# TIBCO® Object Service Broker

## Managing External Data

*Software Release 6.0*
*July 2012*

two-second advantage™

## Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, The Power of Now, TIBCO Object Service Broker, and and TIBCO Service Gateway are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

The TIBCO Object Service Broker technologies described herein are protected under the following patent numbers:

| | | | | | | |
|---|---|---|---|---|---|---|
| Australia: | - | - | 671137 | 671138 | 673682 | 646408 |
| Canada: | 2284250 | - | - | 2284245 | 2284248 | 2066724 |
| Europe: | - | - | 0588446 | 0588445 | 0588447 | 0489861 |
| Japan: | - | - | - | - | - | 2-513420 |
| USA: | 5584026 | 5586329 | 5586330 | 5594899 | 5596752 | 5682535 |

# Contents

**Preface** . . . . . ix

Related Documentation . . . . . x
TIBCO Object Service Broker Documentation . . . . . x

Typographical Conventions . . . . . xv

Connecting with TIBCO Resources . . . . . xviii
How to Join TIBCOmmunity . . . . . xviii
How to Access All TIBCO Documentation . . . . . xviii
How to Contact TIBCO Support . . . . . xviii

**Chapter 1 Introducing External Data Interfaces** . . . . . 1

Overview . . . . . 2
Service Gateway for Files . . . . . 2
Adabas, CA Datacom, and VSAM LDS Access . . . . . 2

External Data Accesses . . . . . 3
IMP (Import) . . . . . 3
EXP (Export) . . . . . 3
VSM (VSAM) . . . . . 3

How TIBCO Object Service Broker Exchanges Data . . . . . 4

Initial Step for Defining Tables . . . . . 5
Invoke the Table Definer . . . . . 5
Specify the Table Type for New Tables . . . . . 5

Using Data Discovery . . . . . 6
Monitoring Copybook Changes . . . . . 6
Running the Change Tracking Agent . . . . . 6

**Chapter 2 Installing and Using TIBCO Service Gateway for Files** . . . . . 9

Overview . . . . . 10
Supported Connectivity . . . . . 10
Optional Field for Service Gateway for Files . . . . . 10
Restrictions in Use . . . . . 11
Shares Infrastructure Used by Peer Servers . . . . . 11
Request Flow . . . . . 11

Distribution Media and Contents . . . . . 12
Installation Media . . . . . 12
Open Systems Distribution . . . . . 12

Installing the Software on an Open Systems Host . . . 13
Installation Types . . . 13
Installation Modes . . . 13
Preparation for Installation . . . 14
Installing the Service Gateway for Files . . . 15
Installation Modes — Using a Response File . . . 16
Post Installation . . . 17
Uninstalling the Software . . . 18
Installing the Software on a z/OS Host . . . 19
Installing Remote on Same Host . . . 19
Installing Remote on Separate Host . . . 22
Configuration of TCP/IP . . . 30
Verification of Installation . . . 32
Reruns of the Installation . . . 33
Restarts of the Installation . . . 34
Uninstalling . . . 35
Configuring Service Gateway for Files . . . 36
Configuring a Service Gateway for Files to Run on Open Systems . . . 36
Configuring a Service Gateway for Files to Run on z/OS . . . 38
Setting Up the Communications Infrastructure . . . 40
Prerequisites . . . 40
Data Object Broker on z/OS to Communicate with a Service Gateway for Files . . . 40
Definition of the Gateway as a Resource . . . 41
Management of Service Gateway for Files . . . 43
Startup of a Service Gateway for Files . . . 43
Shutdown of a Service Gateway for Files . . . 43
Automatic Restart of a Service Gateway for Files . . . 44
Monitoring of Service Gateway for Files . . . 45
Administration . . . 45
Log Files . . . 45
Service Gateway for Files and National Language Support . . . 46

**Chapter 3 Managing IMP Data Definitions . . . 49**

Accessing External Data from TIBCO Object Service Broker . . . 50
Using a Copybook as the Source for the Definition . . . 50
Steps Required to Define an Import Table . . . 51
Task A: Identify the Table . . . 52
Table, Type, Unit, and IDgen Fields . . . 52
Task B: Identify the Data . . . 54
File Field . . . 54
DDname Field . . . 55
Server ID Field . . . 56

Data Cleansing Field . . . . . 56
Task C: Specify Data and Location Parameters. . . . . 57
Data Parameter . . . . . 57
Location Parameter . . . . . 58
Task D: Specify Event Rules . . . . . 60
Event Rule, Typ, and Acc Fields . . . . . 60
Task E: Define Fields . . . . . 62
Specifying External Import Attributes . . . . . 62
Specifying Internal TIBCO Object Service Broker Attributes . . . . . 64

**Chapter 4 Defining Import Tables for Files with Multiple Record Formats. . . . . 67**

How to Define Import Tables for Files with Multiple Record Formats . . . . . 68
What is a Multiple Record Format? . . . . . 68
What is a Repeating Group? . . . . . 68
Definition Requirements . . . . . 68
First Sample File – Personnel . . . . . 70
Types of Records in File . . . . . 70
Table Definitions . . . . . 70
Second Sample File – Inventory . . . . . 72
Types of Records in File . . . . . 72
Table Definitions . . . . . 72
Processing . . . . . 76

**Chapter 5 Manipulating Import Data Using TIBCO Object Service Broker. . . . . 79**

Accessing Import Tables . . . . . 80
Using the Table Browser. . . . . 80
Using Rules . . . . . 80
Considerations . . . . . 81
Retrieval Processing. . . . . 81
Remote Table Access . . . . . 82
Steps to Process a Multi-record Table Remotely . . . . . 82
Sample Rules . . . . . 83
Sample Rule 1: Normal Access . . . . . 83
Sample Rule 2: Accessing Multiple Record Formats . . . . . 83
Explanation of the EMPLOYEE_B Rule . . . . . 84
Listing Members of a PDS (z/OS) or Files in a Directory (Open Systems) . . . . . 85
Example for a PDS in z/OS . . . . . 85
Example for a Directory in Windows or Solaris. . . . . 87
Handling TIBCO Object Service Broker Requests . . . . . 89
ERROR Exception . . . . . 89
ACCESSFAIL Exception. . . . . 89
INTEGRITYFAIL Exception . . . . . 89

External Routines (Pre-processing the Data) . . . 90
Manipulating Data with External Routines (z/OS Only) . . . 90
Parameters Passed to the External Routine . . . 90
Valid Return Codes . . . 91
Example of an Exit . . . 92
Example of an Exit that Normalizes Data . . . 92

**Chapter 6 Managing EXP Data Definitions . . . 95**

Writing TIBCO Object Service Broker Tables to External Files . . . 96
Using a Copybook as the Source for the Definition . . . 96
Steps Required to Define Export Tables . . . 97
Task A: Identify the Table . . . 98
Table, Type, and Unit Fields . . . 98
Task B: Identify the Data . . . 99
File Field . . . 99
DDname Field . . . 100
Server ID Field . . . 101
External Routine Name Field . . . 101
Task C: Specify Data and Location Parameters . . . 102
Data Parameter . . . 102
Location Parameter . . . 102
Task D: Specify Event Rules . . . 104
Event Rule, Typ, and Acc Fields . . . 104
Task E: Define Fields . . . 105
Considerations . . . 105
Specifying External Export Attributes . . . 106
Specifying Internal TIBCO Object Service Broker Attributes . . . 107

**Chapter 7 Processing External Data Using TIBCO Object Service Broker . . . 111**

Writing Data to an Export File and Accessing the Exported Data . . . 112
Using Rules . . . 112
Using COPY_DATA . . . 112
Using Copy Table . . . 112
Accessing the Exported Data . . . 113
Handling TIBCO Object Service Broker Requests . . . 114
Transaction Length . . . 114
Error Handling . . . 114
ERROR Exception . . . 114
ACCESSFAIL Exception . . . 114
INTEGRITYFAIL Exception . . . 115
External Routines (Pre-processing the Data) . . . 116
Manipulating Data with External Routines (z/OS Only) . . . 116

Parameters Passed to the External Routine . . . . . 116
Valid Return Codes . . . . . 117
Example of an Exit that Exports Variable Length Records . . . . . 117

**Chapter 8 Managing VSAM Data Definitions . . . . . 119**

Accessing VSAM Data from TIBCO Object Service Broker . . . . . 120
Using a Copybook as the Source for the Definition . . . . . 120
Steps Required to Define a VSAM Table . . . . . 121
Task A: Identify the Table . . . . . 122
Table, Type, Unit, and IDgen Fields . . . . . 122
Task B: Identify the Data . . . . . 124
File Field . . . . . 124
DDname Field . . . . . 124
Read Only Field . . . . . 125
Load Field . . . . . 125
Data Set Type Field . . . . . 125
Ignore Field . . . . . 125
Server ID Field . . . . . 126
Task C: Specify Data and Location Parameters . . . . . 127
Data Parameter . . . . . 127
Location Parameter . . . . . 128
Task D: Specify Event Rules . . . . . 129
Event Rule, Typ, and Acc Fields . . . . . 129
Task E: Define Fields . . . . . 130
Considerations . . . . . 130
Specifying External VSAM Attributes . . . . . 131
Specifying Internal TIBCO Object Service Broker Attributes . . . . . 132
Behavior of Numeric Key Fields in VSAM Tables . . . . . 134
Data Set Requirements . . . . . 135
KSDS Requirements . . . . . 135
ESDS Requirements . . . . . 135
RRDS Requirements . . . . . 136

**Chapter 9 Defining VSAM Tables for Files with Multiple Record Formats . . . . . 137**

VSAM Tables for Files with Multiple Records . . . . . 138
What is a Multiple Record Format? . . . . . 138
What is a Repeating Group? . . . . . 138
Sample File . . . . . 138
Definition Requirements . . . . . 139
Base and Child Definitions . . . . . 139
Definition Requirements of a Child Table . . . . . 139
Sample Definition (Record A) . . . . . 140

Sample Definition (Record B) . . . . . 140

**Chapter 10 Processing VSAM Data Using TIBCO Object Service Broker. . . . . 143**

Access of TIBCO Object Service Broker VSAM Tables . . . . . 144
Editing or Browsing . . . . . 144
Using the Table Editor . . . . . 144
Using the Table Browser . . . . . 144
Using Rules . . . . . 145
VSAM Files with Multiple Record Formats and Repeating Groups . . . . . 145
Retrieval Processing . . . . . 146
Sample Rules . . . . . 147
Sample Rule 1: Accessing Multiple Record Formats. . . . . 147
Sample Rule 2: Inserting Parent Record with Children . . . . . 147
Handling of TIBCO Object Service Broker Requests . . . . . 149
Synchronization and Recovery . . . . . 149
Error Handling . . . . . 150
ERROR Exception . . . . . 150
ACCESSFAIL Exception . . . . . 151
INTEGRITYFAIL Exception . . . . . 151

**Appendix A Mapping Data Types . . . . . 153**

**Appendix B Mapping File Names for Open Systems . . . . . 161**

Access to External Files . . . . . 161
Providing a File Name . . . . . 161
Using the DSDIR Parameter to Provide a File Name . . . . . 161
Using filespec.dsn to Provide a File Name . . . . . 162
Mapping a Partitioned Data Set. . . . . 164

**Appendix C Data Cleansing . . . . . 167**

Options . . . . . 168
Examples . . . . . 170

**Index . . . . . 171**

# Preface

TIBCO® Object Service Broker is an application development environment and integration broker that bridges legacy and non-legacy applications and data.

This manual describes the interfaces from TIBCO Object Service Broker to import, export, and VSAM data. It also describes the Service Gateway for Files and how to install and use it to access import, export and VSAM data that is remote to your system.

## Topics


- Related Documentation, page x
- Typographical Conventions, page xv
- Connecting with TIBCO Resources, page xviii

# Related Documentation

This section lists documentation resources you may find useful.

## TIBCO Object Service Broker Documentation

The following documents form the TIBCO Object Service Broker documentation set:

### Fundamental Information

The following manuals provide fundamental information about TIBCO Object Service Broker:

- *TIBCO Object Service Broker Getting Started* Provides the basic concepts and principles of TIBCO Object Service Broker and introduces its components and capabilities. It also describes how to use the default developer's workbench and includes a basic tutorial of how to build an application using the product. A product glossary is also included in the manual.
- *TIBCO Object Service Broker Messages with Identifiers* Provides a listing of the TIBCO Object Service Broker messages that are issued with alphanumeric identifiers. The description of each message includes the source and explanation of the message and recommended action to take.
- *TIBCO Object Service Broker Messages without Identifiers* Provides a listing of the TIBCO Object Service Broker messages that are issued without a message identifier. These messages use the percent symbol (%) or the number symbol (#) to represent such variable information as a rule's name or the number of occurrences in a table. The description of each message includes the source and explanation of the message and recommended action to take.
- *TIBCO Object Service Broker Quick Reference* Presents summary information for use in the TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Shareable Tools* Lists and describes the TIBCO Object Service Broker shareable tools. Shareable tools are programs supplied with TIBCO Object Service Broker that facilitate rules language programming and application development.
- *TIBCO Object Service Broker Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

### Application Development and Management

The following manuals provide information about application development and management:

- *TIBCO Object Service Broker Application Administration* Provides information required to administer the TIBCO Object Service Broker application development environment. It describes how to use the administrator's workbench, set up the development environment, and optimize access to the database. It also describes how to manage the Pagestore, which is the native TIBCO Object Service Broker data store.
- *TIBCO Object Service Broker Managing Data* Describes how to define, manipulate, and manage data required for a TIBCO Object Service Broker application.
- *TIBCO Object Service Broker Managing External Data* Describes the TIBCO Object Service Broker interface to external files (not data in external databases) and describes how to define TIBCO Object Service Broker tables based on these files and how to access their data.
- *TIBCO Object Service Broker National Language Support* Provides information about implementing the National Language Support in a TIBCO Object Service Broker environment.
- *TIBCO Object Service Broker Object Integration Gateway* Provides information about installing and using the Object Integration Gateway which is the interface for TIBCO Object Service Broker to XML, J2EE, .NET and COM.
- *TIBCO Object Service Broker for Open Systems External Environments* Provides information on interfacing TIBCO Object Service Broker with the Windows and Solaris environments. It includes how to use SDK (C/C++) and SDK (Java) to access TIBCO Object Service Broker data, how to interface to TIBCO Enterprise Messaging Service (EMS), how to use the TIBCO Service Gateway for WMQ, how to use the Adapter for JDBC-ODBC, and how to access programs written in external programming languages from within TIBCO Object Service Broker.
- *TIBCO Object Service Broker for z/OS External Environments* Provides information on interfacing TIBCO Object Service Broker to various external environments within a TIBCO Object Service Broker z/OS environment. It also includes information on how to access TIBCO Object Service Broker from different terminal managers, how to write programs in external programming languages to access TIBCO Object Service Broker data, how to interface to TIBCO Enterprise Messaging Service (EMS), how to use the TIBCO Service Gateway for WMQ, and how to access programs written in external programming languages from within TIBCO Object Service Broker.

- *TIBCO Object Service Broker Parameters* Lists the TIBCO Object Service Broker Execution Environment and Data Object Broker parameters and describes their usage.
- *TIBCO Object Service Broker Programming in Rules* Explains how to use the TIBCO Object Service Broker rules language to create and modify application code. The rules language is the programming language used to access the TIBCO Object Service Broker database and create applications. The manual also explains how to edit, execute, and debug rules.
- *TIBCO Object Service Broker Managing Deployment* Describes how to submit, maintain, and manage promotion requests in the TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Defining Reports* Explains how to create both simple and complex reports using the reporting tools provided with TIBCO Object Service Broker. It explains how to create reports with simple features using the Report Generator and how to create reports with more complex features using the Report Definer.
- *TIBCO Object Service Broker Managing Security* Describes how to set up, use, and administer the security required for an TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Defining Screens and Menus* Provides the basic information to define screens, screen tables, and menus using TIBCO Object Service Broker facilities.
- *TIBCO Service Gateway for Files SDK* Describes how to use the SDK provided with the TIBCO Service Gateway for Files to create applications to access Adabas, CA Datacom, and VSAM LDS data.

### System Administration on the z/OS Platform

The following manuals describe system administration on the z/OS platform:

- *TIBCO Object Service Broker for z/OS Installing and Operating* Describes how to install, migrate, update, maintain, and operate TIBCO Object Service Broker in a z/OS environment. It also describes the Execution Environment and Data Object Broker parameters used by TIBCO Object Service Broker.
- *TIBCO Object Service Broker for z/OS Managing Backup and Recovery* Explains the backup and recovery features of OSB for z/OS. It describes the key components of TIBCO Object Service Broker systems and describes how you can back up your data and recover from errors. You can use this information, along with assistance from TIBCO Support, to develop the best customized solution for your unique backup and recovery requirements.

- *TIBCO Object Service Broker for z/OS Monitoring Performance* Explains how to obtain and analyze performance statistics using TIBCO Object Service Broker tools and SMF records
- *TIBCO Object Service Broker for z/OS Utilities* Contains an alphabetically ordered listing of TIBCO Object Service Broker utilities for z/OS systems. These are TIBCO Object Service Broker administrator utilities that are typically run with JCL.

### System Administration on Open Systems

The following manuals describe system administration on open systems such as Windows or UNIX:

- *TIBCO Object Service Broker for Open Systems Installing and Operating* Describes how to install, migrate, update, maintain, and operate TIBCO Object Service Broker in Windows and Solaris environments.
- *TIBCO Object Service Broker for Open Systems Managing Backup and Recovery* Explains the backup and recovery features of TIBCO Object Service Broker for Open Systems. It describes the key components of a TIBCO Object Service Broker system and describes how to back up your data and recover from errors. Use this information to develop a customized solution for your unique backup and recovery requirements.
- *TIBCO Object Service Broker for Open Systems Utilities* Contains an alphabetically ordered listing of TIBCO Object Service Broker utilities for Windows and Solaris systems. These TIBCO Object Service Broker administrator utilities are typically executed from the command line.

### External Database Gateways

The following manuals describe external database gateways:

- *TIBCO Service Gateway for Files* Describes the TIBCO Object Service Broker interface to Adabas and Datacom data. Using this interface, you can access external Adabas and Datacom data and define TIBCO Object Service Broker tables based on this data.
- *TIBCO Service Gateway for DB2 Installing and Operating* Describes the TIBCO Object Service Broker interface to DB2 data. Using this interface, you can access external DB2 data and define TIBCO Object Service Broker tables based on this data.
- *TIBCO Service Gateway for IDMS/DB Installing and Operating* Describes the TIBCO Object Service Broker interface to CA-IDMS data. Using this interface, you can access external CA-IDMS data and define TIBCO Object Service Broker tables based on this data.

- *TIBCO Service Gateway for IMS/DB Installing and Operating* Describes the TIBCO Object Service Broker interface to IMS/DB and DB2 data. Using this interface, you can access external IMS data and define TIBCO Object Service Broker tables based on it.
- *TIBCO Service Gateway for ODBC and for Oracle Installing and Operating* Describes the TIBCO Object Service Broker ODBC Gateway and the TIBCO Object Service Broker Oracle Gateway interfaces to external DBMS data. Using this interface, you can access external DBMS data and define TIBCO Object Service Broker tables based on this data.

# Typographical Conventions

The following typographical conventions are used in this manual.

*Table 1 General Typographical Conventions*

| Convention | Use |
|---|---|
| *TIBCO_HOME*<br>*OSB_HOME* | By default, all TIBCO products are installed into a folder referenced in the documentation as *TIBCO_HOME*.<br>On open systems, TIBCO Object Service Broker installs by default into a directory within *TIBCO_HOME*. This directory is referenced in documentation as *OSB_HOME*. The default value of *OSB_HOME* depends on the operating system. For example on Windows systems, the default value is `C:\tibco\OSB`. Similarly, all TIBCO Service Gateways on open systems install by default into a directory in *TIBCO_HOME*. For example on Windows systems, the default value is `C:\tibco\OSBgateways\6.0`.<br>On z/OS, no default installation directories exist. |
| `code font` | Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:<br>Use `MyCommand` to start the foo process. |
| **`bold code font`** | Bold code font is used in the following ways:<br>• In procedures, to indicate what a user types. For example: Type **`admin`**.<br>• In large code samples, to indicate the parts of the sample that are of particular interest.<br>• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, `MyCommand` is enabled: `MyCommand [`**`enable`**` \| disable]` |
| *italic font* | Italic font is used in the following ways:<br>• To indicate a document title. For example: See *TIBCO ActiveMatrix BusinessWorks Concepts*.<br>• To introduce new terms For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal.<br>• To indicate a variable in a command or code syntax that you must replace. For example: `MyCommand` *PathName* |

*Table 1 General Typographical Conventions (Cont'd)*

| Convention | Use |
|---|---|
| Key combinations | Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C.<br>Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q. |
|  | The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances. |
|  | The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result. |
|  | The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken. |

*Table 2 Syntax Typographical Conventions*

| Convention | Use |
|---|---|
| `[ ]` | An optional item in a command or code syntax.<br>For example:<br>`MyCommand [optional_parameter] required_parameter` |
| `\|` | A logical `OR` that separates multiple items of which only one may be chosen.<br>For example, you can select only one of the following parameters:<br>`MyCommand para1 \| param2 \| param3` |

*Table 2 Syntax Typographical Conventions*

| Convention | Use |
|---|---|
| `{ }` | A logical group of items in a command. Other syntax notations may appear within each logical group.<br><br>For example, the following command requires two parameters, which can be either the pair `param1` and `param2`, or the pair `param3` and `param4`.<br><br>`MyCommand {param1 param2} \| {param3 param4}`<br><br>In the next example, the command requires two parameters. The first parameter can be either `param1` or `param2` and the second can be either `param3` or `param4`:<br><br>`MyCommand {param1 \| param2} {param3 \| param4}`<br><br>In the next example, the command can accept either two or three parameters. The first parameter must be `param1`. You can optionally include `param2` as the second parameter. And the last parameter is either `param3` or `param4`.<br><br>`MyCommand param1 [param2] {param3 \| param4}` |

# Connecting with TIBCO Resources

## How to Join TIBCOmmunity

TIBCOmmunity is an online destination for TIBCO customers, partners, and resident experts, a place to share and access the collective experience of the TIBCO community. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

## How to Access All TIBCO Documentation

You can access TIBCO documentation here:

http://docs.tibco.com

## How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

# Chapter 1 Introducing External Data Interfaces

This chapter describes the interfaces to external files. Those interfaces enable you to access external data, such as import, export, and VSAM data, from TIBCO Object Service Broker.

## Topics


- Overview, page 2
- External Data Accesses, page 3
- How TIBCO Object Service Broker Exchanges Data, page 4
- Initial Step for Defining Tables, page 5
- Using Data Discovery on page 6

# Overview

TIBCO Object Service Broker provides you with interfaces to import and export files, Adabas data, CA Datacom data, and VSAM data sets, allowing you to access data in these file types from within your applications. These interfaces ensure data is presented to and received from TIBCO Object Service Broker rules in a manner consistent with TIBCO Object Service Broker behavior. Some of these access are provided with your base TIBCO Object Service Broker installation and others require Service Gateway for Files.

## Service Gateway for Files

Service Gateway for Files is available to enable you to easily access files and VSAM data sets that are remote to your system, without requiring an additional TIBCO Object Service Broker installation on the remote system. This enables, for example, an ODBC request through a TIBCO Object Service Broker on Open Systems to directly access flat files and VSAM files on a z/OS system.

Optionally, you can also use distributed data to access import, export and most VSAM data set types across systems. This requires the installation of an additional TIBCO Object Service Broker system. Refer to the *Managing Data* and *Application Administration* manuals for more information about distributed data.

## Adabas, CA Datacom, and VSAM LDS Access

To access Adabas and VSAM LDS data you require the Service Gateway for Files SDK which is included with the Service Gateway for Files. For information about the SDK refer to the *TIBCO* Service Gateway for Files *SDK User's Guide*.

- Service Gateway for Files is a separately licensed add-on to TIBCO Object Service Broker.
- Service Gateway for Files differs from external database gateways which are discussed in the *TIBCO Service Gateway* manuals.
- Service Gateway for Files has its own installation procedure. Refer to Chapter 2, Installing and Using TIBCO Service Gateway for Files, on page 9 for detailed information.

# External Data Accesses

Accesses to import, export and most VSAM data sets are included with the base TIBCO Object Service Broker product. These are described in the following sections. Access to Adabas and VSAM LDS data are described in the *TIBCO* Service Gateway for Files *SDK User's Guide*.

## IMP (Import)

You use IMP tables to read (import) external files directly into TIBCO Object Service Broker. Depending on the operating system and type of organization, external files can be sequential data sets or members of partitioned data sets (z/OS), or a file or files within a directory (Windows or Solaris).

You use the Table Definer to define an IMP table. At execution time, you specify the file to import and access the data either online or in batch mode.

For more information on IMP tables, refer to Chapter 3, Managing IMP Data Definitions, on page 49, and to Chapter 5, Manipulating Import Data Using TIBCO Object Service Broker, on page 79.

## EXP (Export)

You use EXP tables to write (export) TIBCO Object Service Broker tables directly to external files. Depending on the operating system and type of organization, external files can be sequential data sets or members of partitioned data sets (z/OS), or a file or files within a directory (Windows or Solaris).

You use the Table Definer to define an EXP table and insert data into the table using TIBCO Object Service Broker tools. After data is inserted into the table, you access the data outside of TIBCO Object Service Broker. At execution time, specify the target export file and export this data either online or in batch mode.

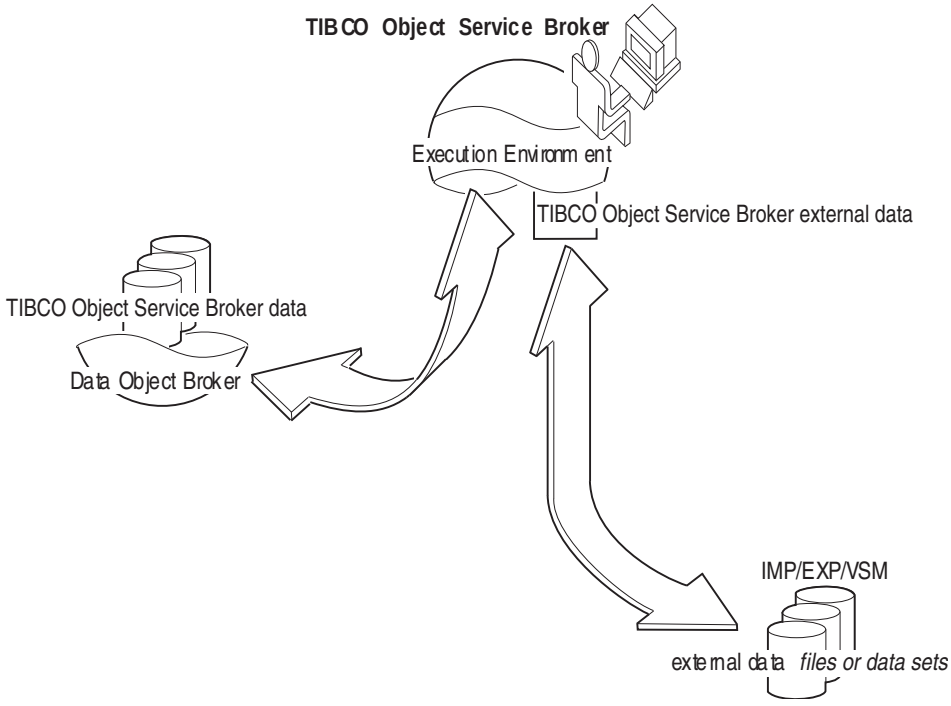
For more information on EXP tables, refer to Chapter 6, Managing EXP Data Definitions, on page 95, and Chapter 7, Processing External Data Using TIBCO Object Service Broker, on page 111.

## VSM (VSAM)

You use VSM tables to access data in a VSAM file. You use the Table Definer to define a VSM table and use TIBCO Object Service Broker tools to access the data. For more information on VSM tables, refer to Chapter 8, Managing VSAM Data Definitions, on page 119, and to Chapter 10, Processing VSAM Data Using TIBCO Object Service Broker, on page 143.

## How TIBCO Object Service Broker Exchanges Data

The following diagram shows how you can read, write, and access external data, or access a VSAM file, while still having access to TIBCO Object Service Broker data:



# Initial Step for Defining Tables

## Invoke the Table Definer

Invoke the Table Definer from the workbench using the DT define table option or the primary command field. You can access an existing definition or define a new TIBCO Object Service Broker table.

## Specify the Table Type for New Tables

After entering the initial Table Definer screen, you change the table type to IMP, EXP, or VSM and press Enter; the appropriate Table Definition screen appears.

| To define a table of type… | Refer to… |
|---|---|
| IMP | Chapter 3, Managing IMP Data Definitions, page 49. |
| EXP | Chapter 6, Managing EXP Data Definitions, page 95. |
| VSM | Chapter 8, Managing VSAM Data Definitions, page 119. |

See Also *TIBCO Object Service Broker Getting Started* for information on invoking workbench tools.

# Using Data Discovery

When you create an IMP, EXP, or VSM table in the TIBCO Object Service Broker UI, you can use a copybook as the source for its definition using Data Discovery.

## Monitoring Copybook Changes

TIBCO Object Service Broker can monitor changes to the copybook from the TIBCO Object Service Broker UI if the source is a member of a PDS. To do so, set the Monitor flag for the table. To check for changes on all monitored tables, run the Change Tracking Agent. Each table is also checked when the definition is viewed in the TIBCO Object Service Broker UI.

## Running the Change Tracking Agent

Member CTA of the JCL data set contains JCL to run the Change Tracking Agent in Batch mode. When you run the Agent, it checks the copybooks that were used to create the tables in your TIBCO Object Service Broker system that have the Monitor flag set. This job must be run from a TIBCO Object Service Broker level-7 user id.

The Change Tracking Agent then indicates (with the “TIMESTAMPS DIFFERENT” message) that changes were made to the copybook since the last time the member statistics were updated for the table. Also, the next time you view the definition of the table in the TIBCO Object Service Broker UI, you see a message telling you that the definition is out of sync with the copybook. To remove this message, save the table (with or without changes) in the TIBCO Object Service Broker UI and confirm the request to reset this message.

### Sample Output

```
Page 1
CHANGE TRACKING - DIFFERENCES REPORT
2007-03-11

-------------------------------------------------------------------------------
OBJECT NAME      DATASET NAME               MEMBER   STATUS
---------------- -------------------------- -------- ----------------

TABLE3           USR40.OSB.COBOL            CBCUST   < TIMESTAMPS DIFFERENT
TABLE            USR40.OSB.COPYLIB          CBCUST   < OK, TIMESTAMPS EQUAL
TABLE2           USR40.OSB.COPYLIB          CBCUST   < OK, TIMESTAMPS EQUAL

*** DIFFERENCES FOUND ***
-------------------------------------------------------------------------------
```

See Also *TIBCO Object Service Broker UI Help* for more information about using Data Discovery to help define IMP, EXP and VSM tables.

# Chapter 2 Installing and Using TIBCO Service Gateway for Files

This chapter describes how to install and operate Service Gateway for Files on z/OS.

## Topics


- Overview, page 10
- Distribution Media and Contents, page 12
- Installing Remote on Separate Host, page 22
- Uninstalling, page 35
- Configuring Service Gateway for Files, page 36
- Setting Up the Communications Infrastructure, page 40
- Management of Service Gateway for Files, page 43
- Monitoring of Service Gateway for Files, page 45
- Service Gateway for Files and National Language Support, page 46

# Overview

Service Gateway for Files provides access to import, export and VSAM files that are remote to a Data Object Broker. Other supported accesses from rules to these file types require a Data Object Broker to run on the target system or the use of distributed data.

Service Gateway for Files (z/OS) provides interfaces to import, export and VSAM files. The Gateway ensures that data is presented to and received from TIBCO Object Service Broker rules in a manner consistent with TIBCO Object Service Broker behavior.

Service Gateway for Files is a separately chargeable component. To order it, contact your regional TIBCO sales office.

The Service Gateway for Files SDK is provided with the Service Gateway for Files. You can use this SDK to build applications to access Adabas, Datacom, and VSAM LDS data from TIBCO Object Service Broker. Refer to *TIBCO* Service Gateway for Files *SDK User's Guide* for information about using the SDK. You must add database objects to your TIBCO Object Service Broker 6.0.0 system to use the SDK. For details, refer to Installing Remote on Separate Host, page 22.

## Supported Connectivity

Service Gateway for Files must run on the same system as the files that are being accessed. A Service Gateway for Files running on a supported platform can communicate freely with a Data Object Broker on the same or another platform as long as the machines are attached through communications and they are properly configured.

## Optional Field for Service Gateway for Files

The table definitions for the IMP, EXP, and VSM tables contain an optional field, Server ID. If you specify a value in this field, a Service Gateway for Files will be used for the data access. This field also identifies which Service Gateway for Files to be used.

Refer to Chapter 3, Managing IMP Data Definitions, on page 49, Chapter 6, Managing EXP Data Definitions, on page 95, and Chapter 8, Managing VSAM Data Definitions, on page 119 for details on defining the table types.

## Restrictions in Use

Location parameters are permitted in the table definition but they must ultimately resolve to your local Data Object Broker as your Service Gateway for Files is logged on there.

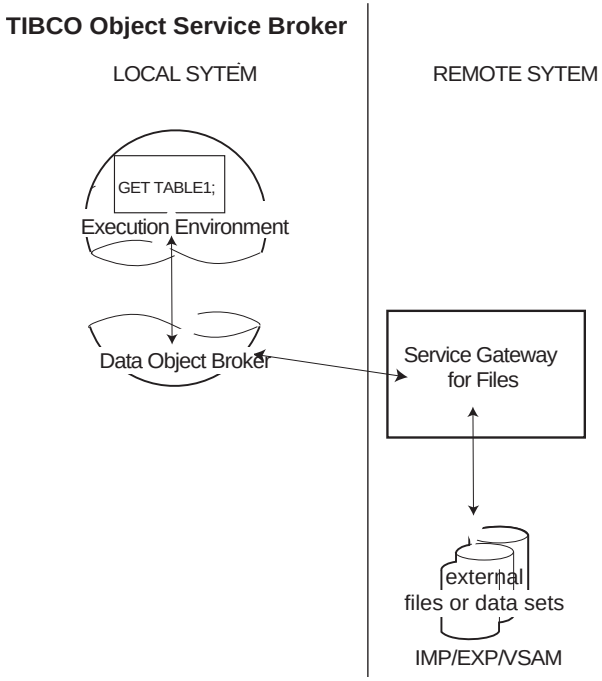
## Shares Infrastructure Used by Peer Servers

The Service Gateway for Files shares much of its infrastructure with Peer Servers and is counted among the Peer Servers by the Data Object Broker.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for details on configuring Peer Servers.

## Request Flow

The following graphic shows a request for access to a remote file using the Service Gateway for Files.



# Distribution Media and Contents

This section describes the software contents.

## Installation Media

You can download the software from the TIBCO site by following these steps:

1. Contact TIBCO Software Inc. for a password, directory information, etc.
2. Connect to the TIBCO site with the required information.
3. Download the appropriate ZIP file as described below.

For details on installation on z/OS, see Installing Remote on Separate Host, page 22.

## Open Systems Distribution

The following zip files comprise the distribution media:

`TIB_srvcgw_file_6.0.0_win.zip` (Windows)

`TIB_srvcgw_file_6.0.0_sol.zip` (Solaris)

## z/OS Distribution

This software is distributed in `.xml` format within a zip file. The file is in a format compatible with IBM System Modification Program/Extended (SMP/E) naming conventions. The product is packaged in SMP/E `txlib` and `lklib` format. The following ZIP file comprises the distribution media:

`TIB_srvcgw_file_6.0.0_zos.zip`

# Installing the Software on an Open Systems Host

## Installation Types

You can install the Service Gateway for Files as follows:

- **Standalone installation** — You have two configuration choices:
  - **Configuration 1: Remote on a separate host** — Install the software on a host in which TIBCO Object Service Broker for Open Systems is not present, that is, it is not local to the Data Object Broker.
  - **Configuration 2: Remote on the same host** — Install the software on the host in which TIBCO Object Service Broker for Open Systems is present, but the components do not share configuration files. This configuration would be suitable for implementations in which the Service Gateway for Files will communicate with a Data Object Broker other than the one on the local host.
- **Local (Enablement)** — Install the software as part of the TIBCO Object Service Broker system, which is locally installed in the machine.

When you run the installer, it detects the presence of TIBCO Object Service Broker for Open Systems if it is installed on the target host. At that time, you are given the choice to install the software with the base product (shared configuration files), or separately from the base product (components do not share configuration files).

## Installation Modes

The installers allow you to run in different modes:

- **GUI mode** — in this mode, the installer presents panels that allow you to make choices about product selection, product location, and so on. When you invoke the installer by double-clicking on the icon, GUI mode is used.
- **Console mode** — this mode allows you to run the installer from the command prompt or terminal window. This is used primarily for non-Windows environments.
- **Silent mode** — installs without prompting you for information. To use this mode, you must first generate a response file (using GUI mode or Console mode) that contains the input values you want to use for the installation. For details, see Installation Modes — Using a Response File on page 16.

## Preparation for Installation

### Prerequisite Software

To run the installers, you must first install JRE/JDK 1.5 or 1.6.

### Installer Disk Space Requirements

Windows

The package is extracted into a temp folder, typically `SystemDrive:\Temp` or `SystemDrive:\Documents and Settings\<user_name>\Local Settings\Temp`.

The installer requires 60 MB of free space in the temp directory.

Solaris

When a regular (non-root) user installs a TIBCO product, the installation registry is maintained in the user's home directory. As more products are installed, entries are added. The user's home directory must at least have 500 KB of free disk space.

### Installer Account

Windows

You must have administrator privileges for the machine on which the Service Gateway for Files is installed. If you do not have administrator privileges, the installer exits. You must then log out of the system and log in as a user with the required privileges, or request your system administrator to assign the privileges to your account.

If you intend to install the product on a network drive, you must ensure that the account used for installation has permission to access the network drive.

When installing on Windows terminal server, there are two modes: `Execute` and `Install`. By default all users are logged on in `Execute` mode, which allows them to run the applications. When you want to install the Service Gateway for Files for use by everyone, the Administrator should change to `Install` mode.

The best way to install the Service Gateway for Files is to use the Add/Remove Programs control panel applet, because this automatically sets the mode to `Install` during the installation and then back to `Execute` at the end.

Alternatively, you can manually change your mode to Install by typing:

```
C:\> change user /install
```

Change back to execute:

```
C:\> change user /execute
```

Check your current mode:

```
C:\> change user /query
```

If you install in the `Execute` mode, the installation registry is maintained in your user home directory. If you install in the `Install` mode, the installation registry is maintained in the *%SystemRoot%* folder.

You must invoke the installer from a TIBCO Object Service Broker-enabled CMD prompt or from a CMD shortcut created by the base installer, in which the TIBCO Object Service Broker environment has already been set.

Solaris

The Service Gateway for Files can be installed by a regular (non-root) user and super-user (root). Different users can install the same product at different locations.

## Installing the Service Gateway for Files

Before starting the installation procedure, ensure that your system meets the hardware and software requirements, and that you have reviewed the pre-installation steps.

### Installation Files

The following are the files for the installation. These files are contained in the zip files described in Open Systems Distribution on page 12:

`TIB_srvcgw_file_6.0.0_win_x86.exe` (Windows)

`TIB_srvcgw_file_6.0.0_sol.bin` (Solaris)

### Installing the Software

After starting the installer and accepting the license agreement, you are prompted to specify the *TIBCO_HOME* directory (if not detected by the installer). Typically, this is the top level installation directory for all TIBCO products; however, you can specify a different directory if desired.

Use one of the following modes to install the software.

**GUI Mode**

This mode allows you input values in panels. Type the following at the command prompt:

**Windows**

```
TIB_srvcgw_file_6.0.0_win_x86.exe
```

**Solaris**

```
./TIB_srvcgw_file_6.0.0_sol.bin
```

**Console Mode**

This mode allows you to install the software in a non-Windows environment. The installer will prompt you for values. Type the following at the command prompt:

**Windows**

```
TIB_srvcgw_file_6.0.0_win_x86.exe -console
```

**Solaris**

```
./TIB_srvcgw_file_6.0.0_sol.bin -console
```

If you wish to install using a response file, see Installation Modes — Using a Response File on page 16 for details.

## Installation Modes — Using a Response File

This section provides instructions on installation using a response file.

**Installing Using Silent Mode**

To use this mode, you must first generate a response file (using GUI mode or Console mode) that contains the input values you want to use for the installation.

You generate a response file using the following command:

<*installer*> `-options-record` <*filename*>

where <*installer*> is the installer executable (or bin file on Solaris), and <*filename*> is the name of the response file to be generated. For an example, see Installing and Generating a Response File below.

Example:

`TIB_srvcgw_file_6.0.0_win_x86.exe -silent -options` <*filename*>

**Installing and Generating a Response File**

You can generate a response file during installation which you can later use to invoke the installer with the selected values as default values (GUI mode) or as selected values (silent mode).

Example:

`TIB_srvcgw_file_6.0.0_win_x86.exe -options-record` <*filename*> (Windows)

**Installation in GUI or Console Mode Using a Response File**

You can use a previously generated response file for installation. For non-silent modes, the response file determines the defaults that are presented. For silent mode, the response file determines what will be installed.

Examples:

`TIB_srvcgw_file_6.0.0_win_x86.exe -options` <*filename*> (GUI Mode)

`TIB_srvcgw_file_6.0.0_win_x86.exe -console -options` <*filename*> `(Console Mode)`

**Combining Options**

You can combine the different available options. For example, to install using Console mode and generate a response file, use:

```
TIB_srvcgw_file_6.0.0_win_x86.exe -console
                                  -options-record <filename>
```

**Installation Choices**

You have two installation choices, enablement and standalone, which apply to all installation modes.

- **Enablement Installation** To make the enablement choice available for installation, you must have installed a TIBCO Object Service Broker base system. Call the installer program from a TIBCO Object Service Broker-enabled CMD prompt or from a CMD shortcut created by the base installer, in which the HURON environment variable is already set.

  When the installer prompts you to choose between enablement and standalone, choose enablement.

  The absence of a choice means that standalone installations already exist in the same *TIBCO_HOME* folder.

- **Standalone Installation** Standalone installation is the default in all other cases if no TIBCO Object Service Broker base system has been installed. This type of installation applies regardless of whether or not other standalone installations exist in the TIBCO_HOME folder.

  Be extra-careful if you use silent mode because the target installation environment plays a primary role in allowing the installation types that are available.

## Post Installation

After installing Service Gateway for Files, you must perform the tasks listed below. The directory names referred to in the following instructions are subdirectories of either OS_ROOT on Windows or ${OS_ROOT} on Solaris if you share configuration files with the base TIBCO Object Service Broker for Open Systems, or in the directory you specified during the install if you do not share configuration files.

**Windows**

Configure the gateway by creating configuration files in the database directory.

Refer to Configuring Service Gateway for Files on page 36 for detailed instructions.

You must modify the batch file `launchFilesgw.bat`, in the utils directory, to supply `osMon` with a `NAME` matching the `NAME` parameter value you choose for Service Gateway for Files in `mon.prm`.

**Solaris** Configure the gateway by creating configuration files in the database directory.

Refer to Configuring Service Gateway for Files on page 36 for detailed instructions.

You must modify the shell script `launchFilesgw`, in the utils directory, to supply `osMon` with a `NAME` matching the `NAME` parameter value you choose for Service Gateway for Files in `mon.prm`.

## Uninstalling the Software

If another product is dependent on the product you wish to uninstall, you are informed that you must uninstall the other product first. The directory names referred to in the following instructions are subdirectories of either `OS_ROOT` on Windows or `${OS_ROOT}` on Solaris if you share configuration files with the base TIBCO Object Service Broker for Open Systems, or in the directory you specified during the install if you do not share configuration files.

**Windows** Use one of the following to uninstall Service Gateway for Files:

- Use Add/Remove Programs from the Control Panel.
- Navigate to the following directory:

  ```
  uninstaller_archives\osbgateway_file\
  ```

  and invoke the `uninstall.exe` program.

**Solaris** Navigate to the following directory:

```
uninstaller_archives/osbgateway_file/
```

and invoke the uninstall program.

# Installing the Software on a z/OS Host

You can install the Service Gateway for Files as follows:

- Installing Remote on Same Host

  The software is installed on the host where TIBCO Object Service Broker for z/OS is present, but the components do not share configuration files. This would be suitable for implementations where the Service Gateway for Files will communicate with a Data Object Broker other than that on the local host.

  You must install and accept (through SMP/E) the TIBCO Object Service Broker base component before installing Service Gateway for Files. Also, you must have the <*HLQ*>.INSTALL data set that was created during that installation.

- Installing Remote on Separate Host

  The software is installed on a host where TIBCO Object Service Broker for z/OS is absent, that is, it is not local to the Data Object Broker.

## Installing Remote on Same Host

If you have acquired the Service Gateway for Files by downloading it from the TIBCO Software site, you must upload the software to the z/OS host system.

### Preparing and Uploading the Product File

1. Download or copy the `TIB_srvcgw_file_6.0.0_zos.zip` file to a PC that can connect to the z/OS host system.
2. Unzip the file to a temporary location on the PC. The zip file contains multiple files; of these, the following file is the only file used in this installation:

   `file.xm1` — compressed file containing Service Gateway for Files.

   The `srvcgw.file.xm1`, `install.bin`, `ostarrec.bin`, `property.bin`, and `OSTAREDC` files are not used in this procedure.
3. Pre-allocate the following sequential data set on the z/OS host system:

   <*HLQ*>.FILE.XM1 (size 18 KB)

Use the same <*HLQ*> that you specified when you uploaded the base component. Below is sample JCL to allocate this data set. Provide a JOB card and submit the JCL.

The value assigned to <HLQ> should be consistent during the entire installation cycle.

```
//ALLOC EXEC PGM=IEFBR14
//DD1 DD DSN=<HLQ>.FILE.XM1,
// DISP=(,CATLG,DELETE),UNIT=SYSDA,
// DCB=(RECFM=FB,LRECL=1024,BLKSIZE=0,DSORG=PS),
// SPACE=(TRK,(2,1))
```

4. FTP the `file.xm1` file in `BIN` mode to the <*HLQ*>`.FILE.XM1` data set.

## Installing the Software

This section describes the procedure for installing the Service Gateway for Files.

You must perform the installation under an ISPF environment only.

You can start the installation if you have the following data sets ready:

- <*HLQ*>**`.INSTALL`**
- <*HLQ*>`.FILE.XM1`

You must use the <*HLQ*>`.INSTALL` data set that was created during the installation of the TIBCO Object Service Broker base component.

To install Service Gateway for Files, perform the following:

1. Edit the properties file by specifying the keywords for installing this component.
2. Install the software.

These steps are described in depth below.

### Task A Edit the Properties File

Edit the `PROPERTY` member in <*HLQ*>`.INSTALL`. Table 3 describes keywords in the properties file for installing this component.

*Table 3 Properties File Keywords*

| Keyword | Description |
|---|---|
| `INSTALL=` | To install Service Gateway for Files, specify `FILES`: `INSTALL=FILES` |

*Table 3 Properties File Keywords*

| Keyword | Description |
|---|---|
| If enabling the CA-Datacom interface, provide the following: | |
| DCOMBASE= | Data set name of your CA Datacom base load library. |
| DCOMIPC= | Data set name of your CA IPC CAILIB load library. |

### Task B Initial Installation

1. Execute File Tailoring EXEC to start installation.
   - Member in: <*HLQ*>.INSTALL
   - Member: INSTALL (EX member)

   The FILE.JCL data set is created at the successful completion of this step.
2. Run Job FILE.JCL.

   This batch job uncompresses the FILE.XM1 file to produce the distribution library.
   - JCL in: <*HLQ*>.FILE.JCL (Edit JOB card to your site's standards)
   - Data Set: <*HLQ*>.FILE.JCL (SUB data set). Do not hold data set open in ISPF edit.

   Uncompressing <*HLQ*>.*FILE.XM1* produces the distribution library <*HLQ*>.FILE.FILEI.
3. Create and customize work copies of data sets.
   - Member in: <*HLQ*>.FILE.FILEI
   - Member: S6F1CUST (EX member)

   The following work copies are created and customized with values specified by OSEMOD variables:

   Customized copy - Library Description

   <HLQNONV>.<INSTVER>.JCL - Sample JCL

   <HLQNONV>.<INSTVER>.FILE.JOBS - Install jobs for Files
4. Initiate install jobs.
   - Member in: <HLQNONV>.<INSTVER>.FILE.JOBS
   - Member: S6F2RUNJ (EX member)

SEND messages are directed to the userid specified in the NOTIFY parameter of each job submitted, informing user of submission and normal completion or abnormal termination. The successful completion of the final job in JOBSF list is accompanied by the message ALL MEMBERS PROCESSED.

This completes the installation process for Files.

**See Also** *TIBCO Object Service Broker for z/OS Installing and Operating* for more information on installing the base component.

## Installing Remote on Separate Host

You can install Service Gateway for Files remotely on separate hosts, where TIBCO Object Service Broker for z/OS is absent, that is, it is *not local* to the Data Object Broker.

### Preparing the Product Files and Utilities for Uploading

1. Download or copy the TIB_srvcgw_file_6.0.0_zos.zip file to a PC that can connect to the z/OS host system.
2. Unzip the file to a temporary location on the PC. The file contains:
   - srvcgw_file.xm1 – A compressed file containing the installation libraries.
   - install.bin – The REXX EXEC to perform the installation.
   - ostarrec.bin – The REXX EXEC to uncompress the .xm1 files.
   - property.bin – A template of mandatory install variables required for product installation.
   - OSTAREDC – A decompression utility program.
3. Preallocate a PDS, fixed block data set on the z/OS host system:

   *HLQ*.INSTALL

   where *HLQ* is any valid high-level qualifier.

The value assigned to *HLQ* should be consistent during the entire installation cycle.

   See sample allocation JCL in Step 4.

4. Preallocate a sequential data set on the z/OS host system:

   *HLQ*.OS.FILE.XM1 (size 50,000 KB)

Use the same *HLQ* as the previous data set. Below is a sample JCL to allocate these data sets. Provide a JOB card and submit the JCL.

```
//ALLOC EXEC PGM=IEFBR14
//DD1 DD DSN=HLQ.INSTALL,
// DISP=(,CATLG,DELETE),UNIT=SYSDA,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
// SPACE=(TRK,(10,5,10))
//DD2 DD DSN=HLQ.OS.FILE.XM1,
// DISP=(,CATLG,DELETE),UNIT=SYSDA,
// DCB=(RECFM=FB,LRECL=1024,BLKSIZE=0,DSORG=PS),
// SPACE=(TRK,(1000,50))
```

5. FTP install.bin, ostarrec.bin and property.bin to your z/OS system in BIN mode to the *HLQ*.INSTALL data set. Name these utilities INSTALL, OSTARREC, and PROPERTY, respectively.
6. FTP the srvcgw_file.xm1 file in BIN mode to the *HLQ*.OS.FILE.XM1 data set.

### Installing the OSTAREDC Program

1. Upload the OSTAREDC file to z/OS in binary format to a data set with LRECL=80 and RECFM=FB.
2. In ISPF 3.4, type the following against this data set:

   "RECEIVE INDA(/)"

   When prompted, specify DA('*HLQ*.INSTLOAD' as the name of the load library where you want the OSTAREDC program restored.
3. Edit OSTARREC as follows:
   - — Issue the command FIND OSTAREDC 1.
   - — Change the constant after the equal sign to contain the full data set name of the program. The string must start with a double quote and a single quote, and end with a single quote and a double quote (the double quotes delimit the string and the single quotes tell TSO that the data set name is fully qualified). For example, change the following:

     OSTAREDC = "'*HLQ*.INSTLOAD(OSTAREDC)'"

     to

     OSTAREDC = "'*your.load.library*(OSTAREDC)'"

     where *your.load.library* is the name of the library referenced in Step 2.

## Installing the Software

You can start the installation if you have uploaded the following data sets as described in Preparing the Product Files and Utilities for Uploading on page 22:

- *HLQ*.INSTALL
- *HLQ*.OS.FILE.XM1

The *HLQ* referenced throughout this chapter is the high-level qualifier you specified when you uploaded the product software. This is the value of the INSTALL and XM1 files you specified. It will be used as the default value for all distribution files created when an XM1 is uncompressed. It is equivalent to the value of symbolic parameter $HLQ$ as described in OSEMOD.

### Overview

To install Service Gateway for Files, perform the following:

1. Determine your system environment values listed in System Environment Checklist.
2. Editing of the Properties File using the values determined in Step 1.
3. Installation.

### Task A System Environment Checklist

Before installation, review the system-environment information in Table 4 and determine whether to use the default value or provide your own value.

*Table 4 OSEMOD Variables*

| Description | OSEMOD Variable | Default Value | Your Value |
|---|---|---|---|
| High-level qualifier for uploaded data sets INSTALL and OS.FILE.XM1 | $HLQ$ | Specified on upload | |
| High-level qualifier for non-VSAM and VSAM data sets you are authorized to create | $HLQNONV$<br>$HLQVSAM$ | TIBCO.TESTNV<br>TIBCO.TESTVS | |
| Second-level qualifier for install files | $INSTVER$ | INS60 | |
| Second-level qualifier for TIBCO Service Gateway system files | $SLQ$ | OSB60 | |
| Second-level qualifier for SMP/E libraries | $SMP$ | SMP60 | |

*Table 4 OSEMOD Variables (Cont'd)*

| Description | OSEMOD Variable | Default Value | Your Value |
|---|---|---|---|
| For SMS Shops – managementclass, dataclass and storageclass, if required | | | |
| For new non-VSAM data sets | $NMGTCLAS<br>$NDATCLAS$<br>$NSTOCLAS$ | STANDARD<br>STANDARD<br>S6BNONV | |
| For new VSAM data sets | $VMGTCLAS<br>$VDATCLAS$<br>$VSTOCLAS$ | STANDARD<br>STANDARD<br>S6BVSAM | |
| High-level qualifier of Language Environment libraries for SCEELKED and SCEEBIND | $CEELIB$ | CEE | |
| High-level qualifier of IBM's Callable Services library CSSLIB | $CSSLIB$ | SYS1 | |
| If enabling the CA-Datacom interface provided with the Service Gateway for Files, provide the following: | | | |
| Data set name of your CA Datacom base load library | $DCOMBASE | CA.DATACOM.CABDLOAD | |
| Data set name of your CA IPC CAILIB load library | $DCOMIPC$ | CA.IPC.CAILIB | |
| Data Object Broker communication ID | $TDS$ | OSBDOB | |

For details, see the *TIBCO Object Service Broker for z/OS Installing and Operating* manual.

### Task B Editing of the Properties File

Use the PROPERTY member in *HLQ*.INSTALL as a template and modify it to suit your requirements. Table 5 describes the keywords in the properties file that correspond to the system-environment variables in System Environment Checklist.

*Table 5 Properties File Keywords*

| Keyword | Description |
|---|---|
| INSTALL= | To install Service Gateway for Files, specify REMOTEGATEWAY:<br>INSTALL=REMOTEGATEWAY |
| SERVICEGATEWAY= | To install Service Gateway for Files, specify FILES:<br>SERVICEGATEWAY=FILES |
| HLQNONV= | High-level qualifier for non-VSAM data sets. |
| HLQVSAM= | High-level qualifier for VSAM data sets. |
| INSTVER= | Second-level qualifier for install files. |
| SLQ= | Second-level qualifier for TIBCO Object Service Broker system files. |
| SMP= | Second-level qualifier for SMP/E libraries. |
| SMS= | YES for SMS site, NO for non-SMS site.<br>**Warning:** If you select the SMS=YES option, be sure to specify SMS-managed data-set names. The SMS automatic class selection (ACS) rules at your site determine whether a data-set name is eligible for SMS management. If the answer is yes, SMS manages that name. Otherwise, the result is unpredictable. |
| COMPAT= | Use if SMS=YES. Valid values: YES for SMS compatible data set name classes; NO for SMS non-compatible data set name classes.<br>If COMPAT=NO, specify the following:<br>• NMGTCLAS – MANAGEMENTCLASS for non-VSAM data sets<br>• NDATCLAS – DATACLASS for non-VSAM data sets<br>• NSTOCLAS – STORAGECLASS for non-VSAM data sets<br>• VMGTCLAS – MANAGEMENTCLASS for VSAM data sets<br>• VDATCLAS – DATACLASS for VSAM data sets<br>• VSTOCLAS – STORAGECLASS for VSAM data sets |

*Table 5 Properties File Keywords (Cont'd)*

| Keyword | Description |
|---|---|
| VOLSER= | If SMS=YES, specify one DASD volume for VSAM data set allocation. Default is USER01. If SMS=NO, specify three DASD volumes separated by commas. Defaults are OSBS06, OSBD18, OSBB02.<br>• vol1 – DASD volser for temp work files<br>• vol2 – DASD volser for install files<br>• vol3 – DASD volser for TIBCO Object Service Broker system files |
| CEELIB= | High-level qualifier of Language Environment libraries. |
| CSSLIB= | High-level qualifier of IBM's Callable Services library CSSLIB. |
| TDS= | Communication ID of the Data Object Broker. |
| If you are enabling the CA-Datacom interface provided with Service Gateway for Files, specify the following: | |
| DCOMBASE | Data set name of your CA Datacom base load library. |
| DCOMIPC | Data set name of your CA IPC CAILIB load library. |

### Task C Installation

To exit the interactive session at any time after executing REXX exec INSTALL, do the following:

1. Press PA1.
2. Enter hi.
3. Press ENTER twice.

**STEP 1: Execute File Tailoring EXEC to start installation.**

Member in: *HLQ*.INSTALL

Member: INSTALL (EX member)

STEP 1 will verify that files can be allocated successfully using the values provided in the PROPERTY file. Test files of type sequential, PDS, PDSE, and VSAM will be allocated then deleted. Installation will stop if any test allocation fails. You should investigate the cause, correct the condition and repeat STEP 1.

The *HLQ*.FILE.JCL data set is created at the successful completion of this step.

**STEP 2:** **Edit the Job card to your site's standards and run Job *HLQ*.FILE.JCL.**

JCL in: *HLQ*.FILE.JCL (Edit Job Card to your site's standards.)

Data Set: *HLQ*.FILE.JCL (SUB data set). Do not hold data set open in ISPF edit.

This batch job will uncompress the OS.FILE.XM1 file to produce the distribution libraries. If you modify the job name, make sure it does not exceed seven characters. The job should successfully complete with a return code of 0.

**STEP 3:** **Edit OSEMOD.**

To make additional changes to the values of the OSEMOD variables, make the changes now.

Member in: *HLQ*.FILECLS

Member: OSEMOD

If Service Gateway for Files is connecting to a Data Object Broker that runs in an Open Systems platform. you must specify the values for these two OSEMOD variables: $HOST$ and $DOBPORT$.

These variables customize the member RELAYTCP in HLQNONV.INSTVER.CNTL before it is copied to the data set HLQNONV.SLQ.RELAYCFG. It is required to establish TCP/IP connectivity. For further details, see Configuration of TCP/IP on page 30.

**STEP 4:** **Create and customize work copies of data sets.**

Member in: *HLQ*.OS.FILE.FILEI

Member: S6M1CUST (EX member)

The following work copies are created and customized with values specified by OSEMOD variables:

`Customized copy – Library Description`

- *HLQNONV*.*INSTVER*.CLIST – CLIST
- *HLQNONV*.*INSTVER*.CNTL – CNTL
- *HLQNONV*.*INSTVER*.JCL – Sample JCL
- *HLQNONV*.*INSTVER*.OS.FILE.JOBS – Install jobs for remote Service Gateway for Files

**STEP 5: Modify STATUS of installation jobs, as required.**

Member in: *HLQNONV*.*INSTVER*.OS.FILE.JOBS

Member: JOBSM (EDIT member)

Jobs in JOBSM are evaluated in the order they are listed and are submitted based upon their specified STATUS. The next job is submitted only if the previous one completed with its expected return code RC.

Valid status: INSTALL (run the job), FUTURE/OPTIONAL (skip the job), DONE (job already completed).

Status is modified from INSTALL to DONE only if the job's completion code is equal to or less than the stated return code.

You can modify the STATUS of any job as per your requirement. For example, if your shop normally ACCEPTs the product FMID at some future time, then change the status of S6M4ACPT from INSTALL to FUTURE.

Skip this step if the default STATUS of all the jobs is acceptable.

**STEP 6: Initiate install jobs.**

Member in: *HLQNONV*.*INSTVER*.OS.FILE.JOBS

Member: S6M2RUNJ (EX member)

SEND messages are directed to the userid specified in the NOTIFY parameter of each job submitted, informing user of submission and normal completion or abnormal termination. The successful completion of the final job in the JOBSM list is accompanied by the message ALL MEMBERS PROCESSED.

This completes the installation process.

**STEP 7:** **Perform the final ACCEPT.**

If you have opted to delay accepting the product as described in step 5 by changing the original STATUS of job S6M4ACPT from INSTALL to FUTURE in member JOBSM, then be sure to accept the Service Gateway for Files product by performing the final ACCEPT when ready. This step is required for future maintenance of the product.

Member in: *HLQNONV*.*INSTVER*.OS.FILE.JOBS

Member: S6M4ACPT (SUB member)

## Configuration of TCP/IP

You need TCP/IP to communicate between TIBCO Object Service Broker components—client processes, Execution Environments, Data Object Brokers, and external database gateways—on a z/OS system and TIBCO Object Service Broker components on z/OS or non-z/OS platforms.

The relay file, RELAYTCP in the CNTL data set, contains information on the TIBCO Object Service Broker components that use TCP/IP. The file associates TCP/IP host names and port numbers with the TIBCO Object Service Broker communication identifiers that are used by those components that run on any supported platform. Also, the file is a text file in XML format, which you must modify when changes to the TCP/IP environment are required. Each component could have a separate relay file or a common file could be shared across a number of components.

If XCF communications relay is deployed, TCP/IP parameters must be merged with the XCF parameters; the combined parameters are in RELAYCFG in the CNTL data set. For details on the XCF parameters, see Configuring XCF Communications in Chapter 2 of the *TIBCO Object Service Broker for z/OS Installing and Operating* manual.

The order of the relay parameters for each node name is the order of selection for that node. If merged with XCF parameters, XCF is considered before TCP/IP.

Run USERMODD in the JCL data set to customize the data set name of the relay file.

If you specify DSNAME=NULLFILE in USERMODD, you disable TCP/IP access. In jobs and started tasks where you want to use TCP/IP, add an S6BRELAY DD statement to point to the relay file. If you specify a non-null relay file in a batch job, it is likely to have a short-term region requirement at startup of over 64 MB as it runs XMLPARSER and might cause jobs to fail. USERMODD with DSNAME=NULLFILE or an S6BRELAY DD DUMMY JCL statement removes this storage requirement.

The installation process for TIBCO Object Service Broker copies RELAYTCP to the data set $HLQNOVN$.$SLQ$.RELAYCFG, which contains your live TCP/IP information. To change your TCP/IP configuration, use the CNTL member RELAYTCP to make and verify your changes, and then copy the new information to $HLQNOVN$.$SLQ$.RELAYCFG. To override the data set name set by USERMODD, add a DDNAME S6BRELAY to your TIBCO Object Service Broker component or any other z/OS components that require TCP/IP communications. If this override is invalid during the component's initialization, support for TCP/IP is disabled until you specify a valid parameter file. Once the relay file has been processed during component initialization, it is freed.

The relay file contains a set of protocol specific parameters, followed by a directory that maps communication identifiers to protocol-specific parameters. When configuring a TIBCO Service Gateway on z/OS to attach with a TIBCO Data Object Broker on Windows or Solaris, be sure to specify the keepalive attribute for the node that describes the TIBCO Data Object Broker (WINDOB in the sample below).

Following is a sample of the TCP/IP section of the HCS relay file:

```
<relay xmlns="http://www.tibco.com/OSB/relayparms.xsd">
    <tcpipparms tcbnum="3" maxtcbsockets="50" />
    <directory>
    <node name="PRODZDOB">
    <tcpip host="zos1.mydomain.com" service="emprec" />
    </node>
    <node name="PRODSDOB">
    <tcpip host="solaris5.mydomain.com" port="26360" />
    </node>
    <node name="FILEZGW">
    <tcpip host="zos1.mydomain.com" port="22636" />
    </node>
    <node name="WINDOB">
    <tcpip host="168.192.0.101" port="26362" keepalive="600"/>
    </node>
    </directory>
</relay>
```

For a detailed description of TCP/IP protocol parameters, see TCP/IP Protocol Parameters for the Relay File in Chapter 2 of the *TIBCO Object Service Broker for z/OS Installing and Operating* manual.

## Verification of Installation

Installation verification for an external DBMS provides a quick method to verify that the installation of a TIBCO Object Service Broker DOB and one or more DBMS Service Gateways was successful. This verifies that the communication between the DOB and a Service Gateway, and a Service Gateway and the DBMS, is functioning properly.

Verification can be accomplished by including sample TIBCO Object Service Broker table definitions for DB2, IMS, CA-IDMS and CA-Datacom. Each of these comes with sample tables or demo databases.

If a DBMS does not have sample tables or a demo database, or these were not included in its installation, you need to manually verify access to one of your databases. Instructions to perform this can be found later in this manual.

The TIBCO Object Service Broker tables are prefixed with the name `S6BIVP*`, for example `S6BIVP_DATACOM`.

### Requirements

Installation verification requires the following:

- The complete installation of a TIBCO Object Service Broker DOB on z/OS, Windows or UNIX.
- The installation of a Service Gateway.
- Sample tables or demo databases, often bundled with a DBMS and installed at most shops, are generally available for use.
- If the DOB is on Windows or UNIX, the `crparm` file must contain in the `OPERATOR` parameter the name of the TSO ID for submitting the `JCL(IVPDAT)` job. For example:

```
OPERATOR = winuser1, TSOID1
# Note:
# - winuser1 is a Windows login ID, from which the Windows DOB is
administered.
# - TSOID1 is a TSO ID, from which the JCL(IVPDAT) job is
submitted.
```

Data set *HLQNONV*`.`*INSTVER*`.JCL(IVPDAT)` contains all the job steps required to execute the verification process. If all requirements are met customize the JCL and run it.

Have your z/OS systems programmer APF-authorize the library *HLQNONV*.*INSTVER*.AUTH so that you can use TCP/IP or z/OS cross-memory communications in IVPDAT.

### Process

The steps in the verification process are as follows:

1. Submit JCL to start the Service Gateway for Files and wait for it to start.
2. Run a TIBCO Object Service Broker batch job to access the predefined DBMS sample table.
3. Shutdown the Service Gateway for Files.
4. Check the results.

## Reruns of the Installation

To do a rerun of the installation:

- Check to make sure you still have these two files:
  - *HLQ*.INSTALL
  - *HLQ*.OS.FILE.XM1
- Run the job *HLQNONV*.*INSTVER*.OS.FILE.JOBS(S6M9CLEN). It will delete all the data sets previously created by job S6M3APLY, i.e., SMP/E-related libraries, etc.
- Manually delete the customized copies of the installation files:
  1. *HLQNONV*.*INSTVER*.CLIST
  2. *HLQNONV*.*INSTVER*.CNTL
  3. *HLQNONV*.*INSTVER*.JCL
  4. *HLQNONV*.*INSTVER*.OS.FILE.JOBS
- Manually delete the source files created from the uncompress of XM1:

  There are 12 of them, with a high-level qualifier of *HLQ* ….
- Go back to Installing Remote on Separate Host on page 22 to do a rerun.

## Restarts of the Installation

To do a restart of the SMP/E installation, assuming that there are no data set name changes:

- Manually run these three jobs in *HLQNONV*.*INSTVER*.OS.FILE.JOBS in the following order:

Insert a

```
//PROCLIB  JCLLIB ORDER=HLQ.OS.FILE.FILEI
```

statement to the first two jobs in order to pick up SMPPROC.

1. S6M3APLY – Should complete with a return code of 4.
2. S6M4ACPT – Should complete with a return code of 4.
3. S6M5CFGR – Should complete with a return code of 0.

To change the second-level qualifier of SMP/E libraries from its default of SMP60 to one of your choice, do the following:

- Run the job *HLQNONV*.*INSTVER*.OS.FILE.JOBS(S6M9CLEN) to delete all previous SMP/E libraries.
- Edit the members S6M3APLY, S6M4ACPT, S6M5CFGR, and S6M9CLEN in *HLQNONV*.*INSTVER*.OS.FILE.JOBS, changing all occurrences of SMP60 to that of your choice and then manually run these jobs:

Insert a

```
//PROCLIB  JCLLIB ORDER=HLQ.OS.FILE.FILEI
```

statement to the first two jobs in order to pick up SMPPROC.

1. S6M3APLY – Should complete with a return code of 4.
2. S6M4ACPT – Should complete with a return code of 4.
3. S6M5CFGR – Should complete with a return code of 0.

# Uninstalling

To uninstall Service Gateway for Files:

1. Run the job *HLQNONV*.*INSTVER*.OS.FILE.JOBS (S6M9CLEN), which deletes all the data sets previously created by the job S6M3APLY, that is, the SMP/E-related libraries and so forth.
2. Manually delete the customized copies of the installation files, as follows:
   - *HLQNONV*.*INSTVER*.CLIST
   - *HLQNONV*.*INSTVER*.CNTL
   - *HLQNONV*.*INSTVER*.JCL
   - *HLQNONV*.*INSTVER*.OS.FILE.JOBS
3. Manually delete the 11 source files with the high-level qualifier of HLQ ….

   Those files were created from the uncompression of XM1.
4. Manually delete the rest of the installation files: HLQ.INSTALL and HLQ.OS.FILE.XM1.

# Configuring Service Gateway for Files

## Configuring a Service Gateway for Files to Run on Open Systems

To configure a Service Gateway for Files for Open Systems, you must edit the following parameter files:

- `mon.prm`
- `session.prm`
- `ee.prm`

The parameter files will be located in the database directory in either `%OS_ROOT` on Windows or `${OS_ROOT}` on Solaris if you share configuration files with the base TIBCO Object Service Broker for Open Systems, or in the directory you specified during the install if you do not share configuration files. Sample template files are provided in these directories (they have the extension `.template`).

If the parameter files already exist in your database directory, modify each file as required for installation. If the parameter files do not already exist, perform the following:

1. Make a copy of each template file.
2. Modify each copy as required for your installation.
3. Rename the files as appropriate to `mon.prm`, `session.prm`, or `ee.prm`.

See Also *TIBCO Object Service Broker Parameters* for details about the Execution Environment parameters.

### Sample Configuration

The following is a sample `mon.prm` file. This file is used by the osMon process which starts the Service Gateway for Files:

```
#mon.prm file: Sample Configuration
NAME=DEFAULT
SERVERS="2 FILEGATEWAYS"
DOB=S6LSDOBA
```

Explanations of the parameters in the sample `mon.prm` file:

| | |
|---|---|
| SERVERS=”2 FILEGATEWAYS” | Two Service Gateway for Files will be started. The name supplied must match a `NAME` parameter in your `session.prm` file. |

| | |
|---|---|
| DOB=S6LSDOBA | S6LSDOBA represents the Communications Identifier of the Data Object Broker your Service Gateway for Files will attach to. |

The following sample `session.prm` file shows the session parameters you must set.

```
#session.prm file: Sample Configuration
NAME=FILEGATEWAYS
EENAME=FGSRVEE
SERVERID=FGSERVER
SERVERTYPE=PRS
DSIXFTYPE=LINE_SEPARATED_ASCII
```

| | |
|---|---|
| NAME=FILEGATEWAYS | Matches the value specified in the `SERVERS` parameter in your `mon.prm` file. |
| EENAME=FGSRVEE | FGSRVEE represents an EE Name, which matches a `NAME` parameter in your `ee.prm` file. |
| SERVERID=FGSERVER | The `SERVERID` parameter supplies the server ID of the Service Gateway for Files. (The corresponding z/OS parameter is `PEERSERVERID`). |
| SERVERTYPE=PRS | Must appear as shown. |
| DSIXFTYPE=LINE_SEPARATED_ASCII | **Optional.** The `DSIXFTYPE` parameter controls, among other things, whether EBCDIC-to-ASCII conversions will be made on IMP and EXP tables. LINE_SEPARATED_ASCII is the default value and is typically the value you will need for remote access from Z/OS to ordinary flat files in an Open Systems environment. |

The following sample `ee.prm` file shows the necessary Execution Environment parameters.

```
#ee.prm: Sample Configuration
NAME=FGSRVEE
DOB=S6LSDOBA
MAXSESSION=20
```

| | |
|---|---|
| NAME=FGSRVEE | Matches the value specified in the `EENAME` parameter in your `session.prm` file. |

| | |
|---|---|
| DOB=S6LSDOBA | S6LSDOBA represents the Communications Identifier of the Data Object Broker your Service Gateway for Files will attach to. |
| MAXSESSION=20 | **Optional.** The MAXSESSION parameter controls how many concurrent sessions will be started before another EE will be started. |

## Configuring a Service Gateway for Files to Run on z/OS

This section shows you how to configure Service Gateway for Files on z/OS through the Execution Environment parameters supplied to the Service Gateway for Files in the JCL that starts it.

See Also *TIBCO Object Service Broker Parameters* for details about the Execution Environment parameters.

### Preliminary Steps

Before starting the Service Gateway for Files, do the following:

1. Configure the communications and target Data Object Broker. See Setting Up the Communications Infrastructure on page 40.
2. Have your z/OS systems programmer APF authorize the library *HLQNONV*.*INSTVER*.AUTH in order to use TCP/IP or z/OS cross memory communications. The S6BRELAY DD statement in the sample JCL configures a TCP/IP path to your remote Data Object Broker.
3. Run EECONFIG from *HLQNONV*.*INSTVER*.JCL. Before submitting job EECONFIG, be sure to specify a valid SVC number in the SVC parameter for members PARMNEE and PARMBAT, which are located in *HLQNONV*.*INSTVER*.CNTL.

### Sample Configuration

The following sample JCL starts a Service Gateway for Files named FGSERVER, and connects to a Data Object Broker with a COMMID of S6LSDOBA.

```
//S6LSQQPP JOB (1),'FILE GATEWAY',MSGCLASS=Y,TIME=1440
/*JOBPARM SYSAFF=*,TIME=1440
//REMSRVR EXEC PGM=S6BDR000,REGION=0M,TIME=1440
//STEPLIB DD DSN=HLQNONV.INSTVER.AUTH,DISP=SHR
//HRNEXTR DD DSN=HLQNONV.INSTVER.AUTH,DISP=SHR
//HRNIN DD *
PEERSERVERNUM=4,
TDS=S6LSDOBA,
PEERSERVERID=FGSERVER
```

```
/*
//*  TCP/IP definitions to communicate with remote DOB
//S6BRELAY DD DISP=SHR,DSN=HLQNONV.SLQ.RELAYCFG
/*
```

See *HLQNONV*.*INSTVER*.JCL(FILEGTWY).

Include the following Execution Environment parameters in your JCL (these are shown with the values supplied for the sample JCL):

| | |
|---|---|
| PEERSERVERNUM=4 | Specifies the number of instances of the Service Gateway for Files. Service Gateway for Files will occupy this number of inbound connections on the Data Object Broker. |
| MDL= | **Optional.** If not specified, defaults to OSB9999 where 9999 is the suffix that represents that a four-digit number, starting at 0001, is to be used by the identifiers as they are assigned. |
| TDS=S6LSDOBA | The communications identifier of the Data Object Broker. Corresponds to the node name in RELAYCFG when using TCP/IP communications. |
| PEERSERVERID=FGSERVER | This is mandatory for a Service Gateway for Files. If this is defaulted, Service Gateway for Files functionality will not be enabled. Note that the parameter PEERSERVERID specifies the server ID of a Service Gateway for Files on z/OS. In the Open Systems environment, the parameter to use is SERVERID. |

# Setting Up the Communications Infrastructure

You can set up communications between a Service Gateway for Files and a Data Object Broker.

## Prerequisites

The operating system of the Service Gateway for Files determines the communications prerequisites, as follows:

Before a Service Gateway for Files running on z/OS can communicate with a remote Data Object Broker, the name and IP port number of the Data Object Broker must be added to the RELAYCFG member in the CNTL data set for Service Gateway for Files.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for details about the RELAYCFG member.

## Data Object Broker on z/OS to Communicate with a Service Gateway for Files

Use ISPF EDIT to modify the contents of the sample TCP/IP Relay configuration data set, identified by the S6BRELAY DD statement in your Service Gateway for Files JCL. You need only modify the last sample entry as follows:

```
        <!--
           Definition for remote/localhost Windows DOB connectivity
            (e.g. for DB2 or ODBC gateway)
        -->
    <node name="WINDOB">
      <tcpip host="192.168.1.1" port="12000"
       keepalive="600" /> </node>
    </node>
    </directory>
</relay>
```

where:

| | |
|---|---|
| node name="..." | Identifies the node name of your remote Data Object Broker |
| tcpip host="..." | Replace with either the IP address or symbolic domain name of your remote Data Object Broker. |

| | |
|---|---|
| `port="..."` | The IP port assigned for exclusive use of your Data Object Broker. |

Ensure that you retain the double quotes around the assigned values.

## Definition of the Gateway as a Resource

A Resource Management entry is automatically created for the Service Gateway for Files when these conditions are satisfied:

- The Data Object Broker is on z/OS
- The Data Object Broker Parameter `DYNAMICRESOURCE = Y`
- There is no permanent resource in the Resource Management Facility that matches the requirements for the Service Gateway for Files

For best results, delete all permanent entries for Service Gateway for Files from the repository when dynamic resource creation is enabled.

If the Data Object Broker is on z/OS and Dynamic Resource Creation is not enabled (Data Object Broker Parameter `DYNAMICRESOURCE = N`) the Service Gateway for Files must be defined through the Resource Management facility available from the Administration menu. Resource Management is option 3 on the Administration menu. This option brings you to the Resource Type List, on which you must supply a TYPE of API and a GROUP corresponding to the server ID of your Service Gateway for Files. You must also have a schedule defined for the resource (a default schedule is provided with TIBCO Object Service Broker).

On the Resource Type Menu, PF5 allows you to configure a new Service Gateway for Files. PF2 can be used to view the configuration of an existing Service Gateway for Files. Once you are viewing such a configuration, PF11 allows you to edit it.

The configuration is substantially the same as for a Peer Server; the following values can be used. Other options can also be set on this menu.

| Field | Value | Notes |
|---|---|---|
| RESOURCE DETAIL FOR | API *name* | *name* is the server ID you wish to use for Service Gateway for Files. This is the same name as in `session.prm`. |
| INTERMEDIATE ROLLBACK | N | |

| Field | Value | Notes |
|---|---|---|
| EARLY RELEASE | Y | |
| COMMIT LEVEL | 0 | Service Gateway for Files provides update support for table types (EXP and VSM) that are always updated immediately rather than via commit processing. As a consequence, it must operate at Commit Level 0. |

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for details about the Resource Management facility.

# Management of Service Gateway for Files

This section describes how to start, shut down, or automatically restart Service Gateway for Files.

## Startup of a Service Gateway for Files

On z/OS, you start the Service Gateway for Files by submitting JCL as that described under Sample Configuration on page 38.

## Shutdown of a Service Gateway for Files

On z/OS, you shut down a Service Gateway for Files using the Data Object Broker operator command STOPSERVER. The following shows the format of the command, where ALLHURON and ALLAPI refer to servers that are either Service Gateway for Files or Peer Servers.

```
MODIFY dob_jobname,Stopserver=ALLHURON
MODIFY dob_jobname,Stopserver=ALLAPI
MODIFY dob_jobname,Stopserver=SRVIDserverid
MODIFY gateway_jobname,SHUTDOWN
```

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for more information about the STOPSERVER command.

Service Gateway for Files can be shut down using their server user ID. Server user IDs are auto-generated in the following format:

```
@EERRRXX
```

where:

| | |
|---|---|
| `@` | is the at sign, with which all server user IDs start, to differentiate servers from regular users. |
| `EE` | contains the first two characters of the name for the Execution Environment. |
| `RRR` | is the three-character string resulting from a hash on the Execution Environment name. |
| `XX` | XX contains the two alphanumeric characters (base 6) derived from the server number for the server in the Execution Environment. An Execution Environment associates a unique identifier for each Service Gateway for Files or Peer Server it starts. Alphanumeric characters are used rather than numbers to allow for more than 99 servers. Example: @SV2WU01. |

## Automatic Restart of a Service Gateway for Files

If a logical error is detected during Service Gateway for Files operation and the server terminates, the Execution Environment logs the error and restarts the server.

When a Service Gateway for Files terminates normally, the server session is not restarted.

# Monitoring of Service Gateway for Files

This section how to browse information on Service Gateway for Files.

## Administration

In the TIBCO Object Service Broker Administration menu, Service Gateway for Files are grouped with Peer Servers. Use the User Activity option (option I) to display Peer Server and Service Gateway for Files activity. Press PF2 to view a region list.

Select SERVER and press PF2 to display servers.

On Open Systems, you can also use the `rsview` utility to view, on standard out (`stdout`), a report about your Peer Server and Service Gateway for Files connections.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for details about the Administration menu.

## Log Files

On z/OS, corresponding messages will be written to the job log for the Data Object Broker.

## Service Gateway for Files and National Language Support

A Service Gateway for Files does not need to operate in the same code page as the Data Object Broker to which it attaches.

You can use the NLS shareable tool to configure this behavior. In the main menu of the NLS tool place the cursor on the field labeled Service Gateway for Files and press PF5 to edit or PF6 to add. On the displayed Table Editor screen, define the code page you intend to use, as follows:

| Field | Value | Notes |
|---|---|---|
| NAME | Server ID | Specifies the server ID of your Service Gateway for Files. |
| TYPE | S | Describes the value that appears in the field VALUE. |
| SYNTAX | C | Describes the value that appears in the field VALUE. |
| LENGTH | 31 | Describes the value that appears in the field VALUE. |
| DECIMAL | 0 | Describes the value that appears in the field VALUE. |
| VALUE | ENGL.IBM-273 | Specifies the code page you intend to use. |
| MODIFY | N | |

Note the following:

- If you do not supply values for LOCAL and REMOTE code pages in the NLS tool, National Language Support will not be enabled, and any code page you specify for the Service Gateway for Files will be ignored.
- A Service Gateway for Files running in the Open Systems environment will perform ascii-to-ebcdic mapping as well as mapping between EBCDIC code pages. The session parameter `DSIXFTYPE` controls this behavior, together with your choice of code pages. If your data appear garbled in an IMP or EXP table access in an Open Systems environment, you may need to adjust the value of this parameter.
- In the Open Systems environment, the following code pages can be specified for the Service Gateway for Files: IBM-037, IBM-273, IBM-277, IBM-278, IBM-280, IBM-282, IBM-284, IBM-285, IBM-297, IBM-500, IBM-1140, IBM-1141, IBM-1142, IBM-1143, IBM-1144, IBM-1145, IBM-1146, IBM-1147, and IBM-2248.

See Also *TIBCO Object Service Broker Shareable Tools* for detailed information about the NLS tool.

*TIBCO Object Service Broker National Language Support* for detailed information about TIBCO Object Service Broker NLS implementation.

*TIBCO Object Service Broker Parameters* for detailed information about the `DSIXFTYPE` parameter.

# Chapter 3 Managing IMP Data Definitions

This chapter describes how to manage IMP data definitions.

## Topics


- Accessing External Data from TIBCO Object Service Broker, page 50
- Task A: Identify the Table, page 52
- Task B: Identify the Data, page 54
- Task C: Specify Data and Location Parameters, page 57
- Task D: Specify Event Rules, page 60
- Task E: Define Fields, page 62

# Accessing External Data from TIBCO Object Service Broker

To read external data directly from TIBCO Object Service Broker, you must define a TIBCO Object Service Broker table of type IMP. An import table can have one or more fields, up to 16 fields in a composite primary key (to a total maximum length of 127 bytes), and optional data and location parameters. The table can also contain multiple record formats.

### Table Definer Screen for an Import Table

```
f
 COMMAND==>                          TABLE DEFINITION

       Table: EMPLOYEE_IMP     Type: IMP   Unit: USR40              IDgen: N

         File:
      DDname:             External Routine Name:
      Server ID:
    Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
   ----------------  -   -- --- --    -    -    '   ---------------- -   -
 _ LOCATION           I   C   16  0    L         ' _
 _                                               ' _
                    ------- IMP --------|--------- Metadata Definition ------
      Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len   Dec Ord Rqd Default
   ---------------- ---- -----  --   -----  -   -   -- ----- ---  -   -  ----------
 _
 _
 _
 _
 _
 _
 _
 _
 PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## Using a Copybook as the Source for the Definition

When you create a table in the TIBCO Object Service Broker UI, you can use a copybook as the source for its definition. You can then have TIBCO Object Service Broker monitor changes to the copybook. For more information, refer to Using Data Discovery on page 6.

## Steps Required to Define an Import Table

After invoking the Table Definer (refer to Initial Step for Defining Tables on page 5 for information on invoking the Table Definer), complete the following tasks to define an import table:

| Task | | Required | Refer to page |
|---|---|---|---|
| A | Identify the table. | Y | 52 |
| B | Identify the data. | Y | 54 |
| C | Specify data and location parameters. | N | 57 |
| D | Specify event rules. | N | 60 |
| E | Define fields. | Y | 62 |

# Task A: Identify the Table

This task is used to:

- Uniquely identify the table
- Verify the table type
- Identify the application or logical unit to which the table belongs
- Specify whether the system should generate unique values for the primary key field

### Table Identification Segment

The following example illustrates the fields used to identify the table:

```
Table: EMPLOYEE_IMP     Type: IMP   Unit: USR40              IDgen: N
```

## Table, Type, Unit, and IDgen Fields

The information for the **Table**, **Type**, **Unit**, and **IDgen** fields is entered by default. You can modify the **Table** and **Unit** fields, if necessary.

| | |
|---|---|
| **Table** | The table name displayed in the **Table** field is the one you specified when invoking the Table Definer. You can type in a new name to save the definition of an existing table under the new name. Refer to *TIBCO Object Service Broker Shareable Tools* for more information on the tools you can use to copy tables. |
| **Type** | The type indicates how data is stored in the table or how data is to be accessed from a table. This field displays IMP, which you changed in Initial Step for Defining Tables on page 5. |
| **Unit** | The unit marks the table as belonging to a particular application or logical unit such as utilities, accounting, or network control. |

| | |
|---|---|
| **IDgen** | The **IDgen** field determines whether values should be generated for the primary key field.<br><br>A value of Y means that the system generates the value for the primary key of each occurrence. Type Y if you are defining an import table for an external file that contains multiple record formats, multiple occurrences, or if you want a unique key for the import table.<br><br>If you specify Y in this field, you must also specify the following for the primary key:<br><br>Semantic Data Type – I<br><br>Syntax – B<br><br>Length – 4<br><br>Key – P |

# Task B: Identify the Data

You can use this task to specify the data you want to import and the name of external routines if you want to manipulate the data at its source before importing.

### Data Identification Segment

The following example illustrates the fields used to identify the data:

```
  File: USR40.EMPLOYEE.IMPORT
DDname:              External Routine Name:
Server ID:
```

## File Field

The **File** field contains the name of the data set or file that contains the data you want to import. The maximum record size is 31,744 bytes for all platforms. The record length for fixed-length files must be at least equal to the total length of the fields. The record length for variable-length files must be at least four more than the total length of the fields.

The **External Routine Name** field, which is valid only for the z/OS platform, contains the name of an external routine that you can use to manipulate data in the source file before importing it into TIBCO Object Service Broker. This routine is a load module resulting from an assembler program and resides in a data set that is concatenated to the external utilities data set. For more information, refer to External Routines (Pre-processing the Data) on page 90.

Notes on the File

- z/OS**:** Under CICS, you do not have to define the data set in a Destination Control Table (DCT) but the CICS region must have external security access to the data set.
- Open Systems**:** To specify the format of the data in external files to be processed by TIBCO Object Service Broker, use the DSIXFTYPE Execution Environment parameter. For more information on the DSIXFTYPE Execution Environment parameter, refer to *TIBCO Object Service Broker Parameters*.
- Open Systems**:** This filename maps to an entry in the filespec.dsn file. This file describes the absolute path name through which this file can be accessed. Refer to Appendix B, Mapping File Names for Open Systems, on page 161 for more information about this file.

- If you use FTP to transfer a variable-length import table file between z/OS and Windows or Solaris, you must reformat the file using the TIBCO Object Service Broker z/OS utility S6BBRFRU. If the import file is fixed length, you do not need to reformat the file.
- If you use FTP to transfer from z/OS to Windows or Solaris, the transfer must be in binary mode with the z/OS FTP LOCSITE subcommand with the NORDW parameter specified.
- For parameterized import tables other than those defining a file of multiple record formats, the File field must specify a partitioned data set or a directory. Neither of these have to exist before you define the import table, and must exist before you access the data. Parameter values you provide become the data set member names or the filenames in the directory.
- For non-parameterized import tables, you can import data from any file or member of a partitioned data set.
- If you are accessing multiple IMP tables from a single file, the File name in all the tables must be the same.
- If you specify both a File name and a DD name, the File name is used.

See Also *TIBCO Object Service Broker for z/OS Utilities* for more information on S6BBRFRU

## DDname Field

The **DDname** field points to the file that contains the data you want to import. The DDname can be specified if you want to import from an uncataloged data set (possibly tape), or if you want to change the import file without changing the table definition.

Notes on DDname

- z/OS**:** You must associate **DDname** with the data set that is the source of the data. To do this, use the DDNAME in your JCL or your TSO **ALLOCATE** command.
- z/OS**:** To import data from a partitioned data set, specify the member in the DD statement of your JCL or TSO **ALLOCATE** command.
- Open Systems**:** In the **DDname** field, specify either the name of the environment variable that contains the fully qualified name of the file, or the name of a DD definition in the filespec.dsn file.
- If you are accessing multiple import tables from a single file, the DDname in all the import tables must be the same.
- The import table cannot have data parameters except for those defining a file of multiple record formats.

- If you specify both a File name and a DD name, the File name is used.

## Server ID Field

The **Server ID** field points to the gateway that is to be used if the table is to be accessed remotely via the Service Gateway for Files. This is an optional specification. The value in this field is determined from the SERVERID Execution Environment parameter.

Refer to Monitoring of Service Gateway for Files on page 45 for details about the Server ID and the Service Gateway for Files.

See Also *TIBCO Object Service Broker Parameters* for information about the SERVERID parameter.

## Data Cleansing Field

Specify the data cleansing attributes as described in Appendix C (z/OS only).

# Task C: Specify Data and Location Parameters

You can use this optional task to specify two types of parameters:

- Data
- Location

### Parameter Segment

The following example illustrates the fields used to define data (DEPTNAME) and location (LOCATION) parameters. To view additional fields, use PF11.

```
    Parameter Name  Typ Syn Len Dec Class Src
   ----------------  -   -  --- --    -    -
 _ DEPTNAME          I   C    8  0    D
 _ LOCATION          I   C   16  0    L
```

## Data Parameter

You can use a data parameter to read data from a partitioned data set or a directory. The parameter value you provide becomes the data set member name or the filename in the directory.

### Data Parameter Validation

The data parameter is validated as follows:

| If the Syntax is | and the Length is | TIBCO Object Service Broker interprets |
|---|---|---|
| C | 8 | the parameter value as a member name of a partitioned data set (z/OS) or a file within a directory (Open Systems). |
| B | 4 | the parameter value as an indicator for multiple record types or repeating groups. Refer to Chapter 4, Defining Import Tables for Files with Multiple Record Formats, on page 67 for more information. |

- If IDgen=Y, you cannot use the data parameter to view different members of a partitioned data set (PDS).
- If IDgen=N, the data parameter must have a syntax of C and be no longer than 8 characters.

## Location Parameter

You use a location parameter to access external data through a peer server associated with another Data Object Broker (remote node). If you do not need to access remote data, use the **D** line command to delete the parameter. If you always access the external file remotely, the node from which you request the access can have either a minimal or full definition.

### Minimal Definition

A minimal definition with a location parameter means you always access data at a remote node. A minimal definition consists of the following:

- The table name, which must be the same at both locations
- The location parameter, which must be the same at both locations

  The location parameter indicates that you always access data at a remote node. The name of the remote node where the full definition is located must be supplied in the **Default** field, **Src** field, or **Src** and **Sourcename** fields.

The table type specified in a minimal definition does not have to match the table type of the full definition on the remote node.

### Full Definition

A full table definition with a location parameter means you can access data at either the local or the remote node. The table type of the full definition must match the data on the local node. For example, a full definition of type TDS used to access TDS data on the local node can also be used to access an import table with the same name on a remote node.

See Also *TIBCO Object Service Broker Managing Data* for more information on defining parameters.

# Task D: Specify Event Rules

Use this optional task to specify event rules to associate business rules and policies with the definition of a table. With these rules you can validate the data and automatically trigger other events based on specific retrieval access to import tables. The rules that you name here are run whenever data in the table is manipulated.

### Event Rule Segment

The following example illustrates the fields used to specify event rules:

```
'      Event Rule   Typ Acc
'   ---------------- -   -
'  _
'  _
```

## Event Rule, Typ, and Acc Fields

The rules that you enter here are run based on defined accesses. You can specify as many rules as you need in any logical order. The rules applying to specific accesses are executed in the order in which they are entered in **Event Rule** field.

| | |
|---|---|
| **Event Rule** | Specify the name of the event rule to be executed when the table is accessed. |
| **Typ** | Specify T (trigger rule) as the type of event rule that is to be accessed. Type V is not applicable for import tables. For a trigger rule there is no restriction on coding except for the following:<br>• The rule must not be a function.<br>• It cannot change the contents of the triggering row.<br>• It cannot use the TRANSFERCALL statement.<br>• Nested triggers are possible. |

| | |
|---|---|
| **Acc** | Specify G (any retrieval) as the type of data access. The type of access can invoke the event rule or specify the manipulation to be performed on the data causing the event rule to be executed. |

See Also *TIBCO Object Service Broker Managing Data* for information on event rules.

# Task E: Define Fields

This task is used to define the external import and internal TIBCO Object Service Broker attributes for the primary key fields and non-key fields of the table.

### Field Definition Segment

This example illustrates the fields used to define the fields of the import table:

```
                    -------- IMP --------|--------- Metadata Definition ------
     Field Name     Xsyn Xlen  Xdec Offset Key Typ Syn Len   Dec Ord Rqd Default
  ----------------- ---- -----  --   -----  -   -   -- ----- ---  -   -  ----------
_ EMPNO             C       5   0              I   C      5   0
_
_
_
_
```

When defining an import table, note the following:

- When defining fields, you can type in external import attributes and the TIBCO Object Service Broker attributes default to the external values, or vice versa.
- The number of fields you can access is dependent upon the Data Object Broker parameter CTABLESIZE. You can use the ESTIMATETBLDFN tool to estimate the size of this parameter.

## Specifying External Import Attributes

The following fields are used to specify the external import attributes. Use PF1 to see valid values for each field:

| | |
|---|---|
| **Field Name** | This field contains the name of the import field. It must be a unique name within the table. You can use the same name as a field in any other table; if you are moving data between this table and another table, giving fields the same names simplifies the process. |

| | |
|---|---|
| **Xsyn** | This field contains the external syntax for the import field. If not specified, the **Xsyn** field defaults to the specified TIBCO Object Service Broker syntax (**Syn** field). On Open Systems, numeric external syntaxes are treated as C or V for TEXT files. For information on external syntax, refer to Appendix A, Mapping Data Types, on page 153.<br><br>External syntax T is not supported for import tables. |
| **Xlen** | This field contains the external length for the import field. If not specified, the **Xlen** field defaults to the specified TIBCO Object Service Broker length (**Len** field). For information on external lengths, refer to Appendix A, Mapping Data Types, on page 153.<br><br>On Open Systems platforms, the **Xlen** field is ignored for TEXT files with a field separator character defined. |
| **Xdec** | This field indicates the external number of decimal places for the import field. If not specified, the **Xdec** field defaults to the specified TIBCO Object Service Broker decimal place (**Dec** field). |
| **Offset** | This field specifies the offset. The offset maps to the start of the external record in the import file. The origin is zero. Overlaps are allowed and you do not have to define fillers, since the offset can be used to skip undefined locations in the occurrence. You can specify offsets in one of three ways:<br><br>• Assign the offset if you know it.<br>• Use PF3 to save the definition; this calculates the offset.<br>• Use PF6 to calculate the offset based on the location of the cursor.<br><br>On the Open Systems platforms, the **Offset** field is ignored for TEXT files with a field separator character defined. |

## Specifying Internal TIBCO Object Service Broker Attributes

The following fields are used to specify the internal TIBCO Object Service Broker attributes. Use PF1 for valid values for each field. Use PF11 to view additional fields.

| | |
|---|---|
| **Key** | This field indicates whether the import fields are to be used in the primary key. You can select any field as the primary key using the **P** line command, without respect to uniqueness of data. You can select up to 16 contiguous fields for a composite primary key, to a maximum length of 127 bytes.<br><br>You must specify a primary key if you want to use the Table Browser or if you want to read multiple record types or repeating groups. |
| **Typ** | This field contains the TIBCO Object Service Broker semantic data type of the field. The default is null. You can specify any valid TIBCO Object Service Broker semantic data type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Syn** | This field contains the TIBCO Object Service Broker syntax of the field. You can specify any valid TIBCO Object Service Broker semantic data type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. If a syntax is not specified, the **Syn** field defaults to an appropriate syntax based on the external syntax and length (**Xsyn** and **Xlen** fields). For more information on external syntax, refer to Appendix A, Mapping Data Types, on page 153.<br><br>Syntaxes F, RD, and UN are not supported for a primary key field. |
| **Len** | This field indicates the length of the import field. The data is padded or truncated as necessary. If a length is not specified, the **Len** field defaults to an appropriate length based on the external syntax and length (**Xsyn** and **Xlen** fields). |

| | |
|---|---|
| **Dec** | This field specifies the number of digits to appear to the right of the decimal point. The data is padded or truncated as necessary. If not specified, the **Dec** field defaults to the specified external number of decimal places (**Xdec** field).<br><br>Depending on the syntax specified in the **Syn** field, define this field as follows:<br><br>• For syntax P, the number of decimal places must be smaller than twice the length of the entire field.<br><br>• For syntaxes B, C, F, RD, UN, and V, the number of decimal places must be 0. |
| **Ord** | This field indicates the order (ascending or descending) in which the occurrences in this field are sorted. The default value of null returns occurrences in ascending order by primary key. When an ordering option is explicitly specified, it takes precedence over the default. When ordering is specified for more than one field, the sort precedence is determined by the order of the fields as they are listed in the table.<br><br>Specifying a value in this field incurs sorting overhead, which can be significant in tables with a large number of occurrences.<br><br>Ordering is not permitted for fields of syntax F (float), RD (raw data) or UN (Unicode).<br><br>In Open Systems, if you have an import table with duplicate records and you want to sort, the sort results can be unpredictable since the internal sort does not maintain the input order of duplicate records. You must specify the ordering for each field to ensure you get the ordering you want. |
| **Rqd** | This field is ignored for an IMP table. |
| **Default** | This field is ignored for an IMP table. |
| **Globalfield Name** | This field displays the name of the global field if you used PF14 to select a field from the global field dictionary. |

See Also

*TIBCO Object Service Broker Shareable Tools* for information on the ESTIMATETBLDFN tool.

*TIBCO Object Service Broker Managing Data* for information on global fields.

*TIBCO Object Service Broker Parameters* for more information about the CTABLESIZE Data Object Broker parameter.

# Chapter 4 Defining Import Tables for Files with Multiple Record Formats

This chapter describes how to define import tables for files with multiple record formats.

## Topics


- How to Define Import Tables for Files with Multiple Record Formats, page 68
- First Sample File – Personnel, page 70
- Second Sample File – Inventory, page 72

# How to Define Import Tables for Files with Multiple Record Formats

## What is a Multiple Record Format?

An import table can reference or access a file of records that have different formats. For example, a personnel file contains records that record employee benefits, payroll information, and training. To access this type of file, you must define multiple, related tables for each record format.

Usually the first table defines the base portion of the record, which identifies subsequent record formats by a flag or record type indicator. To process the record, you GET the first base table and then determine the subsequent record format by examining the flag byte. The corresponding related table is then accessed for the particular record format following the base portion of the record.

## What is a Repeating Group?

A repeating group is a collection of data that occurs multiple times in a record. For example, a time collection system could have 52 repeating groups to record hours worked for each week of the year.

## Definition Requirements

To define an import table for the file shown in First Sample File – Personnel on page 70, define each table as described in Chapter 3, Managing IMP Data Definitions, on page 49, with the following additional considerations:

- Specify Y for the **IDgen** field.
- Specify the same filename or DD name for all the import tables.
- If the file is partitioned, specify the fully qualified filename in the **File** field, for example, `File: USER40.EMPL(ALLMGRS)`. You cannot use a parameter to specify members or files for these types of input files.
- Define a parameter for the import table that describes the second record format. The import table describing the first record format does not have to have a parameter. The parameter must be defined with the following:f

| | |
|---|---|
| Semantic Data Type. | I |
| Syntax. | B |
| Length. | 4 |

- The primary key field of each table must be defined as an **IDgen** field, as follows:

| | |
|---|---|
| Semantic Data Type. | I |
| Syntax. | B |
| Length. | 4 |

Do not provide data for this field; the system generates these values. As a result, the **IDgen** field provides each record in the import table with a unique identifier.

# First Sample File – Personnel

## Types of Records in File

This example illustrates a sample file that has two record types. Each record contains sections of different formats (A and B), including a repeating group (B).

| | | | | | |
|---|---|---|---|---|---|
| A | B | B | B | B | B |
| A | B | | | | |
| A | B | B | B | | |

## Table Definitions

### Sample File Parent Table Definition for Record A

The following example illustrates an import table definition for record A (parent):

```
COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_IMP_A   Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.EMPLOYEE.IMPORT
     DDname:              External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_                                               ' _
_                                               ' _
                  -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ DEPTNO           B       4   0      0      C   B     4   0
_ DEPTNAME         C       9   0      4      S   C     9   0
_
_
_
```

```
_ PFKEYS:3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

To use the Table Browser or read multiple record types or repeating groups, you must specify a primary key. This example does not specify a primary key; therefore, to access the information, you must use rules.

### Sample File Table Definition for Record B

The following example illustrates an import table definition for record B (child):

```
COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_IMP_B   Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.EMPLOYEE.IMPORT
     DDname:             External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ RECORD_A           I   B    4  0    D          ' _
_                                                _
                   -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ KEY              B       4   0      0   P  I   B     4   0
_ LNAME            C      22   0      4      S   C    22   0
_ FNAME            C      10   0     26      S   C    10   0
_
_
_
_

_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

# Second Sample File – Inventory

## Types of Records in File

This Inventory System sample has a transaction file with several record types. Each record is of a specific format (A to D):

```
----+----1----+----2----+----3----+----4----+----5----+----6----+---
AWIDGET VENDOR                  WIDGET-001
AGLUE MAKER                     ADHESIVE-001
ABOARD MAKER                    BOARD-001
B0000100WIDGET-001
B0000010ADHESIVE-001
B0000500BOARD-001
C0000005WIDGET-001                      BUILD-001
C0000002ADHESIVE-001                    BUILD-001
C0000005BOARD-001                       BUILD-001
D0000005PRODUCT-001
```

The record types are:

| | |
|---|---|
| **A** | Addition of new supplier or part to inventory file |
| **B** | Receipt of parts into inventory |
| **C** | Withdrawal of parts from inventory |
| **D** | Record of assemblies built and stored in inventor |

## Table Definitions

### Parent Table

The following example illustrates an import table definition for the header common to all the inventory records:

```
COMMAND==>                          TABLE DEFINITION

      Table: INVENTORY        Type: IMP   Unit: USR40             IDgen: Y

       File: USR40.INVENTORY.IMPORT
     DDname:             External Routine Name:
     ServerID:
```

```
   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
   --------------  -   -- --- --    -     -     '   ---------------- -   -
_ LOCATION          I   C   16  0     L          ' _
_                                                ' _
                  -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
   --------------- ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ GENKEY           B       4   0      0   P  I   B     4   0
_ RCDTYPE          C       1   0      4      S   C     1   0
_
_
_
_
_ PFKEYS:3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

### Record Type A

The following example illustrates an import table definition for record type A:

```
COMMAND==>                          TABLE DEFINITION

      Table: INV_SUPPLIER     Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.INVENTORY.IMPORT
     DDname:             External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
   --------------  -   -- --- --    -     -     '   ---------------- -   -
_ GENKEY            I   B    4  0     D          ' _
_ LOCATION          I   C   16  0     L          ' _
                  -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
   --------------- ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ INV_SUP_KEY      B       4   0      0   P  I   B     4   0
_ SUPPLIER_NAME    C      30   0      4      S   C    30   0
_ PART_NAME        C      30   0     34      S   C    30   0
_
_
_
_
_
 PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

### Record Type B

The following example illustrates an import table definition for record type B:

```
COMMAND==>                          TABLE DEFINITION

      Table: INV_REC_PARTS    Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.INVENTORY.IMPORT
     DDname:             External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ GENKEY            I   B    4  0    D         ' _
_ LOCATION          I   C   16  0    L         ' _
                   -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ INV_REC_PART_KEY B       4   0      0   P  I   B     4   0
_ QTY              N       7   0      4      C   B     4   0
_ PART             C      30   0     11      S   C    30   0
_
_
_
_

_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

### Record Type C

The following example illustrates an import table definition for record type C:

```
COMMAND==>                          TABLE DEFINITION

      Table: INV_WD_PARTS     Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.INVENTORY.IMPORT
     DDname:              External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule    Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ GENKEY            I   B    4  0    D         ' _
_ LOCATION          I   C   16  0    L         ' _
                   -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ----------------  ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ INV_WD_PART_KEY   B       4   0      0   P  I   B     4   0
_ QTY_OUT           N       7   0      4      C   B     4   0
_ PART_OUT          C      30   0     11      S   C    30   0
_ COMPONENT_NAME    C      30   0     41      S   C    30   0
_
_
_

_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

### Record Type D

The following example illustrates an import table definition for record type D:

```
COMMAND==>                          TABLE DEFINITION

      Table: INV_ASSBLY       Type: IMP   Unit: USR40              IDgen: Y

       File: USR40.INVENTORY.IMPORT
     DDname:              External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule    Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ GENKEY            I   B    4  0    D         ' _
_ LOCATION          I   C   16  0    L         ' _
                   -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ----------------  ---- ----  --  -----   -  -   -- ----  --  -   -  ----------
_ INV_ASSBLY_KEY    B       4   0      0   P  I   B     4   0
_ QTY_ASSBLY        N       7   0      4      C   B     4   0
_ PART_ASSBLY       C      30   0     11      S   C    30   0
_
_
```

```
_
_
_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## Processing

### Programming Considerations

- For the parent record, which defines the header, the table definition contains IDgen = Y, no parameters, and a B4 field at the beginning for the key.
- For the child records, the table definition contains IDgen = Y, a parameter of the parent primary key, and a B4 field at the beginning for the key.
- The child record table definitions do not include the common header accounted for by the parent table definition.
- The offsets in both the parent and child record table definitions start at 0 for the IDgen key and continue at 4 for the first field from the external file.

### Sample Rules

```
    RULE EDITOR ===>                                                SCROLL: P
READ_PARENT;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL INVENTORY :                                          | 1
_    CALL INVENTORY_PROC(INVENTORY.RCDTYPE, INVENTORY.GENKEY);|
_    END;                                                     |
_                                                             |
_ ------------------------------------------------------------------------------
```

This is a portion of the called rule, which calls other rules for more detail processing:

```
    RULE EDITOR ===>                                                SCROLL: P
INVENTORY_PROC (RCD, KEY);
_
_ ------------------------------------------------------------------------------
_ RCD = 'A' :                                                 | Y N N N N
_ RCD = 'B' :                                                 |   Y N N N
_ RCD = 'C' :                                                 |     Y N N
_ RCD = 'D' :                                                 |       Y N
```

```
_ ------------------------------------------------------------------+--------------
_ CALL MSGLOG('Record type ' || QUOTE(RCD));                        | 1 1 1 1 1
_ GET INV_SUPPLIER (KEY);                                           | 2
_ CALL MSGLOG('Supp: ' || QUOTE(INV_SUPLIER.SUPPLIER_NAME));        | 3
_ CALL MSGLOG('Part: ' || QUOTE(INV_SUPLIER.PART_NAME));            | 4
_ CALL INV_SUPP_PART_PROC;                                          | 5
_ GET INV_REC_PARTS (KEY);                                          |   2
_ CALL MSGLOG('Qty: ' || QUOTE(INV_REC_PARTS.QTY));                 |   3
_ CALL MSGLOG('Part: ' || QUOTE(INV_REC_PARTS.PART));               |   4
_ CALL INV_REC_PARTS_PROC;                                          |   5
_ GET INV_WD_PARTS (KEY);                                           |     2
_ CALL MSGLOG('Qty: ' || QUOTE(INV_WD_PARTS.QTY_OUT));              |     3
_ CALL MSGLOG('Part: ' || QUOTE(INV_WD_PARTS.PART_OUT));            |     4
_ CALL MSGLOG('Comp: ' || QUOTE(INV_WD_PARTS.COMPONENT_NAME));      |     5
_ CALL INV_WD_PARTS_PROC;                                           |     6
_ GET INV_ASSBLY (KEY);                                             |       2
_ CALL MSGLOG('Qty: ' || QUOTE(INV_ASSBLY.QTY_ASSBLY));             |       3
_ CALL MSGLOG('Assembly: ' || QUOTE(INV_ASSBLY.PART_ASSBLY));       |       4
_ CALL INV_ASSBLY_PROC;                                             |       5
_ SIGNAL UNEXPECTED_REC;                                            |         2
_ ------------------------------------------------------------------------------
_

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE
```

This is the output from this rule with the file shown under Types of Records in File on page 72:

```
-------------------------------------INFORMATION LOG -------------------------
COMMAND ===>                                                  SCROLL ===> P
Record Type 'A'
Supp.:'WIDGET VENDOR'
Part:'WIDGET-001'
Record Type 'A'
Supp.:'GLUE MAKER'
Part:'ADHESIVE-001'
Record Type 'A'
Supp:'BOARD MAKER'
Part:'BOARD-001'
Record Type 'B'
Qty:'0000100'
Part:'WIDGET-001'
Record Type 'B'
Qty:'0000010'
Part:'ADHESIVE-001'
Record Type 'B'
Qty:'0000500'
Part:'BOARD-001'
Record Type 'C'
Qty:'0000005'
Part:'WIDGET-001'
Comp:'BUILD-001'
Record Type 'C'
```

```
Qty:'0000002'
Part:'ADHESIVE-001'
Comp:'BUILD-001'
Record Type 'C'
Qty:'0000005'
Part:'BOARD-001'
Comp:'BUILD-001'
Record Type 'D'
Qty:'0000005'
Assembly 'PRODUCT-001'

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT FIND 12=EXIT 13=PRINT 9=RECALL
```

# Chapter 5 Manipulating Import Data Using TIBCO Object Service Broker

This chapter describes how to manipulate import data using TIBCO Object Service Broker.

## Topics


- Accessing Import Tables, page 80
- Sample Rules, page 83
- Listing Members of a PDS (z/OS) or Files in a Directory (Open Systems), page 85
- Handling TIBCO Object Service Broker Requests, page 89
- External Routines (Pre-processing the Data), page 90

# Accessing Import Tables

When you access external data from TIBCO Object Service Broker, import field data types are translated to the field types defined in the IMP table. You can access the data using:

- Table Browser
- Rules

In the Open Systems versions of TIBCO Object Service Broker, use the DSBIFTYPE Execution Environment parameter to set the file type to LENGTH_PREFIXED_EBCDIC type files when migrating data from one TIBCO Object Service Broker system to another.

## Using the Table Browser

You can browse an import table in the same way you would browse any other table with the following exceptions:

- The table definition *must* have one to eight primary keys, but the data in the table does not have to result in unique values in the key fields for each row.
- If your table definition contains fields of syntax C or V that are longer than 260 bytes, or fields of syntax RD or UN that are longer than 130 bytes, you must use **`SELECT LIKE`** instead of **`SELECT`** to access fields of this length.
- You cannot use the Table Browser to access multiple record formats of an import file.

## Using Rules

- Accessing import data using the rules language is similar to accessing native TIBCO Object Service Broker data. You can access external data online or in batch mode using tools such as COPY_DATA to copy external data to a TIBCO Object Service Broker table that you can update.

### Import Files with Multiple Record Formats

Because multiple record formats for import tables are implemented by accessing parent and child tables, workbench tools such as the Table Browser cannot be used. Use either rules or the TIBCO Object Service Broker Host Language Interface. For a sample rule, refer to Sample Rules on page 83.

## Considerations

When using rules to access import tables, note the following:

- Each transaction stream accessing external data requires its own server thread. Ensure your system administrator is aware of the number of server threads required to accommodate all transaction streams accessing external data in a single transaction.
- If you use the default CTABLESIZE Data Object Broker parameter value, you can access at least 16 import tables per transaction; more, depending on the size of the import table definitions, since the more fields you define, the more space is required to hold the definition in the memory in the Data Object Broker and the Execution Environment.

## Retrieval Processing

A single cursor is used to retrieve import data for the following retrieval statements in your rule:

- GET
- FORALL

Import files with multiple record formats cannot be sorted (ORDERED clause) or selected (WHERE clause) by field values.

### GET Statement

A GET statement to an import table causes the search for the first occurrence that satisfies the specified selection criteria to be started at the beginning of the import table.

A GET … ORDERED statement must retrieve all import data that satisfies the selection criteria and sort it in the Execution Environment before returning the first occurrence that meets the selection criteria.

### FORALL Statement

When using a FORALL statement, occurrences are returned to TIBCO Object Service Broker in the order in which the import server passes them. If you require a different order, you must include an ORDERED clause in your FORALL statement. TIBCO Object Service Broker orders only occurrences specified in the selection criteria.

## Remote Table Access

Remote table access can cause a significant increase in message traffic. This is especially true with import tables that have the **IDgen** field set to Y, which is a normal requirement for handling data in multiple record import tables.

## Steps to Process a Multi-record Table Remotely

To overcome remote table access limitations when processing a multi-record table, complete the following steps:

1. Define the import table with IDgen = N.
2. Access the import table remotely. Refer to the example rule that follows for an example of accessing an import table remotely.
3. Process the nested records locally using MAP tables.

   For more information on MAP tables, refer to *TIBCO Object Service Broker Managing Data*.

### Example Rule

```
    RULE EDITOR ===>                                                SCROLL: P
PEER_ACCESS1;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL REMOTE_TABLE :                                       | 1
_    LOCAL_TABLE.* = REMOTE_TABLE.*;                          |
_    INSERT LOCAL_TABLE;                                      |
_    END;                                                     |
_ ------------------------------------------------------------------------------
```

See Also

- *TIBCO Object Service Broker Parameters* for more information about the DSBIFTYPE Execution Environment parameter and the CTABLESIZE Data Object Broker parameter.
- *TIBCO Object Service Broker Managing Data* for more information on browsing tables.
- *TIBCO Object Service Broker for z/OS External Environments* for more information about the Host Language Interface.
- *TIBCO Object Service Broker Programming in Rules* for more information on transactions and writing rules.
- *TIBCO Object Service Broker Shareable Tools* for more information on the COPY_DATA tool.

# Sample Rules

## Sample Rule 1: Normal Access

The following example illustrates a sample rule for normal access to import data.

```
     RULE EDITOR ===>                                                SCROLL: P
EMPLOYEE_A;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL EMPLOYEE_A :                                         | 1
_    DEPARTMENT.DEPTNAME = EMPLOYEE_IMP.DEPTNAME;             |
_    INSERT DEPARTMENT;                                       |
_    END;                                                     |
_ ------------------------------------------------------------------------------
```

```
PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE
```

## Sample Rule 2: Accessing Multiple Record Formats

The following example illustrates a sample rule for accessing multiple record formats. The tables in this rule are the ones defined in Table Definitions on page 72.

```
    RULE EDITOR ===>                                                 SCROLL: P
EMPLOYEE_B;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL EMPLOYEE_IMP_A :                                     | 1
_    FORALL EMPLOYEE_IMP_B(EMPLOYEE_IMP_A.KEY) :              |
_    CALL MSGLOG(EMPLOYEE_IMP_B.LNAME);                       |
_       END;                                                  |
_    END;                                                     |
_ ------------------------------------------------------------------------------
```

```
PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE
```

## Explanation of the EMPLOYEE_B Rule

To access multi-record formats, a parent import table and one or more child import tables must be defined. In the EMPLOYEE_B rule definition, you need two tables: one for the parent (EMPLOYEE_IMP_A) and a second for the child (EMPLOYEE_IMP_B).

The parameter value of the EMPLOYEE_IMP_B table ensures that all the record formats for this file are accessed from a single physical record.

To access the record B data, you must first access the parent record A data. This causes the import record to be read into a buffer and an internal cursor set to determine the positioning of data being accessed within the buffer.

In this rule, the `FORALL EMPLOYEE_IMP_A:` causes the first record to be read and buffered. The data contained within the format A record is returned to the rule and the cursor set to the end of the record data A area (in this case the first record B data portion).

The second `FORALL EMPLOYEE_IMP_B(EMPLOYEE_IMP_A.KEY):` then returns the first format B data and moves the cursor to the next piece of data (in this case, the second record format B data). Employee last names are written into the message log. This continues until the data within the buffer is exhausted. At this point, the second FORALL terminates and the next parent record A is retrieved by the first FORALL statement.

The parent import occurrence must always be accessed prior to any child table. If the child import table is accessed first, the GET or FORALL operation fails as no current occurrence is buffered and the internal cursor is undefined.

# Listing Members of a PDS (z/OS) or Files in a Directory (Open Systems)

To list the names of the members of a partitioned data set (PDS) or the names of the files in a directory, you define:

1. An import (IMP) table that references the data set or directory in the FILE parameter
2. A parameter value (PRM) table with a source of the IMP table previously defined

## Example for a PDS in z/OS

**To list the member names of a PDS called USR40.EXAMPLE.PDS:**

1. Define an IMP table as follows:

```
COMMAND==>                          TABLE DEFINITION

      Table: PDS_EXAMPLE       Type: IMP   Unit: USR40              IDgen: N

       File: USR40.EXAMPLE.PDS
     DDname:             External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ MEMBER            I   C    8  0    D          ' _
_ LOCATION          I   C   16  0    L          ' _
                   -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len  Dec Ord Rqd Default
  --------------- ---- -----  --   -----  -   -   -- ----- ---  -   -  --------
_ KEY              C       8   0       0   P  S   C      8   0
_ DATA             C      64   0       8      S   C     64   0
_
_
_
_
_

_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

2. Define a PRM table as follows:

```
 COMMAND==>                          TABLE DEFINITION

      Table: PDS_EXAMPLE_PRM   Type: PRM   Unit: USR40
     Source: PDS_EXAMPLE









PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC
```

3. Use rules to access the PRM table, or from the workbench, use the Table Browser to produce the PDS member name list.

```
BROWSING TABLE   :  PDS_EXAMPLE_PRM
COMMAND ==>
                                                              SCROLL: P
    NUMBER      MEMBER
_ ----------- --------
_           1 MEMBER01
_           2 MEMBER02
_           3 MEMBER03









PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 19=SHOW 13=PRINT 3=END 14=EXPAN
```

## Example for a Directory in Windows or Solaris

**To list the names of the files in a directory called D:\ObjectStar\database\JOURNAL:**

1. Define an IMP table as follows:

```
COMMAND==>                          TABLE DEFINITION

      Table: PDS_EXAMPLE       Type: IMP   Unit: USR40              IDgen: N

       File: D:\ObjectStar\database\JOURNAL
     DDname:             External Routine Name:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class Src    '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -    -     '   ---------------- -   -
_ MEMBER             I   C    8  0    D         ' _
_ LOCATION           I   C   16  0    L         ' _
                    -------- IMP --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- -----  --   -----  -   -   -- ----- ---  -   -  ----------
_ KEY              C       8   0       0   P  S   C     8   0
_ DATA             C      64   0       8      S   C    64   0
_
_
_
_
_

_ PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

2. Define a PRM table as follows:

```
 COMMAND==>                          TABLE DEFINITION

      Table: PDS_EXAMPLE_PRM   Type: PRM   Unit: USR40
     Source: PDS_EXAMPLE
```

```
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC
```

3. Use rules to access the PRM table, or from the workbench use the Table Browser to produce the PDS member name list.

```
BROWSING TABLE   :  PDS_EXAMPLE_PRM
COMMAND ==>
                                                                SCROLL: P
  NUMBER     MBMBER
_ ------ --------------
_       1 JOURNAL1
_       2 JOURNAL2
_
_
_
_
_
_
_
_
_
_
_
_
_
_
_
PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND
```

See Also *TIBCO Object Service Broker Managing Data* for more information on creating PRM tables and on browsing tables.

# Handling TIBCO Object Service Broker Requests

The TIBCO Object Service Broker runtime environment signals system exceptions to permit an application to recover from an error. A three-level hierarchy of exceptions exists. The ERROR exception is the top of the hierarchy and is intended to be a catchall exception. Each exception traps the exceptions appear below it in the hierarchy.

All errors encountered when accessing external data through the IMP tables are trapped under one of the following exceptions:

- ERROR
- ACCESSFAIL
- INTEGRITYFAIL

## ERROR Exception

An ERROR exception indicates that an error is detected and no lower-level exception exists in the application.

## ACCESSFAIL Exception

An ACCESSFAIL exception indicates that a table access error is detected. The only valid exception under an ACCESSFAIL exception is a GETFAIL, indicating that no occurrence satisfies the selection criteria.

## INTEGRITYFAIL Exception

An INTEGRITYFAIL exception indicates an attempt to violate data integrity is detected. The following exceptions are valid under a this exception:

| | |
|---|---|
| **DEFINITIONFAIL** | Indicates that the data set or file specified is not found |
| **LOCKFAIL** | Indicates that there is a lock on an occurrence or a table is unavailable |
| **SECURITYFAIL** | Indicates that permission for the requested action on the object is denied |

See Also *TIBCO Object Service Broker Programming in Rules* for more information on exceptions.

# External Routines (Pre-processing the Data)

## Manipulating Data with External Routines (z/OS Only)

When accessing an external data set through an import table in z/OS, you can write an external routine to manipulate the data. The external routine must be written in assembler and can:

- Pass the record to a destination (the import table)
- Cause the record to be skipped
- Alter the record before passing it to its destination
- Insert multiple records to TIBCO Object Service Broker based on one record from a data set

You can concatenate the library in the CLIST that you use to log in to TIBCO Object Service Broker or if you are importing the data in batch, you can concatenate the library in the batch JCL DD statement HRNEXTR.

## Parameters Passed to the External Routine

To write an external routine, use the IMPXPARM macro found in the MACRO data set distributed with TIBCO Object Service Broker to describe the DSECT of the parameters that are passed to the external routine.

| Parameter | Description |
|---|---|
| HRNXRLEN | Represents the record length. If the record length is changed by the external routine, the server uses the changed length for input or output. HRNXRLEN is set to zero upon entry to the import external routine when the end of the file is reached. You can use this to insert additional records into TIBCO Object Service Broker. |
| HRNXRECA | Points to the I/O record buffer. If the buffer pointer is changed by the external routine, the server uses the new buffer for input or output. Do not change the data in this buffer. |
| HRNXWRKA | A 4 KB work area that is preserved between calls to the external routine. It is cleared to X'00' at the start of the FORALL. |

| Parameter | Description |
|---|---|
| HRNXRECE | A 4 KB work area that is not preserved between calls to the external routine. It is also not cleared before the external routine call. It can be used as the work area where you compose the record to be inserted. |
| HRNXTABN | Points to the 16 character TIBCO Object Service Broker table name. |
| HRNXDSN | Points to the 44 character external data set name. |
| HRNXTPRM | Points to the 8 character table parameter value. |

## Valid Return Codes

At the end of the external routine, Register 15 contains a return code that indicates the action the server should take. The valid return codes whose description follows are located in the IMPXPARM macro.

| Return Code | Description |
|---|---|
| @HRNXKEP | Import the record. |
| @HRNXDEL | Do not import the record (equivalent to a logical DELETE). |
| @HRNXDON | Do not call the external routine again for this transaction. No more physical records are read or written and the external routine is not called again for this transaction. For an import table, end of file is indicated when HRNXRLEN is zero and you should return @HRNXDON to prevent its external routine from being called again. |
| @HRNXINX | A new record is to be inserted. The external routine must set HRNXRECA to the address of the new record and HRNXRLEN to the length of the new record. |

## Example of an Exit

The following is a simple example of an external routine. A file is being imported but only records having a length of 32 are imported; other records are ignored.

```
         COPY  EQUATES              STANDARD EQUATES.
         COPY  IMPXPARM             EXIT PARAMETERS.
IMPEXIT  CSECT
         SAVE  (14,12),,*           SAVE THEM ... STANDARD LINKAGE
         USING IMPEXIT,R12          R12 ...
         LR    R12,R15                       BASE REGISTER
         USING HRNXPARM,R1          ADDR CALLING PARMS.
         L     R2,HRNXRLEN          R2 = RECORD LENGTH
         LTR   R2,R2                IS IT A NORMAL RECORD?
         BNZ   NORMAL               YES
*-------------------------------------------------------------------
*        WE'VE BEEN CALLED AT THE END OF ALL NORMAL RECORDS.        *
*-------------------------------------------------------------------
         LA    R15,@HRNXDON         RC = DON'T CALL ANY MORE
         B     RETURN               RETURN
*-------------------------------------------------------------------
*        IGNORE INPUT RECORDS THAT HAVE LENGTHS NOT EQUAL TO 32.    *
*-------------------------------------------------------------------
NORMAL   LA    R15,@HRNXKEP         SET 'KEEP' RC.
         C     R2,=F'32'            GET VARIABLE STRING LENGTH
         BE    RETURN               RETURN.
         LA    R15,@HRNXDEL         SET 'IGNORE' RC.
RETURN   DS    0H
         RETURN (14,12),,RC=(15)
         LTORG
         END
```

## Example of an Exit that Normalizes Data

The following example illustrates a more complex example of an external routine. All records in the data set to be imported begin with a fixed portion but then have a repeating portion on the same record. The number of occurrences of the repeating portion is stored in two bytes after the fixed portion. For a table to describe the data presented to it, the external routine attaches the fixed portion to the beginning of each repeating portion in the same record and each becomes an occurrence in the table.

```
         COPY  EQUATES              STANDARD EQUATES.
         COPY  IMPXPARM             EXIT PARAMETERS.
         EJECT
WORKA    DSECT
VARPOINT DS    F
TIMEFLAG DS    CL1
OCCURS   DS    PL2
```

```
FIXEDDTA DS    CL20
VARDTA   DS    CL40
IMPEXIT  CSECT
         SAVE  (14,12),,*          SAVE THEM ... STANDARD LINKAGE
         USING IMPEXIT,R12         R12 ...
         LR    R12,R15                     BASE REGISTER
         USING HRNXPARM,R1         ADDR CALLING PARMS.
         L     R2,HRNXWRKA         R2 = WORK AREA
         USING WORKA,R2
         CLI   TIMEFLAG,X'01'      SECOND TIME FOR THIS RECORD?
         BE    SECOND              SECOND TIME FOR THIS RECORD
         L     R7,HRNXRLEN         R7 = RECORD LENGTH
         LTR   R7,R7               IS IT A NORMAL RECORD?
         BNZ   NORMAL              YES
*-------------------------------------------------------------------
*        WE'VE BEEN CALLED AT THE END OF ALL NORMAL RECORDS.        *
*-------------------------------------------------------------------
         LA    R15,@HRNXDON        RC = DON'T CALL ANY MORE
         B     RETURN              RETURN
*-------------------------------------------------------------------
*        BUILD FIRST OCCURRENCE FROM VARIABLE OCCURRENCE RECORD      *
*-------------------------------------------------------------------
NORMAL   LA    R15,@HRNXINS        SET 'INSERT' RC.
         L     R4,HRNXRECA         GET ADDRESS OF RECORD
         MVC   OCCURS,20(R4)       GET NUMBER OF VARIABLE OCCURS
         CP    OCCURS,=P'0'        COMPARE NUMBER OF OCCURS TO ZERO
         BE    IGNORE              IGNORE THIS RECORD
         SP    OCCURS,=P'1'        SUBTRACT ONE OCCURANCE FROM COUNTER
         MVI   TIMEFLAG,X'01'      SET FLAG FOR SECOND TIME PROCESS
         L     R8,=F'60'           R8 = NEW RECORD LENGTH
         ST    R8,HRNXRLEN         SAVE NEW RECORD LENGTH
         MVC   FIXEDDTA,0(R4)      MOVE FIXED PART OF RECORD
         LA    R4,22(R4)           INCREMENT R4 TO FIRST OCCUR
         MVC   VARDTA,0(R4)        FIRST VARIABLE PART OF RECORD


         ST    R4,VARPOINT         SAVE THE VARIABLE POINTER
         LA    R8,FIXEDDTA         GET THE ADDRESS OF THE FIXED PART
         ST    R8,HRNXRECA         SAVE POINTER TO RECORD
         B     RETURN
*-------------------------------------------------------------------
*        BUILD SUBSEQUENT OCCURRENCE FROM VARIABLE OCCURRENCE RECORD *
*-------------------------------------------------------------------
SECOND   CP    OCCURS,=P'0'        COMPARE REMAINING OCCURS TO ZERO
         BE    IGNORE              YES IGNORE IT.
         SP    OCCURS,=P'1'        SUBTRACT ONE OCCURANCE FROM COUNTER
         LA    R15,@HRNXINS        SET 'INSERT' RC.
         L     R8,=F'60'           R8 = NEW RECORD LENGTH
         ST    R8,HRNXRLEN         SAVE NEW RECORD LENGTH
         L     R4,VARPOINT         R4 = VARIABLE DATA POINTER
         LA    R4,40(R4)           INCREMENT R4 TO NEXT OCCUR
         MVC   VARDTA,0(R4)        NEXT VARIABLE PART OF RECORD
         ST    R4,VARPOINT         SAVE THE VARIABLE POINTER
         LA    R8,FIXEDDTA         GET THE ADDRESS OF THE FIXED PART
         ST    R8,HRNXRECA         SAVE POINTER TO RECORD
         B     RETURN
*-------------------------------------------------------------------
```

```
*         IGNORE THE CURRENT RECORD AND SETUP FOR FIRST TIME PROCESS   *
*-------------------------------------------------------------------------
IGNORE   LA    R15,@HRNXDEL          SET 'DELETE' RC.
         MVI   TIMEFLAG,X'00'        SET FLAG FOR FIRST TIME PROCESS
RETURN   DS    0H
         RETURN (14,12),,RC=(15)
         LTORG
         END
```

# Chapter 6 Managing EXP Data Definitions

This chapter describes how to manage TIBCO Object Service Broker EXP data definitions.

## Topics


- Writing TIBCO Object Service Broker Tables to External Files, page 96
- Task A: Identify the Table, page 98
- Task B: Identify the Data, page 99
- Task C: Specify Data and Location Parameters, page 102
- Task D: Specify Event Rules, page 104
- Task E: Define Fields, page 105

# Writing TIBCO Object Service Broker Tables to External Files

To write TIBCO Object Service Broker tables to external files, you must define a TIBCO Object Service Broker table of type EXP. An export table can have one or more fields, up to 16 fields in a composite primary key (to a total maximum of 127 bytes), and optional data and location parameters.

### Table Definer Screen for an Export Table

```
 COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_EXP     Type: EXP   Unit: USR40

       File:
     DDname:              External Routine Name:
     Server ID:
    Parameter Name  Typ Syn Len Dec Class            '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -              '   ---------------- -   -
_ LOCATION          I   C   16  0     L              ' _
_                                                    ' _
                  --------- EXP ------|---------- Metadata Definition ------
     Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- -----  --   -----  -   -   -- ----- ---  -   -  ----------
_
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## Using a Copybook as the Source for the Definition

When you create a table in the TIBCO Object Service Broker UI, you can use a copybook as the source for its definition. You can then have TIBCO Object Service Broker monitor changes to the copybook. For more information, refer to Using Data Discovery on page 6.

## Steps Required to Define Export Tables

After invoking the Table Definer (refer to Initial Step for Defining Tables on page 5 for information on invoking the Table Definer), complete the following tasks to define an export table:

| | Task | Required | Page |
|---|---|---|---|
| A | Identify the table. | Y | 98 |
| B | Identify the data. | Y | 99 |
| C | Specify data and location parameters. | N | 102 |
| D | Specify event rules. | N | 104 |
| E | Define fields. | Y | 105 |

# Task A: Identify the Table

This task is used to:

- Uniquely identify the table
- Verify the table type
- Identify the application or logical unit where it belongs

### Table Identification Segment

The following example illustrates the fields used to identify the table:

```
Table: EMPLOYEE_EXP     Type: EXP   Unit: USR40
```

## Table, Type, and Unit Fields

The information for the **Table** and **Unit** fields is entered by default. You can modify the **Table**, **Type**, and **Unit** fields, if necessary.

| | |
|---|---|
| **Table** | The table name displayed in the **Table** field is the one you specified when invoking the Table Definer. To save the definition of an existing table under a new name, type a new name in the field. |
| **Type** | The type indicates how data is stored in the table or how data is to be accessed from a table. This field displays EXP, which you changed in Initial Step for Defining Tables on page 5. |
| **Unit** | The unit marks the table as belonging to a particular application or logical unit such as utilities, accounting, or network control. |

See Also *TIBCO Object Service Broker Shareable Tools* for more information about the tools used to copy tables.

# Task B: Identify the Data

You can use this task to specify the destination of the export table as well as the name of external routines if you want to manipulate the data at its source before exporting.

### Data Identification Segment

The following example illustrates the fields used to identify data:

```
 File:
DDname:              External Routine Name:
Server ID:
```

## File Field

The **File** field contains the name of the file to which you are exporting the data. The maximum record size is 31,744 bytes on all platforms. The record length for fixed length files must be at least equal to, and the record length for variable length files must be at least four more than, the total length of the fields.

Notes on the File

- z/OS**:** Under CICS, you do not have to define the data set in a Destination Control Table (DCT) but the CICS Region must have external security access to the data set.
- Open Systems**:** To specify the format of the data in external files to be processed by TIBCO Object Service Broker, use the DSIXFTYPE Execution Environment parameter. For more information on the DSIXFTYPE Execution Environment parameter, refer to *TIBCO Object Service Broker Parameters*.
- Open Systems**:** This filename maps to an entry in the filespec.dsn file. This file describes the absolute path through which this filename can be accessed along with other file attributes. If the filename is mapped to an entry in the file, special characters (*, ?, <, >) included in the filename are ignored. If a default path name is used, special characters in the filename are replaced and you do not receive the intended filename. For more information on filespec.dsn, refer to the Appendix B, Mapping File Names for Open Systems, on page 161.
- If you use FTP to transfer a variable length export table file between z/OS and Windows or Solaris, you must reformat the file using the TIBCO Object Service Broker z/OS utility S6BBRFRU. If the export file is fixed length, you do not have to reformat the file.
- If you specify both a filename and a DD name, the filename is used.

- For parameterized export tables, you must specify a partitioned data set or a directory. Neither of these has to exist before you define the export table but must exist before you insert data. Parameter values you provide become the data set member names or the filenames in the directory. You can create only one table instance per TIBCO Object Service Broker transaction.

  The parameter must have syntax C. On z/OS, the parameter's length must be 8. In Open Systems, its length must be 8 or less.

- With non-parameterized export tables you can export data to any file or member of a partitioned data set.

See Also *TIBCO Object Service Broker for z/OS Utilities* for more information about S6BBRFRU.

## DDname Field

The **DDname** field points to the file to which you are exporting the data. You specify a **DDname** if you want to change the export data set without changing the table definition or if you want to export to an uncataloged data set (possibly tape).

### Using a DDname

If you use a DDname:

- z/OS**:** You must associate **DDname** with the data set that is the destination of the data. To do this, use JCL or the TSO **ALLOCATE** command.
- z/OS**:** To export data to a partitioned data set, specify the member in the DD statement of your JCL or your TSO **ALLOCATE** command.
- z/OS**:** The **DDname** field is valid in TIBCO Object Service Broker multiple session environments under CICS, IMS TM, or Native Execution Environment only if you use a file disposition other than DISP=SHR in the definition of these environments. This is to prevent concurrent updates.
- Open Systems**:** In the **DDname** field, specify either the name of the environment variable that contains the fully qualified name of the file, or the name of a DD definition in the filespec.dsn file.
- If you are writing multiple export tables to a single file, the DDname in all the export tables must be the same.
- The export table cannot have data parameters.
- If you specify both a File name and a DD name, the File name is used.

## Server ID Field

The Server ID field points to the gateway that is to be used if the table is to be accessed remotely via the Service Gateway for Files. This is an optional specification. The value in this field is determined from the SERVERID Execution Environment parameter.

Refer to Monitoring of Service Gateway for Files on page 45 for details about the Server ID and the Service Gateway for Files.

See Also *TIBCO Object Service Broker Parameters* for information about the SERVERID parameter.

## External Routine Name Field

The **External Routine Name** field, which is valid only for the z/OS platform, contains the name of an external routine that you can use to manipulate data before exporting it to the destination data set. This routine is a load module resulting from an assembler program and resides in a data set that is concatenated to the external utilities data set. For more information, refer to External Routines (Pre-processing the Data) on page 116.

# Task C: Specify Data and Location Parameters

You can use this optional task to specify two types of parameters:

- Data
- Location

### Parameter Segment

The following example illustrates the fields used to define the location (illustrated) and data parameters. To view additional fields, use PF11.

```
   Parameter Name  Typ Syn Len Dec Class
   --------------  -   -- --- --    -
_ LOCATION         I   C   16  0     L
_
```

## Data Parameter

You require a data parameter if you want to write data to a partitioned data set or a directory. The parameter value you provide becomes the data set member name or the filename in the directory. Data parameters must be defined with syntax C. On z/OS, the parameter length must be 8. In Open Systems, the length must be 8 or less.

## Location Parameter

You can use a location parameter to write data to a peer server associated with another Data Object Broker (remote node). If you do not need to access remote data, use the **D** line command to delete the parameter. If you always access the external file remotely, the node from which you request the access can have either a minimal or full definition.

### Minimal Definition

A minimal definition with a location parameter means you always access data at a remote node. A minimal definition consists of the following:

- The table name, which must be the same at both locations
- The location parameter, which must be the same at both locations

The location parameter indicates that you always access data at a remote node. The name of the remote node where the full definition is located must be supplied in the **Default** field, **Src** field, or **Src** and **Sourcename** fields.

The table type specified in a minimal definition does not have to match the table type of the full definition on the remote node.

### Full Definition

A full table definition with a location parameter means you can access data at either the local or the remote node. The table type of the full definition must match the data on the local node. For example, a full definition of type TDS used to access TDS data on the local node can also be used to access an export table with the same name on a remote node.

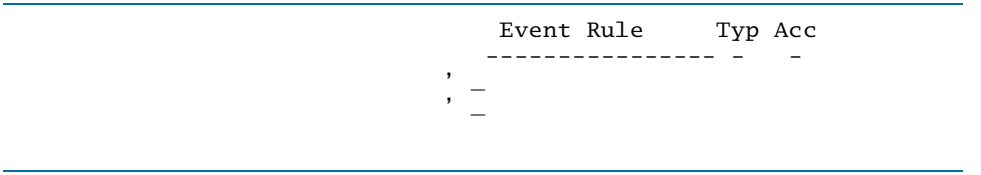
See Also *TIBCO Object Service Broker Managing Data* for more information on parameters.

# Task D: Specify Event Rules

This optional task is used to specify event rules if you need to associate business rules and policies with the definition of a table. These rules allow you to validate and automatically trigger other events based on specific retrieval access to import tables. The rules that you name here are run whenever data in the table is manipulated.

### Event Rule Segment

The following example illustrates the fields used to specify event rules:




## Event Rule, Typ, and Acc Fields

The rules that you enter here are run based on defined accesses. You can specify as many rules as you need in any logical order. The rules applying to specific accesses are executed in the order in which they are entered in this field. For valid values, use PF1.

| | |
|---|---|
| **Event Rule** | Specify the name of the event rule that is to be executed when the table is accessed. |
| **Typ** | Specify the type of event rule that is to be executed. |
| **Acc** | Specify the type of data access that invokes the event rule. Only one access type can be specified for each entry of the **Typ** field. |

See Also *TIBCO Object Service Broker Managing Data* for more information on event rules.

# Task E: Define Fields

You use this task to define the external export and internal TIBCO Object Service Broker attributes for the primary key fields and non-key fields of a table.

### Field Definition Segment

The following example illustrates the fields used to define the fields of an export table:

```
                  --------- EXP ------|---------- Metadata Definition ------
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  -----   -  -   -- ----  --  -  --------------
_
_
_
_
_
```

## Considerations

When defining an export table:

- When defining fields, you can type in external export attributes and the TIBCO Object Service Broker attributes default to the external values, or vice versa.
- The number of fields you can access is dependent upon the Data Object Broker parameter CTABLESIZE. You can use the ESTIMATETBLDFN tool to estimate the size of this parameter.
- Areas of the record not defined as fields contain hex zeros when written.
- If LRECL is not specified in the DCB information at runtime, the position of the last byte of the right-most field (as specified in the external field definition) is used to calculate the LRECL.

## Specifying External Export Attributes

The following fields are used to specify the external export attributes. Use PF1 for valid values for each field:

| | |
|---|---|
| **Field Name** | This field contains the name of the export field within the table definition. You can use the same name as a field in any other table. If you are moving data between this table and another table, giving fields the same names simplifies the process. |
| **Xsyn** | This field contains the external syntax for the export field. If not specified, the **Xsyn** field defaults to the specified TIBCO Object Service Broker syntax (**Syn** field). On Open Systems, numeric external syntaxes are treated as C or V for TEXT files. For information on external syntax, refer to Appendix A, Mapping Data Types, on page 153. |
| **Xlen** | This field contains the external length for the export field. If not specified, the **Xlen** field defaults to the specified TIBCO Object Service Broker length (**Len** field). For information on external lengths, refer to Appendix A, Mapping Data Types, on page 153.<br><br>On the Open Systems platforms, the **Xlen** field is ignored for TEXT files with a field separator character defined. |
| **Xdec** | This field indicates the external number of decimal places for the export field. If not specified, the **Xdec** field defaults to the specified TIBCO Object Service Broker decimal place (**Dec** field). |
| **Offset** | This field specifies the offset of the export field relative to the start of the external record. The origin is zero. You do not need to define fillers, since the offset can be used to skip undefined locations in the row. You can specify offsets in one of three ways:<br>• Assign the offset if you know it.<br>• Use PF3 to save the definition; this calculates the offset.<br>• Use PF6 to calculate the offset based on the location of the cursor.<br><br>On the Open Systems platforms, the **Offset** field is ignored for TEXT files with a field separator character defined. |

## Specifying Internal TIBCO Object Service Broker Attributes

The following fields are used to specify the internal TIBCO Object Service Broker attributes. Use PF1 for valid values for each field. To view additional fields, use PF11:

| | |
|---|---|
| **Key** | This field indicates if the export fields are to be used in the primary key. You can select any field as the primary key using the **P** line command, without respect to uniqueness of data. You can select up to 16 contiguous fields for a composite primary key, to a total maximum length of 127 bytes. |
| **Typ** | This field contains the TIBCO Object Service Broker semantic data type of the export field. The default is null. You can specify any valid TIBCO Object Service Broker semantic data type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Syn** | This field contains the TIBCO Object Service Broker syntax of the export field. You can specify any valid TIBCO Object Service Broker semantic type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. If a syntax is not specified, the **Syn** field defaults to an appropriate syntax based on the external syntax and length (**Xsyn** and **Xlen** fields). For more information on external syntax, refer to Appendix A, Mapping Data Types, on page 153.<br><br>Syntaxes F, RD, and UN are not supported for a primary key field. |
| **Len** | This field indicates the length of the export field. The data is padded or truncated as necessary. If a length is not specified, the **Len** field defaults to an appropriate length based on the external syntax and length (**Xsyn** and **Xlen** fields). |

| | |
|---|---|
| **Dec** | This field specifies the number of digits to appear to the right of the decimal point. The data is padded or truncated as necessary. If not specified, the **Dec** field defaults to the specified external number of decimal places (**Xdec** field).<br><br>Depending on the syntax specified in the **Syn** field, define this field as follows:<br><br>• For syntax P, the number of decimal places must be smaller than twice the length of the entire field.<br><br>• For syntaxes B, C, F, RD, UN, and V, the number of decimal places must be 0. |
| **Ord** | This field indicates the order (ascending or descending) in which the occurrences in this field are sorted. The default value of null returns occurrences unsorted. When an ordering option is explicitly specified, it takes precedence over the default. When ordering is specified for more than one field, the sort precedence is determined by the order of the fields as they are listed in the table.<br><br>Specifying a value in this field incurs sorting overhead, which can be significant in tables with a large number of occurrences.<br><br>Ordering is not permitted for fields of syntax F (float), RD (raw data) or UN (Unicode). |
| **Rqd** | This field indicates whether the user is required to provide a value for each occurrence in the table. The default is null (not required). |
| **Default** | This field contains the default value for the field when it appears. If no data is available, the value provided in this field is used. For example, if you specify a dot (.) as the default, it appears for a field that does not have any values. If you do not specify anything, a blank space appears.<br><br>For numeric fields of Q or C, specify a value such as 0.00 if arithmetic operations are to be performed on the field; arithmetic operations cannot be performed on data containing null values.<br><br>Default values are not permitted for fields of syntax RD or UN. |
| **Globalfield Name** | This field displays the name of the global field if you used PF14 to select a field from the global field dictionary. |

See Also

*TIBCO Object Service Broker Shareable Tools* for information on the ESTIMATETBLDFN tool.

*TIBCO Object Service Broker Managing Data* for information on global fields.

*TIBCO Object Service Broker Parameters* for more information about the CTABLESIZE Data Object Broker parameter.

# Chapter 7 Processing External Data Using TIBCO Object Service Broker

This chapter describes how to process external data using TIBCO Object Service Broker. For a description of how to process Adabas data refer to the Service Gateway for Files *SDK User's Guide.*

## Topics


- Writing Data to an Export File and Accessing the Exported Data, page 112
- Handling TIBCO Object Service Broker Requests, page 114
- External Routines (Pre-processing the Data), page 116

# Writing Data to an Export File and Accessing the Exported Data

When you insert data into an export table, the data is immediately exported to the associated file. You cannot roll back the changes. You can insert data into an export table using:

- Rules
- COPY_DATA tool
- CT copy table workbench option

## Using Rules

When writing data to an export file using rules, you can:

- Update the data before it is exported
- Write data to the file in batch
- Write multiple record formats to the same file or data set

  If the File name or DD name for each export table is the same, you can define multiple export tables to write data to a single data set. Then, if you use rules to insert data into the tables within the same transaction, the data set remains open to receive the occurrences from each table. Providing that the parameter value is the same, the resulting records in the data set have a variety of layouts.

- INSERT is the only statement you can use to access an export table.
- Transactions that run in browse mode can still update the export table.

## Using COPY_DATA

You can use the COPY_DATA tool to copy data from a TIBCO Object Service Broker table to the export table.

## Using Copy Table

You can use the workbench option CT copy table to copy data from a TIBCO Object Service Broker table to the export table.

## Accessing the Exported Data

To access the exported data, exit to your non-TIBCO Object Service Broker system and browse, edit, or process the data as required. You can also terminate the creating transaction and access the file with a different table name of type IMP.

See Also

- *TIBCO Object Service Broker Programming in Rules* for information about the rules language and batch processing.
- *TIBCO Object Service Broker Shareable Tools* for information on the copy tools.

# Handling TIBCO Object Service Broker Requests

The following sections describe how EXP tables handle requests with respect to:

- Synchronization and recovery
- Error handling

By understanding these operations, you can take full advantage available features when you develop applications.

## Transaction Length

An export transaction spans the same length of time as a TIBCO Object Service Broker transaction.

## Error Handling

The TIBCO Object Service Broker runtime environment signals system exceptions to permit an application to recover from an error. A three-level hierarchy of exceptions exists. The ERROR exception is the top of the hierarchy and is intended to be a catchall exception. Each exception traps the exceptions that appear below it in the hierarchy.

All errors encountered when accessing external data the EXP tables are trapped under one of the following TIBCO Object Service Broker exceptions:

- ERROR
- ACCESSFAIL
- INTEGRITYFAIL

## ERROR Exception

An ERROR exception indicates that an error is detected and no lower-level exception exists in the application.

## ACCESSFAIL Exception

An ACCESSFAIL exception indicates that a table access error is detected. The only valid exception under an ACCESSFAIL exception is a GETFAIL, indicating that no occurrence satisfies the selection criteria.

## INTEGRITYFAIL Exception

An INTEGRITYFAIL exception indicates an attempt to violate data integrity is detected. The following exceptions are valid under an INTEGRITYFAIL exception:

| | |
|---|---|
| **DEFINITIONFAIL** | Indicates that the data set or file specified is not found |
| **LOCKFAIL** | Indicates that there is a lock on an occurrence or a table is unavailable |
| **SECURITYFAIL** | Indicates that permission for the requested action on the TIBCO Object Service Broker object is denied |

See Also *TIBCO Object Service Broker Programming in Rules* for more information on exceptions.

# External Routines (Pre-processing the Data)

## Manipulating Data with External Routines (z/OS Only)

When accessing an external data set through an export table in z/OS, you can write an external routine to manipulate the data. The external routine must be written in assembler and can:

- Pass the record to a destination (export to a data set)
- Cause the record to be skipped
- Alter the record before passing it to its destination
- Insert multiple records to the external file based on one record from TIBCO Object Service Broker

You can concatenate the library in the CLIST that you use to log in to TIBCO Object Service Broker or if you are exporting the data in batch, you can concatenate the library in the batch JCL DD statement HRNEXTR.

## Parameters Passed to the External Routine

To write an external routine use the IMPXPARM macro found in the MACRO data set distributed with TIBCO Object Service Broker to describe the DSECT of the parameters that are passed to the external routine.

| Parameter | Description |
|---|---|
| HRNXRLEN | Represents the record length. If the record length is changed by the external routine, the server uses the changed length for input or output. HRNXRLEN is set to zero upon entry to the import external routine when the end of the file is reached. You can use this to insert additional records into TIBCO Object Service Broker. |
| HRNXRECA | Points to the I/O record buffer. If the buffer pointer is changed by the external routine, the server uses the new buffer for input or output. Do not change the data in this buffer. |
| HRNXWRKA | A 4 KB work area that is preserved between calls to the external routine. It is cleared to X'00' at the first rules-based INSERT to an export table. |

| Parameter | Description |
|---|---|
| HRNXRECE | A 4 KB work area that is not preserved between calls to the external routine. It is also not cleared before the external routine call. It can be used as the work area where you compose the record to be inserted. |
| HRNXTABN | Points to the 16 character TIBCO Object Service Broker table name. |
| HRNXDSN | Points to the 44 character external data set name. |
| HRNXTPRM | Points to the 8 character table parameter value. |

## Valid Return Codes

At the end of the external routine, Register 15 contains a return code that indicates the action the server should take. The valid return codes whose description follows are located in the IMPXPARM macro.

| Return Code | Description |
|---|---|
| @HRNXKEP | Export the record. |
| @HRNXDEL | Do not export the record (equivalent to a logical DELETE). |
| @HRNXDON | Do not call the external routine again for this transaction. No more physical records are read or written and the external routine is not called again for this transaction. For an export table, end of file is indicated when HRNXRLEN is zero and you should return @HRNXDON to prevent its external routine from being called again. |
| @HRNXINX | A new record is to be inserted. The external routine must set HRNXRECA to the address of the new record and HRNXRLEN to the length of the new record. |

## Example of an Exit that Exports Variable Length Records

The following example calculates record size based on data in the record. The record size is modified to create variable length records in the output data set.

```
         COPY   EQUATES              STANDARD EQUATES.
         COPY   IMPXPARM             EXIT PARAMETERS.
EXPREC   DSECT
```

```
EXPDLEN  DS    H
EXPTYPE  DS    CL1
         DS    CL1
EXPDATA  DS    4CL1000
SCREXIT CSECT
         SAVE  (14,12),,*           SAVE THEM ... STANDARD LINKAGE
         USING SCREXIT,R12         R12 ...
         LR    R12,R15                       BASE REGISTER
         USING HRNXPARM,R1          ADDR CALLING PARMS.
         L     R2,HRNXRLEN          R2 = RECORD LENGTH
         LTR   R2,R2                IS IT A NORMAL RECORD?
         BNZ   NORMAL               YES
*--------------------------------------------------------------------*
*        WE'VE BEEN CALLED AT THE END OF ALL NORMAL RECORDS.          *
*--------------------------------------------------------------------*
         LA    R15,@HRNXDON         RC = DON'T CALL ANY MORE
         B     RETURN               RETURN
*--------------------------------------------------------------------*
*        CORRECT RECORD LENGTH ON OUTPUT VB RECORD.                   *
*--------------------------------------------------------------------*
NORMAL   L     R9,HRNXRECA          GET ADDRESS OF RECORD
         USING EXPREC,R9            TELL ASSEMBLER
         LH    R3,EXPDLEN           GET DATA LENGTH
         LA    R3,4(,R3)            ADD 4 TO LENGTH FOR HEADER
         ST    R3,HRNXRLEN          SAVE AS RECORD LENGTH
         LA    R15,@HRNXKEP         SET 'KEEP' RC.
RETURN   DS    0H
         RETURN (14,12),,RC=(15)
         LTORG
         END
```

# Chapter 8 Managing VSAM Data Definitions

This chapter describes how to manage TIBCO Object Service Broker VSAM data definitions.

## Topics


- Accessing VSAM Data from TIBCO Object Service Broker on page 120
- Task A: Identify the Table, page 122
- Task B: Identify the Data, page 124
- Task C: Specify Data and Location Parameters, page 127
- Task D: Specify Event Rules, page 129
- Task E: Define Fields, page 130
- Data Set Requirements, page 135

# Accessing VSAM Data from TIBCO Object Service Broker

To access VSAM data directly from TIBCO Object Service Broker, you must define a TIBCO Object Service Broker table of type VSM. A VSAM table can have one or more fields, one or more data parameters (to a total maximum length of 255 bytes), up to 16 fields in a composite primary key (to a total maximum of 127 bytes), and an optional location parameter. The table can also contain multiple record formats.

### Table Definer Screen for a VSAM Table

```
 COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_VSAM_KS Type: VSM   Unit: USR40              IDgen: N

     File  :
     DDname:              Read Only: Y     Load: N     Data Set Type: KSDS
     Ignore:
     Server ID:
   Parameter Name  Typ Syn Len Dec Class             '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -              '   ---------------- -   -
_ LOCATION          I   C   16  0    L              ' _
_                                                   ' _
                   ------- VSAM --------|--------- Metadata Definition ------
     Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- -----  --   -----  -  -   -- ----- ---  -   -  ----------
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## Using a Copybook as the Source for the Definition

When you create a table in the TIBCO Object Service Broker UI, you can use a copybook as the source for its definition. You can then have TIBCO Object Service Broker monitor changes to the copybook. For more information, refer to Using Data Discovery on page 6.

## Steps Required to Define a VSAM Table

After invoking the Table Definer (refer to Initial Step for Defining Tables on page 5 for information on invoking the Table Definer), complete the following tasks to define a VSAM table:

| Task | Required | Refer to page |
|---|---|---|
| Task A: Identify the Table. | Y | 122 |
| Task B: Identify the Data. | Y | 124 |
| Task C: Specify Data and Location Parameters. | N | 127 |
| Task D: Specify Event Rules. | N | 129 |
| Task E: Define Fields. | Y | 130 |

See Also *TIBCO Object Service Broker Managing Data* for more information on minimal definitions.

# Task A: Identify the Table

This step is used to:

- Uniquely identify the table
- Verify the table type
- Identify the application or logical unit to which it belongs
- Specify if the system should generate unique values for the primary key field

### Table Identification Segment

The following example illustrates the fields used to identify the table:

```
Table: EMPLOYEE_VSM_KS  Type: VSM  Unit: USR40          IDgen: N
```

## Table, Type, Unit, and IDgen Fields

The information for the **Table**, **Type**, **Unit**, and **IDgen** fields is entered by default. You can modify these fields, if necessary.

| | |
|---|---|
| **Table** | The table name displayed in the **Table** field is the one you specified when invoking the Table Definer. To save the definition of an existing table under a new name, type in the new name. |
| **Type** | The type indicates how data is stored in the table or how data is to be accessed from a table. This field displays VSM, which you changed in Initial Step for Defining Tables on page 5. |
| **Unit** | The unit marks the table as belonging to a particular application or logical unit such as utilities, accounting, or network control. |

| | |
|---|---|
| **IDgen** | The **IDgen** field determines whether a value should be generated for the primary key field. The default of N means that users who insert data into the table must enter a unique value for the primary key of each occurrence. A value of Y means that the system generates the value for the primary key of each occurrence. Type Y if you are defining a VSAM table for an external file that contains multiple record formats or multiple occurrences, or if you want a unique key for the VSAM table (so you can use the Table Browser).<br><br>If the table contains data and the **IDgen** field is set to Y, you can modify the field; however, if the field is set to N, you cannot modify the field. |

# Task B: Identify the Data

You use this step to specify the location of the data you want to access and the type of data set.

### Data Identification Segment

The following example illustrates the fields used to identify the data:

```
File  : USR40.EMPLOYEE.VSAM
DDname:             Read Only: Y     Load: N     Data Set Type: KSDS
Ignore:
Server ID:
```

## File Field

The File field contains the name of the VSAM file that holds the data you want to access. The maximum record size is 31,744 bytes. The record length must be equal to or greater than the total length of the table fields.

- If you are accessing multiple VSAM files with the same record layout, you can omit both the File name and DD name and instead specify the data set name as the first data parameter (it must have syntax C and a length of 44).
- Under CICS, define the VSAM data set in the File Control Table (FCT) and in the JCL. You must provide the key length of the VSAM records in the File Control Table.
- If you specify both a File name and a parameter, the File name is used.

## DDname Field

The **DDname** field contains the name of the JCL DD statement defining the file containing the data you want to access.

### Using a DDname

If you use a DDname, note the following:

- It must already be allocated by the TIBCO Object Service Broker Execution Environment, using a JCL DD statement or the TSO **ALLOCATE** command.
- If you are accessing multiple VSAM tables from a single data set, the DDname in all the VSAM tables must be the same.

## Read Only Field

The **Read Only** field contains a logical value that determines whether the data in the table can be modified. If **Read Only**=Y, the data can only be read. If **Read Only**=N, the data can be both read and modified.

Field overlaps are allowed only for Read Only tables. If your definition has overlapping fields when the **Read Only** flag is N, you are not allowed to save the definition.

## Load Field

The **Load** field contains a logical value to signal whether the VSAM table is to be initialized with data. The default is N. Type Y if you are initializing the data. You can change the value in this field to N after loading the initial data. You cannot read or change the loaded data; however, records are inserted into KSDS and RRDS types more efficiently.

## Data Set Type Field

The **Data Set Type** field contains a 1 to 4 byte string identifying the type of VSAM data set.

## Ignore Field

The **Ignore** field contains a value to identify the records to ignore in a KSDS data set. The **Ignore** field is defined and used as follows:

- The syntax is V.
- The length is 35.
- The value entered in the field is compared to the start of each record's key.

- If the key field is a string containing any unprintable characters, you can specify a hexadecimal string using X' as a prefix and a single quotation mark (') as a suffix. For example:

  `X'hhh...hh'`

### Processing Records

Records are processed as follows:

- If you request a sequence of records (a FORALL statement) or a non-specific record (a GET statement without a WHERE clause) and the ignore value matches the start of a record's key, that record is ignored and the next one is obtained.
- If you request a specific record (a GET statement with a WHERE clause) or perform update operations, no records are ignored.

## Server ID Field

The **Server ID** field points to the gateway that is to be used if the table is to be accessed remotely via the Service Gateway for Files (z/OS). This is an optional specification. The value in this field is determined from the SERVERID Execution Environment parameter.

Refer to Monitoring of Service Gateway for Files on page 45 for details on ServerID and Service Gateway for Files.

See Also *TIBCO Object Service Broker Parameters* for information about the SERVERVID parameter.

# Task C: Specify Data and Location Parameters

You can use this optional task to specify two types of parameters:

- Data
- Location

### Parameter Segment

The following example illustrates the fields used to specify data and location (illustrated) parameters. To view additional fields, use PF11.

```
   Parameter Name  Typ Syn Len Dec Class
  ---------------  -   -- --- --    -
_ LOCATION          I   C   16  0     L
_
```

## Data Parameter

You can use a data parameter to access external data in different data sets using the same table definition or to access multiple record types or multiple occurrences. Depending on the syntax and length you assign the data parameter, TIBCO Object Service Broker interprets it as follows:

| If IDgen is | and the Syntax is | and the Length is | TIBCO Object Service Broker |
|---|---|---|---|
| N | C | 44 | interprets the parameter value as the name of a data set. |
| Y | Any character | Any length | uses the parameters for reading multiple record types or multiple occurrences. The parameter definitions must match the primary key definition of the base table. |

## Location Parameter

You can use a location parameter to access external data through a peer server associated with another Data Object Broker (remote node). If you do not need to access remote data, use the **D** line command to delete the parameter. If you always access the external file remotely, the node from which you request the access can have either a minimal or full definition.

### Minimal Definition

A minimal definition with a location parameter means you always access data at a remote node. A minimal definition consists of the following:

- The table name, which must be the same at both locations
- The location parameter, which must be the same at both locations

  The location parameter indicates that you always access data at a remote node. The name of the remote node where the full definition is located must be supplied in the **Default** field, **Src** field, or **Src** and **Sourcename** fields.

The table type specified in a minimal definition does not have to match the table type of the full definition on the remote node.

### Full Definition

A full table definition with a location parameter means you can access data at either the local or the remote node. The table type of the full definition must match the data on the local node. For example, a full definition of type TDS used to access TDS data on the local node can also be used to access a VSAM table with the same name on a remote node.

See Also *TIBCO Object Service Broker Managing Data* for more information on defining parameters.

# Task D: Specify Event Rules

This optional task is used to specify event rules if you need to associate business rules and policies with the definition of a table. These rules allow you to validate and automatically trigger other events based on specific retrieval access to VSAM tables. The rules that you name here are run whenever data in the table is manipulated.

### Event Rule Segment

The following example illustrates the fields used to specify event rules:

```
'      Event Rule    Typ Acc
'   ---------------- -   -
'
' _
'  _
```

## Event Rule, Typ, and Acc Fields

The rules that you enter here are run based on defined accesses. You can specify as many rules as you need in any logical order. The rules applying to specific accesses are executed in the order in which they are entered in this field. For valid values, use PF1.

| | |
|---|---|
| **Event Rule** | Specify the name of the event rule that is to be executed when the table is accessed. |
| **Typ** | Specify the type of event rule that is to be executed. |
| **Acc** | Specify the type of data access or manipulation to be performed on the data causing the event to be executed. |

See Also *TIBCO Object Service Broker Managing Data* for more information on event rules.

# Task E: Define Fields

This task is used to define the external VSAM attributes and the internal TIBCO Object Service Broker attributes for the primary key fields and data fields of the table.

### The Field Definition Segment

The following example illustrates the fields used to define the fields of the VSAM table:

```
                 ------- VSAM --------|--------- Metadata Definition ------
     Field Name  Xsyn Xlen  Xdec Offset Key Typ Syn Len  Dec Ord Rqd Default
  ---------------- ---- -----  --   -----  -  -   -- ----- ---  -   -  ----------
_ KEY              B       2   0       0  P  I   B      2   0
_ DEPTNO           B       4   0       2     I   B      4   0
_ DEPTNAME         C       9   0       6     S   C      9   0
_
```

## Considerations

Note the following when defining a VSAM table:

- When defining fields you can type in external VSAM attributes and the TIBCO Object Service Broker attributes default to the external values, or vice versa.
- The number of fields you can access is dependent upon the Data Object Broker parameter CTABLESIZE. You can use the ESTIMATETBLDFN tool to estimate the size of the parameter.
- The number of occurrences allowed in a VSAM table is based on VSAM data set allocation specifications.

## Specifying External VSAM Attributes

The following fields are used to specify the external VSAM attributes. Use FP1 for valid values for each field.

| | |
|---|---|
| **Field Name** | This field contains the name of the VSAM field that must be unique within the table definition. You can use a field name that already exists in any other table; if you are moving data between this table and another table, giving fields the same names simplifies the process. |
| **Xsyn** | This field contains the external syntax for the VSAM field. If Xsyn is not specified, it defaults to the specified TIBCO Object Service Broker syntax (**Syn** field). For information on external syntax, refer to Appendix A, Mapping Data Types, on page 153. |
| **Xlen** | This field contains the external length for the VSAM field. If **Xlen** is not specified, it defaults to the specified TIBCO Object Service Broker length (**Len** field). For information on external lengths, refer to Appendix A, Mapping Data Types, on page 153. |
| **Xdec** | This field indicates the external number of decimal places for the VSAM field. If **Xdec** is not specified, it defaults to the specified TIBCO Object Service Broker number of decimal places (**Dec** field). |
| **Offset** | This field specifies the offset. The offset maps to the start of the external record relative to the VSAM field. The origin is zero. Overlaps are allowed for read-only tables and you do not need to define fillers, since the offset can be used to skip undefined locations in the row. You can specify offsets in one of three ways:<br>• If you know the offset, enter it.<br>• If all offsets are unspecified and you want the offsets calculated from the first field, use PF3 to save the definition; this calculates the offsets.<br>• To calculate the offset from the field where the cursor is located, press PF6 once to verify your intention by reading the message at the bottom of the screen, and a second time to calculate the offsets from that field. |

## Specifying Internal TIBCO Object Service Broker Attributes

The following fields are used to specify the internal TIBCO Object Service Broker attributes. For each field, use PF1 for a list of valid values. To view additional fields, use PF11.

| | |
|---|---|
| **Key** | This field indicates if the VSAM fields are to be used as a primary key. You can select any field as the primary key using the **P** line command, without respect to uniqueness of data. You can select up to 16 contiguous fields for a composite primary key, to a total maximum of 127 bytes.<br><br>For information on how primary key fields handle numeric syntaxes, refer to Behavior of Numeric Key Fields in VSAM Tables on page 134. |
| **Typ** | This field contains the TIBCO Object Service Broker semantic data type of the field. The default is null. You can specify any valid TIBCO Object Service Broker semantic data type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Syn** | This field contains the TIBCO Object Service Broker syntax of the field. You can specify any valid TIBCO Object Service Broker semantic type and syntax combination supported for the external syntax. For valid combinations, refer to *TIBCO Object Service Broker Programming in Rules*. If a syntax is not specified, the **Syn** field defaults to an appropriate syntax based on the external syntax and length (**Xsyn** and **Xlen** fields). For more information on external syntax, refer to Appendix A, Mapping Data Types, on page 153.<br><br>Syntaxes F, RD, and UN are not supported for a primary key field. For information on how primary key fields handle numeric syntaxes, refer to Behavior of Numeric Key Fields in VSAM Tables on page 134. |
| **Len** | This field indicates the length of the VSAM field. The data is padded or truncated as necessary. If a length is not specified, the **Len** field defaults to an appropriate length based on the external syntax and length (**Xsyn** and **Xlen** fields). |

| | |
|---|---|
| **Dec** | This field specifies the number of digits to appear to the right of the decimal point. The data is padded or truncated as necessary. If not specified, the **Dec** field defaults to the specified external number of decimal places (**Xdec** field).<br><br>Depending on the syntax specified in the **Syn** field, define this field as follows:<br><br>• For syntax P, the number of decimal places must be smaller than twice the length of the entire field.<br><br>• For syntaxes B, C, F, RD, UN, and V, the number of decimal places must be 0. |
| **Ord** | This field indicates the order (ascending or descending) in which the occurrences in this field are sorted. The default is null (unsorted).<br><br>Ordering is not permitted for fields of syntax f (float), RD (raw data) or UN (Unicode). |
| **Rqd** | This field indicates if the user is required to provide a value for each occurrence in the table. The default is null (not required). |
| **Default** | This field contains the default value for the field. If no data is available, the value provided in this field is used. For example, if you specify a dot (.) as the default, it appears for a field that has no values. If you do not specify anything, a blank space appears.<br><br>Default values are not permitted for fields of syntax RD or UN. |
| **Globalfield Name** | This field displays the name of the global field if you used PF14 to select a field from the @GLOBALFIELDS table. |

See Also

*TIBCO Object Service Broker Shareable Tools* for information on the ESTIMATETBLDFN tool.

*TIBCO Object Service Broker Managing Data* for information on global fields.

*TIBCO Object Service Broker Parameters* for more information about the CTABLESIZE Data Object Broker parameter.

## Behavior of Numeric Key Fields in VSAM Tables

Unexpected results can occur if a field of syntax B or P is defined as a primary or secondary key field of a VSM table. The results occur because the two numeric syntaxes represent signed values, whereas VSAM key fields are always interpreted as unsigned values. Therefore, records can be placed out of sequence if the data contains both negative and positive values for the key fields. Records are ordered correctly in the following situations:

- The primary or secondary key field contains only negative or positive values
- A non-key field of numeric syntax contains both negative and positive value and an ORDERED clause is used for selection

### Example

The key field has semantic type DATE and syntax B, containing dates both before and after January 1, 1980. Dates before January 1, 1980 are represented as negative integers and dates after this date are represented as positive integers. Since VSAM treats all dates as unsigned integers, the dates prior to January 1, 1980 are ordered after January 1, 1980.

# Data Set Requirements

## KSDS Requirements

Note the following when defining fields for the KSDS data set:

- The sum of the external field lengths must be less than or equal to the VSAM data set record length.
- The data set can be a variable length file.
- Composite key fields must be contiguous but do not have to start at the beginning of the record.
- To access a VSAM file by an alternate index, define the alternate index as the primary key in the TIBCO Object Service Broker table and specify the alternate index path as the data set name.
- To update a VSAM file via its alternate index, it must be defined through IDCAMS using the UPGRADE option.
- Updates to rows with duplicate secondary indexes must be done using a definition that has the primary key set to the VSAM primary key.
- The architecture of KSDS type files specifies that the initial record can only be added by a sequential write operation. There are two ways to handle the initialization of a VSAM KSDS file:
  - The IDCAMS utility can populate the data set using the REPRO statement. This can be done in the same job that initializes the VSAM KSDS data set.
  - Using TIBCO Object Service Broker, ensure the Load option on the Table Definer is set to Y.

  At least one record must then be inserted into the table. When a record is inserted, the Load option can be set to N, if desired.

## ESDS Requirements

Note the following when defining fields for the ESDS data set:

- The sum of the field lengths, excluding the primary key, must be less than or equal to the VSAM data set record length.
- The primary key field must be a 4-byte binary. This field represents a relative byte address (RBA). It specifies the relative offset in the data set of the record being accessed. The first record has an RBA of 0.
- The data set can be a variable length file.

- Access by alternate index can be accomplished by defining the secondary index as the primary key in TIBCO Object Service Broker and using the secondary index path as the data set name.
- Identical records can be inserted into a VSAM table holding the alternate index data.

## RRDS Requirements

Note the following when defining fields for the RRDS data set:

- The sum of the field lengths, excluding the primary key, must be less than or equal to the VSAM data set record length.
- The primary key field must be a 4-byte binary. This field represents a record number that identifies each record in the data set. The first record has a record number of at least 1.
- Alternate indexes cannot be used in the table definitions.

# Chapter 9 Defining VSAM Tables for Files with Multiple Record Formats

This chapter describes how to define VSAM tables for files with multiple record formats.

## Topics


- VSAM Tables for Files with Multiple Records, page 138
- Definition Requirements, page 139

# VSAM Tables for Files with Multiple Records

## What is a Multiple Record Format?

A VSAM table can reference or access a file of records that have different formats. For example, a personnel file has records that record employee benefits, payroll information, and training. To access this type of file with TIBCO Object Service Broker, you need to define multiple, related tables for each record format.

Usually the first table defines the base portion of the record that identifies subsequent record formats by a flag or record type indicator. To process the record, you GET the first base table then determine the subsequent record format by examining the flag byte. The corresponding related table is then accessed for the particular record format following the base portion of the record.

## What is a Repeating Group?

A repeating group is a collection of data that occurs multiple times in a record. For example, a time collection system could have 52 repeating groups to record hours worked for each week of the year. You define VSAM tables for a file containing repeating groups using the same technique as with a file containing multiple record formats. That is, you define separate VSAM tables for the base portion of the record and the repeating group.

## Sample File

The following example illustrates a sample file that has multiple record types. Each record contains sections of different formats (A and B), including a repeating group (B).

| | | | | | |
|---|---|---|---|---|---|
| A | B | B | B | B | B |
| A | B | | | | |
| A | B | B | B | | |

# Definition Requirements

## Base and Child Definitions

You must define a base table and a separate table for each child record format. Define each of your required VSAM tables as described in Chapter 8, Managing VSAM Data Definitions, on page 119.

## Definition Requirements of a Child Table

You must also include the following for each of the child tables:

- Specify the same data set name or DDname for both base and child tables.
- Specify IDgen=Y.
- Define the primary key field as an **IDgen** field with:

| | |
|---|---|
| Semantic Data Type. | I |
| Syntax. | B |
| Length. | 4 |

  Do not provide data for this field; the system generates these values. As a result, each record in the TIBCO Object Service Broker VSAM table has a unique identifier provided by the **IDgen** field.
- Define a data parameter that reflects the format of the associated base table key field. The base VSAM table definition does not require a parameter definition.

### Data Parameter Requirements of a Child Table

- The parameter name can be any valid TIBCO Object Service Broker field name, but TYPE, SYNTAX, LENGTH and DECIMALS must match the base VSAM table definition key fields.
- You must also specify in the REFERENCE field the name of the associated base VSAM table.
- You must set CLASS to D.

## Sample Definition (Record A)

The following is an example of a VSAM table definition for record A shown in the sample file on page , Sample File, on page 138:

```
COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_VSAM_A  Type: VSM   Unit: USR40              IDgen: N

     File  : USR40.EMPLOYEE.VSAM
     DDname:              Read Only: Y     Load: N     Data Set Type: KSDS
     Ignore:
     ServerID:

   Parameter Name  Typ Syn Len Dec Class             '      Event Rule   Typ Acc
  ----------------  -   -- --- --    -               '   ---------------- -   -
                                                     '
_                                                    ' _
_                                                    ' _
                   ------- VSAM --------|--------- ObjectStar Definition ------
     Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
  ---------------- ---- -----  --   -----  -  -   -- ----- ---  -   -  ----------
_ KEY              B       2    0       0  P  I   B      2   0
_ DEPTNO           B       4    0       2     I   B      4   0
_ DEPTNAME         C       9    0       6     S   C      9   0
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## Sample Definition (Record B)

The following is an example of a VSAM table definition for record B shown in the sample file on page <Link> “Sample File”. It includes a value for the REFERENCE field (scroll right to see this field):

```
COMMAND==>                          TABLE DEFINITION

      Table: EMPLOYEE_VSAM_B  Type: VSM   Unit: USR40              IDgen: Y

     File  : USR40.EMPLOYEE.VSAM
     DDname:              Read Only: Y     Load: N     Data Set Type: KSDS
     Ignore:
     ServerID:

   Parameter Name     Reference          Default      '      Event Rule   Typ Acc
  ---------------- ---------------- --------------- '   ---------------- -   -
_ RECORD_A         EMPLOYEE_VSAM_A                  ' _
_                                                   ' _
                   ------- VSAM --------|--------- ObjectStar Definition ------
```

```
     Field Name     Xsyn Xlen Xdec Offset Key Typ Syn  Len  Dec Ord Rqd Default
   ---------------- ---- -----  --  -----  -  -   --  ----- ---  -   -  ----------
_  KEY              B        4   0          P  I   B       4   0
_  LNAME            C       22   0             S   C      22   0
_  FNAME            C       10   0             S   C      10   0
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

# Chapter 10 Processing VSAM Data Using TIBCO Object Service Broker

This chapter describes how to process VSAM data using TIBCO Object Service Broker. For a description of how to process VSAM LDS data refer to the *TIBCO* Service Gateway for Files *SDK User's Guide.*

## Topics


- Access of TIBCO Object Service Broker VSAM Tables, page 144
- Sample Rules, page 147
- Handling of TIBCO Object Service Broker Requests, page 149

# Access of TIBCO Object Service Broker VSAM Tables

## Editing or Browsing

The setting of the **Read Only** field in the TIBCO Object Service Broker VSAM table definition determines how you can access the VSAM data:

- If **Read Only=N**, you can edit the table using the Table Editor or rules.

  You cannot delete records through an ESDS table definition or through a KSDS table definition representing an alternate index of an ESDS data set.

- If **Read Only=Y**, you can only browse the table using the Table Browser or rules.

## Using the Table Editor

You can edit a VSAM table in the same way you would edit any other table with the following exceptions:

- If your table definition contains fields of syntax C or V that are longer than 260 bytes, or fields of syntax RD or UN that are longer than 130 bytes, you must use **`SELECT LIKE`** instead of **`SELECT`** to access fields of this length.
- You can define and access KSDS and ESDS type definitions via alternate indexes. Full read and update access is available if you define the alternate index as the primary key.
- In TIBCO Object Service Broker you cannot cancel changes made in the Table Editor. VSAM changes are immediate; there is no ROLLBACK.

## Using the Table Browser

You can browse a VSAM table in the same way you would browse any other table except for the following:

- You cannot use the Table Browser to access a VSAM file with multiple record formats.
- Using the Single Occurrence Editor from the Table Browser begins a dependent transaction in TIBCO Object Service Broker. Therefore, the table must have a unique key because access in the row transaction is by key.

## Using Rules

You can access VSAM data using the rules language in the same way you would access data in any other table except for the following:

- You cannot roll back INSERTs, DELETEs, or REPLACEs even by transaction failure.
- If you issue a EXECUTE statement within a main (parent) transaction, it creates another transaction stream (child), to a maximum of nine streams.

  The number of streams allowed in a transaction depends on the Execution Environment parameter TRANMAXNUM. Each transaction stream accessing external data requires its own server thread. Additional transactions can cause parent-child locking conflicts if more than one transaction stream is in update.
- Using TRANSFERCALL or DISPLAY & TRANSFERCALL statements in a rule minimizes server threads and reduces the possibility of locking contention.
- Under CICS, write applications that allow pseudo-conversational processing using TRANSFERCALL or DISPLAY & TRANSFERCALL statements. This can minimize recovery problems after an abend.
- If you use the default session parameter values, you can access at least 16 VSAM tables per transaction; more, depending on the size of the VSAM table definitions.

## VSAM Files with Multiple Record Formats and Repeating Groups

When a file contains multiple record formats, you must use rules to access the data.

### Repeating Groups

Note the following about repeating groups:

- Before a repeating group occurrence can be accessed, its immediate parent must be accessed.
- Changes to repeating group occurrences are not written to the VSAM data set until an INSERT or REPLACE on the parent table.
- The maximum number of INSERTs of repeating group occurrences is determined by the VSAM record length.
- For ESDS and RRDS, the record must be full after all INSERTs and DELETEs of repeating group occurrences.

## Retrieval Processing

When a TIBCO Object Service Broker transaction runs in browse mode, locks are not taken on the TIBCO Object Service Broker data; however, locks are taken on the VSAM data in accordance to the transaction mode being set to Browse or No Browse.

The WHERE clause recognizes ranges. For example, if you use the clause `WHERE KEY > 5 & KEY < 10`, the server skips to `> 5` and then stops processing when `Key = 10`. The following two rules statements differ from regular processing:

- GET
- FORALL

VSAM files with multiple record formats cannot be sorted (ORDERED clause) or selected (WHERE clause) by field values because the files are defined by several tables.

### GET Statement

A GET statement retrieves the first occurrence in the VSAM table that satisfies the specified selection criteria.

A GET… ORDERED statement must retrieve all VSAM data that satisfies the selection criteria and sort it in the Execution Environment before returning the first occurrence that meets the selection criteria.

### FORALL Statement

When using a FORALL statement, occurrences are returned in primary key order. If you require a different order, you must include an ORDERED clause in your FORALL statement. TIBCO Object Service Broker orders only occurrences specified in the selection criteria.

See Also

- *TIBCO Object Service Broker Managing Data* for information on the Table Editor.
- *TIBCO Object Service Broker Programming in Rules* for information on using rules, transactions, and table access statements.

# Sample Rules

## Sample Rule 1: Accessing Multiple Record Formats

If the file contains multiple record formats in one physical record, you can access all the records in the file by using rules statements similar to those that follow. To update a record, you must update the base table after updating the other multiple record format tables. The tables in this rule are the ones defined in Sample Definition (Record A) and Sample Definition (Record B) on page 140.

```
 RULE EDITOR ===>                                                    SCROLL: P
VSAMREPEAT(NEW_NAME);
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL EMPLOYEE_VSAM_A :                                    | 1
_    FORALL EMPLOYEE_VSAM_B(EMPLOYEE_VSAM_A.KEY) :            |
_       EMPLOYEE_VSAM_B.LNAME = NEW_NAME;                     |
_       EMPLOYEE_VSAM_B.FNAME = EMPLOYEE_VSAM_B.FNAME ||      |
_         ' STUDENT';                                         |
_       REPLACE EMPLOYEE_VSAM_B(EMPLOYEE_VSAM_A.KEY);         |
_       END;                                                  |
_    REPLACE EMPLOYEE_VSAM_A;                                 |
_    END;                                                     |
_ ------------------------------------------------------------------------------



PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE
```

## Sample Rule 2: Inserting Parent Record with Children

The following is a sample rule illustrating how to insert a parent record with children.

When the physical record is written as indicated by `INSERT EMPLOYEE_A;` the record must be full.

The tables in this rule are the ones defined in Sample Definition (Record A) and Sample Definition (Record B) on page 140 except the data set type is ESDS instead of KSDS.

```
      RULE EDITOR ===>                                                SCROLL: P
INSERT_VSAM;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ EMPLOYEE_A.KEY = 1;                                         | 2
_ EMPLOYEE_A.DEPTNO = 10;                                     | 3
_ EMPLOYEE_B.LNAME = 'DOE';                                   | 4
_ EMPLOYEE_B.FNAME = 'JOHN';                                  | 5
_ INSERT EMPLOYEE_B(EMPLOYEE_A.KEY);                          | 6
_ EMPLOYEE_B.LNAME = 'ANN';                                   | 7
_ EMPLOYEE_B.FNAME = 'MARY';                                  | 8
_ INSERT EMPLOYEE_B(EMPLOYEE_A.KEY);                          | 9
_ EMPLOYEE_B.LNAME = 'DICK';                                  | A
_ EMPLOYEE_B.FNAME = 'TRACY';                                 | B
_ INSERT EMPLOYEE_B(EMPLOYEE_A.KEY);                          | C
_ INSERT EMPLOYEE_A;                                          | D
_ ------------------------------------------------------------------------------


PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE
```

# Handling of TIBCO Object Service Broker Requests

The following sections describe how requests are handled with respect to:

- Synchronization and recovery
- Error handling

## Synchronization and Recovery

About synchronization and recovery:

- A VSAM transaction spans the same length of time as a TIBCO Object Service Broker transaction.
- A COMMIT causes all locks to be released.
- Intermediate COMMITs are necessary when concurrent updates are performed within the same system or cross-system if SHAREOPTIONS 3 is specified for a VSAM file. These intermediate COMMITs avoid data integrity problems, data loss, or unpredictable results (they cause buffered updates to be flushed from the task buffers).
- Under CICS, consider setting the CICSVSAMSYNC Execution Environment parameter to control when SYNCPOINTs are issued by the VSAM server.

These tables summarize the behavior of the VSAM server under CICS:

| Mode | Pseudo-Conversational | | | | Conversational | | | |
|---|---|---|---|---|---|---|---|---|
| Operation | SYNCPOINT | | ROLLBACK | | SYNCPOINT | | ROLLBACK | |
| CICSVSAMSYNC= | YES | NO | YES | NO | YES | NO | YES | NO |
| Event | | | | | | | | |
| DISPLAY | Yes | No | No | No | Yes | No | No | No |
| DISPLAY & TRANSFERCALL | Yes | No | No | No | Yes | No | No | No |
| TRANSFERCALL | Yes | No | No | No | Yes | No | No | No |
| CALL | No | No | No | No | No | No | No | No |

| Mode | Pseudo-Conversational | | | | Conversational | | | |
|---|---|---|---|---|---|---|---|---|
| EXECUTE | Yes [a] | No | No | No | Yes [a] | No | No | No |
| TIBCO Object Service Broker trans. abend | Yes | No | No | No | No | No | No | No |
| CICS trans. abend | No | No | Yes [b] | Yes | No | No | Yes [c] | Yes |
| COMMIT | Yes | No | No | No | Yes | No | No | No |
| ROLLBACK | No | No | No | No | No | No | No | No |
| TIBCO Object Service Broker trans. end | Yes | No | No | No | Yes | No | No | No |

a. SYNCPOINT is done at the end of the executed transaction

b. Only updates since the last DISPLAY are rolled back

c. Only updates since last intermediate SYNCPOINT are rolled back

## Error Handling

The TIBCO Object Service Broker runtime environment signals system exceptions to permit an application to recover from an error. A three-level hierarchy of exceptions exists. The ERROR exception is the top of the hierarchy and is intended to be a catchall exception. Each exception traps the exceptions that appear below it in the hierarchy.

All errors encountered when accessing external data are trapped under one of the following TIBCO Object Service Broker exceptions:

- ERROR
- ACCESSFAIL
- INTEGRITYFAIL

## ERROR Exception

An ERROR exception indicates that an error is detected and no lower-level exception exists in the application.

## ACCESSFAIL Exception

An ACCESSFAIL exception indicates that a table access error is detected. The following exceptions are valid under an ACCESSFAIL exception:

| | |
|---|---|
| **GETFAIL** | Indicates that no occurrence satisfies the selection criteria |
| **DELETEFAIL** | Indicates that the primary key specified for a DELETE statement does not exist |
| **INSERTFAIL** | Indicates that the primary key provided for an INSERT statement already exists |
| **REPLACEFAIL** | Indicates that the primary key provided for a REPLACE statement does not exist |

## INTEGRITYFAIL Exception

An INTEGRITYFAIL exception indicates an attempt to violate data integrity is detected. The following exceptions are valid under an INTEGRITYFAIL exception:

| | |
|---|---|
| **DEFINITIONFAIL** | Indicates that the data set specified is not found |
| **LOCKFAIL** | Indicates that there is a lock on an occurrence or a table is unavailable |
| **SECURITYFAIL** | Indicates that permission for the requested action on the object is denied |
| **SERVERERROR** | External database server error detected |

If a failure occurs when the data set is first opened, either the SERVERERROR or DEFINITIONFAIL exception can result.

See Also *TIBCO Object Service Broker Programming in Rules* for more information on exceptions.

# Appendix A Mapping Data Types

This appendix describes how to map data types for IMP, EXP, and VSM table definitions.

The following table displays the default mapping of external syntax and length (**XSyn** and **Xlen** fields) to TIBCO Object Service Broker syntax and length (**Syn** and **Len** fields) for MAP table definitions. In every case, the TIBCO Object Service Broker decimal (Dec field) value is equal to the external decimal length. TIBCO Object Service Broker syntax is described more fully in *TIBCO Object Service Broker Programming in Rules*.

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| * | Binary or character data, according to internal syntax. Binary data is interpreted according to endian value. Character data is translated according to code page.<br>Valid only for table type MAP. | 1 – 31742 | No default | No default | No default |
| A[o] | Alphabetic (Uppercase). | 1 – 31742 | 0 | C | Xlen[a] |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| B [b, c] | Binary, signed. | 1 & Xdec=0 | 0 | B | 2 |
| | | 2-12 & Xdec=0 | 0 | B | same as Xlen |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2 & Xdec>0 | 5 | P | 3 |
| | | 3 & Xdec>0 | 9 | P | 5 |
| | | 4 & Xdec>0 | 11 | P | 6 |
| | | 5 & Xdec>0 | 13 | P | 7 |
| | | 6& Xdec>0 | 15 | P | 8 |
| | | 7& Xdec>0 | 17 | P | 9 |
| | | 8& Xdec>0 | 19 | P | 10 |
| B16 [e] | Unicode UTF-16-BE encoded string. | 2 – 31742, must be a multiple of 2 | 0 | UN | Xlen[a] |
| B16B [f] | Unicode UTF-16-BE encoded string with BOM. | 4 – 31742, must be a multiple of 2 | 0 | UN | Xlen-2[a] |
| B32 [g] | Unicode UTF-32-BE encoded string. | 4 – 31740, must be a multiple of 4 | 0 | UN | Xlen/2 |
| B32B [h] | Unicode UTF-32-BE encoded string with BOM. | 8 – 31740, must be a multiple of 4 | 0 | UN | (Xlen-4)/2 |
| C[o] | Fixed length character string, with trailing blanks ignored. | 1 – 31742 | 0 | C | Xlen[a] |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| E [c, i] | Little-endian binary, signed. | 1 & Xdec=0 | 0 | B | 2 |
| | | 2-12 & Xdec=0 | 0 | B | same as Xlen |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2 & Xdec>0 | 5 | P | 3 |
| | | 3 & Xdec>0 | 9 | P | 5 |
| | | 4 & Xdec>0 | 11 | P | 6 |
| | | 5 & Xdec>0 | 13 | P | 7 |
| | | 6& Xdec>0 | 15 | P | 8 |
| | | 7& Xdec>0 | 17 | P | 9 |
| | | 8& Xdec>0 | 19 | P | 10 |
| F | Floating point (short, long, or extended). | 4 | 0 | F | 4 |
| | | 8 | | | 8 |
| | | 16 | | | 16 |
| G | Packed, neutral (X'0F') sign when positive and (X'0D') when negative. | 1 – 16 | Xlen * 2 - 1 | P | Xlen |
| H | Hexadecimal. | 1 – 31742 | 0 | RD | 4 + Xlen[a] |
| J [j] | Mixed-case character string in the native code page (EBCDIC for z/OS, ASCII otherwise), with trailing zeroes ignored. | 1 – 31742 | 0 | V | Xlen[a] |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| K [k] | Binary, unsigned. | 1 & Xdec=0 | 0 | B | 2 |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2 - 11 & Xdec=0 | 0 | B | Xlen + 1 |
| | | 2 & Xdec>0 | 5 | P | 3 |
| | | 3 & Xdec>0 | 9 | P | 5 |
| | | 4 & Xdec>0 | 11 | P | 6 |
| | | 5 & Xdec>0 | 13 | P | 7 |
| | | 6& Xdec>0 | 15 | P | 8 |
| | | 7& Xdec>0 | 17 | P | 9 |
| | | 8& Xdec>0 | 21 | P | 11 |
| L | Long packed, up to 31 digits, (X'0F') sign when positive and (X'0D') when negative. Syntax P is recommended in place of this syntax. | 1 – 16 | Xlen * 2 - 1 | P | Xlen |
| L16 [e] | Unicode UTF-16-LE encoded string. | 2 – 31742, must be a multiple of 2 | 0 | UN | Xlen[a] |
| L16B [f] | Unicode UTF-16-LE encoded string with BOM. | 4 – 31742, must be a multiple of 2 | 0 | UN | Xlen-2[a] |
| L32 [g] | Unicode UTF-32-LE encoded string. | 4 – 31740, must be a multiple of 4 | 0 | UN | Xlen/2 |
| L32B [h] | Unicode UTF-32-LE encoded string with BOM. | 8 – 31740, must be a multiple of 4 | 0 | UN | (Xlen-4)/2 |
| M [l] | Numeric (zoned), unsigned. | 1 – 31 | Xlen | P | Xlen/2+1 (round down) |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| N [l] | Numeric (zoned), signed. | 1 – 31 | Xlen | P | Xlen/2+1 (round down) |
| NL | Numeric (zoned), signed, sign leading | 2-32 | Xlen - 1 | P | (XLen - 1) / 2 + 1 (round down) |
| NT | Numeric (zoned), signed, sign trailing | 2-32 | Xlen - 1 | P | (XLen - 1) / 2 + 1 (round down) |
| O | Packed, no sign stored (up to 31 digits). Assignment of negative values is not allowed. | 1 – 16 | Xlen * 2 - 1 | P | Xlen + 1 |
| P [m] | Packed, signed, (X'0C') sign when positive and (X'0D') when negative.(up to 31 digits). | 1 – 16 | Xlen * 2 - 1 | P | Xlen |
| Q[o] | Quoted character string. | 3 – 31742 | 0 | V | Xlen-2[a] |
| R [i, k] | Little-endian binary, unsigned. | 1 & Xdec=0 | 0 | B | 2 |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2 - 11 & Xdec=0 | 0 | B | Xlen + 1 |
| | | 2 & Xdec>0 | 5 | P | 3 |
| | | 3 & Xdec>0 | 9 | P | 5 |
| | | 4 & Xdec>0 | 11 | P | 6 |
| | | 5 & Xdec>0 | 13 | P | 7 |
| | | 6& Xdec>0 | 15 | P | 8 |
| | | 7& Xdec>0 | 17 | P | 9 |
| | | 8& Xdec>0 | 21 | P | 11 |
| RD | Raw data. | 5 – 31742 | 0 | RD | Xlen[a] |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| T | Text numeric. | 2 – 17 | Xlen - 2 | P | Xlen/2 (round down) |
| U [m] | Packed, unsigned, neutral (X'0F') sign when positive (up to 31 digits). Assignment of negative values is not allowed. | 1 – 16 | Xlen * 2 - 1 | P | Xlen |
| U8 | Unicode UTF8 encoded string. | 1 – 31742 | 0 | UN | Xlen (+1 if result is odd)[a] |
| U8N | Null terminated Unicode UTF8 encoded string. | 1 – 31742 | 0 | UN | Xlen (+1 if result is odd)[a] |
| U8B | Unicode UTF8 encoded string with BOM. | 4 – 31742 | 0 | UN | Xlen-3 (+1 if result is odd)[a] |
| U8NB | Null terminated Unicode UTF8 encoded string with BOM. | 4 – 31742 | 0 | UN | Xlen-3 (+1 if result is odd)[a] |
| UN [e] | Unicode. | 2 – 31742, must be a multiple of 2 | 0 | UN | Xlen[a] |
| V [j, o] | Variable character. | 1 – 31742 | 0 | V | Xlen[a] |
| W [o] | Double- & single-byte character string. | 4 – 31742 | 0 | W | Xlen[a] |
| X [j, o] | Fixed length, mixed case character string. | 1 – 31742 | 0 | V | Xlen[a] |
| XCnn [n] | Variable length, mixed case character string in one of a possible 16 user syntaxes. | 1 – 31742 | 0 | UN | Xlen (+1 if result is odd)[a] |

| External Syntax (Xsyn field) | Description | External Length (Xlen field) | Maximum External Decimal Length (Xdec field) | TIBCO Object Service Broker Syntax (Syn field) | TIBCO Object Service Broker Length (Len field) |
|---|---|---|---|---|---|
| Z [j, o] | X'00' fill character. | 1 – 31742 | 0 | V | Xlen[a] |

a.The maximum Len field value is 31723 or 31722 if the syntax requires the length to be even.

b.For the import type LENGTH_PREFIXED_EBCDIC_NATIVE_ENDIAN, the endian is that of the processor where TIBCO Object Service Broker is running. For example, on an Intel machine, the external syntax B for LENGTH_PREFIXED_EBCDIC_NATIVE_ENDIAN import files is little-endian.

c.The default mapping of B and E are identical.

d.On Open Systems, if there are more than 15 significant digits in the field, you must assign the TIBCO Object Service Broker syntax C and length 26.

e.The default mappings of B16, L16, and UN are identical.

f.The default mappings of B16B and L16B are identical.

g.The default mappings of B32 and L32 are identical.

h.The default mappings of B32B and L32B are identical.

i.External syntaxes E and R are not valid on z/OS.

j.The default mappings of J, V, X, and Z are identical.

k.The default mappings of K and R are identical.

l.The default mappings of M and N are identical.

m.The default mappings of P and U are identical.

n.There are 16 possible user syntaxes from XC01 to XC16. These are typically used to map DBCS characters to Unicode. To define the user syntaxes, refer to the procedures described in *TIBCO Object Service Broker for z/OS Installing and Operating* or *TIBCO Object Service Broker for Open Systems Installing and Operating*.

o.Data cleansing is supported for syntaxes A, C, Q, V, W, X, and Z.

About external data translation:

- Numeric nulls are translated to zeros. As a special case, null date fields are interpreted as zero and represented internally as 1980-01-01.
- Null fixed-length character strings are padded with blanks as required. Null variable-length strings (V or W) are imported as is.

# Appendix B Mapping File Names for Open Systems

This appendix describes how to map file names for Open Systems.

## Access to External Files

### Providing a File Name

On z/OS, you must provide a fully qualified data set name when opening a data set. On Open Systems, you can provide a full path name to your external files in your CALL to the file, for example:

```
CALL @OPENDSN('/usr1/osb/dsn/UNLOAD.TST');
CALL @OPENDSN('D:\usr1\osb\dsn\UNLOAD.TST');
```

or you can also provide partial path, for example:

```
CALL @OPENDSN('UNLOAD.TST');
```

If you provide an incomplete file name, the following facilities are used to direct the CALL to the appropriate directory:

- The DSDIR Execution Environment parameter
- Indirect name mapping through the filespec.dsn file

### Using the DSDIR Parameter to Provide a File Name

You can specify the name of the path to be used to create a full file name using the DSDIR Execution Environment parameter. For example, if you provide the value usr1/osb/dsn or D:\usr1\osb\dsn and the following call is made:

```
CALL @OPENDSN('UNLOAD.TST');
```

the full path is /usr1/osb/dsn/UNLOAD.TST or D:\usr1\osb\dsn\UNLOAD.TST.

## Using filespec.dsn to Provide a File Name

You can create a file called filespec.dsn to be used to map a z/OS data set name or Windows or Solaris file name to another value. For example, if the original file name is in mixed case and your TIBCO Object Service Broker application requires it to be in uppercase, you can map the values in filespec.dsn or if you require special characters (*, ?, <, >), map your file name to the filespec.dsn file to avoid having them replaced.

filespec.dsn must be in the same directory as the one you specified in the DSDIR Execution Environment parameter. Using an editor such as Notepad or vi, put each entry on a separate line in the following format:

```
originalfile filetouse rlength [filetype][fldsep][quote]
```

| | |
|---|---|
| *originalfile* | The name to be mapped; it can be the name of a z/OS partitioned or non-partitioned data set, or of a Windows or Solaris file. |
| *filetouse* | The Windows or Solaris file to be used.<br>If a relative file name is specified, the value for *filetouse* is appended to the directory specified in your DSDIR Execution Environment parameter. For example, if the value for *filetouse* is *file2use* and the directory named in the parameter is /usr1/osb/dsn the value for *originalfile* is mapped to the path /usr1/osb/dsn/file2use.<br>If a complete file name is specified, such as /usr1/osb/dsn/unload.tst, the value for *originalfile* is mapped to this file. |
| *rlength* | The length of each record in a native file format. Otherwise, specify 0 (zero). The last record in the file can be shorter than this length. |

| | |
|---|---|
| *filetype* | [Optional] It can be one of the following values.<br>• **LENGTH_PREFIXED_EBCDIC** – Native TIBCO Object Service Broker format. This data is stored in EBCDIC and big-endian format. The length of a record is determined by a length stored in the first two bytes of a record.<br>• **LENGTH_PREFIXED_EBCDIC_NATIVE_ENDIAN** – The native format for the platform. The length of the record is defined using the filespec.dsn mapping file.<br>• **LINE_SEPARATED_ASCII** – ASCII line-oriented data where the end of a line is delimited by the new line character or the end of the file.<br>If *filetype* is not specified, the default value depends on its use: if the file is used in file access calls (@OPENDSN, @READDSN, @WRITEDSN, and @CLOSEDSN, the default value is the value provided by the DSBIFTYPE Execution Environment parameter.<br>If the file is used in import or export tables, the default value is the value provided by the DSIXFTYPE Execution Environment parameter.<br>If *filetype* is not specified, the following options (*fldsep* and *quote*) must not be specified. |
| *fldsep* | Used only if *fieldtype* is LINE_SEPARATED_ASCII; otherwise it is ignored. It specifies the field separator to be used in LINE_SEPARATED_ASCII files for import and export tables. The value for *fldsep* must be one of the following:<br>• **NONE** – No field separator: input is in columns.<br>• **SPACE** – Spaces (tabs and blanks) are used as field separators.<br>• **TAB** – Only tabs are used as field separators.<br>• **COMMA** – Only commas are used as field separators.<br>If *fldsep* is not specified when *filetype* is LINE_SEPARATED_ASCII, the default value is NONE and can be modified by the DSFIELDSEP Execution Environment parameter. |

*quote*

Used only if *filetype* is LINE_SEPARATED_ASCII and *fldsep* is something other than NONE; otherwise it is ignored. It specifies the character to be used for quotation marks in LINE_SEPARATED_ASCII files being used for import and export tables. The value for *quote* must be one of the following:

- **NONE** – No quote characters.
- **SINGLE** – A single quotation mark (') is used as the quote character.
- **DOUBLE** – A double quotation mark (") is used as the quote character.

If *quote* is not specified when required, the default value is SINGLE and can be modified by the DSQUOTE Execution Environment parameter.

#### Example

If the DSDIR Execution Environment parameter is set to /usr1/osb/dsn, to map a file called /usr1/testone/workfile to D:\testone\workfile, the entry for filespec.dsn, is:

```
D:\testone\workfile /usr1/testone/workfile 0 LINE_SEPARATED_ASCII
TAB SINGLE
```

The file type is TEXT, tabs separate each field in the file, and the quotation mark character is a single quotation mark. Record length is ignored for TEXT files and therefore is set to zero (0).

## Mapping a Partitioned Data Set

If a z/OS partitioned data set name is being mapped in filespec.dsn, its name must be set with a "()" (parenthetic) suffix to distinguish it from non-partitioned data sets. The file attributes specified in the map entry are applied to all members in the data set. filetouse becomes the name of the directory for the members in the data set. The directory is created automatically. Names of members are mapped into uppercase file names to be located in the directory filetouse.

#### Example

The partitioned data set PART.DATA.SET has the following entry in filespec.dsn:

```
PART.DATA.SET() partdataset 0 TEXT SPACE SINGLE
```

If the DSDIR Execution Environment parameter is set to /usr1/osb/dsn, the member PART.DATA.SET(MEMBER) is mapped to the file /usr1/osb/dsn/partdataset/MEMBER.

See Also

- *TIBCO Object Service Broker Parameters* for information about the Execution Environment parameters

- *TIBCO Object Service Broker Shareable Tools* for information about the (@OPENDSN, @READDSN, @WRITEDSN, and @CLOSEDSN tools.

# Appendix C Data Cleansing

Data cleansing is a new feature that enables an installation to specify that, for certain external table types, the data must be cleaned up before being used by TIBCO Object Service Broker.

The table types that make use of data cleansing are IMP, VSM, MAP, IMS, IDM, and DAT.

## Topics


- Options, page 168
- Examples, page 170

## Options

Two options for data cleansing are available, as follows:

- Invalid numeric data can be ignored (the field in the table occurrence is set to a null value). This option can apply to all invalid numeric data or only to those whose input contains low values (all bytes X'00'). Numeric data are indicated by the fields with an external numeric syntax, that is, B, F, G, K, L, M, N, NL, NT, O, P, T, and U.
- Character data can have undesirable characters removed or replaced. Character data are indicated by the fields with an external character syntax, that is, A, C, Q, V, W, X, and Z.

You can use the Table Definer to specify data cleansing for a particular table. The Data Cleansing field on the primary Table Definer screen specifies the data cleansing options for the table in question.

The possible options, which you can abbreviate, are as follows:

- LOWVALUES — Treat invalid numeric data as null if low values (all bytes X'00').
- INVALID — Treat invalid numeric data as null.
- TRLOAD=*modname* — Load one or more translate tables from the load module with the name *modname*.
- TRTABLE=*iname* — Load one or more translate tables from the @TRTABLES instance with the TRNAME parameter value *iname*.

For example, INV,TRL=MYTRANS denotes the following:

- Any invalid numeric data input is to be treated as a null field.
- Translate tables for processing character data are to be provided by a load module with the name MYTRANS.

The translate tables to be used are specified as follows:

- For a load module, which must be in a load library accessible to the DOB, there can be one or two entries, each 264 bytes long. Each entry contains an 8-byte blank padded field, which denotes the type of translate table, either REPLACE or DELETE, followed by a 256-byte translate table.
- For a @TRTABLES instance, there can be one or two occurrences. The key field TYPE (IC8) can be REPLACE or DELETE. The field TRTABLE (V256) must contain the corresponding translate table.

If the type is REPLACE, then the 256-byte translate table represents a regular translate table in which each input data byte is replaced by the byte at the corresponding offset in the table.

If the type is DELETE, then the 256-byte translate table contains X'00' bytes for the characters that are to be retained, and non-zero bytes for the input characters that are to be deleted.

Please be aware that changes made to a translate table (either load module or @TRTABLES format) will not be seen by any active table that references that translate table. This is because the translation data is bound to the table definition when the table is first referenced by any user. To be certain of using the updated translate table, it may be necessary to recycle the DOB.

## Examples

Following are a few examples:

- Table SALES2011 is an IMS table that might contain uninitialized packed decimal data represented as low values. To ignore that invalid data, set the Data Cleansing field in the Table Definer to LOW. The input character data will be unchanged.
- Table MYIMP1 is an IMP table that contains character data you would like to clean up before processing it. In addition, to ignore any invalid numeric data, set the Data Cleansing field to INV,TRLOAD=CLEANUP. The load module CLEANUP is based on the following Assembler source file:

```
CLEANUP  CSECT
         DC      CL8'REPLACE'          Function = character replacement
TABLE1   DC      256AL1(*-TABLE1)
         ORG     TABLE1+X'80'
         DC      C' '                  Replace X'80' with a blank
         ORG     TABLE1+X'FA'
         DC      6C' '                 Replace X'FA' to X'FF' with blanks
         ORG     ,
         DC      CL8'DELETE'           Function = character removal
TABLE2   DC      XL256'00'
         ORG     TABLE2+X'0A'
         DC      4X'01'                Characters X'0A'-X'0D' to be deleted
         ORG     ,
         END
```

Using this definition, any carriage returns, line feeds, or form feeds will be removed from the input data before processing. Also, several undesirable characters will be replaced with blanks should they exist in the input data.

- Table MAP47 is a MAP table with character fields that contain X'FF', which you would like to remove. To do so, set Data Cleansing for the table definition to TRTAB=NOFF.
- Table instance @TRTABLES(NOFF) contains one occurrence with the field TYPE = 'DELETE' and the field TRTABLE that contains a 256-byte string with the first 255 bytes zero and the last one, non-zero.

# Index

## Symbols

@CLOSEDSN tool 163, 165
@HRNXDEL return code 91, 117
@HRNXDON return code 91, 117
@HRNXINX return code 91, 117
@HRNXKEP return code 91, 117
@OPENDSN tool 163, 165, 165
@READDSN tool 163, 165
@WRITEDSN tool 163, 165
* (dynamic) external syntax 153

## A

A (alphabetic uppercase) external syntax 153
Acc field
  export tables 104
  import tables 61
  VSAM tables 129
ACCESSFAIL exception 89, 114, 151
accessing
  exported data 112, 113
  external data 50
  external files 161
  import tables 80
  remote tables 82
  VSAM data 120
  VSAM tables 144
Adabas data
  accessing 2, 10
  processing 111
Administration menu, and Service Gateway for Files 45
ALLOCATE command 55, 100, 125
alphabetic uppercase (A) external syntax 153

## B

B16 (Unicode UTF 16-BE) external syntax 154
backup files, transferring with FTP 55
binary
  signed (B) external syntax 154
  unsigned (K), external syntax 156

## C

C, fixed length (external syntax) character string 154
Change Tracking Agent 6
CICSVSAMSYNC Execution Environment parameter 149
commands
  ALLOCATE 55, 100, 125
  D (delete) 58, 102, 128
  P (primary key) 64, 107
  SELECT 80, 144
  SELECT LIKE 80, 144
communications, setting up for Service Gateway for Files 40
configuring
  Service Gateway for Files, sample configuration for z/OS 38
  TCP/IP 30
Copy Table option, using 112
COPY_DATA tool, using 112
CTABLESIZE parameter 62, 105, 130
customer support xviii

## D

D line command 58, 102, 128

data
  accessing
    external 50
    VSAM 120
  exchanging 4
  managing
    import definitions 49
    VSAM definitions 119
data cleansing
  examples 170
  options 168
Data Cleansing field, import tables 56
data identification segment, illustrated
  export tables 99
  import tables 54
  VSAM tables 124
data parameter validation 58
data set requirements 135
Data Set Type field, VSAM tables 125
DDname field
  export tables 100
  import tables 55
  VSAM tables 124
Dec field
  export tables 108
  import tables 65
  VSAM tables 133
Default field
  export tables 108
  import tables 65
  VSAM tables 133
defining
  fields
    export tables 105
    import tables 62
    VSAM tables 130
  import tables for multiple record formats 68
  VSAM tables for multiple record formats 138
DEFINITIONFAIL exception 89, 115, 151
definitions, managing
  export data 95
  import data 49
  VSAM data 119
delete (D) line command 58, 102, 128
DELETEFAIL exception 151
dictionary, global field 65, 108
double-byte and single-byte character string (W)
    external syntax 158
DSIXFTYPE Execution Environment parameter 54, 99
dynamic (*) external syntax 153

## E

E (little-endian binary signed) external syntax 155
ERROR exception 89, 114, 150
error handling 89, 114, 150
ESDS requirements 135
ESTIMATETBLDEFN rule 62, 105, 130
Event Rule field
  export tables 104
  import tables 60
  VSAM tables 129
event rule segment, illustrated
  export tables 104
  import tables 60
  VSAM tables 129
event rules, specifying
  export tables 104
  import tables 60
  VSAM tables 129
exceptions
  ACCESSFAIL 89, 114, 151
  DEFINITIONFAIL 89, 115, 151
  DELETEFAIL 151
  ERROR 89, 114, 150
  GETFAIL 151
  INSERTFAIL 151
  INTEGRITYFAIL 89, 115, 151
  LOCKFAIL 89, 115, 151
  REPLACEFAIL 151
  SECURITYFAIL 89, 115, 151
  SERVERERROR 151
exchanging data 4
EXP server. *See* export server 3
export files
  FTPing 99
  writing data to 112
export server, overview 3

export tables
Acc field 104
data identification segment 99
data parameters, specifying 102
DDname field 100
Dec field 108
Default field 108
defining fields 105
Event Rule field 104
event rule segment 104
External Routine Name field 101
field definition segment 105
Field Name field 106
File field 99
Globalfield Name field 108
identifying data 99
identifying tables 98
internal attributes, specifying 107
Key field 107
Len field 107
location parameter, specifying 102, 102, 128
Offset field 106
Ord field 108
parameter segment 102
Rqd field 108
Server ID field 101
specifying external attributes 106
Syn field 107
Table field 98
table identification segment 98
Typ field 104, 107
Type field 98
Unit field 98
Xdec field 106
Xlen field 106
Xsyn field 106
exported data, accessing 112, 113
exporting
managing data definitions 95
Table Definer illustrated 96
external attributes, specifying
export tables 106
import tables 62
VSAM tables 131
external data set name parameter 91, 117
external data, accessing 50
external files 161
external files, writing TIBCO Object Service Broker tables to 96
External Routine Name field
export tables 101
external routines
manipulating 90
manipulating data 116
overview 116
parameters passed 90, 116
pre-processing data 90
valid return codes 91, 117
external syntaxes
alphabetic uppercase (A) 153
binary signed (B) 154
binary unsigned (K) 156
dynamic (*) 153
fixed length character string (C) 154
fixed length mixed case character string (X) 158
floating point (F) 155
hexadecimal (H) 155
little-endian binary
signed (E) 155
unsigned (R) 157
long packed signed (L) 156
mixed-case character string (J) 155
numeric signed (N) 157
numeric unsigned (M) 156
packed neutral (G) 155
packed signed (P) 157
packed unsigned (U) 158
packed, no sign stored (O) 157
quoted character string (Q) 157
raw data (RD) 157
text numeric (T) 158
U8 (Unicode UTF8 encoded string) 158
U8B (Unicode UTF8 encoded string with BOM) 158
U8N (Null terminated Unicode UTF8 encoded string) 158
U8NB (Null terminated Unicode UTF8 encoded

string with BOM) 158
UN (Unicode) 158
Unicode UTF 16-BE (B16) 154
Unicode UTF 16-LE (L16) 156
Unicode UTF 16-LE with BOM (L16B) 156
Unicode UTF 32-LE (L32) 156
Unicode UTF 32-LE with BOM (L32B) 156
variable character (V) 158
X'00' fill character (Z) 159

## F

F, floating point external syntax 155
field definition segment, illustrated
export tables 105
import tables 62
VSAM tables 130
Field Name field
export tables 106
import tables 62
VSAM tables 131
fields
Acc 61, 104, 129
Data Cleansing 56
Data Set Type 125
DDname 55, 100, 124
Dec 65, 108, 133
Default 65, 108, 133
defining
export tables 105
import tables 62
VSAM tables 130
Event Rule 60, 104, 129
External Routine Name 101
Field Name 62, 106, 131
File 54, 99, 124
Globalfield Name 65, 108, 133
IDgen 53, 123
Ignore 125
Key 64, 107, 132
Len 64, 107, 132
Load 125
Offset 63, 106, 131
Ord 65, 108, 133
Read Only 125
Rqd 65, 108, 133
Server ID 56, 101, 126
Syn 64, 107, 132
Table 52, 98, 122
Typ 60, 64, 104, 107, 129, 132
Type 52, 98, 122
Unit 52, 98, 122
Xdec 63, 106, 131
Xlen 63, 106, 131
Xsyn 63, 106, 131
File field
export tables 99
import tables 54
VSAM tables 124
file, filespec.dsn 99
filespec.dsn file 99
fixed length
character string (C) external syntax 154
mixed case character string (X) external syntax 158
flagging changed copybooks 6
floating point (F), external syntax 155
FORALL statement 81, 146
FTP, using to transfer backup files 55
FTPing
export files 99
import files 55
full definition, location parameter 59, 103, 128

## G

G (packed neutral) external syntax 155
GET statement 81, 146
GETFAIL exception 151
global field dictionary 65, 108
Globalfield Name field
  export tables 108
  import tables 65
  VSAM tables 133

## H

H (hexadecimal) external syntax 155
handling
  errors 89, 114, 150
  TIBCO Object Service Broker requests 89, 114, 149
hexadecimal (H) external syntax 155
HRNXDSN parameter 91, 117
HRNXRECA parameter 90, 116
HRNXRECE parameter 91, 117
HRNXRLEN parameter 90, 116
HRNXTABN parameter 91, 117
HRNXTPRM parameter 91, 117
HRNXWRKA parameter 90, 116

## I

I/O record buffer parameter 90, 116
identifying
  data
    export tables 99
    import tables 54
    VSAM tables 124
  tables
    export 98
    import 52
    VSAM 122
IDgen field 53, 123
Ignore field, VSAM tables 125
import data definitions, managing 49
import files, FTPing 55
import tables 62, 64
  Acc field 61
  accessing 80
  Data Cleansing field 56
  data identification segment illustrated 54
  data parameters, specifying 57
  DDname field 55
  Dec field 65
  Default field 65
  defining fields 62, 130
  Event Rule field 60, 60
  event rule segment illustrated 60
  event rules, specifying 60
  external attributes, specifying 62
  field definition segment illustrated 62
  Field Name field 62
  File field 54
  Globalfield Name field 65
  identifying data 54
  identifying tables 52
  IDgen field 53
  internal attributes, specifying 64
  Key field 64
  Len field 64
  location parameter, specifying 57
  multiple record formats, defining 68
  Offset field 63
  Ord field 65
  parameter segment illustrated 57
  Rqd field 65
  Server ID field 56
  Syn field 64
  Table Definer illustrated 50
  Table field 52
  table identification segment illustrated 52
  Typ field 64
  Type field 52
  Unit field 52
  Xdec field 63
  Xlen field 63
  Xsyn field 63
IMPXPARM macro 90
INSERTFAIL exception 151
installation process 22

INTEGRITYFAIL exception 89, 115, 151
internal attributes, specifying
  export tables 107
  import tables 64
  VSAM tables 132
invoking the Table Definer 5

## J

J (mixed-case character string) external syntax 155

## K

K (binary unsigned) external syntax 156
Key field
  export tables 107
  import tables 64
  VSAM tables 132
KSDS requirements 135

## L

L, long packed signed external syntax 156
L16 (Unicode UTF 16-LE) external syntax 156
L16B (Unicode UTF 16-LE with BOM) external syntax 156
L32 (Unicode UTF 32-LE) external syntax 156
L32B (Unicode UTF 32-LE with BOM) external syntax 156
LDS,VSAM 2
Len field
  export tables 107
  import tables 64
  VSAM tables 132
line commands
  D (delete) 58, 102, 128
  P (primary key) 64, 107
Linear Data Sets, VSAM 2
listing members of a PDS or files in a directory 85
  example for Windows and UNIX 87
  example for z/OS 85
little-endian binary
  signed (E) external syntax 155
  unsigned (R) external syntax 157
Load field, VSAM tables 125
location parameter
  full definition 59, 103, 128
  minimal definition 58, 102, 128
  specifying for
    export tables 102, 128
    import tables 58
LOCKFAIL exception 89, 115, 151
log files, for Service Gateway for Files 45
long packed signed (L) external syntax 156

## M

M (numeric unsigned) external syntax 156
macro, IMPXPARM 90
managing
  export data definitions 95
  import data definitions 49
  VSAM data definitions 119
manipulating
  data with external routines 116
  external routines 90
minimal definition, location parameter 58, 102, 128
mixed-case character string (J) external syntax 155
multiple record formats
  defined 68, 138
  defining VSAM tables 138
  definition requirements 68
  VSAM tables 145

## N

N (numeric signed) external syntax 157
NLS, and Service Gateway for Files 46
Null terminated Unicode UTF8 encoded string (U8N)

external syntax 158
Null terminated Unicode UTF8 encoded string with BOM (U8NB) external syntax 158
numeric
signed (N) external syntax 157
unsigned (M) external syntax 156

## O

O (packed, no sign stored) external syntax 157
Offset field
export tables 106
import tables 63
VSAM tables 131
Ord field
export tables 108
import tables 65
VSAM tables 133

## P

P
line command 64, 107
packed signed external syntax 157
packed
neutral (G) external syntax 155
no sign stored (O) external syntax 157
signed (P) external syntax 157
unsigned (U) external syntax 158
parameter segment, illustrated
export tables 102
import tables 57
VSAM tables 127
parameters
CTABLESIZE 62, 105, 130
data
export tables, specifying 102
import tables, specifying 57
validation 58
VSAM tables, specifying 127
HRNXDSN 91, 117
HRNXRECA 90, 116
HRNXRECE 91, 117
HRNXRLEN 90, 116
HRNXTABN 91, 117
HRNXTPRM 91, 117
HRNXWRKA 90, 116
location
export tables, specifying 102, 128
full definition 59, 103, 128
import tables, specifying 57, 57
minimal definition 58, 102, 128
VSAM tables, specifying 102, 127
passed to external routines 90, 116
primary commands
SELECT 80, 144
SELECT LIKE 80, 144
primary key line command 64, 107
processing records 126

## Q

Q quoted character string external syntax 157
quoted character string (Q) external syntax 157

## R

R (little-endian binary unsigned) external syntax 157
raw data (RD) external syntax 157
RD (raw data) external syntax 157
Read Only field for VSAM tables 125
record length parameter 90, 116
records, processing 126
recovery 114, 149
remote table access 82
repeating groups
defined 68, 138
import tables 68
VSAM tables 145
REPLACEFAIL exception 151

requests, handling for TIBCO Object Service Broker 89, 114, 149
requirements
  data sets 135
  ESDS 135
  KSDS 135
  RRDS 136
retrieval processing 81, 146
return codes
  @HRNXDEL 91, 117
  @HRNXDON 91, 117
  @HRNXINX 91, 117
  @HRNXKEP 91, 117
routines, external 116
Rqd field
  export tables 108
  import tables 65
  VSAM tables 133
RRDS requirements 136
rules
  ESTIMATETBLDEFN 62, 105, 130
  using 80, 112, 145
running the Change Tracking Agent 6

## S

SDK, Service Gateway for Files 2
SECURITYFAIL exception 89, 115, 151
segments
  data identification 54, 99, 124
  event rule 60, 104, 129
  field definition 62, 105, 130
  parameter 57, 102, 127
  table identification 52, 98, 122
SELECT LIKE primary command 80, 144
SELECT primary command 80, 144
Server ID field
  import tables 56
  VSAM tables 126
Server ID field, export tables 101
Server User ID, and Service Gateway for Files 43
SERVERERROR exception 151
SERVERID parameter
  export table 101
  import table 56
  VSAM table 126
Service Gateway for Files 10–47
  log files 45
  managing 43
  monitoring 45
  National Language Support (NLS) 46
Service Gateway for Files SDK, description 2
shutting down, Service Gateway for Files 43
single and double byte character string (W) external syntax 158
specifying
  data parameters
    export tables 102
    import tables 57
    VSAM tables 127
  event rules
    export tables 104
    import tables 60
    VSAM tables 129
  external attributes
    export tables 106
    import tables 62
    VSAM tables 131
  internal attributes
    export tables 107
    import tables 64
    VSAM tables 132
  location parameter
    export tables 102
    import tables 57
    VSAM tables 127
  table types 5
starting, Service Gateway for Files 43
statements
  FORALL 81, 146
  GET 81, 146
  TRANSFERCALL 60
support, contacting xviii
Syn field
  export tables 107
  import tables 64
  VSAM tables 132

synchronization 114, 149
synchronization under CICS 149

## T

T (text numeric) external syntax 158
Table Browser, using 80, 144
Table Definer for
  export tables, illustrated 96
  import tables, illustrated 50
  VSAM tables, illustrated 120
Table Definer, invoking 5
Table Editor, using 144
Table field
  export tables 98
  import tables 52
  VSAM tables 122
table identification segment, illustrated
  export tables 98
  import tables 52
  VSAM tables 122
table name parameter 117
table parameter value 117
table types, specifying 5
tables
  identifying as export 98
  import
    accessing 80
    defining for multiple record formats 68
  names, parameters 91
  parameter values 91
  remote, access 82
  VSAM
    accessing 144
    defining for multiple record formats 138
    multiple record formats 145
    repeating groups 145
TCP/IP, configuration of 30
technical support xviii
text numeric (T) external syntax 158
TIBCO Object Service Broker tables, writing to external files 96
TIBCO_HOME xv
tools
  @CLOSEDSN 163, 165
  @OPENDSN 163, 165, 165
  @READDSN 163, 165
  @WRITEDSN 163, 165
  NLS 46
tools, COPY_DATA, using 112
tracking changes in copybooks 6
TRANSFERCALL statement 60
transferring backup files with FTP 55
Typ field
  export tables 104, 107
  import tables 60, 64
  VSAM tables 129, 132
Type field
  described 52, 122
  export tables 98

## U

U (packed unsigned) external syntax 158
U8 (Unicode UTF8 encoded string) external syntax 158
U8B (Unicode UTF8 encoded string with BOM) external syntax 158
U8N (Null terminated Unicode UTF8 encoded string) external syntax 158
U8NB (Null terminated Unicode UTF8 encoded string with BOM) external syntax 158
UN (Unicode) external syntax 158
Unicode
  UTF 16-BE (B16) external syntax 154
  UTF 16-LE (L16) external syntax 156
  UTF 16-LE with BOM (L16B) external syntax 156
  UTF 32-LE (L32) external syntax 156
  UTF 32-LE with BOM (L32B) external syntax 156
  UTF8 encoded string (U8) external syntax 158
  UTF8 encoded string with BOM (U8B) external syntax 158
Unicode (UN) external syntax 158
uninstallation process 35

Unit field
  export tables 98
  import tables 52
  VSAM tables 122
using
  Copy Table option 112
  COPY_DATA tool 112
  rules 80, 112, 145
  Table Browser 80, 144
  Table Editor 144

## V

V, variable character external syntax 158
valid return codes 91, 117
validation for data parameters 58
variable
  character (V) external syntax 158
VSAM 122
  accessing data 120
  managing data definitions 119
  Table Definer illustrated 120
VSAM LDS data
  accessing 2, 10
  processing 143
VSAM server overview 3
VSAM tables 131
  Acc field 129
  accessing 144
  data identification segment illustrated 124
  data parameters, specifying 127
  Data Set Type field 125
  data, identifying 124
  DDname field 124
  Dec field 133
  Default field 133
  Event Rule field 129, 129
  event rule segment illustrated 129
  external attributes, specifying 131
  field definition segment, illustrated 130
  Field Name field 131
  File field 124
  Globalfield Name field 133
  IDgen field 123
  Ignore field 125
  internal attributes, specifying 132
  Key field 132
  Len field 132
  Load field 125
  location parameters, specifying 127
  multiple record formats 145
  Offset field 131
  Ord field 133
  parameter segment illustrated 127
  Read Only field 125
  repeating groups 145
  Rqd field 133
  Server ID field 126
  Syn field 132
  Table field 122
  table identification segment illustrated 122
  tables, identifying 122
  Typ field 132
  Type field 122
  Unit field 122
  Xdec field 131
  Xlen field 131
  Xsyn field 131
VSM server. *See* VSAM server.

## W

work area parameter 90, 91, 116, 117
writing
  data to export files 112
  TIBCO Object Service Broker tables to external files 96

## X

X (fixed length mixed case character string) external syntax 158
X'00' fill character (Z) external syntax 159
Xdec field
  export tables 106
  import tables 63
  VSAM tables 131
Xlen field
  export tables 106
  import tables 63
  VSAM tables 131
Xsyn field
  export tables 106
  import tables 63
  VSAM tables 131

## Z

Z (X'00' fill character) external syntax 159