info.teradata.com/. 2 Under **Print & CD Publications**, click **How to Order**. 3 Follow the ordering instructions.

Type of Information	Description	Access to Information
General information about Teradata	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	1 Go to Teradata.com. 2 Select a link.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter contains the following topics about how the Teradata access modules work with Teradata Database:

Overview

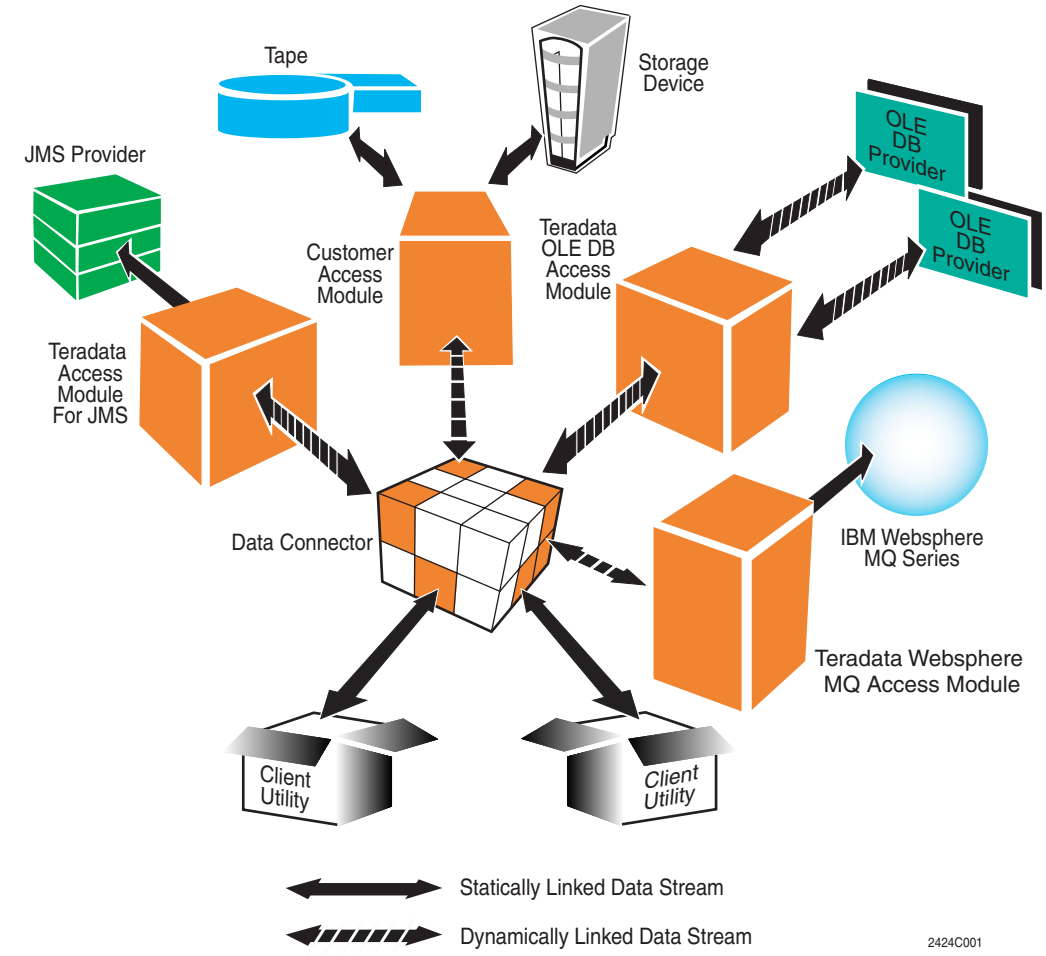
Access modules are dynamically linked software components that provide input and output interfaces to different types of external data storage devices, OLE DB data sources, and message queuing software. Access modules import data from various data sources and return the data to a Teradata utility, which then stores the data in the data warehouse. Access modules are dynamically linked to one or more client utilities by the Teradata Data Connector Application Programming Interface (API).

- Read (import) data flows from access modules to the Teradata Data Connector API. The Data Connector API expects, but does not require, data in blocks that consist of one or more logical records.
- Write (export) data flows from the Teradata Data Connector API to access modules. In most cases, the Data Connector API provides data in blocks consisting of one or more logical records.

Note: Access modules are distinctly different from INMOD and OUTMOD routines.

Figure 1 depicts the relationship between various access modules and the Teradata Data Connector API.

Figure 1: Data Connector API Linkage



Supported Access Modules

Teradata Database supports the following access modules:

- Chapter 2: “Teradata OLE DB Access Module”
- Chapter 3: “Named Pipes Access Module”
- Chapter 4: “Teradata WebSphere MQ Access Module”
- Chapter 5: “Teradata Access Module for JMS”

Supported Teradata Utilities

Teradata access modules work on many operating systems and with the following client load and export utilities:

- BTEQ
- Teradata FastExport
- Teradata FastLoad
- Teradata MultiLoad

- Teradata Parallel Transporter (Teradata PT)
- Teradata TPump

For more information, see the individual access module chapters.

Access Module Calls

Each Teradata client utility invokes access module calls differently. However, all utilities include the following:

- AXSMOD keyword.
- Access module name, which is the file name of the dynamically loadable module providing the access module software.
- Access module initialization string, which is an optional list of operational parameters specified for the access module.

  Initialization strings are specified and delimited according to the requirements of the Teradata client utility. The contents of the string are determined according to the requirements of the specified access module.

To specify an access module in your Teradata utility job script, do the following:

1. Use the syntax for the AXSMOD command or command option as described in the reference documentation for the Teradata utility.
2. Use the syntax for the access module initialization string described in the "Initialization String" subsection of each access module chapter in this reference.

Client Utility Commands

Table 1 lists the client utility commands for specifying an access module.

Table 1: Access Module Commands for Teradata Tools and Utilities

Client Utility	Command	Description
BTEQ	EXPORT IMPORT	See the descriptions of the EXPORT and IMPORT commands in the *Basic Teradata Query Reference*.
FastExport	EXPORT IMPORT	See the descriptions of the EXPORT and IMPORT commands in the *Teradata FastExport Reference*.
FastLoad	AXSMOD	See the descriptions of the AXSMOD command in the *Teradata FastLoad Reference*.
MultiLoad	IMPORT	See the descriptions of the IMPORT command in the *Teradata MultiLoad Reference*.
Teradata PT	*<AccessModuleName>*	See the descriptions of the AccessModuleName attribute in the *Teradata Parallel Transporter Reference*.
TPump	AXSMOD IMPORT	See the descriptions of the AXSMOD and IMPORT commands in the *Teradata Parallel Data Pump Reference*.

Data Connector API

The Data Connector API is the software layer between a client utility and an access module. It is responsible for establishing, maintaining, and monitoring that connection. It accomplishes this by being statically linked to the client modules and dynamically linked to the needed access module. See Figure 1. If you have a Teradata Parallel Transporter version of an access module, the term *Data Connector* throughout this book refers to the Teradata Parallel Transporter DataConnector *operator*.

Specifying an access module in a Teradata client utility job script causes the following actions at runtime:

1. The client utility passes the access module name and initialization information to the Data Connector API.
2. The Data Connector API connects and initializes the specified access module.

Version Identification

All Teradata access modules use the Identification function for version information. This function is requested as the second parameter (*OptParms*) in the main function called *PIDMMain()*. This main function is called by the Data Connector. The Data Connector displays this information in its log file and is available to the client utility to record in its log file.

For the Teradata OLE DB Access Module, product information can be found by accessing **Start>Control Panel>Add or Remove Programs**, or in the **About** menu in the dialog box.

Error Messages

All error return codes documented in this book are errors generated by a specific access module. For Data Connector return codes, refer to the most recent version of the Teradata *Messages* manual.

Session Character Sets

All Teradata access modules can use session character sets.

- **Workstation** - For any workstation access module to run with BTEQ or TPump using UTF-16 session character set, the access module must accept UTF-16 as the attribute value of attribute CHARSET_NAME.
- **Mainframe** - For any mainframe access module to run with BTEQ or TPump using the UTF-8 session character set, the access module must accept UTF-8 as the attribute value of attribute CHARSET_NAME.

CHAPTER 2

Teradata OLE DB Access Module

Topics in this chapter include:

Overview

Teradata OLE DB Access Module is a dynamic link library (DLL) that acts as an interface between Teradata load and export utilities (Teradata FastLoad, Teradata FastExport, Teradata MultiLoad, TPump, Teradata Parallel Transporter [PT], and BTEQ) and data sources for which an OLE DB provider is available. The access module quickly moves data between a OLE DB data sources and Teradata Database without requiring intermediate storage.

The access module can be used to view, edit, and re-run access module job (*.amj*) files, or to perform a quick, one-time copy (to create a new table in Teradata Database with data from an OLE DB data source) or a quick, one-time import from an OLE DB data source to Teradata Database.

The Teradata OLE DB Access Module offers the following options to move data:

- Data source -> OLE DB provider -> Teradata OLE DB Access Module -> Teradata load utility -> Teradata Database
- Teradata Database -> Teradata export utility -> Teradata OLE DB Access Module -> OLE DB provider -> data source
- Teradata Database -> Teradata PT -> Teradata Database

Teradata OLE DB Access Module creates a new table during load operations if no target table already exists, and adds (imports) to existing tables. The exception is when Teradata FastLoad is used for imports, in which case the load operation only works on an empty table. In other words, a load operation that uses FastLoad cannot load to existing tables.

You can use Teradata OLE DB Access Module to accomplish the following processes:

- Use the access module graphical user interface (GUI):
  a. Open the GUI for Teradata OLE DB Access Module (nicknamed **Teradata OleLoad**), and select a data source and destination.
  b. Save the job as an *.amj* file.
  c. Select a Teradata utility to transfer (load or export) the data, generate a script, and run the job.

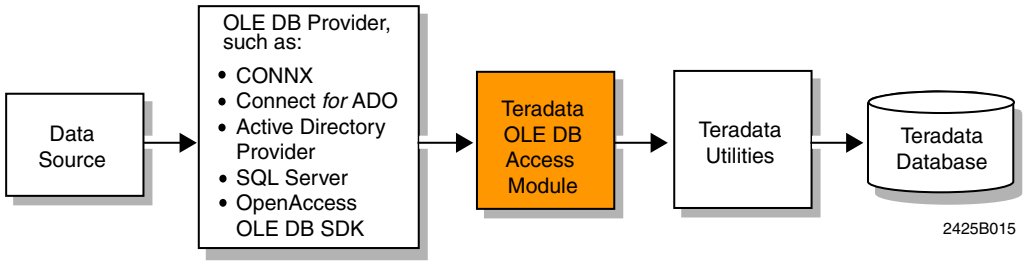
  For more information, see "Loading and Exporting with OleLoad" on page 24.
- Run the access module from a Teradata load and export utility at the command prompt:
  a. In a Teradata utility, write a script that references Teradata OLE DB Access Module and a previously saved *.amj* file.
  b. Run the script.

  For more information, see "Loading and Exporting at the Command Prompt" on page 32.

About Load Operations

Potential data sources for Teradata OLE DB Access Module load operations include flat files, spreadsheets, and databases. Teradata OLE DB Access Module retrieves data using an OLE DB provider, then forwards the data to a Teradata utility that loads data into the target Teradata Database, as shown in Figure 2.

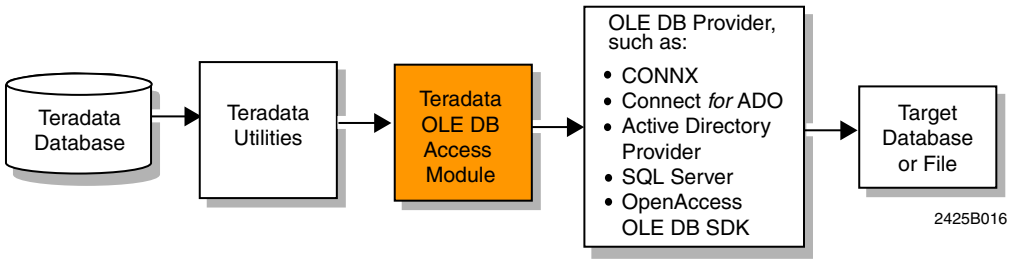
Figure 2: Load Operation Data Flow



About Export Operations

Potential targets for Teradata OLE DB Access Module export operations include flat files, spreadsheets, and databases. Teradata OLE DB Access Module uses a Teradata utility to export data from Teradata Database, then forwards the data to an OLE DB provider that loads or copies data to the target file or database, as shown in Figure 3.

Figure 3: Export Operation Data Flow



About Operating Modes

Teradata OLE DB Access Module can be run in two modes: through a GUI called **Teradata OleLoad**, and through a command prompt. Both modes enable data movement with Teradata load and export utilities, but the access module via the command prompt provides limited functionality.

Operating Requirements

Teradata OLE DB Access Module runs on the Windows 2000, XP, Server 2003, and Vista operating systems. For the most current information about the operating systems supported by Teradata OLE DB Access Module, see Supported Releases in the Preface.

System Prerequisites

Teradata OLE DB Access Module requires the following:

- The Windows version of a Teradata utility (BTEQ, Teradata FastExport, Teradata FastLoad, Teradata MultiLoad, Teradata PT, or TPump) to transport data
- ODBC Driver for Teradata
- An OLE DB provider

  The installation of Teradata OLE DB Access Module includes the following Microsoft data access components (MDAC) and OLE DB providers:

  - Microsoft OLE DB Provider for SQL Server
  - Microsoft OLE DB Provider for Oracle
  - Microsoft OLE DB Provider for Microsoft Jet (to access data from Access, Paradox, dBASE, Excel, FoxPro, text, and more)
  - Microsoft OLE DB Provider for ODBC Driver (to access data traditionally accessed using ODBC)

Note: For more information about OLE DB providers, consult *OLE DB Provider for Teradata Installation and User Guide*, the Microsoft Web site, or the SQL Summit Web site.

Loading and Exporting with OleLoad

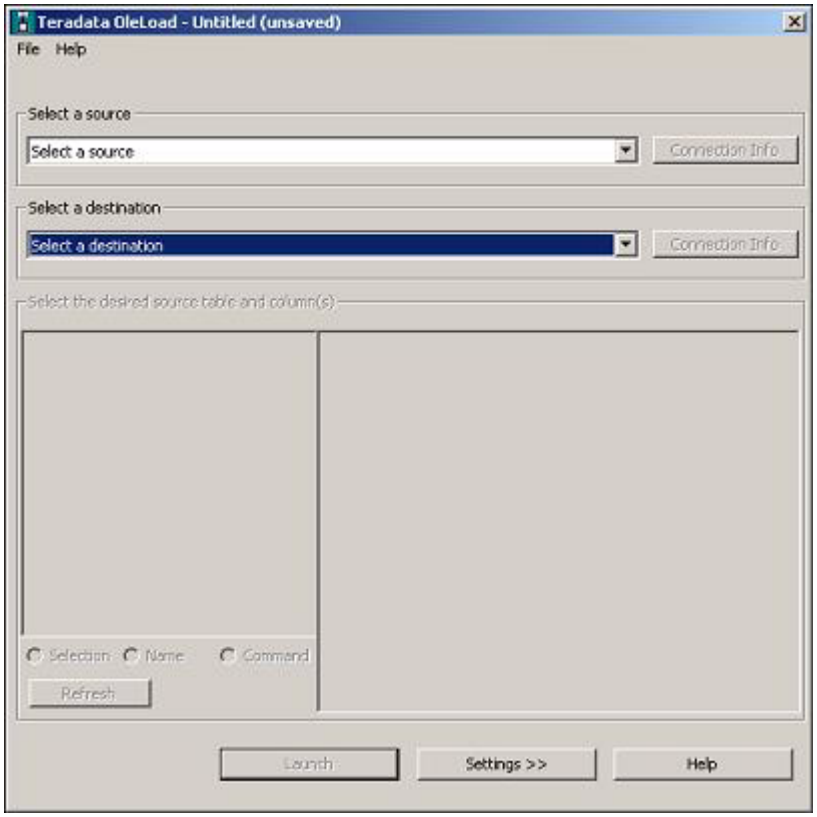
To move (load or export) data using the OleLoad GUI of Teradata OLE DB Access Module, do the following:

- Step 1 - Select a Data Source and Target
- Step 2 - Specify Advanced Settings [optional]
- Step 3 - Launch a Script

Step 1 - Select a Data Source and Target

To specify a data source and target

1 From the desktop, click **Start>Programs>Teradata Client>OleLoad** to open **Teradata OleLoad** dialog box.



2 [Optional] Click **File>Open** to populate the window with parameters from a previously saved *.amj* file.

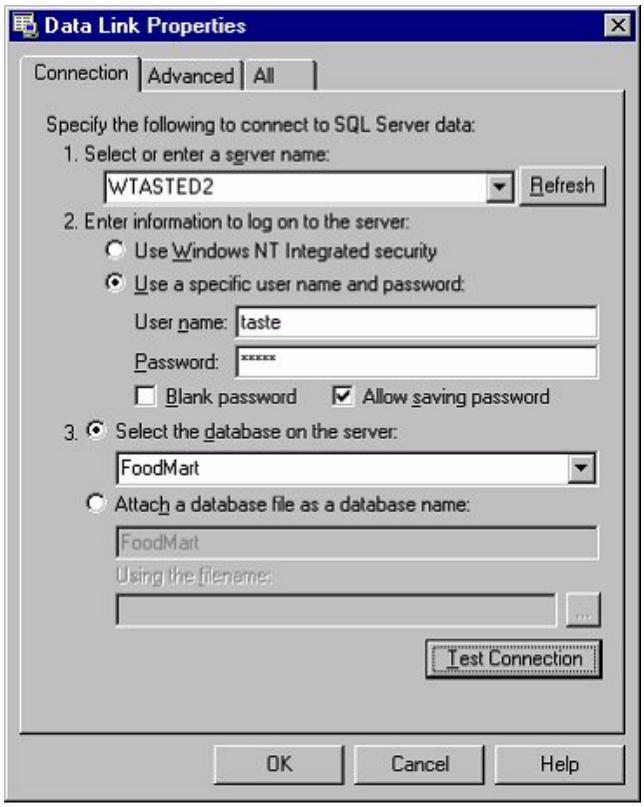
3 Select a data source from the **Select a source** list.

The list contains names of all of the available OLE DB data sources and the Teradata Database. After a data source is selected, you can click **Connection Info** to view system information about a selected data source.

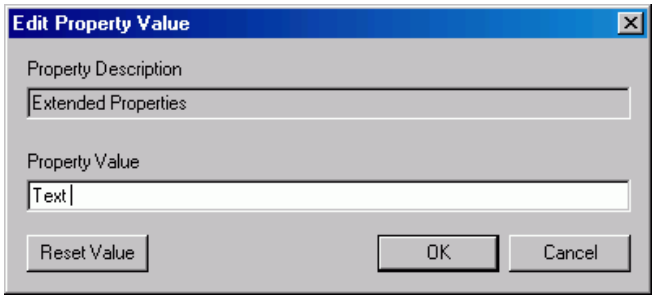
- **Loading to** Teradata Database - Selecting a data source opens the Microsoft **Data Link Properties** dialog box. Use this dialog box to specify the information necessary to connect the source.

  Note: Duplicate rows (multiset tables) are supported.

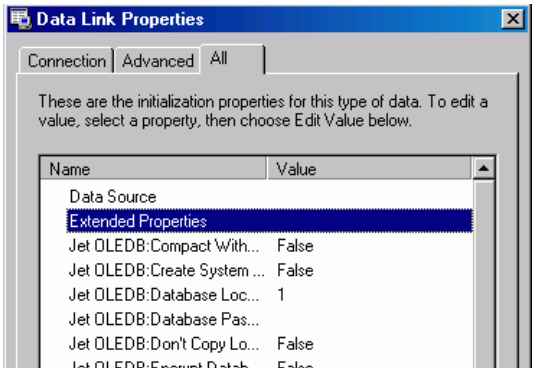
  - If the data source is a text file with multi-lined strings and carriage returns (which are features not supported by Teradata utilities) select **Microsoft Jet 4.0 OLE DB Provider**. For information about how to set up the *schema.ini* file that is needed for this type of transfer, see "Define a Schema File" on page 119.



  - Select the **Blank password** check box if a system administrator allows specific users to log on without a password; clear the check box if an administrator provides the password for accessing the database.
  - Transfers that involve text files need to have the Text property file specifically designated. To do this from the **Data Link Properties** dialog box, click **All**, highlight **Extended Properties**, and click **Edit Value**. In the **Edit Property Value** dialog box, type the word `Text`, and click **OK** to return to **Data Link Properties**.



- Click the **All** tab to verify that the lists of **Name** and **Value** contain an entry for **Extended Properties** and **Text**.



Note: If a Microsoft Data Link Error message occurs, stating that **Failed Properties: Persist Security Info (NOT SUPPORTED)**, ignore the message by clicking **OK**.

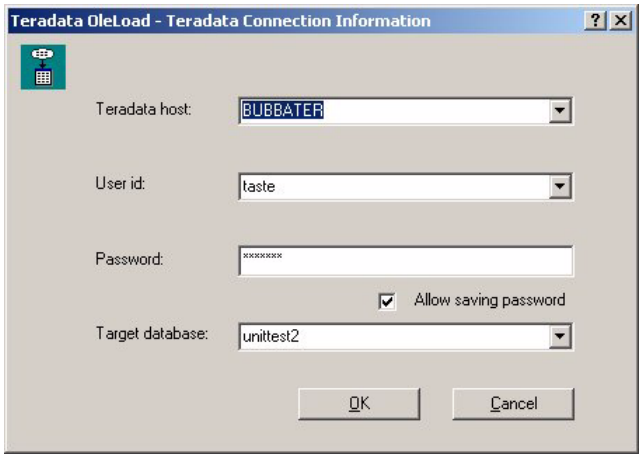
Note: For more help, click **Help** in the **Data Link Properties** dialog box.

- **Exporting from** Teradata Database - Select **Teradata Database** as the data source to open the **Teradata Connection Information** dialog box. Use this dialog box to specify the information necessary to connect to Teradata Database.

  Note: An OLE DB data source cannot be specified as both the source *and* the destination.

  Selecting **Teradata Database** as the data source or the destination for a load or export operation opens the **Teradata Connection Information** dialog box so connection information to the Teradata Database can be configured.

  Note: Although it is possible to export data from one Teradata Database to another Teradata Database, the performance is such that this specific operation is not recommended. Consider using an alternate data source for export operations into a Teradata Database, such as changing the operation to an import operation by selecting **Microsoft OLE DB Provider for ODBC Drivers** or **OLE DB Provider for Teradata** as the data source.



The following fields are available in this dialog box:

- **Teradata host** - Enter the host name for the Teradata Database. The value entered must have a COP entry in the hosts file or in domain name services (DNS).
- **User id** - Enter the user logon id for the connection.
- **Password** - Enter the password associated with the specified user id.
- **Allow saving password** - Select to save the password to the *.amj* file and prevent being prompted again for this information when connecting to Teradata Database.

Caution: If this check box is selected, the *.amj* file might allow unauthorized access because it will contain password information.

- **Target database -** Enter the name of the Teradata Database.

4 The left pane of the **Teradata OleLoad** dialog box displays a hierarchy of the items in the selected data source, if such hierarchies are available. Select one of the following buttons (which are below the left pane) to identify the location of the data needed for the load operation:

- **Selection** - Displays the hierarchy of the data source in the left pane. Select the data needed for the operation.

  This button is only available if the data source supports the TABLE schema rowset.
- **Name** - Type the name of a data source in the left pane.
- **Command** - Type an SQL query in the left pane to retrieve data from the selected data source, then manually update the **Table Name** text box in the **Advanced Settings** dialog box. (With the other options, the text box is automatically updated.)

  Depending on the source, performance might be improved if the command limits the columns that will be returned. For example, instead of using the following command to select only the columns CustomerID and CompanyName,

  ```
  SELECT * FROM "NORTHWIND"."DBO".CUSTOMERS"
  ```

  Instead, use:

  ```
  SELECT CustomerID, CompanyName FROM "NORTHWIND"."DBO".CUSTOMERS"
  ```

  The **Command** button is only available if the data source supports SQL commands. The button is unavailable if **Teradata Database** is the data source.

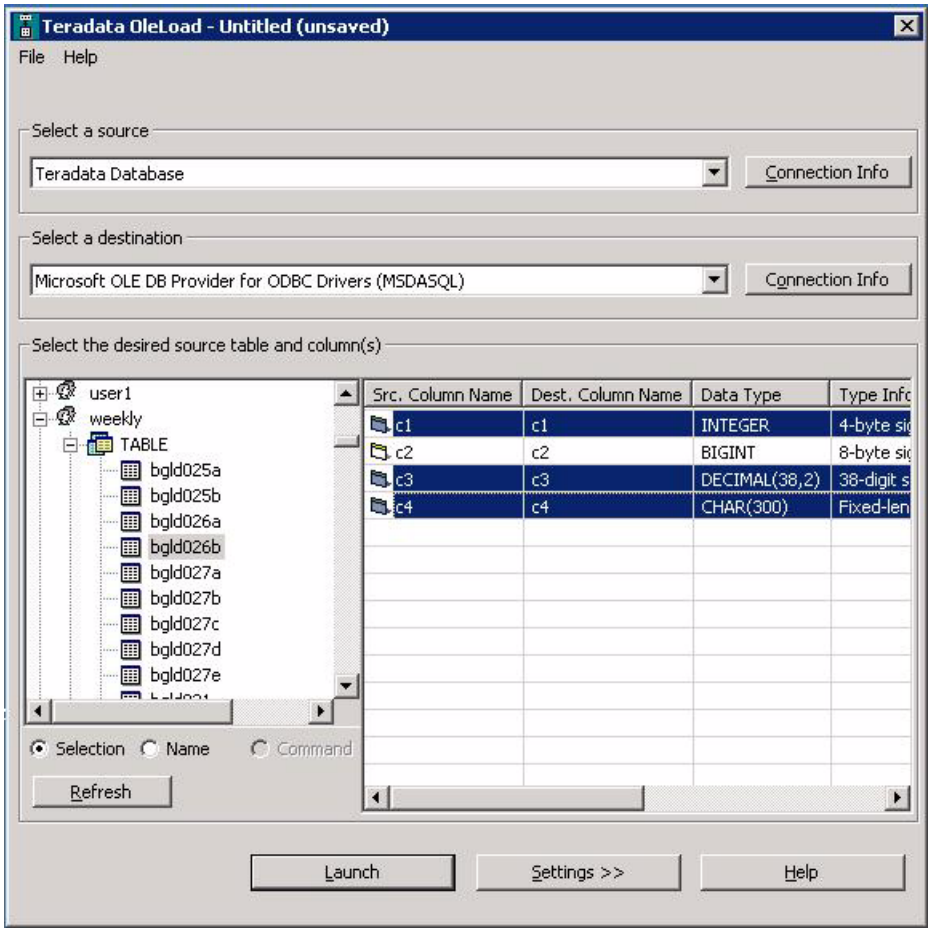
5 From the **Select a destination** list, do one of the following to select a target:

The list contains the names of all of the available OLE DB providers and the Teradata Database. After a destination is selected, you can click **Connection Info** to view system information about a selected destination.

- **Loading to** Teradata Database - Select **Teradata Database**, which opens the **Teradata Connection Information** dialog box. Use this dialog box to specify the information necessary to connect to Teradata Database.
- **Exporting from** Teradata Database - Select an OLE DB provider, which opens the Microsoft **Data Link Properties** dialog box. Use this dialog box to specify the information necessary to connect to the OLE DB provider.

After a destination is selected, you can click **Connection Info** to view system information about a selected destination.

Note: An OLE DB data provider must not be specified as both the source *and* the destination.



6 In the right pane, select the columns that contain the data needed for the operation.

The data in the selected columns will be transferred to the target system that is identified in the **Select a destination** box when the job is launched.

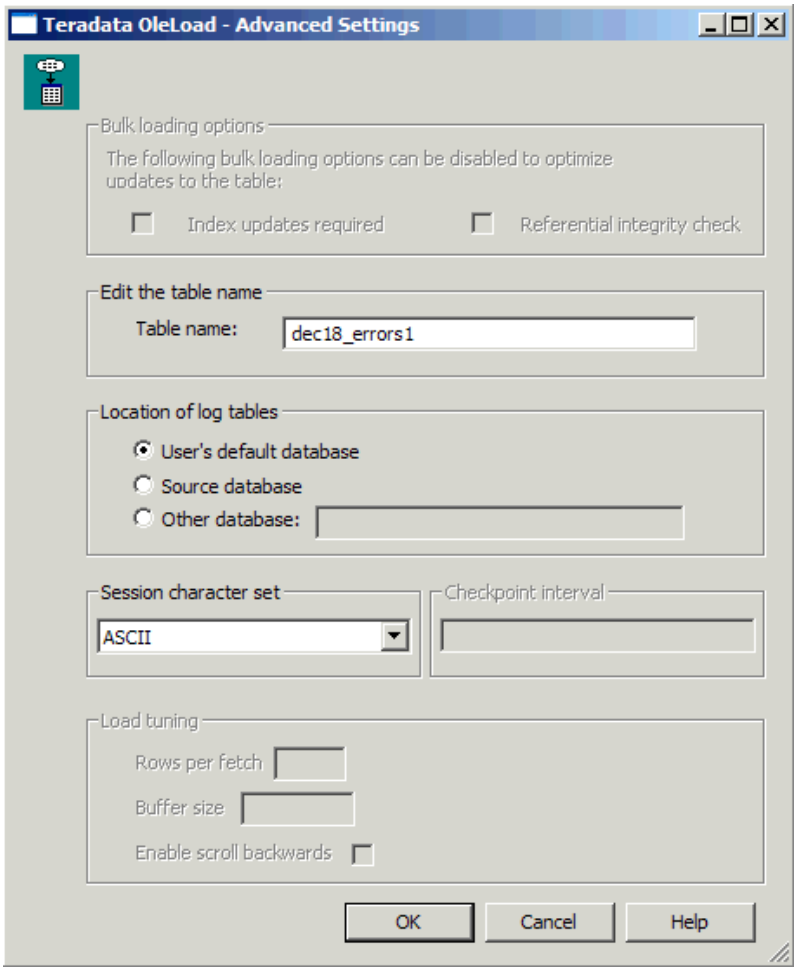
7 Proceed with Step 2 - Specify Advanced Settings [optional].

Step 2 - Specify Advanced Settings [optional]

To set load options, edit a table name, or specify log tables

If these options do not need to be modified, proceed with Step 3 - Launch a Script.

1 [Optional] To set up bulk loading options, edit a table name, or change the location of the log tables involved in the operation, click **Settings** to open the **Advanced Settings** dialog box.



The following items are available in this dialog box:

- **Bulk loading options** - When updating tables, disable either or both of these options to optimize performance. The options are available only when a data source supports the option.
  - **Index updates required** - If this option is cleared, the OLE DB provider is not required to update indexes based on inserts or changes to a row set. This will require indexes to be re-created after changes are made to a row set.
  - **Referential integrity check** - If this option is cleared, the OLE DB provider is not required to check the referential integrity constraints or to enforce changes made to a row set.

- **Table name** - Enter or change the name of the destination table. By default, the quoted and qualified table name is displayed; however, the double quotes must be removed for data providers, such as Oracle, that do not support double quotes. In other words, the table name must be quoted and qualified as required by the destination source.

  If you enter a name for a table that does not exist, a new table will be created.
- **Location of log tables** - Specify the location of the restart log tables for the operation. The name of this database gets included in the LOGTABLE command of Teradata FastExport scripts created by Teradata OLE DB Access Module.
  - **User's default database** - Teradata FastExport scripts search for the log table in the default database that is defined for the user name by the LOGON command.
  - **Source database** - Restart log tables are located in the same database as the tables being exported.
  - **Other database** - Specify a database name other than the default or source database.
- **Session character set** - Specify the character set for the current session used for scripts and data transfer.

  Because the session character set affects the validity of strings allowed for Teradata logon and password, a change in character sets will disconnect a Teradata session.

  For arbitrary Unicode strings to be correctly transferred, the following must occur:
  - In **Session Character Set,** select UTF8 (a Teradata session character set).
  - Ensure that the source (for a load operation) or the destination (for an export operation) properly handles Unicode.
  - Perform the necessary configuration of the data source or OLE DB Provider.

  During an export, character data is received from a export utility (that is, FastExport or BTEQ) in the Teradata session character set encoding. The character data is then converted using the DBTYPE_WSTR (UTF16) data type and passed to the OLE DB Provider that is specified in the **Select a destination** box.

  For additional information character sets, see "Character Set Support" on page 43.
- **Checkpoint interval** - Specify a checkpoint interval. This value is used in the CHECKPOINT specification in load scripts generated for Teradata FastLoad, Teradata MultiLoad, and TPump jobs. The field is enabled for load jobs only.

  If a value is entered in the **Checkpoint interval** field, system performance might be enhanced if values are also added to the **Rows per Fetch** and **Buffer size** boxes, and if the **Enable scroll backwards** option is selected. For more information about how to set these fields, see "Access Module Factors" on page 50.
- **Rows per Fetch** - Enter the number of rows returned for Teradata MultiLoad jobs. Increasing the number of rows might improve efficiency.
- **Buffer size** - Enter a number to represent the size of the buffer for a TPump job. Increasing the buffer size might improve efficiency.
- **Enable scroll backwards** - If this option is selected, restarts begin at the most recent checkpoint; however, this selection might slow performance. If this option is left blank, jobs restart from the beginning of the script.

2 Close the **Advanced Settings** dialog box, and proceed with Step 3 - Launch a Script.

Step 3 - Launch a Script

Scripts create tables and provide the commands necessary to run a job.

To launch a script

1 [Optional] Select **File>Change Default Folder** to change the location of the default folder for saved *.amj* files. The default location is *C:\ My Documents*.

2 After selecting a destination and at least one column of data in a source table, click **File>Save As** to save the file.

Caution: Failure to save the job will result in a failed operation. (You can tell a job is saved if the *.amj* file name appears in the title bar of the dialog box.) Also, if you try select additional columns from a table that was already used in a successful job, an error results unless you save the newly selected columns as a new, separate *.amj* file.

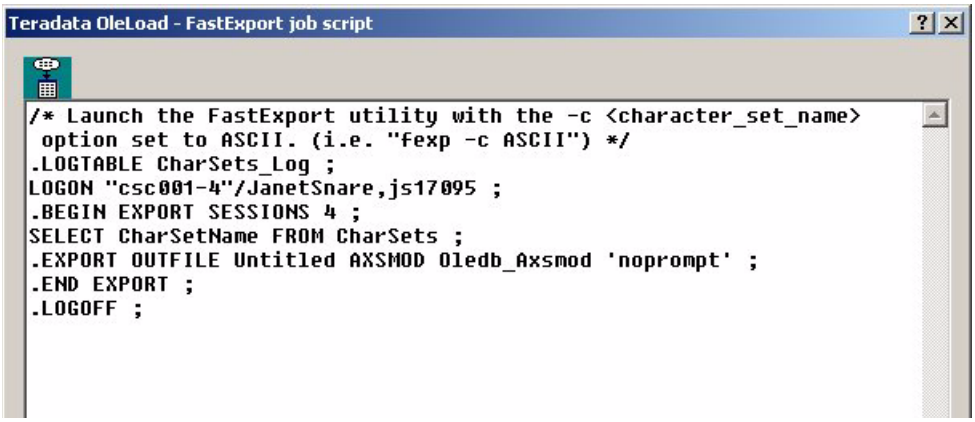
You can save the *.amj* file anywhere you have write access, but it is recommended that all *.amj* files be stored in a single location for easy access. The default storage location of the *.amj* files is the local *My Documents* directory.

3 Click **Launch**, then click one of the utility options in the **Launch** submenu.

Selecting a utility option opens the **Teradata OleLoad - <utility name> job script** dialog box (basically, a text editor), which displays a script generated in response to the selections in the previous dialog box. Scripts are created as Unicode text, then converted to the session character set specified in the **Advanced Settings** dialog box before being passed to the Teradata utility.

4 Edit the script as needed to meet job requirements.

Note: Use **Ctrl-C** and **Ctrl-V** to copy and paste a job script to another application, if needed.



5 Click **OK** to run the job.

Note: Click **Cancel** in the progress dialog box to stop the job. The cancellation, terminates the job without completing it. All modifications to the job script are lost.

Caution: For bulk load operations, the number of rows displayed in the progress dialog box might not be the same as the number of rows returned to the utility if a restart occurs. In this case, the progress dialog box might display a larger number of row retrievals than the number of rows actually retrieved from the data source.

Loading and Exporting at the Command Prompt

Data transfers are also possible using Teradata OLE DB Access Module from an active Teradata load/export session without the Teradata OleLoad GUI. Following is an overview of this process:

1. In a Teradata utility, create a job script that references Teradata OLE DB Access Module as *oledb_axsmod.dll*.

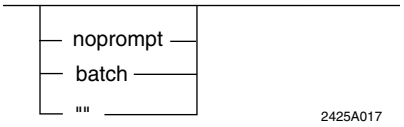
   If the initialization string of the script does not specify *batch* or *noprompt*, the **Teradata OLE DB AXSMOD** dialog box will open when the script is run so you can supply the needed information. (This dialog box is similar to the Teradata OleLoad GUI, but it has limited functionality.)

   For more information, see "Access Module Calls" on page 19 and "Client Utility Commands" on page 19.
2. Run the script from the utility.

About the Access Module Initialization String

When referencing the operating mode in the initialization string, do not use `batch` or `prompt`; only `noprompt` and the blank value are valid. (This does not apply to using Teradata PT.)

Syntax



where:

Option	Description
noprompt	• Removes the prompt for data source specifications. • Shows a progress dialog box during load or export operations.
batch	• Removes the prompt for data source specifications. • Prevents the progress dialog box from being displayed during load operations. • Allows Teradata OLE DB Access Module to operate without user interaction.
""	• Prompts for data source specifications by opening the **Teradata OLE DB AXSMOD** dialog box. • Shows a progress dialog box during load or export operations.

Access Module Commands

The initialization string for Teradata OLE DB Access Module consists of a single specification of the operating mode for the access module; however, to launch Teradata OLE DB Access Module, the initialization string for BTEQ and the load and export utilities must be empty (""). (If the initialization string contains `prompt`, the script fails.) For the specific commands and command options for each supported Teradata utility, see "Access Module Calls" on page 19.

Starting an Access Module Export Job Without Teradata OleLoad

To successfully export data using Teradata OLE DB Access Module without the Teradata OleLoad GUI (that is, using a Teradata utility), a job script must reference a previously saved access module job that designated Teradata Database as the data source in the **Select a source** field of the OleLoad GUI.

Depending on the information designated in the job script, one of the following occurs:

- If *batch* or *noprompt* are specified in the access module initialization string, Teradata OLE DB Access Module processes the job without interruption.
- If *batch* or *noprompt* are not specified in the access module initialization string, the **Teradata OLE DB Access Module** dialog box opens, and displays information from the specified .amj file as default job parameters. Modify the job parameters as needed, then click **Launch** to run the job.

Using BTEQ

To export from Teradata Database using BTEQ

1. Select **Start >Programs >Teradata Client >BTEQ** to open Teradata BTEQ from the Windows desktop.
2. For the EXPORT command, specify *OLEDB_AXSMOD* as the DLL for Teradata OLE DB Access Module.
3. For the FILE parameter and operating specification, use one of the following, where an *.amj* file is a previously saved access module job:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	'noprompt'	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	'noprompt'	Script fails. Instead, specify an *.amj* file name instead of UNTITLED.
"UNTITLED"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens.

BTEQ Export Example

```
.LOGON bubbater/dbc,dbc ;
database bubba ;
.EXPORT indicdata file="C:\fexp.amj" AXSMOD OLEDB_AXSMOD 'noprompt';
SELECT col1, col2 from tst10k ;
.EXPORT reset
.LOGOFF;
```

Using FastExport

To export from Teradata Database using FastExport

1. Select **Start >Programs >Teradata Client >FastExport** to open Teradata FastExport from the Windows desktop.
2. For the EXPORT command, specify *OLEDB_AXSMOD* as the DLL for Teradata OLE DB Access Module.
3. For the .EXPORT OUTFILE and operating specification, use one of the following, where an *.amj* file is a previously saved access module job:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	'noprompt'	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	'noprompt'	Script fails. Instead, specify an *.amj* file name instead of UNTITLED.
"UNTITLED"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens.

FastExport Example

```
.LOGTABLE job_account_Log;
LOGON perform/test,test ;
DATABASE test2 ;
.BEGIN EXPORT SESSIONS 4 ;
SELECT name, salary FROM job_account ;
.EXPORT OUTFILE "Untitled" AXSMOD Oledb_Axsmod 'noprompt';
.END EXPORT ;
.LOGOFF ;
```

Using Teradata PT

To export data from Teradata Database using Teradata PT, ensure that the Teradata PT job script uses the following specifications:

- TYPE definition as DATACONNECTOR CONSUMER
- The AccessModuleName attribute set as OLEDB_AXSMOD
- The FileName attribute set to the pathname of the *.amj* file

For information about exporting data from Teradata Database using Teradata PT, see "Extracting Data" in the *Teradata Parallel Transporter User Guide*.

Starting an Access Module Load Job from a Teradata Utility

Access module load jobs can be initiated from the following Teradata Utilities:

- Using FastLoad
- Using MultiLoad
- Using TPump
- Using BTEQ
- Using Teradata PT

Using FastLoad

To load to Teradata Database using FastLoad

1. Select **Start >Programs >Teradata Client >FastLoad** to open Teradata FastLoad from the Windows desktop.
2. Use the BEGIN LOADING command. Because Teradata OLE DB Access Module returns data in a format that includes indicator bits, specify the INDICATORS option in the FastLoad BEGIN LOADING command.
3. For the AXSMOD specification, use *OLEDB_AXSMOD* as the DLL for Teradata OLE DB Access Module.
4. Use the DEFINE command to specify the fields to load.
5. Specify the one of the following for the file parameter and operating specification:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	"noprompt"	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	"<blank>"	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	"noprompt"	Script fails. Instead, specify an *.amj* file name instead of UNTITLED.
"UNTITLED"	"<blank>"	**Teradata OLE DB AXSMOD** dialog box opens.

FastLoad Example

```
LOGON DELL2300/taste,taste ;
DATABASE test2 ;
CREATE TABLE "unittestmixtablefe1_29"
      (colinteger INTEGER,
      colsmallint SMALLINT,
      Colbyteint SMALLINT);
```

```
BEGIN LOADING "unittestmixtablefe1_29"
   ERRORFILES unittestmixtablefe1_29_errors1, unittestmixtablefe1_29_err
ors2
   INDICATORS;
AXSMOD Oledb_Axsmod "noprompt";
DEFINE colinteger (INTEGER),
       colsmallint (SMALLINT),
       colbyteint (SMALLINT)FILE=Myfile.amj;
INSERT INTO "unittestmixtablefe1_29"(colinteger, colsmallint, and colbyt
eint)
   VALUES (:colinteger, :colsmallint, :colbyteint);
END LOADING;
LOGOFF;
```

Using MultiLoad

To load to Teradata Database using MultiLoad

1. Select **Start >Programs >Teradata Client >MultiLoad** to open Teradata MultiLoad from your Windows desktop.
2. Use the .LAYOUT command. Because Teradata OLE DB Access Module returns data in a format that includes indicator bits, specify the INDICATORS option in the MultiLoad .LAYOUT command.
3. For the AXSMOD specification, use *OLEDB_AXSMOD* as the DLL for Teradata OLE DB Access Module.
4. Use the IMPORT command to specify the following:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	'noprompt'	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	'noprompt'	Script fails. Instead, specify an *.amj* file name instead of UNTITLED.
"UNTITLED"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens.

MultiLoad Example

```
.LOGTABLE newtest.test1_LOG;
.LOGON perform/test,test;
DATABASE test ;
CREATE TABLE "test1"(col1 numeric(3,0));
.BEGIN IMPORT MLOAD TABLES "test1" CHECKPOINT 0;
.LAYOUT test1_layout INDICATORS;
.FIELD col1 * numeric(3,0);
.DML LABEL test1_label;
INSERT INTO "test1"(col1) VALUES (:col1);
.IMPORT INFILE "c:\oledb\test1.amj"
```

```
      AXSMOD OLEDB_AXSMOD 'noprompt'
      LAYOUT test1_layout
      APPLY test1_label;
.END MLOAD;
.logoff;
```

Using TPump

To load to Teradata Database using TPump

1. Select **Start >Programs >Teradata Client >TPump** to open Teradata TPump from the Windows desktop.
2. Use the .LAYOUT command. Because Teradata OLE DB Access Module returns data in a format that includes indicator bits, specify the INDICATORS option in the TPump .LAYOUT command.
3. For the AXSMOD specification, use *OLEDB_AXSMOD* as the DLL for Teradata OLE DB Access Module.
4. Use the TPump IMPORT command to specify the following:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	'noprompt'	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	'noprompt'	Script fails. Instead, specify an *.amj* file name instead of UNTITLED.
"UNTITLED"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens.

TPump Example

```
.LOGTABLE test.precision_log;
LOGON wuscaesc/tester,dbc;
DATABASE test ;
DROP TABLE "TestPrecision";
DROP TABLE "precision_err";
CREATE TABLE "TestPrecision"(
   munie decimal(18,4));
.BEGIN LOAD SESSIONS 1
   ERRORTABLE "precision_err"
   NOMONITOR
   ROBUST ON;
.LAYOUT precision_layout INDICATORS;
.FIELD munie * decimal(18,4);
.DML LABEL precision_label;
INSERT INTO "TestPrecision"(munie) VALUES(:munie);
.IMPORT INFILE "C:\WINNT\Profiles\Personal\sql_test08.amj"
   AXSMOD OLEDB_AXSMOD 'noprompt'
   LAYOUT precision_layout
```

```
   APPLY precision_label;
.END LOAD;
.LOGOFF;
```

Using BTEQ

To load to Teradata Database using BTEQ

1 Select **Start >Programs >Teradata Client >BTEQ** to open Teradata BTEQ from the Windows desktop.

2 Use the IMPORT command to specify the following:

FILE Parameter	Operating Specification	Outcome
"<*pathname*>.amj"	'noprompt'	Script runs if no errors exist. If errors exist, the script fails.
"<*pathname*>.amj"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens so you can run the script using the access module.
"UNTITLED"	'noprompt'	Script fails. Instead, specify an *.amj* file name instead of UNTITLED..
"UNTITLED"	'<blank>'	**Teradata OLE DB AXSMOD** dialog box opens.

BTEQ Load Example

```
.LOGON bubbater/dbc,dbc;
.SET Session Charset "UTF8";
database bubba ;
drop table "Customers";
CREATE TABLE "Customers" (CustomerID CHAR(5) NOT NULL,
      CompanyName VARCHAR(40) NOT NULL,
      ContactName VARCHAR(30),
      ContactTitle VARCHAR(30),
      Address VARCHAR(60),
      City VARCHAR(15),
      Region VARCHAR(15),
      PostalCode VARCHAR(10),
      Country VARCHAR(15),
      Phone VARCHAR(24),
      Fax VARCHAR(24));
.IMPORT indicdata file="C:\mload.amj" AXSMOD OLEDB_AXSMOD 'noprompt';
.REPEAT *using (CustomerID CHAR(5),
      CompanyName VARCHAR(40),
      ContactName VARCHAR(30),
      ContactTitle VARCHAR(30),
      Address VARCHAR(60),
      City VARCHAR(15),
      Region VARCHAR(15),
      PostalCode VARCHAR(10),
      Country VARCHAR(15),
      Phone VARCHAR(24),
      Fax VARCHAR(24))
```

```
   INSERT INTO "Customers"(
      CustomerID, CompanyName,
      ContactName, ContactTitle,
      Address, City,
      Region, PostalCode,
      Country, Phone,
      Fax)
 VALUES(
      :CustomerID, :CompanyName,
      :ContactName, :ContactTitle,
      :Address, :City,
      :Region, :PostalCode,
      :Country, :Phone,
      :Fax);
.LOGOFF;
.QUIT;
```

Using Teradata PT

To load data from Teradata Database using Teradata PT, ensure that the Teradata PT job script contains the following specifications:

- TYPE definition is DATACONNECTOR PRODUCER
- AccessModuleName attribute is OLEDB_AXSMOD
- FileName attribute is set to the pathname of the *.amj* file

For information about loading data from Teradata Database using Teradata PT, see "Loading Data" in the *Teradata Parallel Transporter User Guide*.

Restoring Default Selections

When the **Teradata OLE DB Access Module** dialog box opens, it automatically displays whatever information was last entered in the dialog box, even if the information was not saved.

It can be helpful to restore the default selections, especially when the dialog box is exited with connection information that cannot be restored.

To start OleLoad with all selections blank (default)

1 Click **Start>Run** and type *`OleLoad nosuchfile.amj`*.

  The following message appears:

  ```
  Could not open the requested file, "nosuchfile.amj", because
  nosuchfile.amj was not found.
  ```

2 Click **OK**. The OleLoad GUI starts with blank fields.

To open a previously saved job

✔ Display previously saved *.amj* information in the **Teradata OLE DB Access Module** dialog box (OleLoad) by doing one of the following:

- In Windows Explorer, open an *.amj* file from the local drive.
- Specify the file from the command line. For example, type: `OleLoad C:\<jobname>.amj`.

Access Module Functions

The following functions are available in Teradata OLE DB Access Module:

- Data Type Mapping
- VARCHAR Constraints
- Character Set Support
- Returned Data Format
- Date and Time Data Types
- Checkpoints and Restarts
- Job Files

Data Type Mapping

OLE DB uses standard OLE and Windows data types. To describe a data type, an OLE DB type indicator is used, which is a variable of the enumerated type DBTYPE. Teradata OLE DB Access Module retrieves the C/C++ data type indicated by the OLE DB type indicator and converts the data type to a Teradata Database data type. The Teradata Database data type is based on the DBTYPE and the DBCOLUMNFLAGS values.

Data type mapping is required for transferring data from Teradata OLE DB providers to the Teradata Database. Table 2 lists the mapping of OLE DB type indicators to Teradata Database data types.

Table 2: Mapping OLE DB Data Type to Teradata Database Data Types

DBTYPE	DBCOLUMNFLAG	Teradata Type
DBTYPE_11		BYTE(n)*
DBTYPE_I2		SMALLINT
DBTYPE_I4		INTEGER
DBTYPE_I8		BIGINT
DBTYPE_UI1		SMALLINT
DBTYPE_UI2		INTEGER

Table 2: Mapping OLE DB Data Type to Teradata Database Data Types (continued)

DBTYPE	DBCOLUMNFLAG	Teradata Type
DBTYPE_UI4		DECIMAL(10,0)
DBTYPE_UI8		BIGINT
DBTPYE_R4		FLOAT
DBTYPE_R8		FLOAT
DBTYPE_NUMERIC		DECIMAL(p,s)*
DBTYPE_DECIMAL		DECIMAL(p,s)*
DBTYPE_CY		DECIMAL(18,4)
DBTYPE_BSTR		VARCHAR(64000)
DBTYPE_IDISPATCH		BYTE(n)
DBTYPE_ERROR		DECIMAL(10,0)
DBTYPE_BOOL		BYTEINT
DBTYPE_VARIANT		Unsupported
DBTYPE_IUKNOWN		BYTE(n)**
DBTYPE_GUID		BYTE(n)**
DBTYPE_BYTES	DBCOLUMNFLAGS_ISFIXEDLENGTH	BYTE(n)**
DBTYPE_BYTES		VARBYTE(n)**
DBTYPE_STR	DBCOLUMNFLAGS_ISFIXEDLENGTH	PERIOD(DATE)
DBTYPE_STR	DBCOLUMNFLAGS_ISFIXEDLENGTH	PERIOD(TIME(p))
DBTYPE_STR	DBCOLUMNFLAGS_ISFIXEDLENGTH	PERIOD(TIMESTAMP(p))
DBTYPE_STR	DBCOLUMNFLAGS_ISFIXEDLENGTH	CHAR(n)**
DBTYPE_STR		VARCHAR(n)**
DBTYPE_WSTR	DBCOLUMNFLAGS_ISFIXEDLENGTH	CHAR(n)**
DBTYPE_WSTR		VARCHAR(n)**
DBTYPE_UDT		Unsupported
DBTYPE_DATE		TIMESTAMP(p), DATE, & FLOAT***
DBTYPE_DBDATE		DATE

Table 2: Mapping OLE DB Data Type to Teradata Database Data Types (continued)

DBTYPE	DBCOLUMNFLAG	Teradata Type
DBTYPE_DBTIMESTAMP		TIMESTAMP(p), DATE, & FLOAT***
DBTYPE_ARRAY		Unsupported
DBTYPE_BYREF		Indicates the data points to the real data value. For example, DBTYPE_I2 | DBTYPE_BYREF means that the data contains the address of a two-byte integer. All supported types can be referenced by DBTYPE_BYREF.
DBTYPE_VECTOR		Unsupported
DBTYPE_RESERVED		Unsupported
DBTYPE_NULL		Unsupported
DBTYPE_EMPTY		Unsupported
DBTYPE_DBTIME		FLOAT
DBTYPE_FILETIME		TIMESTAMP(p), DATE, & FLOAT***
DBTYPE_PROPVARIANT		Unsupported
DBTYPE_HCHAPTER		DECIMAL(10,0)
DBTYPE_VARNUMERIC		VARCHAR(n)**

*DECIMAL(p,s) indicates Teradata data type DECIMAL with precision (*p*) and scale (*s*).

**_n_ represents the number of bytes for BYTE(n) and VARBYTE(n), and the number of characters for CHAR(n) and VARCHAR(n).

***All date and time data types are split into a Teradata DATE type for the date portion of the data type, and a Teradata FLOAT for the time portion (in addition to being available as a Teradata TIMESTAMP(p)).

VARCHAR Constraints

By default, Teradata OLE DB Access Module uses the maximum VARCHAR data type specification supported by Teradata Database: 64000 B. Data types like SQL Server "text" can return a length that greatly exceeds the 64000 B maximum. When selecting columns of this type, remember that:

- Rows with a total length exceeding the maximum are not loaded; long string data are not truncated.
- After Teradata OLE DB Access Module returns all rows it can successfully return, in response to the next request for rows the access module returns an error message indicating the number of rows not loaded. This usually causes the job to fail.

When truncating long string data, use an SQL command to create the rowset to be loaded. When using the SQL command, specify the SUBSTRING function in the select-item-list of the SELECT command to truncate the data before it reaches Teradata OLE DB Access Module.

Character Set Support

The following session character sets are supported for transferring data:

- **UTF-8:** Preferred character set because it accommodates a superset of characters handled by the other character sets.
- **ASCII**: Teradata OLE DB Access Module uses the code page of the system locale (also called the system default ANSI code page) when using ASCII. Teradata's ASCII character set does not match Microsoft's code page, so character conversions produce minor differences. If an exact match is required, use the UTF-8 session character set.

  Note: Previous versions of Teradata OLE DB Access Module used the ANSI-Latin1 (1252) code page when the ASCII session character set was used, regardless of the system locale.
- **KANJISJIS_0S**: Teradata's Kanji_SJIS character set does not exactly match Microsoft's Japanese Shift-JIS code page, so character conversions produce minor differences. If an exact match is required, use the UTF8 session character set.
- **LATIN1252_0A**: All UNICODE character strings transferred to the Teradata Database are converted to ANSI-Latin 1 by Teradata OLE DB Access Module before being passed to a Teradata utility.

ANSI-Latin1 (1252) Code Page

When the ASCII character set is specified, Teradata OLE DB Access Module uses the code page of the system locale. If the code page of the system locale is one in which some characters consume more than one byte, Teradata OLE DB Access Module can return longer character fields during execution of a load job than if the 1252 code page were used. Teradata OleLoad generates load job scripts that account for this change.

Update any existing scripts. For example, a system locale of "Chinese (Taiwan)" has a code page of 950, which has a maximum character size of 2 B. If existing FastLoad job scripts using the ASCII session character set to load character data from Teradata OLE DB Access Module, update the scripts to account for the fact that there may be 2 B per character. For example, double the value of <n> in CHAR(<n>) and VARCHAR(<n>) fields located in the CREATE TABLE and DEFINE statements included in the FastLoad job scripts.

Session Character Sets

All Teradata utilities (except for BTEQ) notify Teradata OLE DB Access Module about the session character set they use. For BTEQ, the access module uses the session character set specified in the **Advanced Setting** dialog box of the Teradata OleLoad GUI.

To set session character sets, specify the character set name as follows:

- For jobs launched from the Teradata OleLoad GUI, specify the session character set in the **Advanced Settings** dialog box.
- For jobs launched without the Teradata OleLoad GUI, specify a character set name as follows:
  - **Teradata FastExport, Teradata MultiLoad, or Teradata TPump** - Specify the -c *character-set-name* parameter at runtime in the command line to set the session character set name.

- **BTEQ or Teradata FastLoad** - Specify the .SET SESSION CHARSET command in a script.
- **Teradata PT** - Specify the USING CHAR(ACTER) SET charset-id phrase in the Teradata PT job script.

Returned Data Format

Teradata OLE DB Access Module provides data to and receives data from Teradata utilities in *pmIDFBin1Plus* format with indicator mode bits. In this format, each row consists of the following:

- A 2-byte record-length indicator specifying the length of everything in the row except the record-length indicator
- An indicator bit for each field specifying whether the field has data or is NULL
- The actual row data
- A new-line character signifying the end-of-record

When creating job scripts, be sure to specify the following:

- INDICATORS in the BEGIN LOADING command of a FastLoad job script
- MODE INDICATOR in the .EXPORT command of a FastExport job script
- INDICATORS in the .LAYOUT command of a MultiLoad job script
- INDICATORS in the .LAYOUT command of a TPump job script
- INDICDATA in the .IMPORT command of a BTEQ load job script
- INDICDATA in the .EXPORT command of the BTEQ export job script
- 'Yes' in the IndicatorMode attribute value and 'Formatted' in the Format attribute value of the DATACONNECTOR operator definition of a Teradata PT job script.

Date and Time Data Types

Two columns are synthesized as follows:

- The date portion can be copied to a DATE, and the column name is shown as ColumnName_DATE.
- The time portion can be copied to a FLOAT, and the column name is shown as ColumnName_TIME.

A number can be appended to the new column names as needed to create unique names. For example, if ColumnName_DATE already exists in the table, ColumnName_DATE_2 can be used as the new column name. You can copy the date and time to a TIMESTAMP data type.

If the data source already contains a column named ColumnName_DATE or ColumnName_TIME, Teradata OLE DB Access Module adds a number to the new column names to maintain unique names for the new columns; for example, ColumnName_DATE_2.

Checkpoints and Restarts

Checkpoint and restart operations are *not* supported for export jobs (unloading data); however, checkpoints and restarts are supported for load jobs (loading data). Restarts for load operations are only supported when the following conditions are met:

- The data being loaded does not change between the initial load attempt and the restart.

  Check the source data to make sure it has not changed between the initial load attempt and the restart. Teradata OLE DB Access Module does not check for changed data during the restart operation. If the data has changed, the loaded data may not reflect the true contents of the source table.
- To load data from the most recent checkpoint, the **Enable scroll backwards** option must be selected on the **Advanced Settings** dialog box before the job starts.

The following can occur during checkpoint/restart operations:

- If a load process loses contact with the destination Teradata Database (for example, the destination Teradata Database resets), Teradata OLE DB Access Module backs up to the previous checkpoint location, and resumes retrieving and returning data from that location when contact is restored (at the request of the load utility).
- If a load process terminates unexpectedly, manually restart the load job by reissuing the job. In this case, the Teradata Database detects that a restart is in progress and (at the request of the load utility) Teradata OLE DB Access Module skips forward to the previous checkpoint location, and resumes retrieving and returning data from that location.

Caution: For bulk load operations, the number of rows displayed in the progress dialog box might not be the same as the number of rows returned to the utility if a restart occurs. In this case, the progress dialog box might display a larger number of row retrievals than the number of rows actually retrieved from the data source.

Job Files

An access module job is a set of parameters defining OLE DB data source information to Teradata OLE DB Access Module. Access module jobs include the following parameters:

- Data source details for both of the data sources supplying and receiving data
- Table name for the data source supplying data
- List of columns for the data source supplying data

Teradata OLE DB Access Module uses *.amj* files that include the name of the file containing the information used by the access module when the job was executed; for example, *filename.amj*. An *.amj* file that loads a table from Oracle might include such information as the specific OLE DB provider to be used, the name of the Oracle database server containing the table to be loaded, the user name and password to log on that server, and the name of the table to be loaded.

Note: Teradata OLE DB Access Module does not use the convention of using the name of the data source as the pathname for an operation.

Use the *filename.amj* name for the following:

- FILE = *filename* specification in the IMPORT command in a BTEQ load job script
- FILE = *filename* specification in the EXPORT command in a BTEQ export job script
- OUTFILE *fileid* specification in the .EXPORT command in a FastExport job script
- FILE = *filename* specification in the DEFINE command in a FastLoad job script
- INFILE *filename* specification in the IMPORT command in a MultiLoad or TPump job script
- Value of the FileName attribute specification in the DATACONNECTOR operator definition in a Teradata PT job script.

File Format

Teradata OLE DB Access Module creates *.amj* files, which are saved in an extensible markup language (XML) based text format. You can use Teradata OleLoad to view, edit, and save access module job files.

Note: Access module job files adhere to version 1.0 of the XML specification. Binary format files created by earlier versions can be opened by Teradata OLE DB Access Module; however, XML-based *.amj* files cannot be opened with the versions earlier than 02.02.00.

Some text is *altered* in an access module job file when the text contains illegal characters or characters that already have assigned meaning in XML. Altered text is identical to the unaltered text for all characters except for the following:

- Less-than symbol (<)
- Greater-than symbol (>)
- Equal sign (=)
- Ampersand (&)
- All other characters for which the UTF-16 encoding of the character is not in the range [0x0020 - 0xd7ff] or in the range [0xe000 - 0xfffd]

Altered characters are replaced with an equal sign (=) followed by four hexidecimal digits representing the UTF-16 encoded value of the character. For example, an ampersand is replaced by `=0026` because the UTF-16 encoding for an AMPERSAND is `0x0026`.

Example

The following example, which explains the contents of an *.amj* file, assumes an understanding of the XML specification and the format is subject to change without notice.

Files begin with an XML declaration specifying the XML version and character encoding used and may have comments interspersed as permitted by the XML specification; these comments are discarded when the file is processed.

Following the XML declaration is a processing instruction containing the version number of the first version of Teradata OLE DB Access Module that handled the *.amj* file:

```
OLE_DB_AXSMOD_<FirstCompatibleVersion>
```

After this processing instruction is the root element, named OLE_DB_AXSMOD_Jobs, which has one child element, named Job, which has four child elements: Source, Destination,

CharacterEncoding, and might contain CheckpointInterval, LargeDecimalSupport, RowsPerFetch, BufferSize, and EnableScrollBackwards.

- **Source** - The Source element can contain:
  - One child element named DataSourceParseName
    - A DataSourceParseName element contains an altered parse name uniquely identifying the selected OLE DB provider.
  - One child element named DataSourceProperties
    - A DataSourceProperties element contains the values of all properties needed to initialize the provider, including provider-specific properties. It contains one child PropertySet element for each property set supported by the provider.
    - Each PropertySet element contains one child PropertySetId element followed by one child Property element for each supported property in that property set.
    - A PropertySetId element contains identification of a property set. For some property sets, Teradata OLE DB Access Module associates a symbolic name for the property set; for example, DBPROPSET_DBINIT. For these property sets, the identification is the altered symbolic name. For other property sets (such as, many provider-specific property sets) the identification is an altered textual representation of the GUID identifying the property set; for example, {c200e360-38c5-11ce-ae62-08002b2b79ef}.
    - Each Property element contains one PropertyId element followed by one PropertyType element followed by one PropertyValue element.
    - A PropertyId element contains the identification of a property (within a property set). For some properties, Teradata OLE DB Access Module associates a symbolic name for the property; for example, DBPROP_INIT_TIMEOUT. For these properties, the identification is the altered symbolic name. For other properties (such as, many provider-specific properties), the identification is the altered textual hexadecimal representation of the integer identifying the property; for example, 0x43.

      A property's value may be included in one of two ways. The first method is used when the property value can be converted to a string (VT_BSTR) and then back to the original value using the features supplied by the COleVariant class (which is part of the Microsoft Foundation Classes (MFC)). Otherwise, the second method is used.

      When the first method is used, a PropertyType element contains the data type associated with the property and a PropertyValue element contains the altered textual representation of the value for the property. For some PropertyType elements, Teradata OLE DB Access Module *knows* a symbolic name for the data type; for example, VT_I4. For these types, PropertyType contains the altered symbolic name. For other data types, PropertyType contains the altered textual hexadecimal representation of the type number; for example, 0x43.

      When the second method is used, PropertyType contains "ARCHIVED" and the PropertyValue element contains the altered byte pattern obtained in an archive by

dumping the property value to an archive (managed by an object of the MFC CArchive class) using the COleVariant insertion (<<) operator.

Note: To learn more about OLE DB defined properties, see the *OLE DB Programmer's Reference* at http://msdn.microsoft.com/library/. For documentation of provider-specific properties, consult the documentation or supplier of the relevant OLE DB provider.

- One child element named TableSelection or one child element named TableName or one child element named TableCommand.
  - The TableSelection element is present when a source table is selected from the tree control. It contains one child element named Catalog, followed by one child element named `Schema`, followed by one child element named Name.
  - The Catalog element contains the altered name of the catalog of the table selected by the user. The Schema element contains the altered name of the schema of the table selected by the user. The Name element contains the altered name of the table selected by the user.

    When a source table is specified by name using the edit control, the `Source` element has a child element named TableName containing the altered table name.

  The TableCommand element is present when a source table has been specified by command using the edit control, for example, a SQL SELECT... command.

  Note: The precise manner in which the OLE DB provider uses catalog names, schema names, table names, and commands is specific to the provider. Consult the documentation or supplier of the relevant OLE DB provider for details relating to a particular provider.
- One child element named LocationOfLogTables and one child element named OtherDatabase:
  - The contents of the LocationOfLogTables depends on which option you select in the **Location of log tables** frame on the **Teradata OleLoad - Advanced Settings** dialog box. If you select the **User's default database** option, then the LocationOfLogTables element is zero. This option is enabled in the export job operation. If you select **the Source database** option, then the LocationOfLogTables element is 1. If you select the **Other database** option, then the LocationOfLogTables element is 2.

    Note: The OtherDatabase element contains the altered string from the **Other database** box in the **Teradata OleLoad - Advanced Settings** dialog box.
- One child element named Columns:
  - The Columns element contains one child element named Column for each column in the selected source table. Each Column element contains one empty child element named Selected, if the column is selected. If the column is not selected, the Selected child element is not present. Each Column element also contains one child element named SourceName, followed by one child element named DestinationName, followed by one child element named TypeName.
  - A SourceName element contains the altered name of the source column. This is the name that appears in the **Src. Column Name** column of the list of available columns in OleLoad. When you click **Teradata Database** from the **Select a source** list, this

name is derived from the source Teradata Database by issuing a HELP COLUMN command and removing trailing SPACE (" ") characters from the returned Column Name value. When you select an OLE DB provider from the **Select a source** list, this name is the name returned in the pwszName field of the DBCOLUMNINFO structure by the IColumnsInfo::GetColumnInfo() method when applied to the source table.

- A DestinationName element contains the altered destination column name. This is the name that appears in the **Dest. Column Name** column of the list of available columns in OleLoad. Often this name is the same as the source column name, but it need not be the same. You can change the name using OleLoad.
- A TypeName element contains the altered Teradata data type name for the column. This is the name that appearing in the **Data Type** column of the list of available columns.

- **Destination** - The Destination element contains one child element named TableName, and may contain one child element named DataSourceParseName, one child element named DataSourceProperties, one child element named ReferentialIntegrityIsChecked, and one child element named IndexUpdatesAreRequired.
  - A TableName element that is a child of the Destination element contains the altered name of the destination table. This is the name in the **Table name** box in the **Edit the table name** frame of the **Teradata OleLoad - Advanced Settings** dialog box.

    If you select the **Referential integrity check** check box in the **Bulk Loading Options** frame of the **Teradata OleLoad - Advanced Settings** dialog box, the ReferentialIntegrityIsChecked element is present. This element sets the DBPROPVAL_BO_REFINTEGRITY bit in the DBPROP_ROW_BULKOPS property in the DBPROPSET_ROWSET property set used to create the rowset for the destination table. The ReferentialIntegrityIsChecked element serves as a hint to the destination provider that referential integrity constraints do not need to be checked or enforced for changes made through the rowset. This is effective only during exports to a provider that supports DBPROPVAL_BO_REFINTEGRITY.

    If you select the **Index updates required** check box in the **Bulk Loading Options** frame of the **Teradata OleLoad - Advanced Settings** dialog box, the IndexUpdatesAreRequired element is present. This element sets the DBPROPVAL_BO_NOINDEXUPDATE bit in the DBPROP_ROW_BULKOPS property in the DBPROPSET_ROWSET property set used to create the rowset for the destination table. The IndexUpdatesAreRequired element serves as a hint to the destination provider that the provider is not required to update indexes based on inserts or changes to the rowset. Any indexes need to be recreated following changes made through the rowset.
- **Character Encoding** - The CharacterEncoding element contains the altered character set name for this job.
- **CheckpointInterval** - The CheckpointInterval element contains the altered string from the **Checkpoint Interval** box in the **Teradata OleLoad - Advanced Settings** dialog box when you enter any integral value.

- **LargeDecimalSupport** - The LargeDecimalSupport element contains Supported and NotSupported. If Supported, the access module can return DECIMAL values greater than 18 digits; otherwise the maximum returned DECIMAL values is 18 or less.
- **RowsPerFetch** - The RowsPerFetch element, if present, contains the value specified in the **Advanced Settings** dialog box.
- **BufferSize** - The BufferSize element, if present, contains the value specified in the **Advanced Settings** dialog box.
- **EnableScrollBackwards** - The EnableScrollBackwards element, if present, contains the value specified in the **Advanced Settings** dialog box.

Improving Performance

Both database factors and access module factors can affect the performance of an access module job.

Database Factors

When using the access module products , consider the following database factors, which might improve performance:

- Use the fastest Teradata client load or export utility that meets the requirements. For example, if a new (initially empty) table is being loaded, it is faster to use Teradata FastLoad than TPump. See Appendix B of the *Database Administration* manual for additional information about selecting the appropriate Teradata utility.
- Use the appropriate number of sessions to the Teradata Database. Using multiple sessions in parallel can provide higher throughput.
- Use fast, dedicated networks to connect a data source and Teradata Database. Minimize other traffic on the connections consuming bandwidth.
- Ensure that no other applications and services are running on your system while it is running a load job.

Access Module Factors

In addition to database performance factors that might improve performance, consider the following issues that are specific to Teradata OLE DB Access Module:

- If multiple OLE DB providers can access a particular data source, choose the fastest provider. A list of OLE DB providers is available at http://www.sqlsummit.com/oledbVen.htm.
- Look for bottlenecks in the load operation. Possible locations might be the OLE DB provider on the source side or on the destination (Teradata) side of the transfer. When not in batch mode, observe the statistics in the dialog box that is displayed during the load job execution. Teradata OLE DB Access Module displays how many rows it retrieves from the source OLE DB provider and how many rows it returns to the Teradata load software:

- If the “a” statistic is constantly a good deal higher that the “b” statistic, the data source is supplying the data faster than the load software is requesting it, and thus the speed of the load job is limited by the destination (Teradata Database).
- If the “a” statistic stays close to the “b” statistic then the data source is not able to provide the data as fast as it can be loaded and thus the speed of the load job is limited by the data source side of the transfer.

It might also be possible to improve performance by using batch mode to disable the dialog box that displays the status and progress status of the transfer. This frees the processing resources used to display and updates dialog box.

- If the data sources support SQL commands, use a SQL SELECT command that returns the minimum amount of data to be loaded, that is, only the required columns and rows.
- In load jobs that transfer fixed-length text data (as in CHAR(n) data), if the text data only contains characters that can be properly transferred using the ASCII session character set, use the ASCII session character set instead of the UTF8 session character set. When the UTF8 session character set is used, extra padding characters are added to the end of these fields during the load transfer. These padding characters consume some of the bandwidth available between the Teradata Database and the system running the Teradata client utility. Another option for avoiding this padding is to CAST the columns to a variable-length test data type (such as VARCHAR(n)).
- Run load jobs on a system with at least two CPUs to allow for the retrieval of the data from the data source and the returning of the data to the Teradata Database. For best performance, do these operations in parallel.
- If a value is entered in the **Checkpoint interval** field, performance might be enhanced if values are also added to the **Rows per fetch** and **Buffer size** boxes, and the **Enable scroll backwards** option is selected.
  - If **Rows per fetch** is blank, the default is 10 jobs. Increasing the number too much might prevent rows from being retrieved in parallel, which could decrease performance. Consider limiting the number to one-third or less of the number of rows than can fit into the internal buffer (**Buffer Size)** after deducting from the buffer whatever space is consumed by holding rows between checkpoints when the checkpoint/restart feature is enabled.
  - If **Buffer size** is blank, the default is 128 KB. No less than the default is accepted. It is recommended that the number be three times the rows specified in Rows per fetch; however, if the number is too large, performance might drop because the chances increase that data in the buffer will get paged out to disk, especially on systems with low memory or when running other load task.
  - If **Enable scroll backwards** is selected, restarts begin at the most recent checkpoint, but the internal buffer (**Buffer size**) does not store data for restarts. Selecting this option can cause problems because some data sources can consume all available memory, which can result in job failure. Therefore, caution is recommended when in using this option. If Checkpoint interval is selected while this option is blank, restarts can still succeed if **Buffer size** is large enough to store the rows loaded between checkpoints.

Troubleshooting

Attributes Missing

Problem: Some table attributes might not be propagated from the source table to the destination table because only OLE DB-supported attributes are handled. Teradata OLE DB Access Module lets you connect a Teradata utilities to arbitrary OLE DB providers. The focus is on loading from an OLE DB-supported data source to Teradata Database and exporting from Teradata Database to an OLE DB-supported data source. As such, only OLE DB-supported attributes are handled.

Solution: To preserve Teradata-specific table attributes (such as the display format information for date and time that can be attached to columns), edit the sample script created by Teradata OleLoad to include the desired formatting for the date and time columns. The default date and time formats in the sample load scripts are YYYY-MM-DD and 99:99:99. These come from string entries number 78 and 40 in the resource-only .dll (OLEDB_AXSMODenu.dll).

Informix Not Available

Problem: Tables in Informix cannot be selected.

Solution: The Informix OLE DB provider does not support quoted table names even though it indicates to Teradata OLE DB Access Module that it must use quotes. Consequently, data can be loaded from Informix only if the Command option is selected and the table name is entered without quotes. The Command option is not available for FastExport, so data cannot be exported from Teradata Database to Informix.

Kanji Cannot be Loaded with BTEQ

Problem: Kanji characters cannot be loaded if BTEQ is used with the access module.

Solution: Unlike other Teradata utilities, BTEQ cannot communicate to Teradata OLE DB Access Module information about the session character set that is used to transfer data. Instead, the ASCII default code page is used to transfer data, resulting in possibly inaccurate data. Use an alternate Teradata utility.

Multi-Code Pages

Problem: Attempts fail when trying to load a table that contains columns that use multiple codes.

Solution: Use the UTF8 session character set to transfer tables that use multiple code pages.

Server Data Type is Always Set to Unicode

Problem: The server data type of a character string is always set to Unicode even if KANJSIJIS_OS is specified as the session character set.

Solution: When creating load scripts, Teradata OLE DB Access Module assigns a character data type for a destination table based on the session character set that is specified. If ASCII or LATIN1252_0A is specified, then a LATIN character data type is assigned; otherwise, a Unicode character data type is assigned. Scripts can be modified to use any supported character data type.

Inaccessible Data After Errors

Problem: Access is locked to the portion of a table that was loaded before an error occurs.

Solution: When a FastLoad job halts due to an error during the loading phase, the job is put into the paused state. In this state, the table that was being loaded and the two error tables are locked and cannot be accessed. To unlock these tables, submit a FastLoad job that contains a BEGIN LOADING command and an END LOADING command.

Note: After executing the END LOADING command, you will be able to access the tables, but the original job cannot be restarted.

Unexpected Exceptions

Problem: Unexpected exceptions occur while using Teradata OLE DB Access Module.

Solution: It is possible to find diagnostic information for an unexpected exception using Dr. Watson for Windows.

Note: This process is *not* supported by Teradata.

CHAPTER 3

Named Pipes Access Module

Teradata Named Pipes Access Module provides an interprocess communication link between a writer process, such as Teradata FastExport, and a reader process, such as Teradata FastLoad. Topics include:

Supported Operating Systems

Teradata Named Pipes Access Module runs on the following operating systems:

- Linux 32-bit and 64-bit, and Opteron 32-bit and 64-bit
- HP-UX 32-bit and 64-bit
- IBM® AIX 32-bit and 64-bit
- SUN® Solaris SPARC 32-bit and 64-bit
- SUN Solaris Opteron 32-bit and 64-bit
- Microsoft® Windows 2000, XP, Server 2003, and Vista

Note: Unless stated otherwise in this chapter, the term *Windows* means any Windows 2000/XP/2003/Vista system.

For the most current information about supported operating systems, see Supported Releases in the Preface.

Supported Teradata Utilities

Teradata Named Pipes Access Module can be used on a number of different operating systems. The following table lists supported Teradata load and unload utilities by operating system.

Table 3: Teradata Utilities Supported by the Named Pipes Access Module

Operating System		Teradata Named Pipes Access Module Operates with					
		BTEQ	Teradata FastExport	Teradata FastLoad	Teradata MultiLoad	Teradata TPump	Teradata Parallel Transporter
Linux	32-bit	Yes	Yes	Yes	Yes	Yes	Yes
UNIX	HP-UX						
	32-bit	Yes	Yes	Yes	Yes	Yes	Yes
	64-bit	Yes	Yes	Yes	Yes	Yes	Yes
	IBM AIX (32-bit and 64-bit)	Yes	Yes	Yes	Yes	Yes	Yes
	SUN Solaris 10 Opteron 32-bit	Yes	Yes	Yes	Yes	Yes	No
	SUN Solaris on SPARC 8, 9, and 10						
	32-bit	Yes	Yes	Yes	Yes	Yes	Yes
	64-bit	Yes	Yes	Yes	Yes	Yes	Yes
Windows	2000	Yes	Yes	Yes	Yes	Yes	Yes
	XP (32-bit and 64-bit)	Yes	Yes	Yes	Yes	Yes	Yes
	2003	Yes	Yes	Yes	Yes	Yes	Yes
	Vista	Yes	Yes	Yes	Yes	Yes	Yes

Access Module Names

Depending on your operating system, Teradata Named Pipes Access Module can be either a shared library, a shared object, or a dynamic link library. To use Teradata Named Pipes Access Module with a Teradata utility you must reference the file name in your job script. The following table lists the Teradata Named Pipes Access Module name to use as the AXSMOD *name* specification in your Teradata utility job script.

Table 4: AXSMOD Name Specifications for Teradata Named Pipes Access Module

Operating System	Teradata Parallel Transporter Version	Named Pipes Access Module
Linux	Shared library file called *np_axsmod.so*	Shared library file called *np_axsmod.so*

Table 4: AXSMOD Name Specifications for Teradata Named Pipes Access Module (continued)

Operating System		Teradata Parallel Transporter Version	Named Pipes Access Module
UNIX	HP-UX PA-RISC	Shared library file called *np_axsmod.sl*	Shared library file called *np_axsmod.sl*
	HP-UX IA64	Not Available	Shared library file called *np_axsmod.so*
	IBM-AIX	Shared object file called *np_axsmod.so*	Shared object file called *np_axsmod.so*
	Solaris SPARC	Shared object file called *np_axsmod.so*	Shared object file called *np_axsmod.so*
	Solaris Opteron	*np_axsmod.so*	*np_axsmod.so*
Windows 2000/XP/2003		Dynamic link library file called *np_AXSMOD.dll*	Dynamic link library file called *np_AXSMOD.dll*

Data Flow

Pipes are bidirectional interprocess communication mechanisms on UNIX, Linux, Windows 2000/XP/2003 systems. They provide input and output file structures accessed by different applications and processes on a first-in-first-out (FIFO) basis.

Using pipes, instead of disk or tape files, provides a substantial performance improvement for data transfer operations between two complementary data extract and load utilities, such as FastExport, FastLoad, and TPump. However, standard pipe mechanisms do not support checkpoint and restart functions because FIFO file structures are not cached, leaving them no way to revert to an earlier position after a system failure or restart. Teradata Named Pipes Access Module caches pipe output data stream in a fallback data file that supports checkpoint and restart functions and provides quick, easy recovery from the following:

- Restarts on the destination database
- Crashes on the system running the load utility

Note: Teradata Named Pipes Access Module does not support checkpoint or restart operations on the source database.

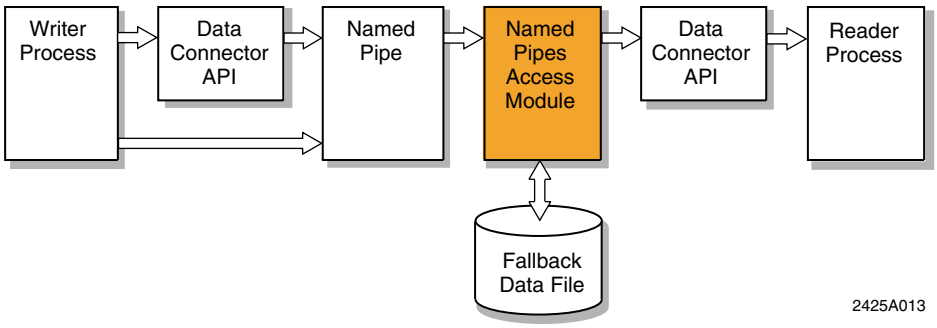
You can use Teradata Named Pipes Access Module a variety of ways:

- On UNIX with client load and unload utilities with the following configurations:
  - Named pipes
  - Unnamed pipes and file descriptor devices
  - Teradata Parallel Transporter
- On Windows 2000/XP/2003

With Load and Unload Utilities

Figure 4 shows how Teradata Named Pipes Access Module works with client load and unload utilities.

Figure 4: Data Flow Between Named Pipes and Load/Unload Utilities



A writer process writes an input data stream to a specified pipe either directly or through the Data Connector API. In response to requests from a reader process, Teradata Named Pipes Access Module does the following:

- Reads the data from the specified pipe
- Copies it to the fallback data file
- Submits it to the reader process

Teradata Named Pipes Access Module can also read data from an ordinary file or from a file descriptor file system device on UNIX systems. However, using the module with an ordinary file is not recommended and does not yield the same performance as using a flat file without the access module.

The Named Pipes Access Module supports only read operations from a named pipe. You cannot use the Named Pipes Access Module to write to a pipe. Additionally, the access module is a block data transfer module; it is neither data-type nor record sensitive.

Writer Process

The writer process can be a client extraction utility, such as FastExport, or any other application, data source, or device that can provide data in a format supported by the reader process. On UNIX systems the writer process must run on the same system as the reader process. UNIX pipes cannot span a network.

The interface between the writer process and the Named Pipes Access Module can be either a named pipe or the Data Connector API. On Solaris SPARC systems, the interface can also be unnamed pipes. However, Teradata Named Pipes Access Module does *not* support interconnection through unnamed pipes on the following operating systems because these systems do not have file descriptor file systems:

- HP-UX
- IBM AIX
- Linux

- Windows 2000
- Windows XP
- Windows 2003

Reader Process

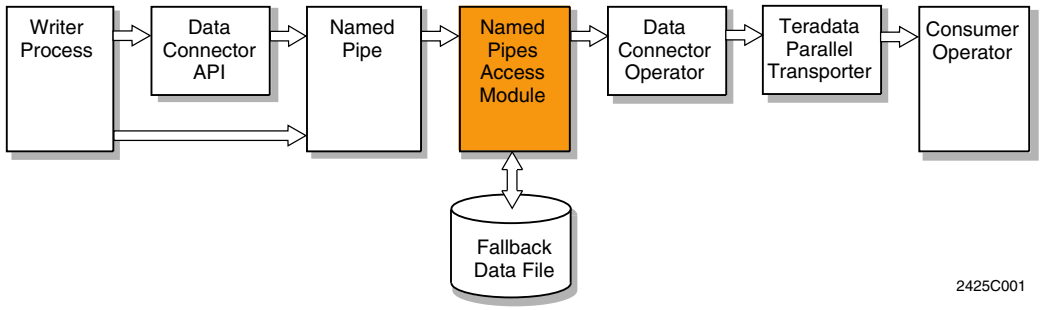
The reader process can be a client load utility, such as Teradata FastLoad, Teradata MultiLoad, or Teradata TPump. Because Windows named pipes can span networks, the reader and writer processes can reside on different network-connected Windows systems.

The Data Connector API provides the interface between the reader process and Teradata Named Pipes Access Module.

With Teradata Parallel Transporter Infrastructure

The Named Pipes Access Module can be used transparently with any Teradata Parallel Transporter (Teradata PT) consumer operator through the DataConnector operator. Figure 5 shows how the UNIX Named Pipes Access Module communicates with Teradata PT.

Figure 5: Data Flow Between Teradata Named Pipes and Teradata Parallel Transporter



Data flows through a UNIX named pipe between a writer process, such as Teradata FastExport, and Teradata PT with a consumer operator, such as Load operator, and the SQL Inserter operator, Teradata PT communicates with the Data Connector operator, which communicates with the UNIX Named Pipes Access Module.

On UNIX systems, the module supports transfer with Teradata PT through named pipes and files. Refer to *Teradata Parallel Transporter Reference* for more information.

Writer Process

Because UNIX pipes cannot span a network, both the writer process and the Teradata PT process must reside on the same UNIX system. The writer process does not require an instance of the Data Connector module. The writer process can be any third-party application that supplies data through a named pipe in the format supported by the reader process.

Reader Process

The reader process is the UNIX Named Pipes Module, *np_axsmod.so*, which tracks data flow and copies inbound data to a fallback data file. If Teradata PT determines it must fall back to

an earlier point in the data stream, it issues the standard File Set Position access command to *np_axsmod.so*, which supplies subsequent reads from the data it saved in the fallback data file.

Using Teradata Named Pipes Access Module

You can use Teradata Named Pipes Access Module a variety of ways:

- On UNIX with client load and unload utilities with the following configurations:
  - Named pipes
  - Unnamed pipes and file descriptor devices
  - Teradata PT
- On Windows 2000/XP/2003

With Client Load and Unload Utilities

On UNIX systems, you can use the Named Pipes Access Module with named pipes. On Solaris SPARC systems, you can also use the access module with unnamed pipes that correspond to open file descriptor file system devices (*/dev/fd* devices).

With Named Pipes

To use Teradata Named Pipes Access Module on UNIX with named pipes:

1. Use the UNIX *mknod* command with the *p* option to create a named pipe. In the following example, */tmp/mypipe* is the name of the pipe:

   ```
   /sbin/mknod /tmp/mypipe p
   ```

2. Program the writer process to send its output stream to the named pipe, as in the following FastExport script example:

   ```
   .EXPORT OUTFILE /tmp/mypipe;
   ```

3. Program the reader process to read from the named pipe as in the following FastLoad script example:

   ```
   axsmod np_axsmod.so "fallback_directory=...";
   define file= /tmp/mypipe;
   ```

4. Launch both the writer and the reader processes, as in the following example where *flod.cmds* is the name of the FastLoad job script file:

   ```
   fexp <fexp.cmd & fastload <flod.cmds &
   ```

   In this example, UNIX connects both processes through the named pipe */tmp/mypipe*.

With Unnamed Pipes and File Descriptor Devices

To use Teradata Named Pipes Access Module with unnamed pipes and file descriptor devices:

1. Program the writer process to send its output to a file descriptor device greater than 2 (*stderr*), such as */dev/fd/4*, as in the following FastExport script example:

   ```
   .EXPORT OUTFILE /dev/fd/4;
   ```

2 Program the reader process to read from a file descriptor device greater than 2 (*stderr*), such as */dev/fd/3*, as in the following FastLoad script example:

```
axsmod np_axsmod.so "fallback_file=...";
define file= /dev/fd/3...;
```

3 Plumb the resulting file descriptors together using a shell pipeline, such as:

```
fexp <fexp.cmds 4>&1 >fexp.out | \
fastload 3<&0 <flod.cmds >flod.log
```

In this example UNIX diverts the FastExport standard output to the file *fexp.out* and:

- *fexp.out* is the name of the diverted FastExport output file
- *flod.cmds* is the name of the FastLoad job script file

With Teradata Parallel Transporter

To use Teradata Named Pipes Access Module with Teradata PT on UNIX:

1 Create a named pipe (for example */tmp/mypipe*).

2 Create a Teradata PT script that specifies */tmp/mypipe* as the filename opened by the module. For example, the script *tbuild.txt* contains a statement similar to the following, defining the Teradata PT Data Connector Operator:

```
DEFINE OPERATOR DataConnector ()
  TYPE DATACONNECTOR PRODUCER
  OUTPUT SCHEMA Tab3schema
  ATTRIBUTES
    (
    VARCHAR FileName                = '/tmp/mypipe',
    VARCHAR PrivateLogName          = 'DcImport.log',
    VARCHAR AccessModuleName        = 'np_axsmod.so',
    VARCHAR AccessModuleInitStr     = 'ld=. fd=.',
    VARCHAR IndicatorMode           = 'N',
    VARCHAR OpenMode                = 'Read',
    VARCHAR Format                  = 'Formatted'
    );
```

The following attributes are relevant to the Named Pipes Access Module:

- *FileName* specifies the name of the named pipe.
- *AccessModuleName* specifies the Named Pipes Access Module. For example, on Solaris SPARC, the *AccessModuleName* is *np_axsmod.so*.
- *AccessModuleInitStr* specifies the access module's initialization string.

All other Teradata PT Data Connector Operator attributes are transparent to the access module.

3 Code an Export Operator script named *fexp.coms* containing a statement similar to the following:

```
.EXPORT OUTFILE /tmp/mypipe ;
```

4 Launch Export Operator and Teradata Parallel Transporter with shell commands similar to the following:

```
fexp < fexp.cmds > fexp.out & tbuild -f tbuild.txt &
```

In this example, UNIX connects both processes through the named pipe */tmp/mypipe*.

The Load operator can direct the access module to save a checkpoint in case the job must be restarted.

For Windows

On a Windows system, use the following procedure to Teradata Named Pipes Access Module with client load and unload utilities, such as FastLoad or FastExport.

Named pipes follow the Microsoft Universal Naming Convention (UNC) for networked entities: \\ *system_name*\ *pipe_name*

Specifying the system name as part of the pipe name enables data transfer between networked systems. Using a period character (.) as the *system_name* specifies the local system. Unlike UNIX, named pipes on Windows are not persistent. The access module creates them automatically, and they are destroyed when the access module closes them.

1. Program the reader process to specify the Named Pipes Access Module, as in the following FastLoad script example:

   ```
   axsmod np_AXSMOD.dll "fallback_directory=...";
   define file=\\.\pipe\mypipe...;
   ```

2. Program the writer process to send its output stream to the named pipe, as in the following Fast Export script example:

   ```
   .EXPORT OUTFILE \\.\pipe\mypipe;
   ```

3. Launch the reader and writer process from two different command windows (CMD.EXE).

   Note: On Windows systems, the reader process *must* be running and in the wait mode for a pipe read operation before you launch the writer process. The FastLoad utility, for example, indicates this status by displaying: `Starting to send to RDBMS with record 1.`

   a. To launch FastLoad in one command window, enter the following, where *flod.cmd* is the name of your FastLoad job script file:

      ```
      fastload <flod.cmd
      ```

   b. Wait until FastLoad has initialized and is ready for a pipe read operation.

   c. To launch FastExport in another command window, enter the following:

      ```
      fexp <fexp.cmd
      ```

Restarting a Job

Restarting the Named Pipes Access Module depends on the utilities being used.

With Client Load and Unload Utilities

The Named Pipes Access Module supports normal checkpoint and restart/recovery operations on the reader process system, but has no such interaction with the writer process.

Routine recovery operations on the reader process system are handled automatically by the Named Pipes Access Module. They require no manual intervention:

- If the reader process terminates unexpectedly, restart the job, which causes the Named Pipes Access Module to use the fallback data file to relocate to the last checkpoint in the data stream.
- If the writer process terminates unexpectedly, manually abort the reader process and resynchronize the job.

The writer process generally restarts from its beginning, while the reader process falls back to the last checkpoint, allowing the Named Pipes Access Module to synchronize the two. You might need to take the following additional steps to complete the restart operation, depending on which client load and unload utilities are used:

1 Prepare the writer process source for a clean start.

  The FastExport utility, for example, uses a log table to determine that a task was interrupted. To start an interrupted FastExport job from its beginning, you must first drop the FastExport log table.

  Note: The log table is specified by the LOGTABLE command in the FastExport job script.

2 Modify the reader process job script. For example, in the FastLoad utility, do the following:

  a Remove any statement that drops the table being loaded.

  b Remove any statement that creates the table being loaded.

3 Launch both the writer and the reader processes as described in "Using Teradata Named Pipes Access Module" on page 60. The Named Pipes Access Module uses the fallback data file from the interrupted job to locate the restart position in the data stream from the writer process.

With Teradata Parallel Transporter

If a transfer terminates unexpectedly, you can restart the transfer. Under Teradata Parallel Transporter, the pipe writer process must restart from its beginning, but the Teradata Parallel Transporter infrastructure falls back to its last checkpoint, allowing the access module to synchronize the two. This saves time because the Teradata Parallel Transporter infrastructure does not have to reinsert into the destination table those records that have already passed through the pipe.

To specify a restart, issue the tbuild command with the *-r* option, as in the following example:

```
fexp < fexp.cmds > fexp.out & tbuild -r -f tbuild.txt &
```

In Teradata Parallel Transporter, the consumer operator cannot automatically direct the access module to restart, as do the client load utilities. All restarts require operator intervention.

Operational Considerations

The following things should be considered when using the Named Pipes Access Module.

Fallback Data File Space Requirements

The fallback data file requires enough space to save all of the data between two successive checkpoints. If the directory that you specify for the fallback data file does not have enough free space to save all of the data read during the checkpoint interval, then the Named Pipes Access Module will not be able to provide all of the data to support a subsequent restart operation.

Always specify a directory for the fallback data file that has enough free space to store all of the data expected during the checkpoint interval.

Example

The "Checkpoint Tradeoffs" subsection in Chapter 2 of *Teradata FastLoad Reference* recommends checkpoints be taken as follows:

- Every 100,000 records if each record is less than 4 Kb
- Every 50,000 records if each record is more than 4 Kb

These guidelines imply that the fallback data file should range from 200 to 400 Mb in size for small- to medium-sized databases. For large databases, the fallback data file might need to be 1 Gb or greater.

Deleting the Fallback Data File

By default, the Named Pipes Access Module reports an end-of-file condition and deletes the fallback data file when the last record written to the pipe is read and sent to the reader process. But because the module handles only raw data streams and disregards data formats imposed by the writer or reader processes, it cannot distinguish between a normal end-of-file condition and an end-of-data condition that would occur if the writer process halts prematurely.

If the writer process terminates before achieving a normal end-of-file condition, the Named Pipes Access Module, by default, interprets the empty pipe as a normal end-of-file condition and deletes the fallback data file. In this case, the data would no longer be available to support a subsequent reader process restart operation.

To override automatic deletion of the fallback data file, include the confirm_fallback_deletion=y (cfd=y) parameter in the Named Pipes Access Module initialization string. The Named Pipes Access Module will then prompt you to confirm the action before deleting the fallback data file.

Fallback Level Restriction

The fallback data file is limited to one level. The data stream from each successive checkpoint interval overwrites that of the prior interval. Attempting to restart at a file position earlier than the last checkpoint position produces an error condition.

Deleting the Log File

The Named Pipes Access Module neither deletes nor restricts the growth of the optional log file. If you specify a log file in the Named Pipes Access Module initialization string, you must also determine when to truncate or delete the file.

Open Pipes Restriction

The number of open pipes is limited to one per instance of the Named Pipes Access Module under the following conditions:

- When you specify a fallback data file name

  The one-to-one relationship between open pipes and fallback data files imposes this restriction. To avoid this constraint, allow the Named Pipes Access Module to assign the fallback data file names by specifying only a directory name or accepting the default directory name for fallback data files.
- When you use unnamed pipes on a Solaris SPARC system

  Because a file descriptor file system device can be opened by more than one process at the same time, you cannot allow the Named Pipes Access Module to generate default fallback data file names when using unnamed pipes. Instead, you must specify a fallback data file name for the unnamed pipe, thereby imposing the one-pipe-per-instance restriction.

Teradata Parallel Transporter Restrictions

The following restrictions apply to using the Teradata Parallel Transporter:

- Unlike the client load utilities, Teradata Parallel Transporter cannot use the file descriptor file system devices to connect a writer process with the reader process through the Access Module. This is a limitation of the Teradata Parallel Transporter infrastructure, because the *tbuild* process does not accept data through its standard input.
- A Teradata Parallel Transporter job must be restarted from its beginning.

Named Pipes Access Module Log File

The Named Pipes Access Module log file is an ordinary text file that the access module creates to save operational status information. You can access the log file using standard UNIX and Windows text-editing utilities.

Name and Location

By default, the Named Pipes Access Module log file is named *namedpipes.log*, which is located in */tmp* on UNIX systems and in *%TEMP%* then *C:* on Windows 2000/XP/2003 systems.

You can specify a different location for the log file using the log_directory parameter in the Named Pipes Access Module initialization string. See “Initialization String” on page 75.

Format

Each entry in the Named Pipes Access Module log file is a numbered five-field record formatted as follows:

```
time-stamp process level message_number text
```

where:

Field Name	Description.
time-stamp	Time-stamp locale-specific date and time stamp information, such as: `Thur Jan 20 16:33:57 2000`
process	Decimal process ID of the logged task.
level	Indication of the logging level, either: CRITICAL INFO ERROR DEBUG WARNING TRACE
message-number	Decimal message number associated with the log message.
text	Log message text string.

See the table in the next section for a description of the log file messages.

Messages

The following table describes each message, listed by message number, that can be written in the Teradata Named Pipes Access Module log file.

Many of the message descriptions cite access module functions generated by the Data Connector API in response to your client utility commands. For these messages, refer to *Teradata Tools and Utilities Access Module Programmer Guide* for information about the Data Connector API Access Module functions.

Table 5: Teradata Named Pipes Access Module Error Messages

Number	Log Level	Message Text	Description/Response
6	2/Error	`The Named Pipes Access Module currently does not support writing.`	The client utility attempted either a File Open for write or a File Write operation. Revise the utility job script to avoid using the access module for those operations.
8	2/Error	`The Access Module still has open files.`	A File Open operation was attempted on a file that was already open.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
12	2/Error	`Error encountered during memory mgmt.`	The access module did not allocate memory for an internal buffer, structure or string. On UNIX systems, verify that the *ulimit* for process data is not too low.
13	2/Error	`Invalid open mode: value.`	An undefined open mode of decimal value *value* was used with a File Open operation. Revise the Client utility job script to not use this access module for that operation.
15	2/Error	`Invalid Block_size value: value`	The *value* value is not a valid block size. Revise the access module initialization string in the utility job script to specify a valid block_size value.
16	2/Error or 3/Warning	`Data validation failed during synchronization of the write side. The data read from the pipe did not match the previous data read during data recovery. This could be due to the database changing between transfer attempts.`	Logged as a level-2 Error message when the Client utility attempts to recover an earlier session, the signature_check level is 2, and the restarted write process data stream fails the signature check. To guarantee a consistent transfer, unlock the destination table and start the operation from the beginning. If the signature_check level is 1, then the message is logged as a level-3 Warning, but no error condition is returned to the Client utility.
23	2/Error	`Invalid positioning data length was passed.`	The Client utility passed position data of an invalid length. This should not occur.
24	2/Error	`Named Pipes Access Module has received an unsupported request "mnemonic"(code code) from the calling software, "process name". It is possible that the calling software version is incompatible with this access module.`	The Client utility attempted an operation that is not supported by the access module. If it is available, the message text includes the standard command mnemonic (*mnemonic)*, along with the decimal command code (*code*). Possibly the Client utility is not able to use the access module.
32	2/Error	`The Position value did not match the available values. Only the last one returned should be used.`	A File Set Position operation used position information that did not match one of the last two values returned by the access module. Unlock the destination table and start the operation from the beginning.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
33	2/Error	`An unrecognized attribute name was specified.`	The Data Connector API issued one or more of the following functions that are not implemented by the Named Pipes Access Module: • pmiPIDMOptPutA_A • pmiPIDMOptGetA_A • pmiPIDMOptPutF_A • pmiPIDMOptGetF_A **Note:** This message occurs frequently when using the FastLoad, TPump, and MultiLoad utilities.
34	1/Critical or 2/Error	`An unexpected critical exception occurred in the description.`	General failure message.
37	2/Error	`Error encountered: description`	On UNIX systems a library detected and reported an exception described in *description*.
38	2/Error	`The Named Pipes Access Module was entered on a different thread. This is not supported.`	On Windows systems a thread entered the access module with a thread ID that was different from that of the thread that originally the module. The Named Pipes Access Module does not support multiple threads.
39	2/Error	`Cannot get module filename because: explanation`	On Windows systems a WIN32 API GetModuleFileName() call failed.
40	2/Error	`Cannot get size of module version information because: explanation`	On Windows systems a WIN32 API GetFileVersionInfoSize() call failed.
41	2/Error	`Cannot get module version information because: explanation`	On Windows systems a WIN32 API GetFileVersionInfo() call failed.
42	2/Error	`Cannot get version information from version information resource.`	On Windows systems a WIN32 API VerQueryValue() call failed.
43	2/Error	`Cannot allocate n bytes of memory because: explanation`	On Windows systems a memory allocation for n bytes of data failed.
44	2/Error	`No pipe name or path has been provided to Named Pipes Access Module.`	The File Open function passed a zero-length string as the pipe name.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
45	2/Error	`Could not open the requested Pipe, "pipename", because: explanation`	The Named Pipes Access Module could not create or open the pipe or device named *pipename* for the reason described in *explanation*. Check the pipe name for correct spelling, permissions, or existence, and update the Client utility job script if necessary.
49	2/Error	`Could not format error message because: number`	On Windows systems a call to the WIN32 API FormatMessage() failed. The reason is given in the decimal error number *number*, from the SDK file WINERROR.H.
51	2/Error	`Write to the fallback file failed. The file is deleted because recover data is now corrupted, but the transfer will continue.`	This message indicates that a subsequent recovery operation may not be possible. It does not, by itself, indicate that the transfer failed. Verify that sufficient space is available in the fallback data directory.
52	2/Error	`The fallback file is missing so data is not available to be recovered.`	A File Set Position occurred but no fallback data file was available. The restart operation failed. Verify that sufficient free space is available in the fallback directory.
53	2/Error	`Error received during recovery while reading the Fallback file: filename.`	The recovery operation failed. Determine why the file named filename could not be read.
54	2/Error	`The initialization string "keyword" has no value.`	Assign a value to the given Named Pipes Access Module initialization string keyword in the Client utility job script.
58	2/Error	`Invalid Signature_Check value: value`	The signature_check parameter has been assigned a value other than 0, 1, or 2 in the initialization string. The invalid value is shown as *value*.
61	2/Error	`Cannot read from file because: explanation`	An I/O error occurred while reading from the pipe or file.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
72	1-6/(any)	`Log Initialize: ProcessName=processname, UtilityID=id, FileName=filename, LogLevel=level`	This informational message occurs when a new process opens the log file or the log level changes. The operating system process name as *processname*. This process name is associated with the process ID number for this entry as well as all subsequent entries with the same ID. The utility ID appears as *id* unless the NPLOGDIR environment variable is defined. The name of the log itself appears as *filename* and the new log level is the decimal value *level*.
73	2/Error	`The module rejected an attempt to open a second device because the initialization string specifies a fallback file (vs. a directory).`	Update the initialization string in the Client utility job script to use a fallback directory instead of a fallback file.
75	2/Error	`Invalid Fallback file: filename`	During writer-process restart a File Set Position occurred, the signature check failed, and the initialization string specified signature check level 2. This implies that the data sent through the pipe has changed from the earlier attempt. In this case, delete the destination table and start the transfer from the beginning. Alternately: • A File Open occurred, an existing fallback file was specified in the initialization string, but the file did not have the standard format or referred to a different pipe name. Retry the transfer, specifying a different fallback file. • A File Set Position occurred, resulting in a failed seek within the fallback data file. This implies that the file is corrupt and should be deleted.
77	2/Error	`A memory allocation failed.`	On UNIX systems a memory allocation failure occurred. Check the process *ulimit* value.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
78	1-6/(any)	`Previous entry occurs n times.`	This informational message indicates that the preceding message occurred *n* times, even though only one instance appears in the log file. (This message does not count as an occurrence.)
79	2/Error	`Could not open the fallback file "filename", because: explanation`	Because the fallback data file could not be accessed, the checkpoint and restart functions are effectively disabled. This does not indicate, however, that the transfer failed. Verify that the initialization string fallback_directory and fallback_file parameters are valid.
80	4/ Information	`Successfully created the fallback file "filename".`	This informational message documents the creation or re-creation of a specified or automatically generated fallback data file. This occurs on File Open or on File Get Position if a previous write error occurred.
81	2/Error	`Invalid Log Level value: level`	The initialization string specifies a log_level parameter with a value other than 0 through 6. The erroneous value appears in *level*.
82	2/Error	`Undefined initialization string keyword: keyword`	The Named Pipes Access Module initialization string uses an invalid keyword. Replace the invalid keyword parameter specification, *keyword*, in the initialization string.
83	2/Error or 5/Debug	`Fallback file header does not contain the correct magic number!`	If the initialization string assigns a fallback file and the fallback file is found with a bad magic number, this message is logged at level 2 and an error is returned in response to the File Open command. **Note:** The magic number is the string "Named Pipes Data Access Module Fallback File *vv.vv.vv**n*" in the fallback data file header that identifies the version number of the file format as *vv.vv.vv**n*. If a fallback directory is specified, this message is logged at level 5 and the file is ignored.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
84	2/Error or 5/Debug	`Fallback file header does not contain the correct pipe name!`	If the initialization string assigns a fallback file and the fallback file is found with a pipe name that does not match that passed during File Open, this message is logged at level 2 and an error is returned in response to the File Open command. If a fallback directory is specified, this message is logged at level 5 and the file is ignored.
85	2/Error or 5/Debug	`Fallback file header does not contain the correct block size parameter.`	If the initialization string assigns a fallback file and the fallback file is found with a block size that does not match that specified in the initialization string, this message is logged at level 2 and an error is returned in response to the File Open command. If a fallback directory is specified, this message is logged at level 5 and the file is ignored.
86	4/ Information	`Deleting fallback file "filename" of size n.n megabytes after successful transfer.`	This informational message provides file size information that you can use in capacity planning for the fallback data file.
87	2/Error	`The initialization string keyword "keyword" specifies the nonexistent directory "directory_name"!`	The Named Pipes Access Module initialization string includes a log_directory or a fallback_directory parameter (*keyword*) that specifies a nonexistent directory (*directory_name*). Either create the specified directory or modify the initialization string to specify a valid directory.
88	4/ Information	`Recovered fallback file "filename" for synchronizing the pipe writer.`	This informational message indicates that the Named Pipes Access Module: • Has located and validated the fallback data file (*filename*) in response to a File Open command • Is awaiting a File Set Position command to initiate a recovery operation
89	2/Error	`The module rejected an attempt to open the directory "directory_name" for input. Please specify a named pipe or a file instead.`	This message occurs on File Open if the Client utility job script erroneously specifies a directory name instead of a pipe, a device, or a file name.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
90	2/Error	`The module failed to create the directory "directory_name".`	On UNIX systems this message occurs during File Open if the access module cannot create an intermediate directory in the path of the specified pipe.
91	4/ Information	`The module created the named pipe "pipename".`	On UNIX systems this message indicates that the access module successfully created the specified named pipe during File Open. **Note:** The pipe remains after the access module has closed it.
92	2/Error	`The module failed to create the named pipe "pipename". Explanation.`	On UNIX systems this message indicates that the access module could not create the named pipe *pipename*. Check for spelling errors in the Client utility job script, as well as permissions of the directories in the path.
93	4/ Information	`The Data Connector issued a Read when a Set Position was expected. Abandoning recovery and reusing fallback file "filename".`	This informational message indicates that the access module had validated a fallback data file as a candidate for recovery, but is recycling the file because the Client utility did not initiate a recovery operation.
94	4/ Information	`The Data Connector issued a Get Position when a Set Position was expected. Abandoning recovery and reusing fallback file "filename".`	This informational message indicates that the access module had validated a fallback data file as a candidate for recovery, but is recycling the file because the Client utility did not initiate a recovery operation.
95	4/ Information	`The pipe reader has successfully re-synchronized with the pipe writer.`	This informational message occurs on File Set Position after the module has successfully restarted the write process system.
96	2/Error	`The initialization string keyword "keyword" specifies the unwritable directory "dir"!`	This message occurs if the initialization string contains a log_directory or a fallback_directory keyword that refers to a read-only directory. Either change the properties of the directory to be writable or modify the initialization string to refer to a writable directory.

Table 5: Teradata Named Pipes Access Module Error Messages (continued)

Number	Log Level	Message Text	Description/Response
97	2/Error	`The environment variable NPLOGDIR specifies the directory "dir", which is inaccessible or unwritable!`	This message occurs if the environment variable NPLOGDIR refers either to a nonexistent or to a write-protected directory. Either create the directory, add write permission, or modify the initialization string to refer to a valid directory.

WIN32 Named Pipes API

When creating a Win32 client application that writes to a named pipe, it is necessary to know how the named pipes server creates the named pipe. This is important to be able to code the corresponding options on the client application's CreateFile() system call.

The Win32 Named Pipes Access Module creates the named pipe using the following system call:

```
CreateNamedPipe(PipeName, // Pipe name
                    PIPE_ACCESS_INBOUND, // Open mode
                    PIPE_TYPE_BYTE,      // pipe mode
                    4,            // Number of instances
                    BuffSize,     // output buffer size
                    BuffSize,    // input buffer size
                    100,// default timeout in milliseconds
                    NULL); // pointer to security attributes
```

where:

PipeName is any string of the form \\.*pipe**pipename* and *pipename* is the pipe name agreed upon between the client and server applications and *BuffSize* is a value twice the number of bytes specified in the initialization string "Block_size" parameter.

A user-developed client application must code the GENERIC_WRITE (but not the GENERIC_READ) option on its CreateFile() system call because the access module allows data to flow only from client to server.

Example

```
HANDLE hPipe =
CreateFile(PipeName,// pipe name
  GENERIC_WRITE,// write access
  0,              // no sharing
  NULL,           // no security attributes
  OPEN_EXISTING,  // opens existing pipe
  0,              // default attributes
  NULL);          // no template file
```

Alternately, the client utility can use standard `C' stream I/O, opening the pipe with the following:

```
FILE *fp = fopen(PipeName, "wb");
```

As a named pipes server, the access module both connects to the pipe and reads from the pipe using synchronous, blocking I/O. The client is free to write to the pipe either synchronously or asynchronously.

The access module does not impose a message structure on the data flowing through the pipe. Thus, the client application may write to the named pipe either as a stream of bytes or as a stream of messages since the access module always reads from the pipe as a stream of bytes.

For details on programming Win32 named pipes, see the article "Named Pipes" under Platform SDK in the *Microsoft Developer's Network On-Line Library*.

Initialization String

The initialization string for the Named Pipes Access Module is in the form keyword=value where:

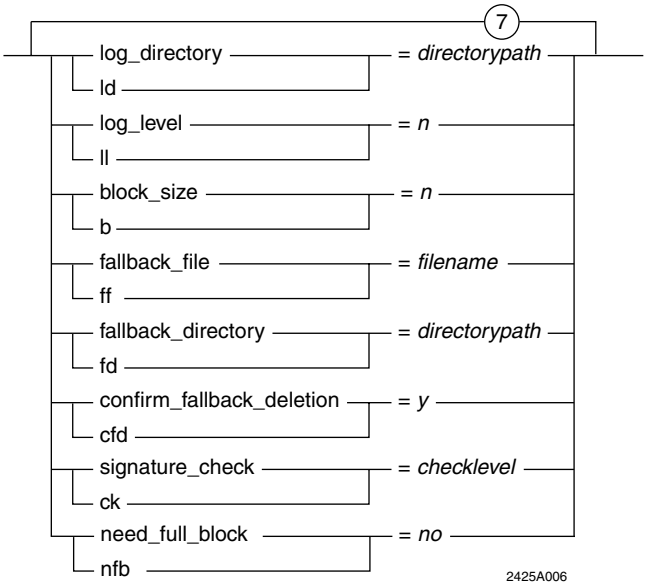
- *keyword* identifies the initialization parameter
- *value* is either an integer or a string, sometimes enclosed in single- or double-quote characters

Function

Use the initialization string in your Teradata Client utility job script to specify or override the default settings of the following parameters:

- Log Directory
- Log Level
- Block Size
- Fallback Data File Name
- Fallback Data Directory Path
- Fallback Data File Deletion
- Signature Checking

Syntax



Note: Because the Named Pipes Access Module applies each parameter as it parses the initialization string, you can ensure the earliest possible acquisition of log information by doing one of the following:

- Specify the log_directory parameter first
- Use the NPLOGDIR environment variable to specify the log directory

where:

Table 6: Initialization Syntax

Syntax Element	Description
block_size=*n*	Block size, in bytes, of the data transfer operation where *n* is an integer from 1 to 2147483647. The default, if you do not specify the block_size parameter, is 65536 bytes. Due to the limitations in the Data Connector, the block size must be greater than or equal to the size of the largest record passing through the pipe. When the module is used in a trickle feed environment, as with TPump, the shortest latency occurs when the block size is set to the maximum record size.
confirm_fallback_deletion=y	Specification to make the Named Pipes Access Module present a confirmation prompt before deleting the fallback data file.
fallback_directory=*directorypath*	Path of the fallback data file directory. The default directory, if you do not specify the fallback_directory parameter, is: • */opt/np_axsmod* on UNIX systems • *%TEMP%* or *%WINDIR%\temp* on Windows systems
fallback_file=*filename*	Name of the fallback data file.

Table 6: Initialization Syntax (continued)

Syntax Element	Description
log_directory=*directorypath*	Path of the log file directory. If you do not specify the log_directory parameter, the default is: • */tmp* on UNIX systems • *%TEMP%* or *C:* on Windows systems
log_level=*n*	Specification that sets the level of detail to be posted to the log file, where: • 0 = Disabled—No logging. This is the default log_level value if you do not specify a log_directory path. • 1 = Critical—Logs events where a critical resource, such as memory or its message strings, cannot be obtained. • 2 = Error—Logs error conditions. This is the default if you specify a log_directory path but do not specify a log_level value. • 3 = Warning—Logs unusual events that do not halt processing. • 4 = Information—Logs operational events or statistics. • 5 = Debug—Logs details normally required for debugging. • 6 = Trace—Logs all available information about each I/O operation.
need_full_block=no	Explicitly enables buffer flushing within client scripts when this parameter is set to "no." This parameter must be set to "no" when TPump is using latency option with IMPORT. Buffer flushing needs to perform check-point restarting somewhat differently then when buffer flushing is *not* enabled (the default), the fallback recovery data generated is different, and is *not* interchangeable between flushing and non-flushing modes.
signature_check=*checklevel*	Specification that sets the level of detail to be posted to the log file, where: 0 = Disabled—A signature is neither calculated nor checked. 1 = Enabled/No Return— Calculates and checks a signature and logs an error if the check fails, but does not return an error condition. 2 = Enabled/Return— Calculates and checks a signature and logs and returns an error condition if the check fails.

Specifying Directory Name on Windows

On Windows, if you specify a directory name for the initialization string that has spaces, you must enclose that name within single quotes, for example: `'c:\documents and settings'`

CHAPTER 4

Teradata WebSphere MQ Access Module

Teradata WebSphere® MQ Access Module allows Teradata utilities to import data using IBM® WebSphere MQ message queuing middleware. Topics include:

Supported Operating Systems

Teradata WebSphere MQ Access Module operates on the following systems:

- HP-UX
- IBM AIX
- IBM MVS/ESA
- Linux, Linux Opteron
- Solaris SPARC, Solaris 10 Opteron
- Windows 2000, XP, and Server 2003

For the most up-to-date information about supported releases, see Supported Releases in the Preface.

Installation

Teradata WebSphere MQ Access Module and Teradata Parallel Transporter (Teradata PT) version of the WebSphere MQ Access Module are included in the Teradata software on the following operating systems:

- MVS/ESA on magnetic cartridge
- AIX, HP-UX, Linux, Solaris SPARC, and Windows on the Teradata Tools and Utilities Load/Unload CD

For information about installing the access module, refer to the installation guide appropriate for your operating system. For information about installing WebSphere MQ, refer to IBM's documentation.

Access Module Name

The version of the access module available on your system depends on the operating system and whether the WebSphere MQ Server or the WebSphere MQ Client was installed. Each supported operating system also has a Teradata PT Server and Client version of the access module.

To use Teradata WebSphere MQ Access Module with a Teradata utility you must reference the access module names in your job script. The following table lists the Teradata WebSphere MQ Access Module name to use as the AXSMOD *name* specification in your Teradata job script.

Table 7: AXSMOD Name Specification for Teradata WebSphere MQ Access Module

Operating System	WebSphere MQ Version	Access Module Name	Mode
AIX	WebSphere MQ Client	*libmqsc.so*	32-bit and 64-bit
	WebSphere MQ Server	*libmqs.so*	32-bit
HP-UX	WebSphere MQ Client	*libmqsc.sl*	32-bit and 64-bit
	WebSphere MQ Server	*libmqs.sl*	32-bit and 64-bit
Linux	WebSphere MQ Client	*libmqsc.so*	32-bit
Linux Opteron	WebSphere MQ Client	*libmqsc.so*	32-bit
MVS/ESA	WebSphere MQ Server	*LIBMQS*	32-bit
	WebSphere MQ Server-Teradata PT	*LIBMQST*	
Solaris SPARC, Solaris Opteron	WebSphere MQ Client	*libmqsc.so*	32-bit and 64-bit
	WebSphere MQ Server	*libmqs.so*	32-bit
Solaris Opteron 10	WebSphere MQ Client	*libmqsc.so*	32-bit
	WebSphere MQ Server	*libmqs.so*	32-bit
Windows 2000/XP/2003	WebSphere MQ Client	*libmqsc.dll*	32-bit
	WebSphere MQ Server	*libmqs.dll*	32-bit

Features

A basic knowledge of WebSphere MQ and its APIs is required before attempting to configure and use Teradata WebSphere MQ Access Module. However, no direct access to APIs exists in the WebSphere MQ Access Module.

Teradata WebSphere MQ Access Module allows Teradata utilities to import data using IBM WebSphere MQ message queuing middleware. Teradata WebSphere MQ Access Module transfers data between the WebSphere MQ client or server and the Data Connector. The Data Connector is the interface between Teradata WebSphere MQ Access Module and Teradata utilities.

WebSphere MQ provides connectivity between different types of applications that might be written in different languages or on different operating systems or networks. Platform-specific versions of Teradata WebSphere MQ Access Module provide the appropriate connectivity between different operating systems. This connectivity capability eliminates the need for the application developer to write code that provides this connectivity. Eliminating the need to write that code results in the following benefits:

- Faster application development time
- Transactional integrity (which means that messages are assured delivery)
- Asynchronous delivery of messages

Teradata WebSphere MQ Access Module supplements the reliability and convenience of message queues with checkpoint and restart capability by caching data in a fallback data file. When used with utilities such as TPump to transfer data, the access module allows quick recovery from the following:

- Restarts in the destination database
- Crashes on the system running the import utility
- Crashes on the data source, subject to the limitations of WebSphere MQ technology

Supported Utilities are denoted in

Table 8: Teradata Utilities Supported by the WebSphere MQ Access Module

Operating System		BTEQ	FastLoad	FastExport	MultiLoad	TPump	Teradata PT
Linux	32-bit	Yes	Yes	No	Yes	Yes	Yes
UNIX	HP-UX 32-bit and 64-bit	Yes	Yes	No	Yes	Yes	Yes
	IBM AIX 32-bit and 64-bit	Yes	Yes	No	Yes	Yes	Yes
	IBM MVS 32-bit	Yes	Yes	No	Yes	Yes	Yes

Table 8: Teradata Utilities Supported by the WebSphere MQ Access Module (continued)

Operating System		BTEQ	FastLoad	FastExport	MultiLoad	TPump	Teradata PT
SUN Solaris	SPARC 32-bit and 64-bit	Yes	Yes	No	Yes	Yes	Yes
	Opteron 32-bit	Yes	Yes	No	Yes	Yes	Yes
Windows XP/2000/2003	32-bit	Yes	Yes	No	Yes	Yes	Yes

Standard Output Files

Teradata WebSphere MQ Access Module

Teradata WebSphere MQ Access Module directs standard output (stdout) to the terminal or screen.

Teradata PT Version of Teradata WebSphere MQ Access Module

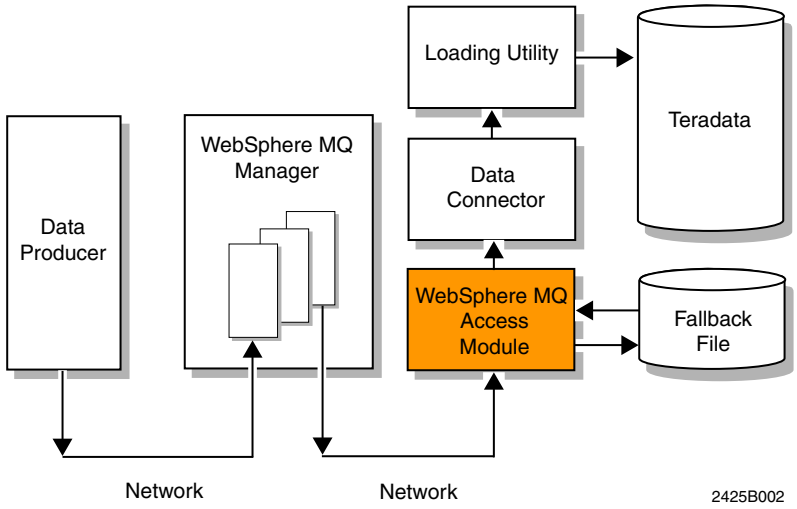
On all operating systems except for MVS/ESA, the Teradata PT version of the WebSphere MQ Access Module directs standard output to *WAMstdout.txt* in the current directory. You can find results of the help command in this file.

On MVS/ESA, the access module directs the output to SYSPRINT.

Data Flow

Figure 6 shows how Teradata WebSphere MQ Access Module imports data to a Teradata Database through a client load utility, such as BTEQ, FastLoad, MultiLoad, or TPump. Although the figure shows the data producer, the queue manager, and the load utility on separate systems, these entities can be on the same system. If you are using the Teradata PT version of the WebSphere MQ Access Module, the term *DataConnector* in this figure and throughout this chapter refers to the Teradata PT DataConnector operator.

Figure 6: Importing Data through WebSphere MQ with Load and Unload Utilities



The flow of data between a data producer, such as an eBusiness application, and a table in Teradata begins as the data producer does the following:

1. Establishes a network connection with a WebSphere MQ queue manager using standard WebSphere MQ interfaces
2. Composes database records into messages (the load utility defines the database records' format)
3. Sends the messages to a message queue under control of the queue manager

At this point, the queue manager stores the incoming messages until the load utility reads and removes them through the following sequence of events:

1. The WebSphere MQ Access Module establishes a connection to the queue manager.
2. The WebSphere MQ Access Module reads messages from the message queue when directed to do so by the Data Connector.
3. The WebSphere MQ Access Module delivers the data to the Data Connector.
4. The Data Connector delivers the data to the load utility.
5. The load utility loads the data into a table within Teradata.

Note: If the load utility is on the same system as the queue manager, the queue manager can be configured to trigger the load utility.

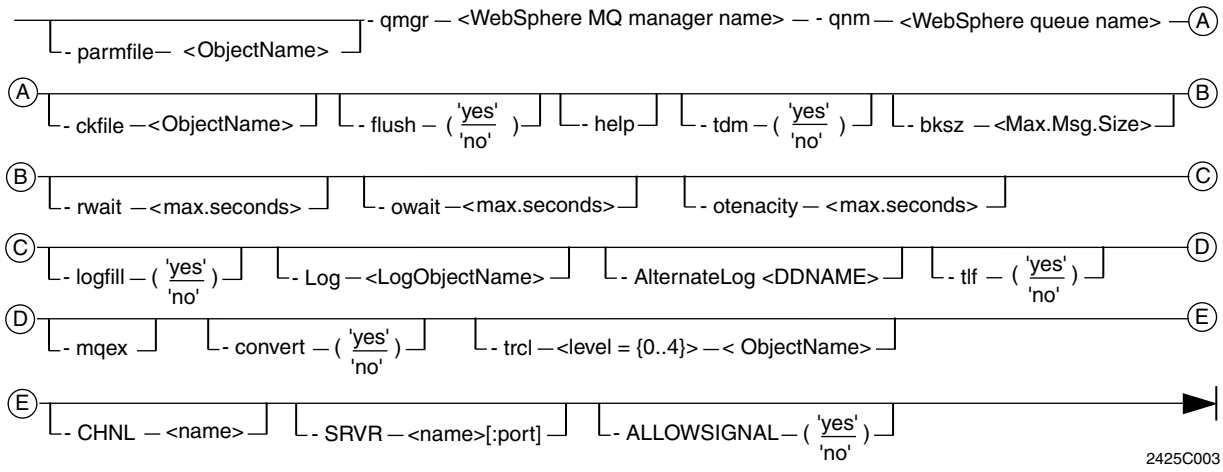
Initialization String

The initialization string for the WebSphere MQ Access Module consists of keywords and corresponding parameters. You can specify all keywords within the initialization string. If desired, you can specify additional keywords in a file. For information on this approach, see the description below for the *parmfile* keyword.

Note: You must enter a hyphen before each keyword.

See"Access Module Calls" on page 19 for information about the specific AXSMOD command or command option syntax for each Teradata utility.

Syntax



Note: When defining parameter attribute in an initialization string or in a parameter file (-parmfile), keywords are case-insensitive, and unrecognized keywords with invalid values cause an immediate termination.

where:

Table 9: Initialization Syntax

Syntax Element	Description
parmfile *<ObjectName>*	Optional object name for the file in which more access module parameters may reside. On MVS, this is a DDNAME that is defined in the JCL. • Each line in the parameter file may have a single keyword (parameter) and its value(s). • Empty lines will be ignored. • Lines beginning with the pound # character will be ignored. • Keywords must begin in the first column. See "Example: JCL Excerpt" on page 91.
help	Optional keyword that requests the display of a list of valid parameter file keywords.
QMGR *<WebSphere MQ Manager Name>*	Required parameter that specifies the WebSphere MQ Manager, which coordinates and routes messages to the appropriate queue. If this parameter is not provided, the operator terminates with a fatal error.
qnm*<WebSphere Qname>*	Required keyword that specifies the queue name.

Table 9: Initialization Syntax (continued)

Syntax Element	Description

CKFILE <*ObjectName/ Statement*>	Optional parameter that allows a request for checkpoint support through the use of disk facilities.
	On MVS/ESA, the <ObjectName> to which you want to write checkpoint data is a DDNAME. For example:
	`CKFILE <DDNAME>`
	where <*DDNAME*> refers to a previously defined DDNAME within the JCL.
	On MVS/USS, <*Statement*> defines the DD statement. For example:
	`ckfile dsorg=ps,dsn=tpt.x2cuss,disp=new,ndisp=catlg, cdisp=catlg,track,primary=100,secondary=20,recfm=u, blksize=32760,lrecl=0`
	For restarts, set the disp value to *shr*. If the disposition value is equal to *new*, and the data set already exists, the allocation and job will fail. Refer to IBM MVS documentation for explanations of other parameters.
	Enough disk storage to hold all checkpointed messages, plus space to copy the file, is required.
	If this parameter is not provided, no file-based checkpoint support is available.
	Note: On workstation platforms, to facilitate a checkpoint during a recovery, a temporary file is created and resides in the same directory as the checkpoint file. Named *MQAMtmp.*<*pid*>, this file supports the process of removing obsolete records from the checkpoint file. The file is removed upon completion of the checkpoint.

FLUSH <*Yes/No*>	Optional parameter that ensures all checkpoint data is written to the media. If you select *Yes*, the access module writes all checkpoint data to the media. You must have file-based checkpoint support in effect to use this option. If you select *No*, checkpoint data is not written to the media. Significantly fewer system resources are consumed if you choose this option. If this parameter is not provided, the default is *No*.
TDM <*Yes/No*>	Optional parameter that allows termination of duplicate messages. If you select *Yes*, the access module returns an EOF if a duplicate message is suspected. If you select *No*, the access module returns the duplicate message. Duplicate messages only occur during a recovery from a crash or ABEND that occurred between the time that a message was written to the checkpoint file and the time that the message was committed through a MQCMIT command. If this parameter is not provided, the default is *Yes*.

Table 9: Initialization Syntax (continued)

Syntax Element	Description
BKSZ <*MaximumMsgSize*>	Optional parameter that specifies the block size for WebSphere MQ messages. You should define the block large enough to accommodate the largest message anticipated. The parameter BLKZ cannot be a value that exceeds the value of the DCB BLKSIZE specified in the JCL within the checkpoint DD statement. Acceptable Range: 1 byte - 4 MB **Note:** On AIX platforms only, the upper limit on blocksize is 4MB rather than 1MB. If this parameter is not provided, the default is 32,000 bytes. If an unacceptable value is provided, the following error is issued, and the access module fails: `Teradata WebSphere MQ AMOD/OpenCkPtFile(!ERROR!): Checkpoint DD DCB BLKSIZE (<DCB BLKSIZE value>) less than potential required (<BLKZ + 4>) (a function of keyword 'BKSZ')`
RWAIT <*MaximumSeconds*>	Optional parameter that specifies the MQGET wait time, which is the time that MQGET waits if no message is immediately available. MQGET is called iteratively in a loop with maximum iterations set to the read wait interval value. To ensure that signals handlers are called, a sleep(0) call will be within the body of the loop. Acceptable Range: Enter 1 - 600 seconds or -1 for unlimited wait time. If this parameter is not provided, the default is 1 second. A zero length message encountered on the queue will trigger an EOF to the application. This may be useful when using the unlimited wait feature (RWAIT -1).
ALLOWSIGNALS <*Yes/No*>	Optional parameter that determines whether signals handlers are called. The default is *Yes,* which calls signal handlers while waiting for message arrive. If *No,* signal handlers are not called.
OWAIT <*MaximumSeconds*>	Optional parameter that specifies the MQOPEN retry time interval, which is the wait time between an unsuccessful MQOPEN call and the next call attempt. Acceptable Range: 1 - 600 seconds See the OTENACITY parameter for related information. If this parameter is not provided, the default is 5 seconds.
OTENACITY <*MaximumSeconds*>	Optional parameter that specifies the amount of time that the access module attempts to open the designated Queue via an MQOPEN call. Acceptable Range: 1 - 6000 seconds If this parameter is not provided, the default is 5 seconds.

Table 9: Initialization Syntax (continued)

Syntax Element	Description
LOGFILL<*Yes/No*>	Optional parameter that packs log records to minimize wasted disk space. If you select *Yes,* the module fills each disk block to the block size specified in the Data Control Block. If you select *No*, the module fills each disk block with a single message. If this parameter is not provided, the default is *Yes*.
LOG <*LogObjectName /LogObjectStatement*>	Optional parameter that requests and directs a Log object. All messages received from MQ are echoed to the specified log. On MVS/USS, <*LogObjectStatement*> defines the DD statement. Example `log dsorg=ps,dsn=tpt.x2luss,disp=new,ndisp=catlg, cdisp=catlg,track,primary=100,secondary=20,recfm=u, blksize=2000,lrecl=0` To add to an existing log, set the *disp* value to `mod`. Setting the *disp* value to *`new`* begins a new log, therefore a data set should not already exist. Refer to IBM MVS documentation for explanations of other parameters. If this parameter is not provided, no log is maintained.
ALTERNATELOG <*DDNAME*>	Optional parameter for MVS version of WebSphere MQ Access Module that opens an alternate log. Specifying an alternate log allows you to close the primary log for external processing. An external notify exit routine sends a signal to the access module to swap between the destination defined by *-LOG <DDNAME>* and that defined by *-AlternateLog <DDNAME>*. Swaps between the logs will occur each time a signal is received. If this parameter is not provided, no alternate log is maintained.
TLF <*Yes/No*>	Optional parameter that specifies whether a log failure should cause a fatal error or not. If you select *Yes*, the access module terminates if it is unable to write to the log for any reason. If you select *No*, logging is disabled while the access module continues to operate. If this parameter is not provided, the default is *Yes*.
MQEX	Optional parameter that allows the process to open the MQ named queue exclusively. If this parameter is not provided, it will not be possible for another process to open this queue.
CONVERT <*Yes/No*>	Optional parameter that requests the WebSphere MQ message character set. If you select *Yes,* the access module converts data to the resident character set. If you select *No,* the access module processes messages without converting them. If this parameter is not provided, the default is *No.*

Table 9: Initialization Syntax (continued)

Syntax Element	Description
TRCL<*level*> <*ObjectName*>	Optional parameter for diagnostics, where levels 0 through 4 indicate: 0: no diagnostic trace 1: job events only reported 2: I/O events 3: I/O buffers 4: detailed information The default is zero (0), indicating no trace. Indicate a level of 1 or higher for diagnostics or checkpoint support. The filename designation on MVS is a DDNAME. **Note:** The load utility you are using, not the access module nor the DataConnector, creates job logs.
CHNL<*name*>	Establishes the WebSphere MQ channel name on Windows platforms. For UNIX platforms, the channel name is defined in environmental variables (for example, MQSERVER) via the shell and system. The channel name is optional on the following platforms: • AIX • HP-UX • Linux • Linux Opteron • Solaris SPARC • Solaris Opteron • Windows 2000/XP/2003 If this parameter is not specified, the channel name is defined in environmental variables via the shell and system. For more information about the proper syntax and use of the environmental variable, go to the WebSphere documentation at http://ibm.com/webspheremq.

Table 9: Initialization Syntax (continued)

Syntax Element	Description
SRVR<*name*>[:*port*]	Establishes the WebSphere MQ server name on Windows platforms. For UNIX platforms, the server name is defined in environmental variables (for example, MQSERVER) via the shell and system. The server name is optional on the following platforms: • AIX • HP-UX • Linux • Linux Opteron • Solaris SPARC • Solaris Opteron • Windows 2000/XP/2003 If this parameter is not specified, the server name is defined in environmental variables via the shell and system. For more information about the proper syntax and use of the environmental variable, go to the WebSphere documentation at http://ibm.com/webspheremq.

Checkpoint Processing

If checkpointing is enabled, the access module reads messages from the queue, then copies the data into a fallback file before delivering the data to the Data Connector. If the load utility decides to restart at an earlier point in the data stream, it directs the Data Connector to issue the File Set Position command to the access module. The module then supplies data from the fallback data file until the restart is complete.

After a position request is made to the access module, additional processing by the client (and probably the database) occurs. A checkpoint request is considered successful only after a subsequent request from the client is received by the access modules. A checkpoint followed by a function request for a reposition is considered successful by virtue of the contents of the position information provided at the reposition. Support for a previous checkpoint is not removed until the most recent checkpoint is successful. This approach allows recovery from a checkpoint failure. On workstation platforms, a temporary file named MQAMtmp.<pid> may be created and resides in the same directory as that of the checkpoint file (specified by the CKFILE keyword).

When a reposition request immediately follows an open request, a restart is conducted. In this case, one or more messages are retrieved from the checkpoint file before the messages are sent back to WebSphere MQ.

Caution: Attempting restarts with multi-volume checkpoint files (checkpoint file allocated on multiple disk volumes) can result in the loss or corruption of data. To ensure proper restarts, make sure that the dataset for the DD statement MQCHKPT in the WebSphere MQ job step resides *entirely* on one single disk volume.

Repeatability of Messages

After recovery from a checkpoint failure, it is possible that messages read by one process may not be repeated to the same process in the same order because another process has read the rolled back messages in the queue. All messages sent to the Teradata utility after the most recent checkpoint request are rolled back into the WebSphere MQ queue under the following conditions:

- A reposition following a database recovery is requested
- The client process fails via an ABEND

If you require repeatability, you should establish a reserved queue name convention in which a single reader process, such as TPump, uses the exclusive option to ensure that it is the only reader of that queue. No other process should use that reserved queue name.

If the checkpoint file is populated and the first request made to the access module following the opening of the named queue is a reposition, the first message returned is retrieved from the checkpoint file. Otherwise, there is no effect because only a single record is maintained in the checkpoint file. All other messages are implicitly rolled back in the case of a client ABEND.

MVS JCL Requirements

The job JCL includes the following required DDNAMES.

Table 10: Required DDNAME Parameters

DDNAME	Description
JOBLIB DD (or the STEPLIB DD for the utility using the access module)	Must include: • MQ Access Module - DSN in which the MQ Access Module resides • MQ Libraries - DSN in which the MQ support modules reside
Parameter DDNAME	The DDNAME must match the *parmfile* keyword value. For example, for the initialization string `-parmfile mqparms`, include DDNAME MQPARMS.

The job JCL includes the following optional DDNAMES.

Table 11: Optional DDNAME Parameters

DDNAME	Description
AlternateLog DDNAME	Include AlternateLog DDNAME to swap between the log and an alternate log.
Checkpoint DDNAME	Include DDNAME MQCHKPT for any checkpoint request.

Table 11: Optional DDNAME Parameters (continued)

DDNAME	Description
Journal DDNAME	The DDNAME must match the *jrnl* keyword value. For example, for the keyword `jrnl mqjrnl,` include DDNAME MQJRNL.
Trace DDNAME	Include DDNAME MQTRACE for a trace request.

Example: JCL Excerpt

For the JCL excerpt that follows, the initialization string would be `-parmfile mqparms`.

```
//* Include the following in the STEPLIB (or JOBLIB) for the utility that
//* will be utilizing the WebSphere MQ Access Module (LIBMQS).
//STEPLIB  DD  DISP=SHR,DSN=<DS containing MQ Access Module>
//         DD  DSN=TER2.SASC700C.LINKLIB,DISP=SHR
//         DD  DSN=MQS.V1R2.SCSQAUTH,DISP=SHR
//*
//* E.g., initialization string provided via utility script:
//*    "-help -parmfile mqparms"
//* will cause the following parameters to be accepted.
//MQPARMS  DD  *
# Request file based checkpointing
ckfile mqchkpt
# Set Access Module trace level to "1" and output to DDNAME MQTRCE
TRCL 1 MQTRCE
# REQUIRED parameter, define Q manager
qmgr CSQ1
# REQUIRED parameter, define named Q
qnm TLB1
/*
//SYSPRINT DD  SYSOUT=*
//* If requested via utility script,
//* the PMTRCE DD will direct DataConnector trace.
//PMTRCE   DD  SYSOUT=*
//* WebSphere MQ trace output (DDNAME define in "TRCL" parameter)
//MQTRCE   DD  SYSOUT=*
//JRNL     DD  DISP=SHR,DSN=TPT.JOURNAL
 //* The following DD statement defines the checkpoint dataset.
//MQCHKPT DD DISP=(NEW,CATLG,CATLG),DSN=<checkpointDS>,
//           SPACE=(CYL,(2,1)),VOL=SER=<VolSer>,
//           UNIT=SYSDA,DCB=(RECFM=U,BLKSIZE=32760)
//* This form of the DD statement should be used for recovery after
//* a client failure.
//MQCHKPT DD DISP=(OLD,DELETE,KEEP),DSN=<checkpointDS>
```

CHAPTER 5

Teradata Access Module for JMS

Teradata Access Module for Java Message Service (JMS) is a utility in the Teradata Tools and Utilities product set that offers a fast, asynchronous method to import and export data between Teradata Database and any JMS-enabled messaging system, which are generally referred to as Message Oriented Middleware (MOM) or Enterprise Services Bus (ESB) systems. JMS provides simplified and vendor-independent communication with messaging service providers.

Teradata Access Module for JMS can be invoked by certain Teradata load/unload utilities, specifically Teradata FastLoad, Teradata MultiLoad, Teradata FastExport, Teradata Parallel Transporter (PT) DataConnector Operator, BTEQ, and TPump.

Topics in this chapter include:

Supported Platforms and Teradata Utilities

Teradata Access Module for JMS can be used on a number of operating systems. Table 12 lists supported Teradata load/export utilities according to operating system.

Table 12: Platform and Utility Support for Teradata Access Module for JMS

Operating System		Supported Utilities					
		BTEQ	Teradata FastExport	Teradata FastLoad	Teradata MultiLoad	Teradata TPump	Teradata PT
HP-UX PA-RISC	32-bit and 64-bit	Yes	Yes	Yes	Yes	Yes	Yes

Table 12: Platform and Utility Support for Teradata Access Module for JMS (continued)

Operating System		Supported Utilities					
		BTEQ	Teradata FastExport	Teradata FastLoad	Teradata MultiLoad	Teradata TPump	Teradata PT
IBM AIX Power PC	32-bit and 64-bit	Yes	Yes	Yes	Yes	Yes	Yes
UNIX SUN Solaris on SPARC 8, 9, and 10	32-bit and 64-bit	Yes	Yes	Yes	Yes	Yes	Yes
Windows 2000/2003/XP	32-bit	Yes	Yes	Yes	Yes	Yes	Yes
Linux	32-bit	Yes	Yes	Yes	Yes	Yes	Yes

For the most up-to-date information about supported operating systems, see Supported Releases in the Preface.

Access Module Names

Teradata Access Module for JMS is designed to be invoked by the Data Connector. The Data Connector is a software component linkage between Teradata utilities and an access module. Table 13 lists the access module names.

Table 13: AXSMOD Name Specifications

Operating System		Access Module Name
Linux	RedHat and SUSE	*libjmsam.so*
UNIX	HP-UX	*libjmsam.sl*
	IBM-AIX	*libjmsam.so*
	Solaris SPARC	*libjmsam.so*
Windows 2000/XP/2003		*libjmsam.dll*

Data Flow

Messaging is a way of communicating between software components. A client component sends a message to a destination, and other receiver components can retrieve the message from the specified part of this destination. The sender does not need to know anything about the receiver except for the message format and destination. The sender and receiver

components communicates with a messaging service, called Message Oriented Middleware (MOM), which establishes the rules for communication and data exchange. Teradata Access Module for JMS uses a JMS API to interface with a MOM service provider. The JMS API is a common queuing messaging interface must be installed on the client system. For more about the roles of the sender and receiver, see "Messaging Models" on page 98.

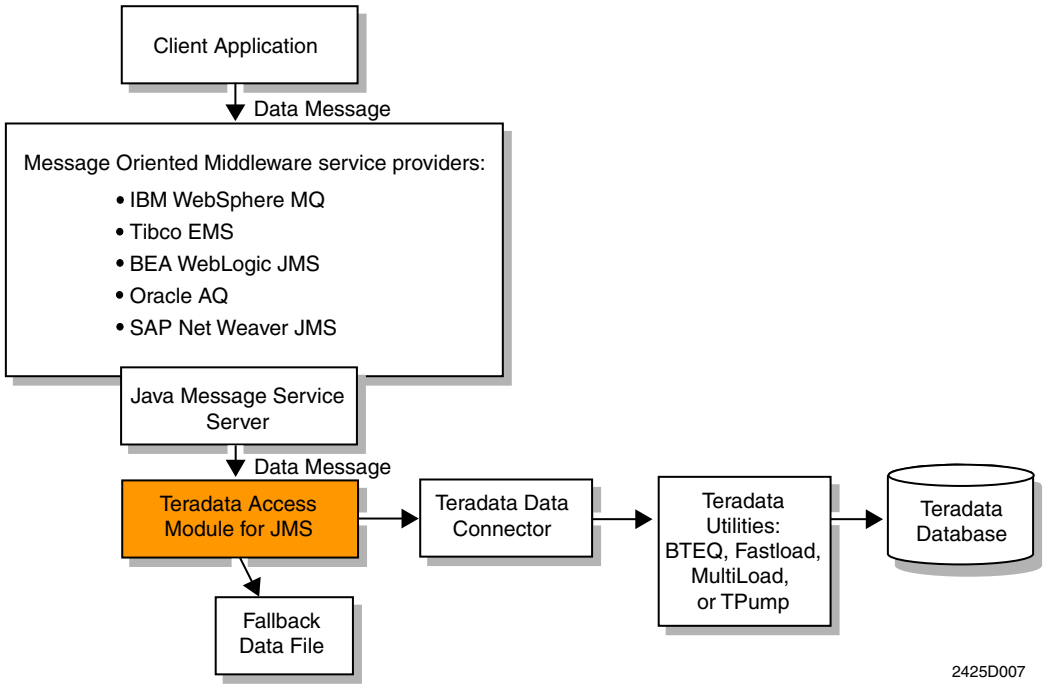
Importing Data

Figure 7 on page 96 is an overview of how data is imported from a JMS client application to Teradata Database. In general, the data flow is as follows:

1. The client component sends a data message to a MOM service provider.
2. Depending on the messaging model, the JMS message server retrieves the data message from a queue or topic within a MOM service provider. Conversely, the JMS message server retrieves a data message from a queue or topic.
3. At the software code level, Teradata Access Module for JMS uses JMS connection calls to access the configured JMS-administered objects to connect to the JMS server.

   Each service provider has its own proprietary implementation of connections to communicate with the JMS server and an administrative tool to configure the connection and queue objects. Teradata Access Module for JMS is expected to use **Java Naming Directory Interface** JNDI namespace to locate those JMS-administered objects.
4. The Data Connector initiates a sequence of instructions so Teradata Access Module for JMS can get the data messages from the queue or topic. This sequence of instructions may include the access module Initialize, File Open, File Read, File Get Position, and Shutdown commands. See Table 14 on page 102 for more information on access module functions.
5. Teradata Access Module for JMS reads the data from the queue or topic. It copies the data to a fallback data file for checkpoint and restart purposes and then delivers it to the Data Connector.
6. The Data Connector transfers the data to a Teradata utility, such as BTEQ, FastLoad, MultiLoad, or TPump.
7. The Teradata load utility processes and loads the data into a table in the Teradata Database.

Figure 7: JMS Importing to Teradata Database

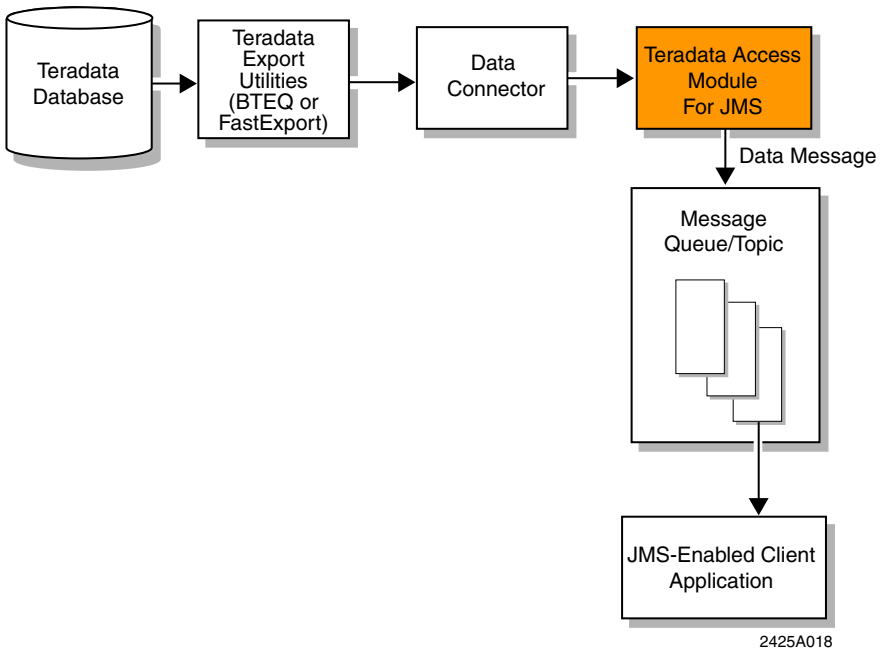


Exporting with Teradata Export Utilities

Figure 8 on page 97 is an overview of how data is exported from Teradata Database to a JMS-enabled client application through a Teradata export utility, namely FastExport or BTEQ. In general, the data flow is as follows:

1. The Teradata export utility retrieves data from Teradata Database and transfers the data to Teradata Data Connector.
2. Data Connector transfers the data to Teradata Access Module for JMS.
3. Teradata Access Module for JMS uses JMS connection calls to access configured JMS administered objects, and connects to the JMS server.
4. Teradata Access Module for JMS composes database records into messages (the export utility defines the database record format), and sends the messages to a JMS server.
5. The JMS server puts the messages on a specified queue or a certain destination, called a topic.
6. The JMS server stores the messages until any other JMS-compliant application reads and removes them.

Figure 8: Exporting with Teradata Export Utilities

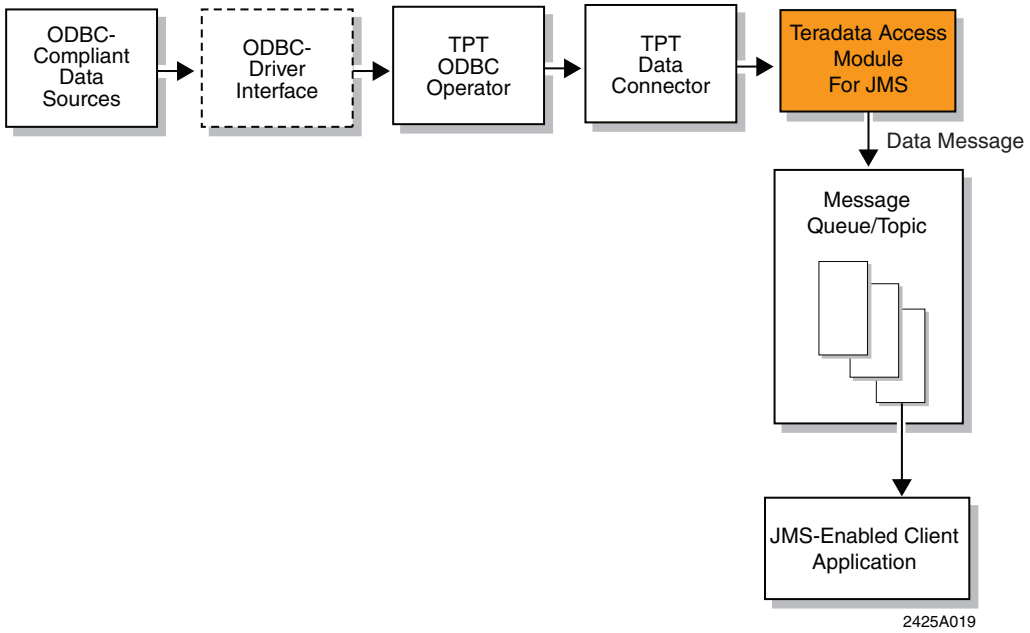


Exporting from an ODBC-Compliant Data Source (Using Teradata PT)

The Teradata Access Module for JMS can also be transparently used with any Teradata Parallel Transporter (PT) producer operator, such as the ODBC Operator, through the Teradata PT DataConnector Operator. Figure 9 on page 98 is an overview of the data flow between any ODBC-compliant data source and a JMS client application using Teradata Access Module for JMS. In general, the process is as follows:

- Teradata PT ODBC Operator connects to an ODBC-compliant data source, for example, Oracle, SQL Server, or DB2.
- The operator reads data close to the sources (from the same machine where Teradata PT is running, as opposed to across a network).
- The operator feeds the data (via data stream) to Teradata PT Data Connector.
- Teradata PT Data Connector transfers the database records to Teradata Access Module for JMS.
- Teradata Access Module for JMS uses the JMS connection calls to access configured JMS administered objects, and connects to the JMS server.
- Teradata Access Module for JMS composes database records into messages (the export utility defines the database record format), and sends the messages to a JMS server.
- The JMS server puts the messages on a specified queue or a certain destination, called a topic.
- The JMS server stores the messages until any other JMS-compliant application reads and removes them.

Figure 9: Exporting from ODBC-Compliant Data Sources



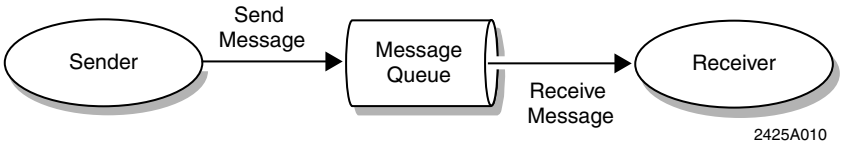
Messaging Models

Teradata Access Module for JMS supports both point-to-point and publisher-subscribe messaging models, both of which use the concepts of *sender* and *receiver*. The following information defines the two messaging models, then describes the roles of sender/receiver in relation to imports and exports with Teradata Access Module for JMS.

Point-to-Point Messaging

A point-to-point messaging model is built on the concept of sender, receiver, and message queue. Data messages are sent to a specific queue, then fetched and processed by a single receiver. This model provides a one-to-one relationship between the sender and receiver. There are no timing dependencies for sending or receiving messages. The sender and the receiver operate independently.

Figure 10: Point-to-Point Messaging Model



Publish-Subscribe Messaging

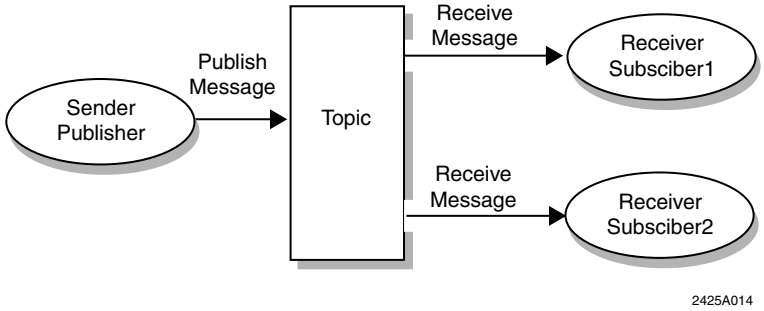
A publish-subscribe messaging model is built around the concepts of topics, publishers, and subscribers. Data messages are sent to a topic by a publisher (sender), then received by one or more subscribers (receivers) of that topic. This model has a one-to-many relationship.

Publishers and subscribers operate independently of each other, and they do not need to know anything about each other.

A subscriber can either be *durable* or *nondurable*:

- A nondurable subscriber only gets messages that are published while it is connected.
- A durable subscriber gets all the messages that are published, including the ones published while it is disconnected.

Figure 11: Publish-Subscribe Messaging Model



Component Roles During Data Loads

- For load jobs, the *sender* is a customer-supplied component responsible for the following:
  - Format a data message that is defined by the Teradata load utility. The data message type can either be a simple text message (Type TextMessage) or a stream of bytes (Type ByteMessage).
  - Place a data message on a specified queue or at certain destination, called a topic.
  - Send a control message (zero-length message) to indicate the end of the message stream.
- For load jobs, the *receiver* is Teradata Access Module for JMS, which is responsible for the following:
  - Perform a JNDI lookup to retrieve JMS-administered objects from the naming space.
  - Establish communication with the JMS provider.
  - Retrieve the data message from the queue or topic.
  - Copy the data message to a fallback file for checkpoint and restart operations.
  - Pass the data message to the Teradata load utility, which then processes and loads the data into a Teradata Database.
  - Continue retrieving data messages from the queue or topic until a control message (zero-length message) is received, or until a time out is received that signals the end of the message stream.

Component Roles During Data Exports

- For export jobs, the *sender* is Teradata Access Module for JMS, which is responsible for the following:

    - Format a data message that is defined by the Teradata export utility. The data message type can either be a simple text message (Type TextMessage) or a stream of bytes (Type ByteMessage).
    - Perform a JNDI lookup to retrieve JMS-administered objects from the naming space.
    - Establish communication with the JMS provider.
    - Place a data message on a specified queue/topic.
    - Send a control message (zero-length message) to indicate the end of the message stream.
- For export jobs, the *receiver* can be Teradata Access Module for JMS or any other JMS-enabled application. The receiver is responsible for the following tasks:
    - Perform a JNDI lookup to retrieve JMS-administered objects from the naming space.
    - Establish communication with the JMS provider.
    - Retrieve the data message from the queue/topic.
    - Copy the data message to a fallback file for checkpoint and restart.
    - Pass the data message to the Teradata load utility, which then processes and loads the data into a JMS-enabled application.
    - Continue fetching data messages from the queue or topic until a control message (zero-length message) is received or a time out that signals the end of the message stream.

Interfaces

Teradata Access Module for JMS can interface with a JMS provider or with the Teradata Data Connector.

Interface with a JMS Provider

Teradata Access Module for JMS uses Java Message Service (JMS) to access a JMS provider. The JMS is a Java API that allows applications to create, send, receive, and read data messages between software components.

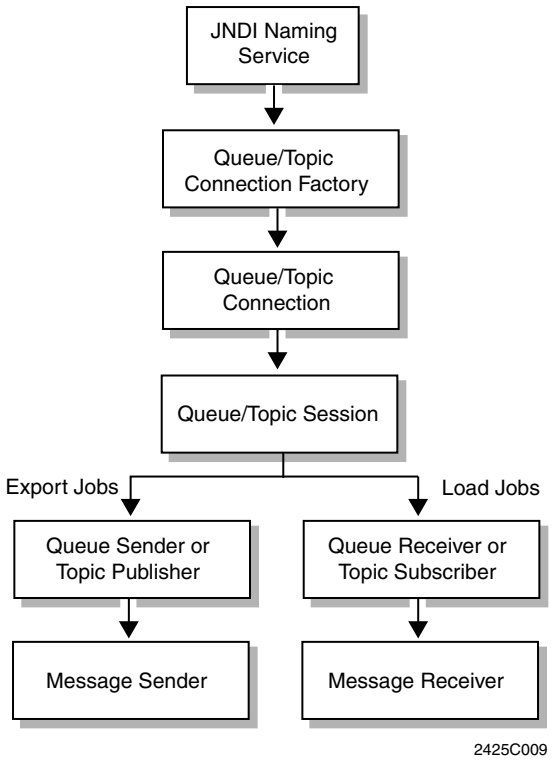
Following are the main parts of Teradata Access Module for JMS that interface with JMS providers:

- **JNDI Naming Service** looks up naming and directory services by setting up initial context.
- **Queue or Topic Connection Factory** creates queue or topic connections with a JMS provider.
- **Queue or Topic Connection** creates queue or topic session objects.
- **Queue or Topic Session** creates queue receiver, topic subscriber, queue sender, or topic publisher objects.
- **Queue Receiver or Topic Subscriber** receives messages that are delivered to a queue or topic during a load job.

- **Queue Sender or Topic Publisher** sends messages from a queue or topic session during an export job.
- **Message Receiver** processes messages from a queue receiver or topic subscriber during a load job.
- **Message Sender** processes messages from a queue sender or topic publisher during an export job.

Figure 12 shows an overview of Teradata Access Module for JMS processing.

Figure 12: Interface Components



Interface with the Data Connector

Teradata Access Module for JMS must provide a main function called *PIDMMain()*, which is called by the Data Connector to request I/O operations. This function contains two parameters:

```
void PIDMMain (pmiCmdBlock_t *Opts, void *OptParms)
```

The first parameter (*Opts*) specifies one of the access module functions in Table 14 on page 102. The second parameter (*OptParms*) contains context specific information for each of the functions specified in the table.

Table 14: PIDMMain's Opts Parameter Values

Access Module Function	Description
Access Module Initialization	Initializes Teradata Access Module for JMS and parses the initialization string.
Access Module Identification	Identifies Teradata Access Module for JMS and version.
File Open	Opens the queue or topic connection.
File Close	Closes the queue or topic connection.
File Read	Gets and processes data messages from a specified queue or topic.
File Write	Puts and processes data messages to a specified queue or topic.
File Get Position	Returns the current position of a data file for checkpoint and restart purposes.
File Set Position	Restart at a given point of a data file for checkpoint and restart purposes.
Get Attribute	Returns an attribute to the Teradata load/export utilities. Primarily for exchanging information between Teradata Access Module for JMS and the load/export utilities.
Put Attribute	Sends an attribute to Teradata Access Module for JMS. This is for exchanging information.
Shutdown	Terminates the access module.

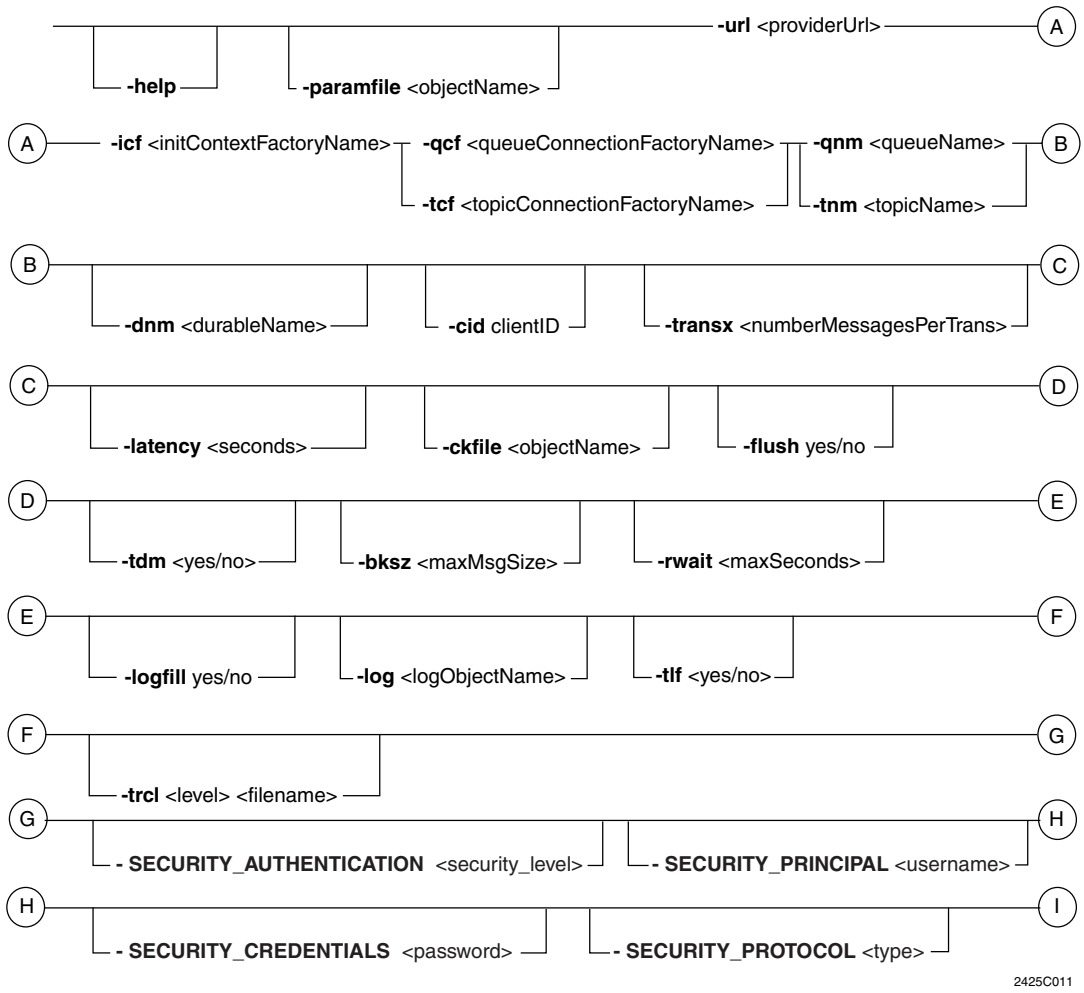
Initialization Strings

Following are the initialization strings for loading and exporting data using Teradata Access Module for JMS.

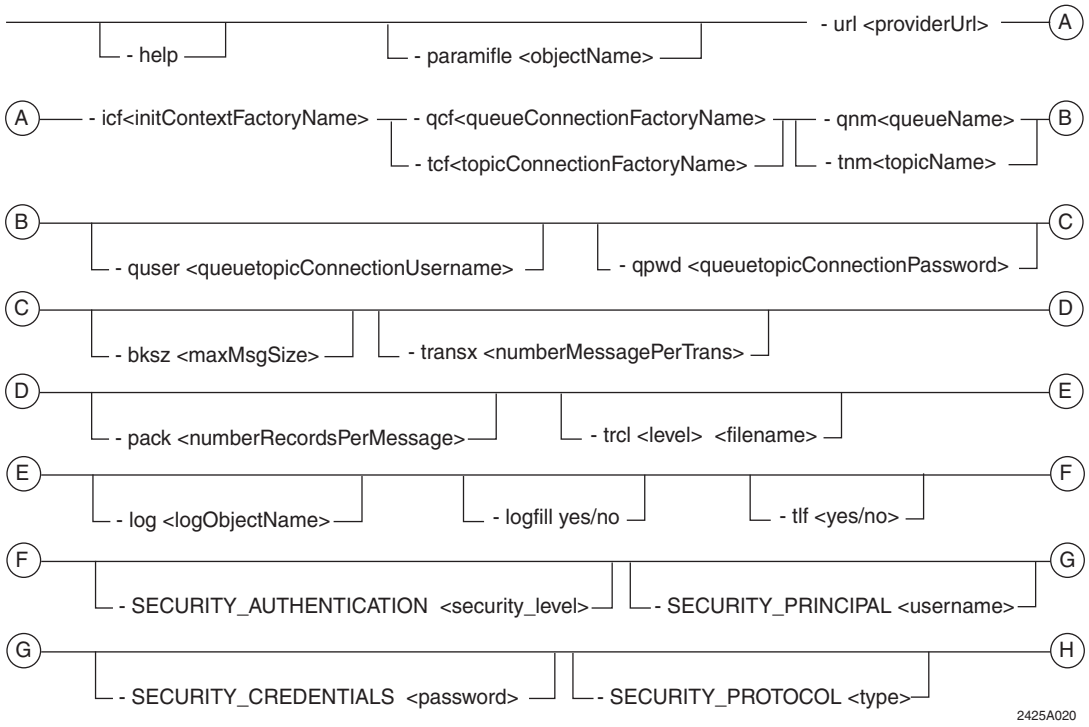
Syntax

The initialization string consists of a series of keyword and value pairs separated by blanks. Each keyword is preceded by a hyphen, and it is not case-sensitive. The value is either an integer or a string. Keywords can be specified within the initialization string or in a file. See "Access Module Calls" on page 19 for information about specific AXSMOD command or command option syntax for each Teradata utility.

Load Syntax


-url <providerUrl>
A
-help
-paramfile <objectName>
A
-icf <initContextFactoryName>
-qcf <queueConnectionFactoryName>
-qnm <queueName>
B
-tcf <topicConnectionFactoryName>
-tnm <topicName>
B
C
-dnm <durableName>
-cid clientID
-transx <numberMessagesPerTrans>
C
D
-latency <seconds>
-ckfile <objectName>
-flush yes/no
D
E
-tdm <yes/no>
-bksz <maxMsgSize>
-rwait <maxSeconds>
E
F
-logfill yes/no
-log <logObjectName>
-tlf <yes/no>
F
G
-trcl <level> <filename>
G
H
- SECURITY_AUTHENTICATION <security_level>
- SECURITY_PRINCIPAL <username>
H
I
- SECURITY_CREDENTIALS <password>
- SECURITY_PROTOCOL <type>
2425C011

Export Syntax



where:

Table 15: Export Syntax

Syntax Element	Description
bksz <*maxMsgSize*>	(Optional) Specifies the maximum message length in bytes. Default value is *32,000*. For example: `-bksz 1024`
cid <*clientID*>	(Optional) Specifies the unique client ID for the connection in a load job. Use with durable subscriptions. For example: `-cid clientID`
ckfile <*objectName*>	(Optional) Specifies the name of the file that contains checkpoint data from load jobs only. For example: `-ckfile sampleCheckpointFile`
dnm <*durableName*>	(Optional) Supplies the durable subscriber name for a load job. The subscriber receives all published messages, including the ones published while it is disconnected. For example: `-dnm sampleDurableName`
flush <*yes/no*>	(Optional) Ensures that all checkpoint data (from load jobs) is written to media. Default value is *no*. For example: `-flush yes`

Table 15: Export Syntax (continued)

Syntax Element	Description
help	(Optional) Displays a list of parameter keywords and valid values. For example: `-help`

icf <*initContextFactoryName*>	(Required) Specifies the class name of the initial context factory. For example:
	`-icf com.ibm.websphere.naming.WsnInitialContextFactory`
	`-icf weblogic.jndi.WLInitialContextFactory`
	`-icf com.sun.jndi.fscontext.RefFSConextFactory`

latency <*seconds*>	Specifies the time delay between receiving a current load message and the next message within a transaction, if *transx* is greater than 1. Default value is five seconds. For example: `-latency 4`
log <*logObjectName*>	(Optional) Specifies the name of a log file or STDOUT. All messages received from a queue or topic are displayed in the specified file. Default value is *no*. For example: `-log myLogFile`
logfill <*yes/no*>	(Optional) Packs log records to minimize wasted disk space. Default value is *yes*. `-logfill no`
mesgtype <*messageType*>	Specifies the type of export message. Valid values: • Text • Binary (default) For example: `-text`
pack <*numberRecordsPerMessage*>	Specifies the number of records per message. For example: `-pack 10`
paramfile <*objectName*>	(Optional) Supplies the object name for a file that includes more parameters. For example: `-paramfile myParmFile`
qcf <*queueConnectionFactoryName*>	Specifies the name of the queue connection factory. Required if *tcf* is not specified. For example: `-qcf sampleQCF`
qnm <*queueName*>	Specifies the queue name. Required if *qcf* is defined. For example: `-qnm myQueue`

Table 15: Export Syntax (continued)

Syntax Element	Description
qpwd <*queue/topicConnectionPassword*>	(Optional) Specifies the password to create the queue or topic connection. For example: `-qpwd password`
quser <*queue/topicConnectionUsername*>	(Required if *qpwd* is specified.) Specifies the username for queue or topic connection. For example: `-quser userName`

rwait <*maxSeconds*>	(Optional) Specifies the wait time in seconds if no load message is immediately available. The default value is 1 second. Valid values for an unlimited wait time are one of the following:
	• 1 to 600 seconds, or
	• minus 1.
	For example:
	`-rwait 500`
	`-rwait -1`

SECURITY_AUTHENTICATION <*security_level*>	(Optional) Context property that specifies the security level for data transfer. The behavior is determined by the service provider. Valid values:
	• none
	• simple
	• strong
	For example:
	`-SECURITY_AUTHENTICATION none`
	`-SECURITY_AUTHENTICATION simple`
	`-SECURITY_AUTHENTICATION strong`

SECURITY_CREDENTIALS <*password*>	(Optional) Context property that specifies the credentials of the principal for authenticating the caller to the service for data transfer. The value depends on the authentication scheme of the service provider. For example, it could be any of the following or more: • a hashed password • a clear-text password • a key • a certificate
SECURITY_PRINCIPAL <*username*>	(Optional) Context property that specifies the identity of the principal for authenticating the caller to the service for data transfer. The format depends on the authentication scheme of the service provider.
SECURITY_PROTOCOL <*type*>	(Optional) Specifies the security protocol for data transfer. The value is a string determined by the service provider, such as *ssl*. For example: `-SERVICE_PROTOCOL ssl`
tcf <*topicConnectionFactoryName*>	(Required if *qcf* is not specified.) Provides the topic connection factory name. For example: `-tcf sampleTCF`

Table 15: Export Syntax (continued)

Syntax Element	Description
tdm <*yes/no*>	(Optional) Terminates duplicate load messages. Default value is *yes*. For example: `-tdm no`
tlf <*yes/no*>	(Optional) Specifies that a log failure will cause a fatal error. Default value is *yes*. For example: `-tlf no`
tnm <*TopicName*>	(Required if *tcf* is defined.) Specifies the topic name. For example: `-tnm myTopic`
transx <*numberMessages*>	(Required) Specifies the number of messages in a transaction. For example: `-transx 4`
trcl <*level*><*fileName*>	(Optional) Specifies the trace level and file name of diagnostic activities. The trace level record these events: • 1=No diagnostic trace • 2=Job events • 3=I/ O events • 4=I/O buffers • 5=Detailed trace information The default trace level is 1. `-trcl 4 myFileName`

url <*providerUrl*>	(Required) Specifies the JNDI location of the administered objects. For example:
	`-url iiop: //myhost:9001 (from IBM Webshere MQ)`
	`-url t3://myhost:7001 (from BEA WebLogic JMS)`
	`-url file:/support/jms/samplet3 (file system context)`

Session Character Sets

All Teradata utilities (except for BTEQ) notify Teradata Access Module for JMS about the session character set they use. Specify the session character set name for each utility as follows:

- **Teradata FastExport, Teradata MultiLoad, or Teradata TPump** - Specify the *-c character-set-name* parameter at runtime in the command line to set the session character set name.
- **BTEQ or Teradata FastLoad** - Specify the *.SET SESSION CHARSET* command in the script.
- **Teradata PT** - Specify the *USING CHARACTER SET <charset-id>* option preceding the DEFINE JOB statement in the script.

Teradata Access Module for JMS uses a fixed mapping of Teradata session character sets to Java character sets. This mapping (Table 16) is a default properties file, called *charsets.properties*, in the installation folder of the access module. For a given Teradata session character set, the corresponding Java character set is used to encode bytes sent to the message queue system, and to decode bytes received from the queue system. You can add new character sets to the properties file and edit existing mappings.

Table 16: Character Set Mapping

Teradata Session Character Set	Java Character Set
ASCII	ASCII
UTF8	UTF-8
UTF16	UnicodeBigUnmarked (for big-endian platforms)
UTF16	UnicodeLittleUnmarked (for little-endian platforms)
EBCDIC037_0E	Cp037
EBCDIC273_0E	Cp273
EBCDIC277_0E	Cp277
HANGULEBCDIC933_1II	Cp933
HANGULKSC5601_2R4	MS949
KANJIEBCDIC5026_0I	Cp930
KANJIEBCDIC5035_0I	Cp939
KANJIEUC_0U	EUC_JP
KANJISJIS_0S	MS932
KATAKANAEBCDIC	CP930
LATIN1_0A	ISO8859_1
LATIN1252_0A	Cp1252
LATIN9_0A	ISO8859_15_FDIS
SCHEBCDIC935_2IJ	Cp935
SCHGB2312_1T0	EUC_CN
TCHBIG5_1R0	BIG5
TCHEBCDIC937_3IB	Cp937

If ASCII is specified as the character set, Teradata Access Module for JMS uses the platform default encoding. If that default encoding cannot represent certain characters in a platform-specific byte sequence, character conversions might produce minor differences. If an exact match is required, use the UTF8 session character set to avoid conversions between character sets.

Checkpoint Processing

If checkpointing is enabled, the access module reads messages from the queue, then copies the data into a fallback file before delivering the data to the Data Connector. If the load utility decides to restart at an earlier point in the data stream, it directs the Data Connector to issue the File Set Position command to the access module. The module then supplies data from the fallback data file until the restart is complete.

After a position request is made to the access module, additional processing by the client (and probably the database) occurs. A checkpoint request is considered successful only after a subsequent request from the client is received by the access module. A checkpoint followed by a function request for a reposition is considered successful by virtue of the contents of the position information provided at the reposition. Support for a previous checkpoint is not removed until the most recent checkpoint is successful. This approach allows recovery from a checkpoint failure.

When a reposition request immediately follows an open request, a restart is conducted. In this case, one or more messages are retrieved from the checkpoint file before the messages are sent back to the access module.

Only the load feature of Teradata Access Module for JMS supports checkpoint and restart capability for data recovery. Checkpoint and restarts are not supported during exports.

Repeatability of Messages

After recovery from a checkpoint failure, it is possible that messages read by one process may not be repeated to the same process in the same order because another process has read the rolled back messages in the queue. All messages sent to the Teradata utility after the most recent checkpoint request are rolled back into the message queue under the following conditions:

- A reposition following a database recovery is requested
- The client process fails via an ABEND

If you require repeatability, you should establish a reserved queue name convention in which a single reader process, such as TPump, uses the exclusive option to ensure that it is the only reader of that queue. No other process should use that reserved queue name.

If the checkpoint file is populated and the first request made to the access module following the opening of the named queue is a reposition, the first message returned is retrieved from the checkpoint file. Otherwise, there is no effect because only a single record is maintained in the checkpoint file. All other messages are implicitly rolled back in the case of a client ABEND.

Code Sample

Following is an example of an initialization function of the OptParms parameter for a load job.

```
PIDMMain(pmiCmdBlock_t*opts, void*optParms)

//Initialization Function
opts.Reqtype = pmiPIDMOptInit;

strcpy (optParms.EyeCatcher, pmiEC_Init_t); //struct eyecatch string

//Pass initialization string to access module
strcpy (optParms.InitStr, InitStr);

optParms.InitStrL = strlen(optParms.InitStr);  //Length of InitStr
optParms.StructLength=sizeof(optParms);        //Total structure length
optParms.InterfaceVerNo=pmiInterfaceVersionD;  //Header version number
of pmdcomt.h
optParms.InterfaceVerNoD=pmInterfaceVersion;   //Header version number
of pmddamt.h

//File Read Function
opts.Reqtype=pmiPIDMOptRead;
strcpy(optParms.EyeCatcher, pmiEC_RW_t);     //Stuct eyecatcher string

//Pointer to a buffer into which the access module is to return the data
block
optParms.Buffer=(char*)malloc(1024);

optParms.bufferLen=1024
optParms.StructLength=sizeof(optParms);     //Total structure length
```

Messages

Table 17 is a numeric listing of the access module error messages that can be written in the access module trace file. Many of the message descriptions cite access module functions that are generated by the Data Connector API in response to client utility commands. For information about these messages, see *Teradata Tools and Utilities Access Module Programmer Guide*.

Table 17: Error Messages

Number	Message Text	Description/Response
9	`The Access Module has reached EOF.`	The end-of-file character has been reached by the access module.
15	`Invalid Block_size value: value`	The value *value* is not a valid block size. Revise the access module initialization string in the utility job script to specify a valid block_size value.
18	`Bad version number.`	The running utility was built using a different version of the DataConnector other than the version that has been invoked (loaded) at runtime. Refer the problem to the client utility.

Table 17: Error Messages (continued)

Number	Message Text	Description/Response
22	`Invalid command option in Init string.`	An invalid access module command option was detected in the provided access module initialization string. Refer the problem to access module support.
24	`Access Module for JMS has received an unsupported request "mnemonic"(command code) from the calling software, "process name". It is possible that the calling software version is incompatible with this Access Module.`	The Client utility attempted an operation that is not supported by the access module. If it is available, the message text includes the standard command mnemonic (*mnemonic)*, along with the decimal command code (*code*). Possibly the Client utility is not able to use the access module.
26	`Access Module has not been initialized.`	A DataConnector function call other than for initialization has been received before the DataConnector was initialized via *pmInit*. Refer the problem to the client utility.
30	`Redundant initialize calls`	The access module has encountered another initialization statement after being initialized.
33	`An unrecognized attribute name was specified.`	The Data Connector API issued one or more of the following functions that are not implemented by the access module: • *pmiPIDMOptPutA_A* • *pmiPIDMOptGetA_A* • *pmiPIDMOptPutF_A* • *pmiPIDMOptGetF_A* **Note:** This message occurs frequently when using the FastLoad, TPump, and MultiLoad utilities.
34	`An unexpected critical exception occurred in the description.`	General failure message.

APPENDIX A

How to Read Syntax Diagrams

This appendix describes the conventions that apply to reading the syntax diagrams used in this book.

Syntax Diagram Conventions

Notation Conventions

Item	Definition / Comments
Letter	An uppercase or lowercase alphabetic character ranging from A through Z.
Number	A digit ranging from 0 through 9. Do not use commas when typing a number with more than 3 digits.
Word	Variables and reserved words. • UPPERCASE LETTERS represent a keyword. Syntax diagrams show all keywords in uppercase, unless operating system restrictions require them to be in lowercase. • lowercase letters represent a keyword that you must type in lowercase, such as a UNIX command. • *lowercase italic letters* represent a variable such as a column or table name. Substitute the variable with a proper value. • **lowercase bold letters** represent a variable that is defined immediately following the diagram that contains the variable. • UNDERLINED LETTERS represent the default value. This applies to both uppercase and lowercase words.
Spaces	Use one space between items such as keywords or variables.
Punctuation	Type all punctuation exactly as it appears in the diagram.

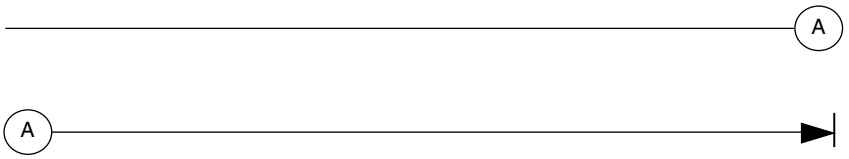
Paths

The main path along the syntax diagram begins at the left with a keyword, 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.

The only part of a path that reads from right to left is a loop.

Continuation Links

Paths that are too long for one line use continuation links. Continuation links are circled letters indicating the beginning and end of a link:



FE0CA002

When you see a circled letter in a syntax diagram, go to the corresponding circled letter and continue reading.

Required Entries

Required entries appear on the main path:

FE0CA003

If you can choose from more than one entry, the choices appear vertically, in a stack. The first entry appears on the main path:

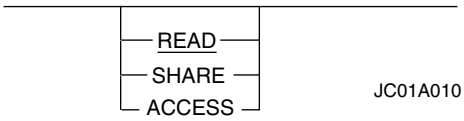
FE0CA005

Optional Entries

You may choose to include or disregard optional entries. Optional entries appear below the main path:

FE0CA004

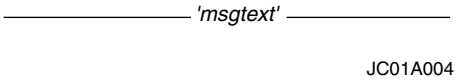
If you can optionally choose from more than one entry, all the choices appear below the main path:



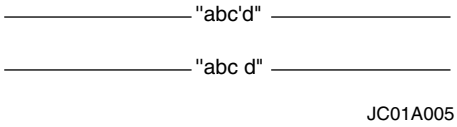
Some commands and statements treat one of the optional choices as a default value. This value is UNDERLINED. It is presumed to be selected if you type the command or statement without specifying one of the options.

Strings

Strings appear in single quotes:



If the string text includes a single quote or a blank space, the string appears in double quotes:



Abbreviations

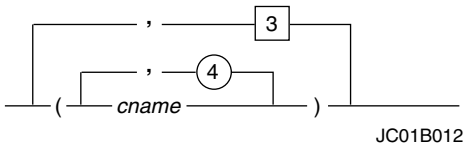
If a keyword or a reserved word has a valid abbreviation, the unabbreviated form always appears on the main path. The shortest valid abbreviation appears beneath.

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:



Read loops from right to left.

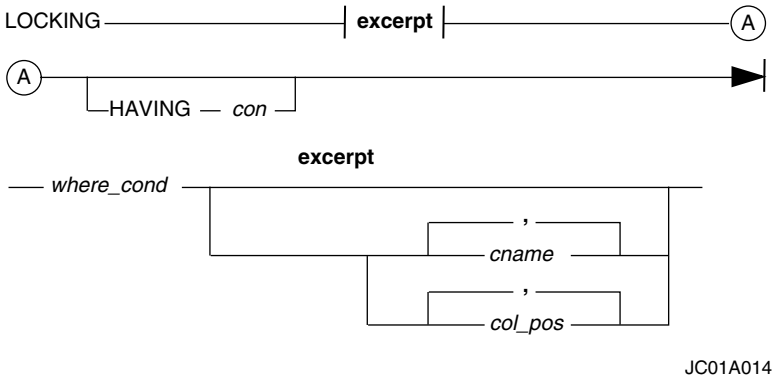
The following conventions apply to loops:

Loop Convention	Description
A maximum number of entries is allowed.	The number appears in a circle on the return path. In the example, you may type *cname* a maximum of 4 times.
A minimum number of entries is required.	The number appears in a square on the return path. In the example, you must type at least three groups of column names.
A separator character is required between entries.	The character appears on the return path. If the diagram does not show a separator character, use one blank space. In the example, the separator character is a comma.
A delimiter character is required around entries.	The beginning and end characters appear outside the return path. Generally, a space is not needed between delimiter characters and entries. In the example, the delimiter characters are the left and right parentheses.

Excerpts

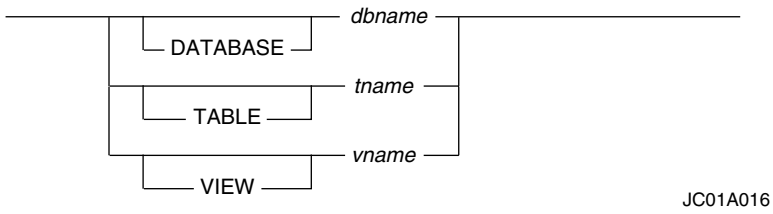
Sometimes a piece of a syntax phrase is too large to fit into the diagram. Such a phrase is indicated by a break in the path, marked by (|) terminators on either side of the break. The name for the excerpted piece appears between the terminators in boldface type.

The boldface excerpt name and the excerpted phrase appears immediately after the main diagram. The excerpted phrase starts and ends with a plain horizontal line:



Multiple Legitimate Phrases

In a syntax diagram, it is possible for any number of phrases to be legitimate:



In this example, any of the following phrases are legitimate:

- *dbname*
- DATABASE *dbname*
- *tname*
- TABLE *tname*
- *vname*
- VIEW *vname*

Sample Syntax Diagram

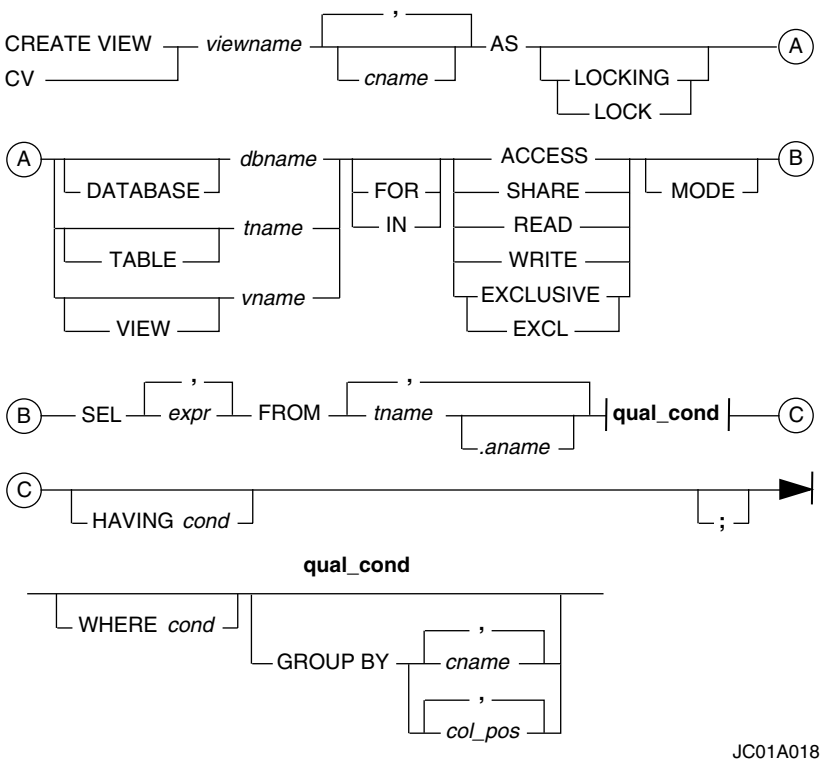


Diagram Identifier

The alphanumeric string that appears in the lower right corner of every diagram is an internal identifier used to catalog the diagram. The text never refers to this string.

APPENDIX B

Creating Schema Files

This appendix describes how to create a schema file that can be used as a data source by Teradata OLE DB Access Module.

Define a Schema File

The format of a text file is determined by using a schema information file. This file, named *schema.ini*, is always kept in the same directory as the text data source, and is required for accessing fixed-length data. Use a *schema.ini* file when a text table contains DateTime, Currency, Decimal data, or whenever more control is needed to handle table data.

Create a *schema.ini* file with entries to specify each of the following five characteristics for the table that needs to be created.

- **Text File Name** - The first entry in the *schema.ini* file must be the name of the text source file enclosed in square brackets.
- **File Format** - The file format option specifies how the text file fields are delimited or the length of the fields in a file that uses a fixed length format, as shown in Table 18. If used, the file format settings in the *schema.ini* file override file-by-file settings in the Windows Registry.

Table 18: Schema File Formats

Format specifier	Delimiter Description	Format statement example
Tab Delimited	Fields in the file are delimited by tabs	`Format=TabDelimited`
CSV Delimited	Fields in the file are delimited by commas (comma-separated values)	`Format=CSVDelimited`
Custom Delimited	Fields in the file are delimited by the character specified in the Format statement. All characters are allowed (even the blank character) except the double quote (")	`Format=Delimited(custom charater)`
Fixed Length	Fields in the file are a fixed length	`Format=FixedLength`

- **Field Names, Widths, and Types** - Specify the field names in a character-delimited text file in one the following ways:

- Set the ColNameHeader to True and include the field names in the first row of the table.

  For example, the following *schema.ini* file sets the ColNameHeader to true, keeping the column definitions and names.

  ```
  [Text_In_Out.txt]
  ColNameHeader=True
  Format=Delimited(#)
  MaxScanRows=25
  CharacterSet=ANSI
  ```

  The ColNameHeader setting overrides the FirstRowHasNames setting.
- Set ColNameHeader to False and specify each column by number and designate the column name, data type, and width for fixed-length types.

  For example, the following *schema.ini* file sets the ColNameHeader to false and redefines the columns.

  ```
  [Text_In_Out.txt]
  Format=Delimited(#)
  ColNameHeader=False
  MaxScanRows=25
  CharacterSet=ANSI
  Col1=SOR Char Width 255
  Col2=ID_Integer
  Col3=CREATE_DT Date
  Col4=FREETEXT LongChar
  Col5=EOR Char Width 255
  ```

  The ColNameHeader setting overrides the FirstRowHasNames setting.
- Use the MaxScanRows option to indicate how many rows to scan when determining the column types. If MaxScanRows is set to zero, the entire file is scanned.

  The MaxScanRows setting in the *schema.ini* file overrides the setting file by file.
- Use the column number (Col*n*) option. This option is required for fixed-length files; it is optional for character-delimited files.

  ```
  Coln=ColumnName type [width #]
  ```

  Table 19 describes each part of the Col*n* statement. The following example shows the *schema.ini* entries for two fields. The fields are specified as the fifth and sixth in the row format, with PartName defined as a text field with a width of ten, and PartNumber also defined as a text field, with a width of 30.

  ```
  Col[5]=PartNumber text width[10]
  Col[6]=PartName text width[30]
  ```

  Table 19: Coln Statement Parameters

Parameter	Description
ColumnName	Text name of the column. If the column name contains embedded spaces, enclose them in double quotation marks.

Table 19: Coln Statement Parameters (continued)

Parameter	Description	
type	MicroSoft Jet data types: • bit • byte • short • long • currency • single • double • datetime • text • memo	ODBC data types • char (same as text) • float (same as double) • integer (same as short) • longchar (same as memo) • date *date format*
width	Literal string value that specifies the width of the column (required for fixed-length files and optional for character-delimited files.	
#	Integer value that designates the width of the column. Required if width is specified.	

- **Character Sets** - Two character sets are available: ANSI and OEM. This setting overrides the setting in the Windows Registry. The following code sample sets the character set to ANSI:

  ```
  CharacterSet=ANSI
  ```

- **Currency Data Formats and Conversions** - Table 20 lists the settings and valid values for formatting currency data.

  Note: If any entry is omitted, the default value in the Windows Control Panel is used.

Table 20: Data Formats and Descriptions

Data Format	Description
CurrencyDigits	Specifies the number of digits in the fractional part of a currency amount.
CurrencyDecimalSymbol	Set to any single character that separates the whole from the fractional part of a currency amount.
CurrencyNegFormat	Set to one of the following values: • -$1, $-1, $1-, -1$, 1-$, or 1$- • Formats with one character separation: -1 $, -$ 1, 1 $-, $ 1-, $ -1, or 1- $ • Formats with parentheses: ($1) or (1$) • Format with parentheses and one character separation: ($ 1) or (1 $) **Note:** These examples use the dollar sign as a symbol value. Use the CurrencySymbol format to set the symbol value.

Table 20: Data Formats and Descriptions

Data Format	Description
CurrencyPosFormat	Set to one of the following values: • Currency symbol prefix with no separation ($1) • Currency symbol suffix with no separation (1$) • Currency symbol prefix with one character separation ($ 1) • Currency symbol suffix with one character separation (1 $)
CurrencySymbol	Specifies the currency symbol to use for currency values in the text file.
CurrencyThousandSymbol	Indicates the single-character symbol to use for separating currency values in the text file by thousands.
DateTimeFormat	Specifies a format string used for all dates and times. If set, all the date and time fields in the load or export job are handled with the same format. If not specified, the Windows Control Panel short date picture and time options are used. **Note:** All Microsoft Jet formats are supported except A.M. and P.M.
DecimalSymbol	Set to any single character used to separate the integer from the fractional part of a number.
NumberDigits	Provides the number of decimal digits in the fractional portion of a number.
NumberLeadingZeros	Set to specify if a decimal value less than one and greater than negative one should contain leading zeros. Valid values are False (no leading zeros) or True (insert leading zeros).

For more information about the settings in a *schema.ini* file, go to:
search.msdn.microsoft.com/search/default.aspx?siteId=0&tab=0&query=schema.ini

Glossary

A

application programming interface (API) The calling conventions by which an application program accesses the 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 to ensure the portability of code. May also provide an interface between a high-level language and lower-level services.

B

Basic Teradata Query (BTEQ) A general-purpose, command-based program that allows users on a workstation to communicate with one or more Teradata Database systems, and to format reports for both print and screen output. A Teradata product offering.

C

client module Any problem solving application that requires conventional I/O support.

D

Domain Name Services (DNS) Data query service chiefly used on the Internet for translating hostnames into Internet addresses. Also, the style of hostname used on the Internet, though such a name is properly called a *fully qualified domain name*. DNS can be configured to use a sequence of name servers, based on the domains in the name being looked for, until a match is found.

F

FIFO First in, first out An access method for a queue data structure pertaining to how data is loaded and retrieved.

J

Java Message Service (JMS) Java Message Oriented Middleware (MOM) application programming interface (API) for sending messages between two or more clients. The main elements of a JMS system include provider, client, producer, consumer, message, queue and topic elements.

Java Naming Directory Interface (JNDI) An API that allows clients to find data and objects via a name independent of the underlying implementation. An JNDI can also specify a service provider interface (SPI) that allows directory service implementations to be plugged into a framework. The vendor can implement the service as a server, a flat file, or a database.

Java virtual machine (JVM) A specification for software which interprets Java programs that have been compiled into byte-codes, and usually stored in a *.class* file. The JVM instruction set is stack-oriented with variable instruction length. The JVM itself is written in C and can be ported to run on most platforms. It needs thread support and I/O (for dynamic class loading). The Java byte-code is independent of the platform. There are also hardware implementations of the JVM.

J2EE Java 2 Platform, Enterprise Edition

J2SE Java 2 Platform, Standard Edition

M

MOM Message Oriented Middleware is a category of inter-application communication software that relies on asynchronous message passing as opposed to a request and response relationship. Most MOM software is based around a message queue system, although some implementations rely on a broadcast or multicast message system.

N

non-partitioned primary index (NPPI) Used in full table scans. See partitioned primary index (PPI).

O

open database connectivity (ODBC) An application programming standard that defines common database access mechanisms to simplify the exchange of data between a client and server. ODBC-compliant applications connect with a database through the use of a driver that translates the application's ODBC commands into database syntax.

OLE DB A set of Microsoft COM interfaces that allow uniform and consistent access to diverse data sources.

OLE DB Access Module See Teradata OLE DB Access Module.

OLE DB Provider A software module that exposes OLE DB interfaces to allow access to a specific data source. For example, "Microsoft OLE DB Provider for SQL Server" is an OLE DB provider that provides access to data located in a Microsoft SQL Server database.

OleLoad The GUI for the Teradata OLE DB Access Module, used to create, view, and edit *.amj* files. Teradata OleLoad can launch a Teradata utility.

P

partitioned primary index (PPI) An indexing mechanism in Teradata Database that improves performance for large tables when queries that specify a range constraint are submitted. PPI allows for the reduction of the number or rows processed by using partition elimination.

S

small computer system interface (SCSI) A popular processor-independent standard, via a parallel bus, for system-level interfacing between a computer and intelligent devices including hard disks, floppy disks, CD-ROM, printers, scanners, and more.

structured query language (SQL) An industry-standard language for creating, updating, and querying relational database management systems. Originally developed by IBM, it is often embedded in general-purpose programming languages.

T

Teradata OLE DB Access Module An access module created by Teradata that provides a basic input/output interface between Teradata load and export utilities and OLE DB data sources. It supports load operations to and export operations from a Teradata Database.

Teradata OLE DB Access Module allows you to select a data source from OLE DB data sources, create and save an access module job (*.amj*) file, and then use the *.amj* file to load the source data to a Teradata Database.

U

universal naming convention (UNC) Used in IBM PC networking to completely specify a directory on a file server.

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W