# TIBCO® Object Service Broker

## Managing Data

*Software Release 6.0*
*July 2012*

two-second advantage™

TIBCO®
The Power of Now®

## 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

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

This manual describes how to define, manipulate, and manage data that is required for a TIBCO Object Service Broker application. It does not cover data that is stored in external data sources; refer to the appropriate *Service Gateway* manuals for external data accesses.

## Topics

# 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 rules 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 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 Overview of Data Structures

This chapter provides an overview of the data structures used by TIBCO Object Service Broker.

## Topics

# Relationships of Data

## How is Data Viewed in TIBCO Object Service Broker?

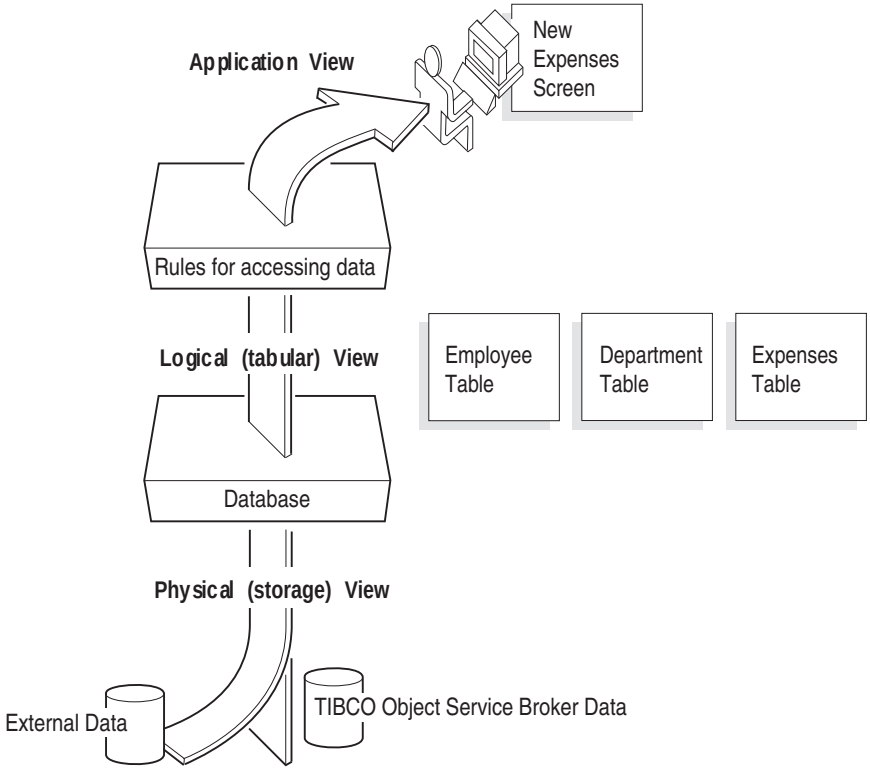
TIBCO Object Service Broker uses, accesses, and stores data from three different perspectives, as shown in the following illustration. The three views are referred to as:

- Application view
- Logical view
- Physical store view

The applications that you write use the application and logical views only to access the data that they require. For more detail, refer to the following sections.

## Three Views of Data

The following diagram illustrates the relationship between these views:



# Views of Data

## First Level: Application View

The application view is the view of data as it is presented to an end user. It is a selective view of data that you as an application developer define.

When you create an application you use many component pieces, for example, presentation screens, input fields, and report writing capabilities. In TIBCO Object Service Broker, all these diverse components are stored as tables. Your application requires all or only some of the data that is defined to these tables.

### How Do You Selectively View Data?

You selectively view data by writing programs that reference specific fields within specific tables. Within TIBCO Object Service Broker this construction is referred to as a *table.field* (*t.f*) format.

## Second Level: Logical View

The logical view is a tabular view of data. You use this view to store and access data. Using a definer, you must define a tabular view of data before an application can use the data.

## Third Level: Physical Store

The actual data that your application uses is stored in physical storage devices. The data is accessed via a server layer referred to as the TIBCO Object Service Broker engine. This physical store can contain data known as TIBCO Object Service Broker data that is native to TIBCO Object Service Broker. It can also contain data, known as external data, that belongs to external database management systems (DBMSs) and external operating systems.

# Your Data Store

## What is Your Data Store?

Your data store is any data that can be accessed from a TIBCO Object Service Broker system. In TIBCO Object Service Broker, your data is stored in a Pagestore. Other DBMSs use their own storage structures. Your data can reside on your TIBCO Object Service Broker local or remote node, or in files and databases external to TIBCO Object Service Broker. This data can be persistent or transitory (non-persistent) in nature. For more information on the Pagestore, refer to Storage of TIBCO Object Service Broker Data on page 7.

## Stored Types of Data

The type of data that you access determines the range of operations and behavior that your data exhibits. You can access both TIBCO Object Service Broker and external data.

### TIBCO Object Service Broker Data

TIBCO Object Service Broker data is data that belongs to TIBCO Object Service Broker. TIBCO Object Service Broker determines its storage, allowable syntax and semantic type, and accessibility. It is physically stored as a table in the Pagestore. You can access Pagestores that are local or remote to your node.

### External Data

External data is data stored outside of TIBCO Object Service Broker. An external operating system or DBMS controls its storage, its syntax and semantic type, and some aspects of its security. The physical storage structure of the data depends on its DBMS or operating system. TIBCO Object Service Broker uses an additional definition layer to translate the logical structure to the physical structure. This definition layer is provided through the appropriate Table Definer.

## Persistent Data

Persistent data is data that is physically stored in a storage device. This data can be either TIBCO Object Service Broker or external data. Data that is accessed via the following types of tables is persistent in nature:

| Table Type | Description | External Data |
|---|---|---|
| ADA | Accesses Adabas data. | Y |
| DAT | Accesses Datacom data. | Y |
| DB2 | Accesses DB2 data. | Y |
| EXP | Exports data to an external file. | Y |
| IDM | Accesses CA-IDMS data. | Y |
| IMS | Accesses IMS/DB data. | Y |
| IMP | Imports data from an external file. | Y |
| SLK | Accesses various types of SQL databases. | Y |
| SUB | Holds a subview of data stored in another type of table. | |
| TDS | Contains data stored in the default TIBCO Object Service Broker format. | N |
| VSM | Accesses data stored in VSAM files. | Y |

## Non-persistent Data

Non-persistent data is data that lives only for the duration of a TIBCO Object Service Broker session or transaction. The data is based on some other data and is not stored in a physical storage device. The following table types are used for data that is non-persistent:

| Table Type | Description |
|---|---|
| CLC | Holds calculated values based on data stored in a TDS table. |
| EES | Holds data in Execution Environment level storage. The data lasts for the duration of the Execution Environment. |

| Table Type | Description |
|---|---|
| MAP | Interfaces a memory storage area (DSECT or STRUCT). |
| PRM | Holds the parameter values for a parameterized TDS table. |
| RPT | Holds data for report generation. The data lasts as long as your transaction. |
| SCR | Holds data for the purpose of a display.<br>It is used to display a text-based screen via a display statement within a rule. The data lasts as long as your transaction unless a DISPLAY & TRANSFERCALL statement is used in as the display statement, in which case the data is brought into the subsequent transaction. |
| SES | Holds data in session managed storage. The data lasts for the duration of your session. |
| TEM | Holds data in transaction level storage. The data lasts for the duration of the transaction. |

# Storage of TIBCO Object Service Broker Data

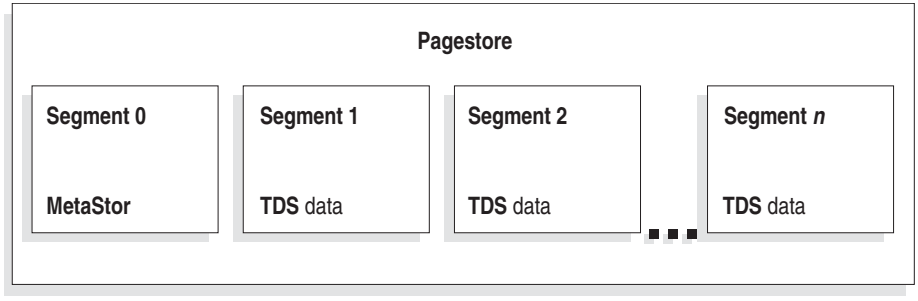
TIBCO Object Service Broker data is stored in TDS tables. These tables are stored in the Pagestore.

## Pagestore

The Pagestore is a physical store of data that is divided up into segments. A segment holds TDS data. TDS storage supports the B+ tree index structure, which allows fast direct access to a sequenced set of data. Each autonomous TIBCO Object Service Broker system, known as a node, has its own Pagestore.

### Description of the Data

The description of the data, including its location, is contained in a metadata file called the MetaStor. The MetaStor is located on segment 0. The following illustration shows this inter-relationship:



# Use of Tables in TIBCO Object Service Broker

## What is a Table?

A table is a logical view of your stored data. At a conceptual level, each table is comprised of rows and columns. A row is referred to as an occurrence and a column is referred to as a field.

You must create a table definition before a table exists as an object that can be used by other objects.

## What is the Purpose of a Table?

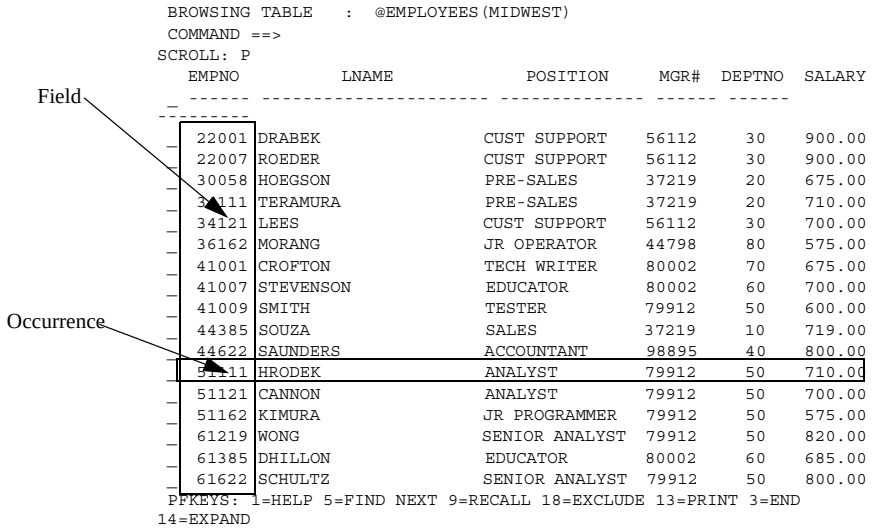
In TIBCO Object Service Broker, tables are used to store data that is native to TIBCO Object Service Broker, access data that is external to TIBCO Object Service Broker, provide a location for the data, and access data that is contained in other tables.

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to its online help.

## What Does a Table Look Like?

The following illustration shows how a TDS table, which is the default table type, appears to a user when browsing the table:



# Available Table Types

## What is a Table Type?

The table type is a characteristic of a table that determines how data can be stored in the table or how data can be accessed from the table. When you define a table you must define its table type.

## Determining the Appropriate Table Type

The following table describes the table types available, what each type is used for, and where you can get additional information about it.

| To … | Specify a table of type … | | Refer to the following manual… |
|---|---|---|---|
| Access Adabas databases. | ADA | TIBCO Object Service Broker Adabas table. | *TIBCO Service Gateway for Files SDK* |
| Count a unique combination of field values in TDS tables. | CLC | Calculation table. | This manual |
| Access CA Datacom databases. | DAT | TIBCO Object Service Broker Datacom table. | *TIBCO Service Gateway for Files SDK* |
| Access DB2 databases. | DB2 | TIBCO Object Service Broker DB2 table. | *TIBCO Service Gateway for DB2 Installing and Operating* |
| Create a temporary table for the life of an Execution Environment. | EES | Execution Environment table | This manual |
| Export data to a sequential file. | EXP | Export table. | *TIBCO Service Gateway for Files Installing and Operating* |

| To … | Specify a table of type … | | Refer to the following manual… |
|---|---|---|---|
| Access CA-IDMS databases. | IDM | TIBCO Object Service Broker IDMS/DB table. | *TIBCO Service Gateway for IDMS/DB Installing and Operating* |
| Import data from a sequential file. | IMP | Import table. | *TIBCO Service Gateway for Files Installing and Operating* |
| Access IMS/DB databases. | IMS | TIBCO Object Service Broker IMS table. | *TIBCO Service Gateway for IMS/DB Installing and Operating* |
| Interface to an external program through data. | MAP | MAP table. | This manual |
| List all the parameter values for a data table. | PRM | Parameter table. | This manual |
| Create a report. | RPT | Report table. | *TIBCO Object Service Broker Defining Reports* |
| Create a text-based screen. | SCR | Screen table. | *TIBCO Object Service Broker Defining Screens and Menus*. |
| Create a temporary table for the life of a session. | SES | Session table. | This manual |
| Access SQL databases. | SLK | TIBCO Object Service Broker SLK table. | *TIBCO Service Gateways for ODBC and for Oracle Installing and Operating* |
| Provide a selection view of a table with one or all of the following: excluded fields, derived fields, or derived values. | SUB | Subview table. | This manual |

| To … | Specify a table of type … | | Refer to the following manual… |
|---|---|---|---|
| Store data in the default TIBCO Object Service Broker format. | TDS | Table data store table. | This manual |
| Create a temporary table for the life of a transaction. | TEM | Temporary table. | This manual |
| Access VSAM files. | VSM | TIBCO Object Service Broker VSAM table. | *TIBCO Service Gateway for Files Installing and Operating* |

# Re-usability of Data

Much of the data available to you can be used in a number of different ways. By choosing the appropriate table types for your applications, you can limit the number of objects that require modifications as your applications evolve.

## Design Options

The following design options can be used to promote re-usability of application code and data. You can use one or a combination of these design options.

| Option | Description |
|---|---|
| 1 | Parameterized tables. |
| 2 | Minimal definitions. |
| 3 | Non-persistent data. |
| 4 | Global fields. |

## Option 1: Using Parameterized Tables

A parameterized table is a table with one or more defined attributes that are used to partition the data in the table. These attributes can be specific to the actual data, the location of the data, or a combination of both the data and the location.

- Parameter attributes that are specific to data are referred to as data parameters.
- A parameter attribute that is specific to a location is referred to as a location parameter. You would use a location parameter if you require access to the table from a number of different TIBCO Object Service Broker nodes.

### Example of Data Parameterization

There are many employees submitting expenses to a company and it is important to know when individual expenses are processed. In a case such as this, you could define a parameterized expense table using employee number and processing date as the parameters, for example, `EXPENSES(empno, procdate)`.

When the data is accessed, you need to reference only the instance of the table that is specific to your requirements. For example, in a rule, to get the first expense for employee number 79912 for the date 2000-03-15, you would use the statement:

```
GET EXPENSES(77912, '2000-03-15');
```

If required, you can access all the instances of the table by defining a parameterized (PRM) table on the base table. Refer to Defining a Parameter Value (PRM) Table on page 55 for more information.

### Example of Location Parameterization

A company stores data on one node and want access to the data from other nodes. For example, all the information regarding exchange rates is stored at a central location and employee expenses are stored at the location where each employee works. In this case, the location of a rates table and its data can be specified through the use of a location parameter and it is possible that the expenses table does not require a location parameter.

## Option 2: Using Minimal Definition

A minimal definition is a table definition that consists of only a table name and a location parameter. It is used in environments where the same data is accessed from a number of different nodes. The minimal definition points to the full definition and data located on a node remote to where the initial access is being made to the data.

## Option 3: Using Non-persistent Data

Often, when building applications, a particular view of data is required that can be obtained from an already existing data source. Because the real data already exists, less maintenance and storage overhead is incurred if you can make use of non-persistent data in your applications.

### Table Types that Use Non-persistent Data

The following table types use non-persistent data:

| | |
|---|---|
| CLC | Calculation table. |
| EES | Execution Environment table |
| PRM | Parameter value table. |
| RPT | Report table. |

| | |
|---|---|
| SCR | Screen table. |
| SES | Session table. |
| TEM | Temporary table. |

Refer to Defining a Calculation (CLC) Table on page 52 for information on defining calculation (CLC), parameter value (PRM), Execution Environment (EES), session (SES), and temporary (TEM) tables.

### Example of Using Non-persistent Data

The data required for a parameter value (PRM) table, which holds the values of the data parameters of a parameterized table, can be obtained from the data already stored for the parameterized table. You do not need to add additional data to the data store to be able to use these values. However, it is possible that you require this specialized view of data to meet the requirements of your application.

## Option 4: Using Global Fields

A global field is a field defined by an application administrator to be used across applications. The definition of the field and the help associated with it are stored in the @GLOBALFIELDS table. A link to the table can be made when you are defining the fields of your tables. Your administrator determines your site's global fields implementation. Refer to Selecting Global Fields on page 27 for more information.

# Accessing Table Data

## How Do I Access Data?

You access data through TIBCO Object Service Broker tools, programs that are written in the TIBCO Object Service Broker rules language, and programs that are external to TIBCO Object Service Broker. Language accesses to your external and TIBCO Object Service Broker data are specified using a *table.field* construction. You can use the same methods to access external and TIBCO Object Service Broker data.

## Requirements before Accessing Data

Before you can access data, you must meet the following requirements:

1. You must have a table defined for the data.
2. You must have security access to the table definition and data.
3. The table must be loaded with data.
4. If the table is located on a remote node, the server that accesses the remote data must be connected to your local node. You must have definition and data access to the table at the remote node.
5. If the table is an external table, the server accessing the external data must be running. You must have external security access to the external data.

## Locating the Data

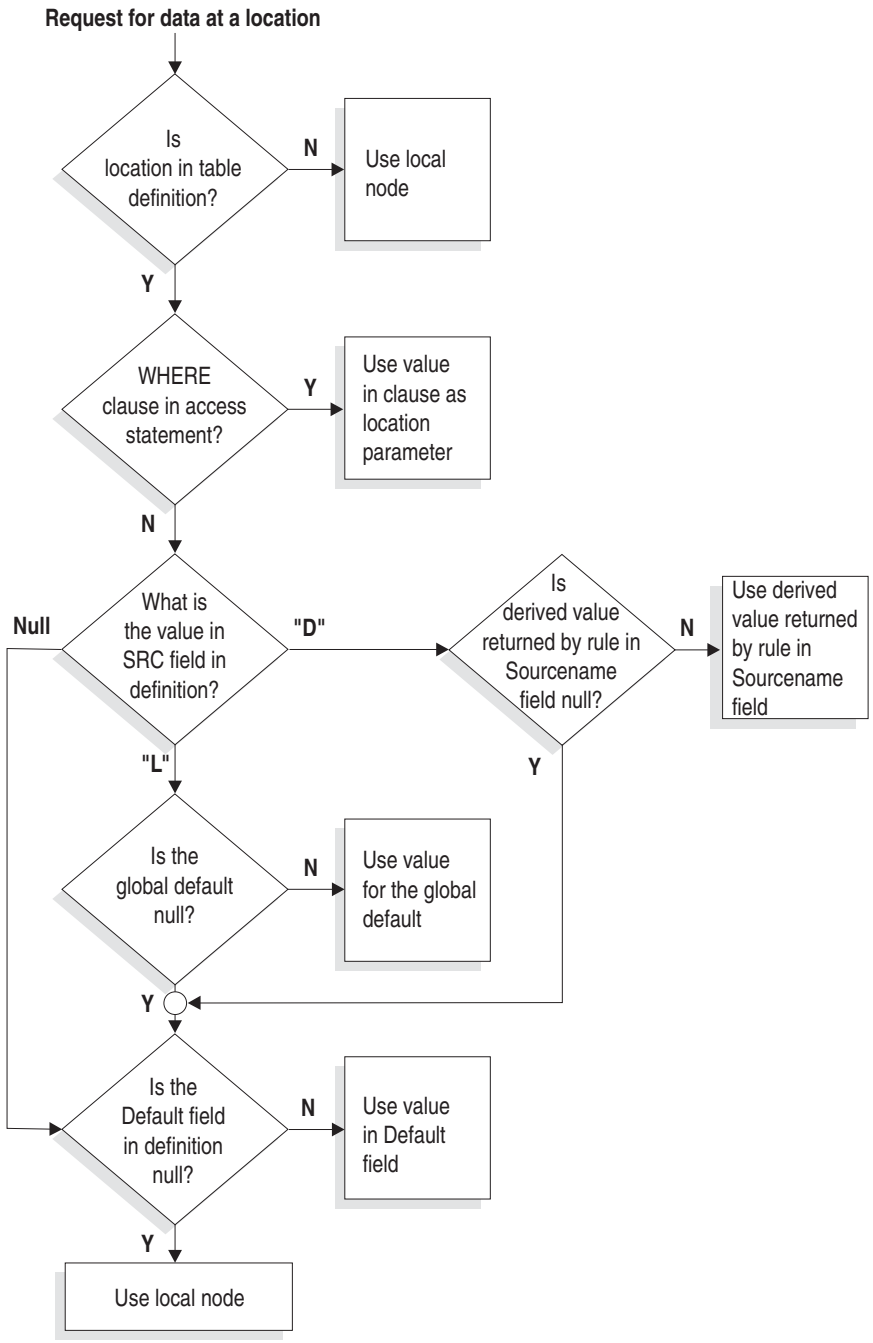
The location of the data used by an application is provided through a value for a location parameter. If a location parameter is not defined to the table, the data is retrieved from the local data store.

The value used to determine location is the node name for the TIBCO Object Service Broker system where the required data resides. At installation time, a TIBCO Object Service Broker system is assigned a node name using the Data Object Broker parameter NODENAME.

You can load and unload definitions of all table types using the LOAD and UNLOAD tools. Only TDS, EES and SES table data can be loaded using these tools.

## Order of Evaluation to Determine Location

When data is accessed by a rule, the location of the data is determined by a defined order of evaluation. The evaluation is based on a parameter defined as CLASS=L (location) in the table definition. This evaluation continues until either a non-null location value is found or the end of the list is reached, as shown in the following illustration.


Request for data at a location
Is location in table definition?
N
Use local node
Y
WHERE clause in access statement?
Y
Use value in clause as location parameter
N
What is the value in SRC field in definition?
Null
"D"
Is derived value returned by rule in Sourcename field null?
N
Use derived value returned by rule in Sourcename field
"L"
Y
Is the global default null?
N
Use value for the global default
Y
Is the Default field in definition null?
N
Use value in Default field
Y
Use local node

See Also

This manual describes how to define, load, and manipulate data for TIBCO Object Service Broker tables. For additional information on how to access data for presentation and processing and on how to access remote and external data, refer to the following manuals:

| Topic | Manual |
|---|---|
| Processing data | *TIBCO Object Service Broker Programming in Rules* and *TIBCO Object Service Broker Shareable Tools*. |
| Presenting data | *TIBCO Object Service Broker Defining Screens and Menus* and *TIBCO Object Service Broker Defining Reports*. |
| Setting up security | *TIBCO Object Service Broker Managing Security* |
| Starting a server for a remote node | *Installing and Operating* for your operating environment. |
| Accessing external data | *TIBCO Service Gateway for Files Installing and Operating* and the external *TIBCO Service Gateway* manuals for the various supported external data types. |
| LOAD and UNLOAD tools | *TIBCO Object Service Broker Shareable Tools* |

# Chapter 2 The Table Definer

This chapter describes the table definer.

## Topics

# What is the Table Definer?

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to the TIBCO Object Service Broker UI online help.

## Overview

You use the Table Definer to specify and modify the definitions of your data tables. This tool, as it is described here, is accessible from the developer's workbench via the Define Table option. It is also available from the administrator's workbench. Chapter 3, Defining TDS, EES, SES, and TEM Tables, on page 29 and Chapter 4, Defining a View of a Source Table, on page 45 describe the Table Definer as it is used to define the following table types:

- CLC
- EES
- PRM
- SES
- SUB
- TDS
- TEM

For descriptions of how to define other table types, refer to Chapter 1, Overview of Data Structures, on page 1.

## Accessing the Table Definer

You can access the Table Definer from the workbench by doing one of the following:

- Type a new or existing table name to the right of the DT define table option and press Enter. This displays the initial Table Definer screen.

- Position your cursor to the right of the DT define table option and press Enter. This displays the Object Manager screen for the Table Definer. Select an object from this screen to invoke the Table Definer screen.
- Type DT and a new or existing table name in the command line.
- Execute the tool DEFINE_TABLE (*tbl_name*) where *tbl_name* is the name of a table.

You can also access the Table Definer from within the OS object set workbench option. To access the Table Definer, from within the first screen, position your cursor on the name of a table object and press PF9.

## Layout of the Screen

The initial screen that appears when you enter the Table Definer is used to define TDS, TEM, EES and SES tables. This screen is divided into four segments:

- Table identification
- Parameter
- Event rule
- Data field

The layout of the initial screen changes if you are editing the definition of a table of type SUB, CLC, or PRM. For additional information refer to Chapter 4, Defining a View of a Source Table, on page 45.

## Example

The following example shows a definition screen for the @EMPLOYEES table.

```
 COMMAND==>                          TABLE DEFINITION

      Table: @EMPLOYEES       Type: TDS   Unit: DOCEXMPL        IDgen: N

    Parameter Name  Typ Syn Len Dec Class             '      Event Rule   Typ Acc
   ----------------  -   -  ---- --   -               '   ---------------- -   -
 _ REGION            I   C   16  0    D               ' _ DELETEMPNO       T   D
 _ LOCATION          I   C   16  0    L               ' _ VALIDEMPNO       V   I
      Field Name   Typ Syn Len Dec Key Ord Rqd    Default          Reference
   ---------------- -   -  ---- --  -   -   - ---------------- ----------------
 _ EMPNO             I   P     3  0  P
 _ LNAME             S   C    22  0  S       Y
 _ POSITION          S   C    14  0
 _ MGR#              I   P     3  0                              MANAGER
 _ DEPTNO            I   B     2  0
 _ SALARY            Q   P     4  2             0.00
 _ ADDRESS           S   V    38  0
```

```
_ CITY              S    C     20   0
_ STATE_PROV        S    C      4   0
_ ZP_CODE           S    C      7   0
_ HIREDATE          D    B      4   0
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC
```

See Also *TIBCO Object Service Broker Application Administion* for information on the administrator's workbench.

# Tools Available from the Table Definer

## What Tools are Available?

The following tools are available from within the Table Definer:

- Table Editor
- Table Browser
- Single Occurrence Editor (SOE)
- Global Field Selector

## Table Editor, Table Browser, and Single Occurrence Editor

Use the Table Editor and Single Occurrence Editor to add or delete data in your table. Use the Table Browser to view the data in your table. For more explanation of how to use these tools, refer to Chapter 6, Manipulating Data in a Table, on page 69.

To access the Table Editor or Table Browser, use PF21 from within your Definer session. The table type that you are defining determines whether the Table Browser or the Table Editor is invoked, as shown in the following table:

| Table Type | Tool Invoked |
|---|---|
| SUB | Table Editor. |
| TDS | Table Editor. |
| CLC | Table Browser. |
| EES | Table Browser. |
| PRM | Table Browser. |
| TEM | Table Browser. |
| SES | Table Browser. |

## Global Field Selector

Use the Global Field Selector to select predefined fields to be included in your table definition. Global fields allow for the standardization of definitions across the database. These fields are defined by your application administrator and are stored in the @GLOBALFIELDS shared table. For more information on global fields and how to use them, refer to Selecting Global Fields on page 27.

# Selecting Global Fields

The use of global fields in your environment is determined by your application administrator. This implementation is enforced at a table-type level.

Global fields are not supported in the graphical Table Definer supplied with the TIBCO Object Service Broker UI. You must use the text-based definer if you are going to use global fields.

## Types of Implementation

The following implementations are available for the @GLOBALFIELDS table. Check with your application administrator for the implementation used in your development environment.

| Field Names | Field Attributes |
|---|---|
| Each field must be linked to a global field or you are not able to save the definition. | The attributes of the field must match the attributes of the global field to which you are linked or you are not able to save the definition. |
| You are warned if a field is not linked to any field in the global field dictionary. | You are warned if the field attributes do not match the attributes of the global field to which you are linked. |
| The fields are not linked to the global field dictionary. | The field attributes are not linked to the global field dictionary. |

## Types of Information Stored with a Global Field

The following types of information are stored with a global field:

- Field attributes such as name, unit, type, and syntax.
- A description of the field. It is optional for this to be defined.
- A display mask and/or a display length if the field is to be used in screens or reports.
- Help specific to the field. It is optional for help to be defined.

## Global Fields Selector Illustrated

When you press PF14 from within the Table Definer screen, a screen similar to the following appears:

```
                            Globalfields                              Scroll: P
COMMAND ==>                                                        Select All: N
Location:                                                        Deselect All: N
                                                            Show selection specs: Y
========================== Selection Specification ============================
      Selection: NAME LIKE '*'
AND               Op                          Value
                  ---- ------------------------------------------------------------
             NAME
    BUSINESSNAME
            UNIT
         CREATED
          AUTHOR
================================================================================
  Name              Businessname                          Unit        Created Aut
  ---------------- ------------------------------------ -------- ---------- ---
_ ACCESSTYPE        ACCESS FOR DBMS                       EXT      2000-03-15 AMD
_ DATE              CURRENT DATE                          USR40    1995-03-01 USR
_ DEPTNO            DEPARTMENT NUMBER                              1992-01-22 MGR
S MANAGER           MANAGER NAME                          EMP      1989-03-01 WEA
_ TIME              CURRENT TIME
S USERID            USERID OF EMPLOYEE
PFKEYS: ENTER=UPDATE 3=SAVE 12=CANCEL
```

## Using the Global Fields Selector

To select global fields to copy from the Global Field dictionary, complete the following tasks:

1. Press PF14. This displays a listing of the Global Field dictionary.
2. Type S beside the fields you want to copy.

   The Selection Specification section of the screen can be used to narrow the selection list by using specified selection criteria. The list of fields appears in the lower portion of the screen. You can use more than one type of selection criteria. For a list of valid values for each of these fields, position your cursor on the field and press PF1. To narrow your selection, beside the appropriate selections specify an operator in the **Op** field and appropriate values in the **Value** field.
3. Press PF3 to save or copy. The Table Definer screen appears with the global fields appended to the table, one per line in the order in which they are stored in the @GLOBALFIELDS table.

# Chapter 3 Defining TDS, EES, SES, and TEM Tables

This chapter describes how to define the TDS, EES, SES, and TEM tables.

## Topics

# Overview

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to the TIBCO Object Service Broker UI online help.

## Tasks for Defining a Table

Complete the following tasks to define TDS, EES, SES, and TEM tables after you invoke the Table Definer (refer to Accessing the Table Definer on page 22):

| Task | Required | Refer to page… |
|---|---|---|
| Task A: Define Table Properties. | Yes | 31 |
| Task B: Define Parameters. | No | 33 |
| Task C: Specify Event Rules. | No | 37 |
| Task D: Define Primary Keys. | Yes | 39 |
| Task E: Define Non-key Fields. | Yes | 42 |

Processing of Execution Environment tables has unique characteristics. For details, see Understanding EES Table Considerations on page 87.

# Task A: Define Table Properties

## Purpose of this Task

In this task you:

- Uniquely identify the table
- Specify the table type
- Identify the application or developmental unit where it belongs
- Specify if the system should generate unique values for the primary key field

Use the table identification segment of the screen for this task.

### Table Identification Segment

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

```
Table: @EMPLOYEES      Type TDS     Unit: DOCEXMPLS      IDgen: N
```

## Table, Type, Unit, and IDgen Fields

The information for the **Table**, **Type**, **Unit**, and **IDgen** fields appears by default. You can modify the **Table** and **Unit** fields, if necessary. You cannot modify the **Type** field if data already exists in the table. For valid values, press PF1.

| | |
|---|---|
| **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 a new name. For more information on how to copy TIBCO Object Service Broker objects, refer to *TIBCO Object Service Broker Shareable Tools*. |
| **Type** | The type indicates how data is stored in the table or how data can be accessed from a table. If the table is new, a default value of TDS is supplied. You can change it to one of the allowed table types by typing in a valid value or pressing PF1 and selecting from the supplied list.<br><br>Refer to Available Table Types on page 10 to determine the appropriate table type to define for your application requirements. |

| | |
|---|---|
| **Unit** | The unit indicates that the table belongs to a particular application or logical unit such as utilities, accounting, or network control. |
| **IDgen** | The **IDgen** field determines whether TIBCO Object Service Broker should generate values 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. You must then use rules to populate the table with data. For more information, refer to Chapter 6, Manipulating Data in a Table, on page 69.<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: Define Parameters

## Purpose of this Task

In this optional task you specify parameters. There are two types of parameters you can specify using the parameter segment of the screen:

- Data
- Location

### Parameter Segment

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

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

## Considerations for Defining Parameters

Use the following table to determine if you should define parameters and the type of parameters to define if they are required.

| Considerations | Parameterize | Type of Parameter |
|---|---|---|
| The data is single dimensional. It does not partition easily or there is no need to partition the data. | N | |
| The data has more than one dimension. It logically breaks down into one or more partitions (called instances). These instances can then be used to access the data. | Y | Data |
| The same type of data is used in more than one location and can be stored in different TIBCO Object Service Broker databases. | Y | Location |

## Data Parameters

Data parameters are used to partition data. They can be defined for TDS, Execution Environment ((EES), temporary (TEM), and session (SES) tables.

## Defining Data Parameters

A maximum of four data parameters can be specified for a table, to a total maximum length of 240 bytes. The following fields are used to define data parameters. You can enter the information into the fields in any order. For valid values, press PF1.

| | |
|---|---|
| **Parameter Name** | Enter the name of the data parameter. The order in which the names are entered determines the order in which supplied values are processed. Data parameters are relocated to the top of the parameter list when the table definition is saved. |
| **Typ** | Enter the semantic data type of the parameter. Refer to *TIBCO Object Service Broker Programming in Rules* for more information and a list of valid values. |
| **Syn** | Enter the syntax of the parameter. Refer to *TIBCO Object Service Broker Programming in Rules* for more information and a list of valid values. |
| **Len** | Specify the length of the parameter value. The value is in bytes and valid values are determined by the syntax of the parameter. For valid lengths, refer to *TIBCO Object Service Broker Programming in Rules*. Each data parameter must have a non-null value. |
| **Dec** | Specify the number of digits to the right of the decimal point. In most cases this is optional. It is relevant only for syntax P. |
| **Class** | Specify D to indicate that the parameter is a data parameter. |
| **Reference** | [Optional] Specify the name of a reference table. Specify a value only if a table is to be referenced when a user is inserting or replacing a value in the parameter. You then ensure that the user enters valid values for the data parameter. The reference table must be non-parameterized. If the new value does not exist as a primary key value in the referenced table, the action fails validation. |

## What is a Location Parameter?

A location parameter is used to identify the location (MetaStor) where the data in a table is stored. It cannot be defined for SES tables. The name of a location parameter is provided by default when a new table is being defined. Use the **D** line command to delete the parameter if it is not required. The location parameter is moved to the bottom of the parameter list when the table definition is saved.

When you access a table via a rule, which passes a location parameter, the value for the location parameter must not be NULL or blank.

## Defining a Location Parameter

The following fields are used to define the location parameter. You can enter the information into the fields in any order. For valid values, press PF1.

| | |
|---|---|
| **Parameter Name** | Change the name of the location parameter, if required. The value of LOCATION is provided by default but it can be modified or deleted. The name of the location parameter must be the last parameter name listed. |
| **Typ** | Type I for the semantic data type. |
| **Syn** | Type C for the syntax of the parameter. |
| **Len** | Type 16 for the length. |
| **Class** | Type L to indicate that the parameter is a location parameter. Only one location parameter can be specified. |
| **Reference** | Specify the name of a reference table (this is optional). Specify a value only if a table is to be referenced when a user is inserting or replacing a value in the table instance. You then ensure that the user enters valid values for the location parameter. The reference table must be non-parameterized. If the new value does not exist as a primary key value in the referenced table, the action fails validation.<br><br>The value in the reference field is ignored for a minimal table definition. Refer to Defining a Minimal Definition on page 64. |

| | |
|---|---|
| **Default** | Specify the name for the default node to be used on data access (this is optional). Even if a value is provided, when an access is made to the data, the location is determined using the order of evaluation described in Order of Evaluation to Determine Location on page 17. |
| **Src** | Indicate how the name of the node is to be determined (this is optional). The value for the node name can be derived from a functional rule named in the **Sourcename** field or provided through the users' session options. Even if a value is provided, when an access is made to the data, the location is determined using the order of evaluation described in Order of Evaluation to Determine Location on page 17. |
| **Sourcename** | Identify the name of the source rule. This is used only if the **Src** field is set to D. The rule named must be a functional rule that returns the value for the node name. It must have an argument for the table name and arguments for each of the data and location parameters. For more information about coding considerations, refer to Chapter 7, Coding Considerations for Event, Location, and Derived Value Rules, on page 91. |

# Task C: Specify Event Rules

## Purpose of this Task

In this optional task, you specify event rules if you need to associate business rules and policies with the definition of a table. The rules that you name here are run whenever data in the table is manipulated. There are two types of event rules that you can specify using the event rule segment of the screen:

- Trigger rules
- Validation rules

For more information on trigger and validation rules, refer to Coding Event Rules on page 92.

### Event Rule Segment

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

```
                Event Rule    Typ Acc
              --------------- -   -
          '_ DELETEMPNO       T   D
          '_ VALIDEMPNO       V   I
```

## Considerations for Defining Event Rules

Use the following table to determine if you should define event rules, the type of event rules to define if they are required, or the TIBCO Object Service Broker option to use if they are not required.

| Considerations | Use an Event Rule | Type of Event Rule | Other Option |
|---|---|---|---|
| You need to validate data using rules processing as it is being accessed. | Y | Validation. | |
| Simple validation of data is required. | N | | Reference table. |

| Considerations | Use an Event Rule | Type of Event Rule | Other Option |
|---|---|---|---|
| Additional processing should be initiated when a table is accessed, for example, to maintain a one-to-one relationship between two tables. | Y | Trigger. | |
| Logging of accesses is required. | N | | Logging on option available from TIBCO Object Service Broker security. |

## Event Rule, Typ, and Acc Fields

The rules that you enter here are run based on defined accesses. All the rules that apply to a specific access are executed in the order in which they are entered in the event rule segment. For valid values, press PF1.

| | |
|---|---|
| **Event Rule** | Specify the name of the event rule to be executed. For information about coding considerations, refer to Chapter 7, Coding Considerations for Event, Location, and Derived Value Rules, on page 91. |
| **Typ** | Specify the type of event rule 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 to the **Typ** field. |

# Task D: Define Primary Keys

## Purpose of this Task

In this task you define the primary key fields for the table. At least one primary key field must be specified for each table definition for a table of type TDS, EES SES, and TEM. The value provided for the primary key field of each occurrence must be unique to the table. If the table is parameterized, the value provided for each primary key field must be unique to the table instance. The primary key fields is used to identify and retrieve occurrences of data. Primary keys are defined in the field segment of the Table Definer screen.

### Field Segment

The following example illustrates the fields used to define the primary key (EMPNO). To view additional fields, press PF11.

```
     Field Name      Typ Syn Len  Dec Key Ord Rqd      Default         Reference
   ----------------  -   -- ----  --  -   -   -  ---------------- ----------------
- EMPNO               I   P     3   0  P
-
```

Secondary keys can also be specified for data access but they are not defined with the standard definer. For more information, refer to *TIBCO Object Service Broker Application Administion*.

## Composite Primary Keys

You can specify a single field or up to sixteen fields to be the primary key, to a total maximum length of 127 bytes. The primary key is always stored as the first field or series of fields.

Although TIBCO Object Service Broker allows a maximum of sixteen fields in a composite primary key, you can use only the Table Browser and the Table Editor on tables that have a maximum of eight fields in the composite primary key. You must use rules to access tables with more than eight fields in a composite primary key.

## Methods Available for Defining Primary Key Fields

You can use the following methods to define primary key fields:

- Use fields with predefined attributes.

  For information about how to use fields with predefined attributes, refer to Selecting Global Fields on page 27.

- Copy values from an existing definition.

  For information about copying a definition, refer to Copying a Definition on page 62.

- Create a new field.

  The following section describes how to create a new primary key field.

## Creating a New Primary Key Field

To define a primary key, type information in the following fields in any order in the fields segment. Primary key fields must be the first fields in the definition. For a list of valid values, press PF1.

| | |
|---|---|
| **Field Name** | Enter the name of the primary key field. The name of each field must be unique to the table. |
| **Typ** | Enter the semantic data type for the primary key field. For valid values for users to supply, refer to *TIBCO Object Service Broker Programming in Rules*. If the table has the **IDgen** field set to Y, the semantic type must be defined as I (identifier). |
| **Syn** | Enter the syntax of the primary key field. For valid values for users to supply, refer to *TIBCO Object Service Broker Programming in Rules*. If the table has the **IDgen** field set to Y, the syntax must be defined as B (binary). |
| **Len** | Specify the length of the primary key field. The value is in bytes and the length is determined by the syntax of the field. For valid lengths, refer to *TIBCO Object Service Broker Programming in Rules*. Each primary key field must have a non-null value. If the table has the **IDgen** field set to Y, the length must be defined as 4 bytes. |
| **Dec** | Specify the number of digits to the right of the decimal point. In most cases this is optional. It is relevant only for syntax P. |
| **Key** | Type P to indicate that the field is a primary key field. At least one field must be specified as the primary key field. |

| | |
|---|---|
| **Ord** | [Optional] Specify the ordering of the primary key field. The default value of null returns occurrences in ascending order by primary key. Specifying a value in this field incurs sorting overhead, which can be significant in tables with a large number of occurrences. |
| **Rqd** | [Optional] Indicate if a value for the primary key field is required. By definition a value for a primary key field is required. |
| **Reference** | [Optional] Specify the name of a reference table. Specify a value only if a table is to be referenced when a user is inserting or replacing a value in the field. You then ensure that the user enters valid values for the primary key field. The reference table must be non-parameterized. If the new value does not exist as a primary key value in the referenced table, the action fails validation. |

# Task E: Define Non-key Fields

## Purpose of this Task

In this task you define the non-key fields for your table. Non-key fields are defined in the field segment of the Table Definer screen.

### Field Segment

The following example illustrates the fields you use to define the fields of your table. To view additional fields, press PF11.

```
    Field Name      Typ Syn Len  Dec Key Ord Rqd       Default          Reference
   ---------------- -   -- ---- --  -   -   -   ---------------- ---------------
- EMPNO              I   P     3   0  P
- LNAME              S   C    22   0  S       Y
- POSITION           S   C    14   0                                MANAGER
- MGR#               I   P     3   0
- DEPTNO             I   B     2
- SALARY             Q   P     4   2              0.00
- ADDRESS            S   V    38   0
- CITY               S   C    20
- STATE_PROV         S   C     4   0
- ZP_CODE            S   C     7   0
- HIREDATE           D   B     4   0
```

## Methods Available for Defining Data Fields

- Use fields with predefined attributes.

  For information about how to use fields with predefined attributes, refer to Selecting Global Fields on page 27.

- Copy values from an existing definition.

  For information about copying a definition, refer to Copying a Definition on page 62.

- Create a new field.

  The following section describes how to create a new field.

## Creating a New Non-key Field

To define a new non-key field type information in the following fields in any order, after the primary key fields. For valid values, press PF1.

| | |
|---|---|
| **Field Name** | Enter the name of the field. The name of each field must be unique to the table. |
| **Typ** | Enter the semantic data type of the field. For an explanation and valid values, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Syn** | Enter the syntax of the field. For an explanation and valid values, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Len** | Specify the length of the field. The value is in bytes and valid values are determined by the syntax of the field. For valid lengths, refer to *TIBCO Object Service Broker Programming in Rules*. |
| **Dec** | Specify the number of digits to the right of the decimal point. In most cases this is optional. It is relevant only for syntax P. |
| **Ord** | Specify the ordering of the field (this is optional). 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>Specifying a value in this field incurs sorting overhead, which can be significant in tables with a large number of occurrences.<br>Ordering is not permitted for fields with syntax RD (raw data) or UN (Unicode). |
| **Rqd** | Specify if a value for the field is required (this is optional). |
| **Default** | Specify the value to be used for the field if none is supplied by the user.<br>Default values are not permitted for fields with syntax RD or UN. |

| | |
|---|---|
| **Reference** | Specify the name of a reference table (this is optional). Specify a value only if a table is to be referenced when a user is inserting or replacing a value in the field. You then ensure that the user enters valid values for the field. The reference table must be non-parameterized. If the new value does not exist as a primary key value in the referenced table, the action fails validation. |

# Chapter 4 Defining a View of a Source Table

This chapter describes how to define a view of a source table and how to define SUB, CLC, and PRM tables.

## Topics

# Views of a Source Table

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to the TIBCO Object Service Broker UI online help.

## What is a View of a Source Table?

A view of a source table is a predefined means of accessing specific kinds of data stored in the source table. Views do not contain any data of their own, although they look like tables and can be processed as though they are real tables. Views simplify database access and enhance re-usability of data.

## What Types of Views Can be Specified?

Providing you have adequate security access, you can create the following types of views of a table:

- A view of selected fields, occurrences, or instances in a subview (SUB) table
- A count of specified fields in a calculation (CLC) table
- A view of all data parameter values in a parameter value (PRM) table

# Defining a Subview (SUB) Table

## Why Define a Subview?

A subview, which is a limited view of a source data table at either a local or remote location, aids in data privacy, increases data security, and often simplifies data access. Changes made in a subview are reflected in the source table but a user of a subview does not have full access to all the data in the source table. You must have DEF_VIEW access to the source table to define a subview of it.

## Behavior of Subviews

The following table describes the behavior of fields, event rules, and parameters as defined in the source and subview tables:

| Elements Defined in Source Table | When Accessed Through the Subview |
|---|---|
| Required fields. | Constraints apply in the subview. |
| Referenced fields. | Constraints apply in the subview. |
| Event rules. | Are executed when access is made to the subview. |
| Fields in the source table not defined in the subview. | Take on default values in the source table. |
| Fields in the source table defined in the subview. | Take on the new values provided through the subview. |
| Data parameters. | Can be part of either the selection criteria or defined as parameters to the subview. |
| Location parameters. | Must be defined as a location parameter in the subview. |

## Tasks Required to Define a Subview

Complete the following tasks to define a subview table. These tasks are described in the following sections.

| Task | Required | Go to page ... |
|---|---|---|
| Specify the table type. | Y | 48 |
| Specify the source table. | Y | 49 |
| Specify the lock mode. | Y | 49 |
| Select the data. | Y | 49 |
| Define the parameters. | N | 50 |
| Define the fields. | Y | 50 |

See Also Refer to Defining a Subview on a Minimal Definition on page 66 for the tasks required to define a subview on a minimal table definition.

### Task A Specify the table type

After entering the initial Table Definer screen, described in Accessing the Table Definer on page 22, you must change the **Type** field to SUB. When you press Enter, a screen similar to the following appears.

```
COMMAND==>                          TABLE DEFINITION

     Table: @EMPLOYEES_SUB   Type: SUB   Unit: DOCEXMPL
     Source:
     Select:

                               Lock Mode:
   Parameter Name  Typ Syn Len Dec Class Src   Source Name        Default
  ----------------  -   -- ---  --    -    -  ---------------- ----------------
_
_
                     Synt    Dec   Order
     Field Name    Type   Len    Key    Rqd     Default     Src  Source Name
  ----------------  - -- ---- -- -   -  -  ---------------- - ----------------
_
_
_
_
_
_
_
 PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC
```

```
Table type changed
```

### Task B Specify the source table

In the **Source** field, type the name of the table that is your source of data. The source table must already exist and it must be one of the following types of tables: TDS,VSM, DB2, DAT, IDM, IMS, SLK, ADA, or IMP.

When the source name is provided and you press Enter, the screen is populated with parameter and field names from the source table. The parameter types and primary key are also identified. The values supplied can be edited.

### Task C Specify the lock mode

In the **Lock Mode** field, type in B if you do not want locks to be taken on the data. If locks are not taken, users can only browse the data. Type in D if you want the transaction to determine whether updates are allowed.

### Task D Select the data

In the **Select** field, enter selection criteria for data for the subview. Occurrences being inserted into or deleted from the table must meet the selection criteria. For occurrences being replaced, both the old and new must meet the selection criteria. If no selection criteria is specified, all occurrences in the table are available for use.

The selection criteria must be a field name or a data parameter name from the source table, followed by a relational operator, followed by one of: a value; a field name from the source table; a new subview parameter; or an expression. The expression can include *only* a source field and a constant, or a defined parameter and a constant.

If a data parameter of the source table is not specified in the parameter area of the subview, it must be selected in the selection field as: *source_parameter_name=value.*

If a data parameter of the subview is a field of the source table, it must be selected as: *source_field_name=subview_parameter_name.*

### Example of Selection

To select occurrences that meet the following criteria: position of ANALYST or employees earning more than $400.00, and the source field MGR# set to the parameter MANAGER# in the subview, enter the following:

```
POSITION='ANALYST' | SALARY > 400.00 & MGR# = MANAGER#
```

### Task E Define the parameters

Define the parameters in the same way as described in Task B: Define Parameters on page 33, except for the following differences:

- A data parameter of a source table, if it is not defined in the subview, must be selected in the selection area of the subview screen as previously described.
- A data parameter in a subview can be a field of the source table, but it must be selected in the selection area of the subview screen as previously described.
- The names of the parameters can differ between the subview and the source table. If the names differ, the value in the Src field must be set to S and the name of the parameter in the source table must be provided in the Source Name field.

### Task F Define the fields

Define the key and non-key fields in the same way as described in Task D: Define Primary Keys on page 39 and Task E: Define Non-key Fields on page 42, noting the following:

- Key fields in the source table must be defined as key fields in the subview.
- Some or all of the non-key fields named in the subview can be the same as those in the source table.
- Some of the fields in the subview could be renamed from the source table but still derive their values from the source table.
- Only the name and primary key setting of the source fields are imported from the source table. The other attributes, such as syntax and length, are left unspecified. Unless you override these other attributes in the subview table, they are inherited from the source table.
- Ordering and default values are not permitted for fields with syntax RD (raw data) or UN (Unicode).
- Some of the fields could be new fields unique to the subview and derive their value through a functional rule.

### Setting a Value or Name

Use the **Src** field and **Source Name** field to set the value or name to be used, as shown in the following table:

| Value Src Field | Value in Source Name Field | Value is … |
|---|---|---|
| Blank | Blank. | Same as source. |

| Value Src Field | Value in Source Name Field | Value is … |
|---|---|---|
| S | Name of source field. | Derived from the source table. |
| D | Name of a functional rule. | Derived from a functional rule. |

If a functional rule is named, the table must be viewed and edited using the shareable tools STEBROWSE and STE. For information about these tools, refer to *TIBCO Object Service Broker Shareable Tools*. For information about coding the functional rule, refer to Chapter 7, Coding Considerations for Event, Location, and Derived Value Rules, on page 91.

## Sample Subview Table

The following example illustrates a sample subview table:

```
BROWSING TABLE    :  @EMPLOYEES_SUB(MIDWEST)
COMMAND ==>
                                                                  SCROLL: P
  EMPNO     POSITION     MANAGER# DEPTNO  SALARY
_ ------ --------------  ------  ------ ---------
_  22001 CUST SUPPORT      56112      30    900.00
_  22007 CUST SUPPORT      56112      30    900.00
_  30058 PRE-SALES         37219      20    675.00
_  34111 PRE-SALES         37219      20    710.00
_  34121 CUST SUPPORT      56112      30    700.00
_  36162 JR OPERATOR       44798      80    575.00
_  41001 TECH WRITER       80002      70    675.00
_  41007 EDUCATOR          80002      60    700.00
_  41009 TESTER            79912      50    600.00
_  44385 SALES             37219      10    719.00
_  44622 ACCOUNTANT        98895      40    800.00
_  51111 ANALYST           79912      50    710.00
_  51121 ANALYST           79912      50    700.00
_  51162 JR PROGRAMMER     79912      50    575.00
_  61219 SENIOR ANALYST    79912      50    820.00
_  61385 EDUCATOR          80002      60    685.00
_  61622 SENIOR ANALYST    79912      50    800.00
PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND
```

# Defining a Calculation (CLC) Table

## Why Define a Calculation View?

Often, when accessing data, a simple count of the fields that contain the same values within a table is required as part of the data access. Using a calculation table, the values can be accessed via a source table and counted. The values are dynamically updated for you each time a change is made to the source table. You must have DEF_VIEW access to the source table to define a calculation view of it.

Fields with either a syntax of F or with a definition length of more than 127 cause the table to be unusable in a rule.

## Behavior of Calculation Tables

Note the following when using a calculation table:

- The primary key values are assigned dynamically and are also updated with each update to the source table.
- For performance reasons, if only one field is being counted, the field should have a secondary key built on it, if this is allowed for its syntax, before the calculation table is accessed.

See Also

- *TIBCO Object Service Broker Programming in Rules* for information on which fields, depending on their syntax, can be defined as secondary keys.
- *TIBCO Object Service Broker Application Administion* for more information on secondary keys.

## Tasks Required to Define a Calculation Table

Complete the following tasks to define a calculation table:

| Task | Required | Go to Page ... |
|---|---|---|
| Specify the table type. | Y | 53 |
| Specify the source table. | Y | 53 |
| Verify the parameters. | N | 53 |
| Define the primary key and non-key fields. | Y | 54 |

The following sections provide more information on these tasks.

### Task A Specify the table type

After entering the initial Table Definer screen described in Accessing the Table Definer on page 22, you must change the table type field to CLC. When you press Enter, a screen similar to the following appears.

```
COMMAND==>                          TABLE DEFINITION

     Table: @EMPLOYEES_CLC   Type: CLC   Unit: DOCEXMPL
     Source:

   Parameter Name  Typ Syn Len Dec Class     Default      Src    Sourcename
  ----------------  -   -- --- --    -   ---------------- --- ----------------
_
_
     Field Name    Type Syntax Length Decimal  Key Ord Src   Source Name
  ----------------   -      --   ----     --     -   -   -  ----------------
_
_
_
_
_
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC
Table type changed
```

### Task B Specify the source table

In the **Source** field, enter the name of the table that is your source of data. The source table must already exist and it must be a TDS table. When you provide the source name and press Enter, the screen is populated with values from the source table.

### Task C Verify the parameters

The parameter values are provided for you when you press Enter after specifying a source table name. These values must not be changed.

### Task D Define the primary key and non-key fields

The field values are provided for you when you press Enter after specifying a source table name. In addition, a new primary key field named **KEY** and a new non-key field named **COUNT** are added and defined, and two new fields called **Src** and **Source Name** are added.

Except for the **KEY** and **COUNT** fields, delete the fields that are not to be counted. You can use the **D** line command to delete the fields. The names for the **KEY** and **COUNT** fields can be changed but you must not change the default attributes for these fields.

### Sample Calculation Table

The following example illustrates a sample calculation table:

```
 BROWSING TABLE   :  @EMPLOYEES_CLC(MIDWEST)
COMMAND ==>
                                                                SCROLL: P
      KEY         POSITION      MGR#      COUNT
_ ----------- -------------- ------ -----------
_           1 ACCOUNTANT      98895           1
_           2 ANALYST         79912           3
_           3 CUST SUPPORT    56112           3
_           4 EDUCATOR        80002           2
_           5 EDUCATOR        98895           1
_           6 JR OPERATOR     44798           1
_           7 JR PROGRAMMER   79912           1
_           8 MANAGER         99999           2
_           9 PRE-SALES       37219           2
_          10 SALES           37219           1
_          11 SENIOR ANALYST  79912           2
_          12 TECH WRITER     80002           2
_          13 TESTER          79912           1


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

# Defining a Parameter Value (PRM) Table

## Why Define a Table for Parameter Values?

When accessing the data of a parameterized table, you often require access to data in all the instances of the table. By defining a table to hold the data parameter values, you can relate this information in your data access. This relationship can be coded explicitly using a FORALL statement or implicitly using a tool interface. You must have DEF_PRM access to the source table to define a subview of it.

## Behavior of Parameter Tables

Note the following when using a parameter value table:

- The primary key values are assigned dynamically and are also updated with each update to the source table.
- Only the values of data parameters are maintained in a parameter value table.

## Tasks Required to Define a Parameter Value Table

Complete the following tasks to define a parameter value table:

| Task | Required | Go to Page ... |
|---|---|---|
| Specify the table type. | Y | 55 |
| Specifying the source table. | Y | 56 |

The following sections provide more information on these tasks.

### Task A Specify the table type

After entering the initial Table Definer screen described in Accessing the Table Definer on page 22, you must change the table type field to PRM. When you press Enter, a screen similar to the following appears.

```
COMMAND==>                          TABLE DEFINITION

      Table: $@EMPLOYEES       Type: PRM    Unit: DOCEXMPL
     Source:
```

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

### Task B Specifying the source table

In the **Source** field, type in the name of the table that is used as your source of data. The source table must already exist and it must be a parameterized table of one of the following:

- IMP
- EES
- SES
- TDS
- TEM
- VSM

## Using the NUMBER Field in a Parameter Table

A parameter table is a virtual table, meaning that its data is extracted from dynamic source information. The **NUMBER** field of the parameter table depends on the following:

- Current state of the parameter instances
- Access path, for example, if the initial subset of the key information is available
- Selection string criteria

Therefore, since the values in the **NUMBER** field are dynamic, it cannot be used in any meaningful way and should be ignored. For example, the **NUMBER** field for parameter instance A has a value of one but on the following day the **NUMBER** field (for parameter instance A) could equal 2 because a new parameter instance is added.

## Sample Parameter Value Table

The following example illustrates a sample parameter value table:

```
 BROWSING TABLE   :  $@EMPLOYEES
COMMAND ==>
                                                                  SCROLL: P
    NUMBER          REGION
_ ----------- ----------------
_           1 CANADA
_           2 MEXICO
_           3 MIDWEST
_           4 SOUTHWEST









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

## Example Rule for Parameter Values for a Table

The following rule uses FORALL statements to access and process data parameter values for a table.

```
RULE EDITOR ===>                                                SCROLL: P
DEPT_EXPENSE(MEDIA);
_
_ ------------------------------------------------------------------------
_ ------------------------------------------------------------+-----------
_ FORALL $EMP_EXPENSE :                                       ¦ 1
_    DEPT_EXPENSE.MONTH_NUMBER = $EMP_EXPENSE.MONTH;          ¦
_    FORALL EMP_EXPENSE($EMP_EXPENSE.MONTH) :                 ¦
_       DEPT_EXPENSE.* = EMP_EXPENSE.*;                       ¦
_       INSERT DEPT_EXPNSE('DEPT_EXPENSE');                   ¦
```

```
_       END;                                                                ¦
_    END;                                                                   ¦
_ CALL $RPTPRINT('DEPT_EXPNSE', MEDIA);                                     ¦ 2
_ ------------------------------------------------------------------------------
```

### Explanation of the Rule

The rule does the following:

1. Uses a FORALL statement to access parameter values held in the table $EMP_EXPENSE.
2. Uses a FORALL statement to access all the instances of the source table EMP_EXPENSE.
3. Assigns data to the report table DEPT_EXPENSE.
4. Inserts the data into the report table of a report also called DEPT_EXPENSE.
5. Prints the report to the specified output.

See Also *TIBCO Object Service Broker Programming in Rules* for information about the FORALL statement.

# Chapter 5 Editing a Table Definition

This chapter describes how to edit, copy, and delete a table definition.

## Topics

# Editing a Definition

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to the TIBCO Object Service Broker UI online help.

## Overview

The Table Definer provides editing facilities for making changes and additions to a table definition. Permissible changes do not affect existing data. However, changes are restricted in ways that ensure data integrity and security.

## Updating Specifications with a Definition

You can update the specifications within a definition by:

- Positioning the cursor and over-typing
- Inserting or deleting characters
- Using line commands
- Using primary commands

Your updates are validated when you press PF3. An appropriate message appears if the updates are not valid.

## Permissible Editing Changes

After you load the table with data, you can still make the following selective changes to the table definition:

- Change field names for fields that are not indexed (primary or secondary)
- Change required fields
- Change the unit
- Change the parameter names
- Add/delete location parameters

- Change event rule specifications
- Change the semantic type
- Increase length of syntaxes B, C, RD, UN, V, W, and P
- Add fields (the fields must be added to the end of the table definition)

## Non-permissible Editing Changes

After you load the table with data, the following changes are not permitted to the table definition:

- Change table name (changing the table name creates a new table definition)
- Change the table type
- Change the IDgen specification
- Add/delete data parameters
- Change the length of parameters
- Change key fields or indexed fields
- Change the syntax type
- Change the length of syntax F
- Change the number of decimal places
- Delete fields
- Decrease the length of a field

## Commands and PF Keys Available

To view a list of line and primary commands and PF keys available from the Table Definer, press PF1.

# Copying a Definition

There are a number of methods that you can use to copy an existing table definition. Using an existing definition and modifying only the portions required simplifies coding and the assignment of data. In the case of distributed development, it also assists in the management of table definitions across nodes as the table containing the data must be defined on the local and remote nodes.

You can copy a definition from within the Table Definer or through the use of shareable tools. If the source table is on a TIBCO Object Service Broker node that is remote to the one you are working on, you must copy the definition using a shareable tool.

## Copying a Definition Using the Table Definer

When you are within an existing definition in the Table Definer, you can copy a definition by doing one of the following:

- Changing the name of the definition that you are viewing to a new name and pressing Enter.
- Using the **`COPY`** command.

  To use this command, at the primary command field type `COPY tablename` where *tablename* is the name of the table that you are copying.

When using the **`COPY`** command:

- The command fails if the displayed table already contains data.
- The name of the table that you are viewing is retained but all the other information in the definition is overwritten with the values from the copied table.

## Copying a Definition Using Shareable Tools

Two shareable tools, COPY_DEFN and COPYDEFN, are available to you to copy existing definitions. Using these tools, if the correct security access is set up, you can copy definitions across TIBCO Object Service Broker nodes or within your local node. COPY_DEFN is called from within a rule and COPYDEFN executed via the workbench option CD copy defn.

### Example Rule

The following rule calls the tool COPY_DEFN to copy the definition of the DEPARTMENTS table from Node A to Node B. Because a table definition is being copied, values are not required for the arguments *library*, *environment*, or *parentonly:*

```
COPY_TABLEDEFN;
_
_ ------------------------------------------------------------------------
_ ------------------------------------------------------------+-------------
_ CALL COPY_DEFN('TABLE', 'DEPARTMENTS', '', '' , 'NODEA',    | 1
_    'NODEB', '');                                           |
_------------------------------------------------------------------------
```

See Also *TIBCO Object Service Broker Shareable Tools* for information on the tools.

# Editing a Definition for Distributed Development

## Purpose of the Definition

If you are working in a distributed data environment and you are accessing data across nodes, you must establish a relationship with the tables to be used. You establish this relationship through the definitions of the related tables. When this relationship is established, your data can be stored remotely or locally to the definition.

## Definition and Data Requirements for Distributed Data

To access data that resides on another node, the following conditions for the table definitions must be met:

- The table containing the data must be defined with the same table name and the same location parameter name on both the local and the remote nodes.
- If data is to reside on a particular node, the *full* definition for the table must exist on that node.
- If data is not to reside on a particular node, only a minimal definition of that table is required at that node. Optionally, you can have a full definition on both nodes if you want.
- If data resides on both a local and remote node, the data on *both nodes must be accessible* when the data on the remote node is being accessed.

## Copying a Definition

The simplest way to relate definitions is to copy a full definition from one node to another. If required, you can modify it to be only a minimal definition by following the tasks below. Because your tables are to be used in a distributed data environment, you must use the shareable copy tools COPY_DEFN and COPYDEFN to copy the definition. These tools are discussed in Copying a Definition on page 62 and described fully in *TIBCO Object Service Broker Shareable Tools*.

## Defining a Minimal Definition

As previously noted, you can use a minimal definition on a node so long as data is not to reside with the definition. A minimal definition consists of:

- The table name.

- The location parameter name and attributes.
- The name of the remote node where the full definition is located. This value is determined using the order of evaluation described in Order of Evaluation to Determine Location on page 17.

### Modifying an Existing Full Definition

If you have already used one of the shareable copy tools, complete the following tasks to create a minimal definition from the full definition. Refer to Definition and Data Requirements for Distributed Data on page 64 for restrictions on values.

1. Invoke the Table Definer with the name of your table.

   The name must be same as the name of the table with the full definition.

2. Delete the values in all the fields of the table identification portion, leaving only the Table field defined.
3. Using the **D** line command, delete all the parameters except for the LOCATION parameter, and delete all the fields.
4. Define the location parameter as described in Defining a Location Parameter on page 35.

   The name of the location parameter must be the same as the location parameter for the table with the full definition.

When defined, the source for the full definition is available through the location parameter.

### Creating a New Minimal Definition

If you have not already copied the definition, complete the following tasks to create a minimal definition.

1. Invoke the Table Definer with the name of your table.

   The name must be the same as the name of the table with the full definition.

2. Delete the values in all the fields of the table identification portion, leaving only the Table field defined.
3. Define the location parameter as described in Defining a Location Parameter on page 35.

   The name of the location parameter must the same as the location parameter for the table with the full definition.

When defined, the source for the full definition is available through the location parameter.

### Defining a Subview on a Minimal Definition

You can also define a subview on the minimal definition of a table. The tasks required are:

1. On the remote system complete each of the following:
   - Define the full definition of the base table.
   - Define a subview of the base table.
2. On the local system complete each of the following:
   - Define a minimal definition of the base table.
   - Define a subview table with the same name as the remote subview and its source is the minimal definition defined above.

# Deleting a Definition

There are a number of methods that you can use to delete a table definition. Refer to the following table to determine the appropriate method to use:

| If the table … | Use … |
|---|---|
| Was promoted to a target system. | Promotion system for that location. |
| Was not promoted and is on your local node. | Table Definer or DELETE_DEFN. |
| Was not promoted and is on a node remote to the node where you are presently working. | DELETE_DEFN. |

## Considerations when Deleting an Object

If an object such as a table definition was promoted to another (target) system, you must submit a change request through the Promotion system (of the source system) to extend the deletion to the target system. If you do not issue a change request to delete the definition, the following occurs:

- The table exists on the target system and no rights are associated with it on the source system.
- If a new object with the same name is created on the source system, the creator is unable to promote the object to the target system because an object with the same name already exists there.

## Deleting a Definition Using the Table Definer

When you are within an existing definition in the Table Definer, you can delete a definition by doing one of the following:

- Press PF22.
- Use the **DELETE** command.

In either case you are prompted to confirm the deletion.

## Deleting a Definition Using a Shareable Tool

The shareable tool DELETE_DEFN is available to you to delete existing definitions. Using this tool, if the correct security access is set up, you can delete definitions across TIBCO Object Service Broker nodes or within your local node. DELETE_DEFN is called from within a rule.

### Example Rule

The following rule calls the DELETE_DEFN tool to delete the definition of the DEPARTMENTS table from Node A. Because a table definition is being deleted, values are not required for the arguments *library*, *environment*, or *parentonly*:

```
DELETE_TABLEDEFN;
_
_-------------------------------------------------------------------------
 _ ------------------------------------------------------------+--------------
 _ CALL DELETE_DEFN('TABLE', 'DEPARTMENTS', '', '' , 'NODEA',  | 1
 _      '');                                                   |
 _-------------------------------------------------------------------------
```

See Also

- *TIBCO Object Service Broker Shareable Tools* for information about the tools.
- *TIBCO Object Service Broker Managing Deployment* for information about change requests.

# Chapter 6 Manipulating Data in a Table

This chapter describes how to manipulate data in a table, including processing differences for EES tables.

## Topics

# Data Manipulation Tools

## TIBCO Object Service Broker UI

This chapter describes how to perform various tasks in TIBCO Object Service Broker using the text-based workbench. You can also perform these tasks using the TIBCO Object Service Broker UI, which provides a graphical environment for TIBCO Object Service Broker development. For information about using the TIBCO Object Service Broker UI, refer to the TIBCO Object Service Broker UI online help.

## Available Methods

A number of methods are available for you to manipulate the data in your tables from within TIBCO Object Service Broker. You can use:

- The Table Browser
- The Table Editor
- The Single Occurrence Editor (SOE)
- Workbench options
- Rules statements
- Shareable tools

## Choosing a Tool

The method that you choose determines the tool that you use. Your choice depends on when and how you want to manipulate the data. Use the following table to determine the best option for your requirements:

| To do the following… | Use the… |
|---|---|
| View the data without locking the table. | Table Browser. |
| Do updates without locking the table. | Single Occurrence Editor. |
| Make only a small set of updates. | Single Occurrence Editor. |
| Insert new data. | Table Editor or rules. |

| To do the following… | Use the… |
|---|---|
| Update a table that has event rules defined. | Execute the shareable tool STE or use rules. |
| Copy data. | CT copy table workbench option or shareable tools. |
| Delete data from the whole table. | CL clear table workbench option or shareable tools. |

## Browsing Data with the Table Browser

You can use the Table Browser to browse data defined in the MetaStor. When you are using the Table Browser, you do not lock access to data defined in tables. The following section describe what you can and cannot do when browsing a table with the Table Browser.

### What You Can Do With the Table Browser

When browsing data with the Table Browser, you can:

- Browse TDS, SUB, CLC, EES, SES, IMS, PRM, VSM, DAT, DB2, ADA, SLK, and IDM table types.

  Refer to Available Table Types on page 10 for a description of each table type.
- Browse IMP table types if they have a primary key defined.
- Browse tables that have a maximum of eight fields in the composite primary key, although TIBCO Object Service Broker allows a maximum of 16 fields in a composite primary key.
- Browse tables with fields of syntax W (double-byte and single-byte character strings).
- Browse tables with fields of syntax UN (UTF-16 Big Endian Unicode character strings). Unicode strings are shown as sequences of groups of four hexadecimal digits, each group representing a Unicode character.
- Browse tables with fields of syntax RD (strings of raw binary data bytes). Raw data strings are shown as sequences of groups of two hexadecimal digits, each group representing a raw data byte.
- Use the primary command **`SHOW`** (or its corresponding function key PF19) to display and/or re-display fields excluded from the screen because of a long primary key or because the **`EXCLUDE`** command was issued.

### What You Cannot Do With the Table Browser

When browsing data with the Table Browser, you cannot:

- Browse EXP, RPT, or SCR table types.
- Browse tables that have more than eight fields in a composite primary key. You must use rules to access the tables.
- Use Table Browser operations on fields of syntax W.

A dot (.) appears when the field length exceeds the maximum display size. To view the field, use the Single Occurrence Editor.

## Manipulating Data with the Table Editor

You can use the Table Editor to edit an entire table in the database. It combines the features of a full-screen editor and a line editor. The Table Editor locks the table instance if the table is parameterized that you are using so that other users cannot access it to make editing changes. Therefore, if only minor changes are required to the table, use the Single Occurrence Editor.

Note the following considerations when manipulating data with the Table Editor.

### What You Can Do With the Table Editor

When manipulating data with the Table Editor, you can:

- Edit TDS, VSM, IMS, DB2, DAT, ADA, SLK, and SUB table types.
- Edit tables that have a maximum of eight fields in the composite primary key, although TIBCO Object Service Broker allows a maximum of 16 fields in a composite primary key.
- Use the primary command **`SHOW`** (or its corresponding function key PF19) to display and/or re-display fields excluded from the screen because of a long primary key or because the **`EXCLUDE`** command was issued.

### What You Cannot Do With the Table Editor

When manipulating data with the Table Editor, you cannot:

- Edit PRM, IDM, IMP, SCR, RPT, and EXP table types.
- Roll back editing changes made to VSM type tables. Changes are committed as soon as they are made.
- Edit a screen other than the current screen when using line commands.

- Edit tables that have more than eight fields in a composite primary key. You must use rules to access the tables.

A dot (.) appears when the field length exceeds the maximum display size. To view the field, use the Single Occurrence Editor.

## Manipulating Data with the Single Occurrence Editor

You can use the Single Occurrence Editor to update or display a single occurrence in a table.

Note the following considerations when manipulating data with the Single Occurrence Editor.

### What You Can Do With the Single Occurrence Editor

When manipulating data with the Single Occurrence Editor, you can:

- Make changes to tables while using the Table Browser or Table Editor. The Single Occurrence Editor locks a single occurrence rather than the whole table or table instance if the table is parameterized.
- Display and edit fields that are too large for the display table in either the Table Browser or the Table Editor.

### What You Cannot Do With the Single Occurrence Editor

You cannot use the Single Occurrence Editor on IMP tables.

# Invoking the Table Browser, Table Editor, and Single Occurrence Editor

## Invoking the Table Browser

You can access the Table Browser from the workbench by doing one of the following:

- Type an existing table name to the right of the BR browse table option and press Enter. This displays the initial Table Browser screen.
- Position your cursor to the right of the BR browse table option and press Enter. This displays the Object Manger screen for the Table Browser. Select an object from this screen to invoke the Table Browser screen.
- Type BR and an existing table name in the command line.
- Execute the tool STEBROWSE(*input*) where *input* is a string containing the table name (and parameters, if any).

## Invoking the Table Editor

You can access the Table Editor from the workbench by doing one of the following:

- Type an existing table name to the right of the ED edit table option and press Enter. This displays the initial Table Editor screen.
- Position your cursor to the right of the ED edit table option and press Enter. This displays the Object Manager screen for the Table Editor. Select an object from this screen to invoke the Table Editor screen.
- Type ED and an existing table name in the command line.
- Execute the STE(*tablename*) tool where *tablename* is the name of the source table and any parameters.

## Invoking the Single Occurrence Editor

You can access the Single Occurrence Editor by doing one of the following:

- From either the Table Browser or Table Editor, type **S** in the line command field of the required occurrence and press Enter.
- Execute the tool SOE(*tablespec*) where *tablespec* is the name of the table and any parameters containing the single occurrence that is to be edited.

If you do not supply a value for *tablespec*, pressing Enter displays a screen prompting for a value.

See Also *TIBCO Object Service Broker Shareable Tools* for information about the tools.

# Replacing Data

## Replacing Data Using the Table Editor

To replace values in a field using the Table Editor, you can:

- Overtype existing values
- Delete existing values
- Use the primary command **CHANGE**

When you overtype or delete existing values, press Enter to update the table with the new values.

## Replacing Data Using the CHANGE Command from the Table Editor

When using the **CHANGE** command, you must enter the field name, the value that is changed, and the new value. For example, type `change deptno=10 60` in the primary command field. Deptno is the field name, 10 is the value that is changed, and 60 is the new value.

### Notes on the CHANGE Command

- Enclose values (constants) with spaces or non-numeric characters in single quotation marks. For example, type `change position='staff pe' 'resources'` in the primary command field.
- The search starts at the cursor position, not the beginning of the table.
- Press PF6 to change the next occurrence.
- Press PF5 if you want to inspect the occurrences before changing them.

## Controlling the Scope of the CHANGE Command

You have three options that control the scope of the **CHANGE** command:

- PAGE
- REST
- ALL

The options REST and ALL, if used on large tables, could cause the commit limit to be reached. If the commit limit is reached, save your updates to this point, re-enter the command, and continue making the changes.

If you have a large table, use rules to replace the data. Refer to Replacing Data Using a Rules Statement on page 78 for more information.

### Controlling the Scope with the PAGE Option

The PAGE option limits the scope of the **CHANGE** command to occurrences on the displayed page. Occurrences above or below the displayed page remain unchanged. All changes are visible.

For example, typing `change deptno=10 60 page` in the primary command field changes all occurrences that you can currently see with department number 10 to department number 60. When the changes are made, the message line displays the number of changes made (possibly 0).

### Controlling the Scope with the REST Option

The REST option limits the scope of the **CHANGE** command to occurrences on the displayed page or on the pages after the displayed one. Occurrences on pages before the displayed one remain unchanged. Some changes could be invisible.

For example, typing `change deptno=10 60 rest` in the primary command field changes all occurrences on the display screen and below with department number 10 to department number 60. When the changes are made, the message line displays the number of changes made (possibly 0).

### Controlling the Scope with the ALL Option

The ALL option changes the scope of the **CHANGE** command to the entire table. All occurrences with the specified field value are changed to the new value.

For example, typing `change deptno=10 60 all` in the primary command field changes all occurrences with department number 10 to department number 60. The message line displays the number of changes made (possibly 0).

## Replacing Data Using the Single Occurrence Editor

To replace data using the Single Occurrence Editor, invoke the Single Occurrence Editor and overtype or delete the existing data. Type **SAVE** or press PF3 to save the changes, update the table, and exit from your editing session.

## Replacing Data Using a Rules Statement

To replace data using a rules statement, use the REPLACE statement. First retrieve the data that you want to modify using a GET or FORALL statement. You cannot select a field using the REPLACE statement but you can use a WHERE clause to select a table instance. You also cannot replace a primary key value using the REPLACE statement.

### Example Rule for of Replacing Data

```
REPLACE_DEPTNO(region, empno, deptno);
_
_-------------------------------------------------------------------------
_ ------------------------------------------------------------+--------------
_ FORALL @EMPLOYEES WHERE REGION = REGION :                   |   1
_    @EMPLOYEES.DEPTNO = DEPTNO;                              |
_    REPLACE @EMPLOYEES WHERE REGION = REGION;                |
_    END;                                                     |
_ ------------------------------------------------------------------------
```

### Explanation of the Rule

The previous rule:

1. Retrieves an occurrence in the @EMPLOYEES table based on the parameter value provided for the argument REGION and the value provided for EMPNO.
2. Assigns a new value for DEPTNO, based on the argument DEPTNO.
3. Replaces the existing value for DEPTNO with the new value provided by the argument DEPTNO.

See Also *TIBCO Object Service Broker Programming in Rules* for information on rules and the REPLACE statement.

# Inserting Data

## Inserting Data Using the I Line Command in the Table Editor

The line command `I` creates a new occurrence or multiple occurrences. It is the only line command that can be used on the top line command field. To use the line command `I` to insert data, complete the following tasks:

1. Type I in a line command field.

   You can type `I` in multiple line command fields at once.

2. Press Enter.

   A new occurrence is created below the selected one. The cursor is positioned in the first column of the primary key field, indicated by the ampersand (&).

3. Enter a unique value for the primary keys.
4. Type in values for the fields.
5. Press Enter.

## Inserting Data Using PF4 in the Table Editor

To use PF4 to insert a new line, complete the following tasks:

1. Position the cursor anywhere on an occurrence.
2. Press PF4.

   A new occurrence is created below the selected one, indicated by the ampersand (&).

3. Type in a unique primary key for the new occurrence.
4. Type in values for the fields.
5. Press Enter.

## Inserting Data Using the Single Occurrence Editor

To insert an occurrence from the Single Occurrence Editor, complete the following tasks:

1. Type S in the line command field of an existing occurrence.

   This invokes the Single Occurrence Editor.

2. Type CLEAR in the primary command field.

3. Press Enter.

   This clears all the existing data from the occurrence.

4. Enter a unique value in the primary key field.
5. Enter the values for the other fields as required.

### Repositioning of Data

If used from the Table Browser, the new occurrence is inserted in its correct position based on the primary key as soon as you save from the Single Occurrence Editor. If used from the Table Editor, the new occurrence is first inserted after the selected occurrence and repositioned after you save from the Table Editor.

## Inserting Data Using a Rules Statement

To insert data with a rules statement, use the INSERT statement. You can use a WHERE clause to select a table instance. You cannot insert an occurrence into a table if the primary key value for the occurrence already exists; you must delete the occurrence first using the DELETE statement.

### Example of Inserting Data

The following example illustrates a rule that inserts data:

```
    RULE EDITOR ===>                                                SCROLL: P
INSERT_EMPLOYEE(REGION, EMPNO);
_
_-------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL @EMPLOYEES WHERE REGION = REGION:                    |   1
_    @EMPLOYEES.EMPNO = EMPNO ;                               |
_    INSERT @EMPLOYEES WHERE REGION = REGION;                 |
_    END ;                                                    |
_ ------------------------------------------------------------------------------
```

This rule:

1. Gets the table instance of the @EMPLOYEES table based on the parameter value provided for the argument REGION.
2. Assigns a value for EMPNO, based on the argument EMPNO.
3. Inserts the new occurrence into the table.

See Also *TIBCO Object Service Broker Programming in Rules* for information on commits, the use of rules, and the INSERT and DELETE statements.

# Replicating Data

## Purpose of Replicating Data

Replicating an occurrence provides a template for a new occurrence. No value is provided for the primary key field for the replicated occurrence; you must enter a unique value for the new occurrence. Any of the following methods leaves the original occurrence unaltered and adds a new occurrence to the table.

## Replicating Data Using the R Line Command in the Table Editor

From the Table Editor, use the **R** line command to create a replica of one or more occurrences (you can replicate several occurrences at one time):

1. Type R in the line command field beside the occurrences you want to replicate.
2. Press Enter.

   The replicated occurrences appears below the selected lines.
3. Enter a unique value for the primary keys.
4. Change the values in the fields, where appropriate.

   The new occurrences are repositioned after you enter a unique primary key and save.

## Replicating Data Using the Single Occurrence Editor

You can replicate an existing occurrence by invoking the Single Occurrence Editor for an existing occurrence. Change the primary key value to a value that does not currently exist. You can then edit values for other fields as required.

For example, if the occurrence with primary key 80003 currently appears in the Single Occurrence Editor and you change this value to 81033 (perhaps changing values of other fields as well), a new occurrence is created.

### Repositioning of Data

If used from the Table Browser, the new occurrence is inserted in its correct position based on the primary key as soon as you save from the Single Occurrence Editor. If used from the Table Editor, the new occurrence is first inserted after the selected occurrence and repositioned after you save from the Table Editor.

# Deleting Data

## Deleting Data Using the D Line Command in the Table Editor

Using the **D** line command, you can delete multiple occurrences. To use the **D** line command, complete the following tasks:

1. Type D in the line command field beside the occurrences you want to delete.
2. Press Enter.

   The occurrences is deleted.

## Deleting Data Using PF16 in the Table Editor

You can use PF16 to delete one occurrence (where the cursor is positioned) at a time. To use PF16, complete the following tasks:

1. Place the cursor anywhere on the occurrence you want to delete.
2. Press PF16 to delete the occurrence.

   The occurrence is deleted.

## Deleting Data Using the Single Occurrence Editor

From the Single Occurrence Editor you have two options for deleting the occurrence:

- The **DELETE** primary command
- PF22

To use either method, complete the following tasks:

1. Press PF22 or type DELETE in the primary command field.
2. Press PF22 to confirm the deletion.

   If you decide to cancel the deletion, use any PF key other than PF22. When the deletion is completed the screen re-appears with the updated table.

## Deleting Data Using a Rules Statement

To delete data with a rules statement, use the DELETE statement. This statement requires the primary key to be available, either via a previous GET or FORALL on the table, or by explicit selection of the primary key.

You can use the WHERE clause and the primary key value to specify which occurrence to delete (that is, WHERE *primary key value* =).

### Example of Deleting Data

The following rule deletes a manager from the MANAGER table:

```
     RULE EDITOR ===>                                                SCROLL: P
DELETE_MANAGER(mgr#);
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ DELETE MANAGER WHERE MANAGER_NUM = MGR#;                    |    1
_ ------------------------------------------------------------------------------
```

## Deleting Data Using a Workbench Option

To delete data using the workbench, use the CL clear table option. To use this option, complete the following tasks:

1. Position your cursor to the right of CL clear table ==>.
2. Press Enter.

   A prompt screen appears.
3. At the Table Name prompts, type in the table name and parameter values, if the table is parameterized.

   For example, to delete data from the MIDWEST region of the @EMPLOYEES table, type: `@EMPLOYEES(MIDWEST)`.
4. If selection of data is required, at the Select Occurrences Where prompt, type your selection criteria.

   For example, to delete the occurrence for employee 22312, type: `EMPNO=22312`
5. Press Enter.

### Alternative Method to Delete Data

Alternatively, you can use the CL clear table option as follows:

1. Position your cursor to the right of CL clear table ==>.
2. Type in the name of the table, and the parameter values if the table is parameterized, followed by a comma (,).

   For example, to delete data from the MIDWEST region of the @EMPLOYEES table, type: `@EMPLOYEES(MIDWEST),`

3. Press Enter.

## Deleting Data Using a Shareable Tool

A number of shareable tools are available for you to delete data from a table. Use the tools as follows:

- To delete data at either a local or remote location, use DELETE_DATA. You can selectively delete data using this tool.
- To delete data locally, use $CLRTAB. You can do only a selection on parameter values using this tool.

See Also

- *TIBCO Object Service Broker Shareable Tools* for information on the tools.
- *TIBCO Object Service Broker Programming in Rules* for information about the use of rules and the DELETE statement.

# Copying Data

## Copying Data Using a Workbench Option

To copy data using the workbench, use the CT copy table option. To use this option, complete the following tasks:

1. Position your cursor to the right of CT copy table ==>.
2. Press Enter.

   A prompt screen appears.
3. At the Source Table Name prompt, type in the table name and, if the table is parameterized, the parameter values.

   For example, to copy data from the MIDWEST region of the @EMPLOYEES table, type: `@EMPLOYEES(MIDWEST)`.
4. At the Destination Table Name prompt, type in the table name and, if the table is parameterized, the parameter values.

   For example, to copy data to the TEST region of the @EMPLOYEES table, type: `@EMPLOYEES(TEST)`.
5. If selection of data is required, at the Select Occurrences Where prompt, type your selection criteria.

   For example, to copy the occurrence for employee 22312, type: `EMPNO=22312`.
6. Press Enter.

## Copying Data Using a Shareable Tool

You can copy data from one table or table instance to another using the shareable tool COPY_DATA. You can use this tool to copy data locally or remotely.

See Also *TIBCO Object Service Broker Shareable Tools* for information on the tools.

# Committing Changes

Tables are not actually updated until one of the primary commands that commits the changes is issued from the primary command field. These commands are:

- **SAVE**
- **SELECT**
- **ORDERED**
- **BROWSE**
- **EDIT**

Using PF3 also commits changes.

## Committing Changes Using the Table Editor

To commit changes and continue with your editing session, use the primary command **SAVE**. This commits changes without exiting the current session. To commit changes and exit to the workbench, press PF3.

The Table Editor displays a message at the bottom of the screen to indicate whether the changes are committed. For example, you see a message similar to: `Updates to table "EMPLOYEE(NORTHWEST)" saved`. When changes are committed, they are saved in the ascending order of the primary key.

When changes are not committed, a message at the bottom of the display screen gives a reason for the failure and the Table Editor positions the cursor where corrections are required.

## Committing Changes Using the Single Occurrence Editor

To commit changes to an occurrence, press PF3 or use the primary command **SAVE**. In either case, the new occurrence or the updates to an existing occurrence are committed, the table is updated, and the editing session is terminated.

## Committing Changes Using Rules

At the end of every transaction, TIBCO Object Service Broker implicitly commits changes to the database made since the last synchronization point. Synchronization points can be established using the COMMIT and ROLLBACK statements from within a rule.

## Understanding EES Table Considerations

This section defines EES tables and describes the procedure for processing them.

### Definition

When the table type in the table definer is initially changed to EES, Object Service Broker adds two predefined permanent fields to the bottom, as shown here:

```
    Table: EES_EXAMPLE      Type: EES   Unit: CLARKD            IDgen: N
   Source:
 Parameter Name  Typ Syn Len Dec Class            '      Event Rule   Typ Acc
---------------- -   -- --- --   -               '   ---------------- -   -
LOCATION          I   C   16  0    L              ' _
                                                 ' _
   Field Name   Typ Syn  Len Dec Key Ord Rqd    Default         Reference
---------------- -   -- ----- --  -   -   - ---------------- ----------------
@@UPDATE_COUNT   Q   B      4  0
@@REF_COUNT      Q   B      4  0
```

- The field `@@UPDATE_COUNT` maintains integrity of the table data.
- The field `@@REF_COUNT` contains a count of the number of times any given row in the table has been referenced.

Do not modify the definitions of those two fields, which must always remain as the last two fields in the table definition. If they are absent, Object Service Broker rejects access to the EES table.

### Processing

Because EES tables can be shared in a multiuser Execution Environment, although their processing is similar to that of SES and TEM tables, some changes are necessary to maintain data integrity. Read on for the details.

#### Table Access

For a nonupdate processing to an EES table, Object Service Broker obtains a shared global lock for all the EES tables for the duration of the request. Whenever a row in the table is referenced by a nonupdate request, Object Service Broker increments the field `@@REF_COUNT` by 1.

### Updates

For an update processing to an EES table, Object Service Broker obtains an exclusive global lock for all the EES tables for the duration of the update request. That is, once control is to return to the processing rule, Object Service Broker releases the lock.

### Insertion of Rows

To insert a row in a table in addition to normal processing of the fields, Object Service Broker sets the `@@UPDATE_COUNT` and `@@REF_COUNT` fields to 1.

### Replacement of Rows

To replace a row in a table, if the field `@@UPDATE_COUNT` exists in the replacement buffer, Object Service Broker compares it with the value in the actual table data. Depending on the result, either of the following occurs:

- If the comparison succeeds, Object Service Broker replaces the row in the actual table data and increments the fields `@@UPDATE_COUNT` and `@@REF_COUNT` by 1.
- If the comparison fails, Object Service Broker raises a `LOCKFAIL` exception and no replacement occurs.

If the field `@@UPDATE_COUNT` does not exist in the replacement buffer, Object Service Broker replaces the row in the actual table data and increments the fields `@UPDATE_COUNT` and `@@REF_COUNT` by 1.

### Deletions of Rows

To delete a row in a table, if the field `@@UPDATE_COUNT` exists in the deletion buffer, Object Service Broker compares it with the value in the actual table data. Depending on the result, either of the following occurs:

- If the comparison succeeds, Object Service Broker deletes the row.
- If the comparison fails, Object Service Broker raises a `LOCKFAIL` exception and no deletion occurs.

If the field `@@UPDATE_COUNT` does not exist in the row buffer, Object Service Broker deletes the row.

## Example: How the `@@UPDATE_COUNT` Field Maintains Data Integrity

Consider two transactions, A and B, both of which are attempting to update row X in an EES table. Assume that row X is currently in its initially inserted state, that is, fields `@@REF_COUNT` and `@@UPDATE_COUNT` are set to 1.

To maintain integrity, the rules for processing the table must read the data row before attempting to replace it, as follows:

- Transaction A gets row X in its row buffer: The field `@@UPDATE_COUNT` is 1 and the field `@@REF_COUNT` is 2.
- Transaction B gets row X in its row buffer: The field `@@UPDATE_COUNT` is 1 and the field `@@REF_COUNT` is 3.
- Transaction A updates the data in its row buffer and replaces the row. Since `UPDATE_COUNT` is 1 in both its row buffer and the table data, the replacement succeeds. Accordingly, the field `@@UPDATE_COUNT` becomes 2 and `@@REF_COUNT` becomes 4.
- Transaction B updates the data in its row buffer and replaces the row. Since field `@@UPDATE_COUNT` is 1 in its replacement buffer and is now 2 in the actual table data because of the replacement operation by transaction A, the replacement fails. Object Service Broker raises a `LOCKFAIL` exception.

See Also *TIBCO Object Service Broker Programming in Rules* for information on the COMMIT and ROLLBACK statements and synchronization of the database.

Chapter 7

# Coding Considerations for Event, Location, and Derived Value Rules

This chapter describes coding considerations for event, location and derived value rules.

## Topics

# Coding Event Rules

Using the event rule feature, you can associate business rules and policies with the definition of a table. These rules are run whenever data in the table is manipulated. You can code two types of event rules:

### Validation rule

A validation rule is used when the table is being modified. It checks the validity of modified values of fields in the table.

### Trigger rule

A trigger rule causes additional processing to take place when a table is accessed. For example, it can be used to create an audit trail, or maintain a one-to-one relationship between two tables.

These rules run based on defined accesses. All the rules that apply to a specific access are executed in the order in which they are entered in the event rule section.

## Conditions for Validation Rules

The following conditions apply to the coding of a validation rule:

- No updates to tables holding persistent data (TDS, DB2, and so on) are allowed during the validation process.
- Updates to tables holding temporary data (TEM, SES, and EES) are allowed.
- The rule must be a function. A successful validation must return Y. An unsuccessful validation can return any non-Y value. It is recommended that the rule return either N or a message explaining why the validation was not successful.
- The validation must explicitly handle its own exceptions. If a validation does not handle an exception, the transaction that caused the validation is terminated.

## Conditions for Trigger Rules

Trigger rules can be nested up to a maximum depth of five accesses. For example, an access to one table can trigger access to another table, which in turn can trigger access to another table, and so on. The following restrictions apply to the coding of a trigger rule:

- It must not be a function; however, other rules invoked within it can be functions.
- It cannot alter the contents of the triggering row.
- It must not terminate the transaction and therefore cannot contain the TRANSFERCALL statement.
- It must explicitly handle its own exceptions. If a trigger does not handle an exception, the transaction that caused the trigger is terminated.
- It should not contain COMMIT or ROLLBACK statements. If you need to explicitly keep changes to data, start a new transaction using the EXECUTE statement.

### When are Locks Released?

Locks taken by an event rule are released at the end of the transaction in which it is running.

## Search Path

The search path for the application invoking the event rule determines the search path for the event rule. Because the Table Editor on the developer's workbench has a search path of S (system library), when you are accessing the table from the workbench you must execute the shareable tools STEBROWSE or STE to browse or edit a table using an event rule. For more information about these tools, refer to *TIBCO Object Service Broker Shareable Tools*.

## Sample Set of Event Rules

The sample table has two event rules defined, DELETEMPNO and VALIDEMPNO. The following illustrations show the coding of these rules.

### Sample Trigger Rule

The first rule, DELETEMPNO, inserts data into the table @DELETEMPNO when an occurrence is deleted from the @EMPLOYEES table.

```
DELETEMPNO;
_
_-------------------------------------------------------------------------
_ ------------------------------------------------------------+-----------
_ @DELETEMPNO.EMPNO = @EMPLOYEES.EMPNO;                       | 1
_ INSERT @DELETEMPNO;                                         | 2
_ ------------------------------------------------------------------------
```

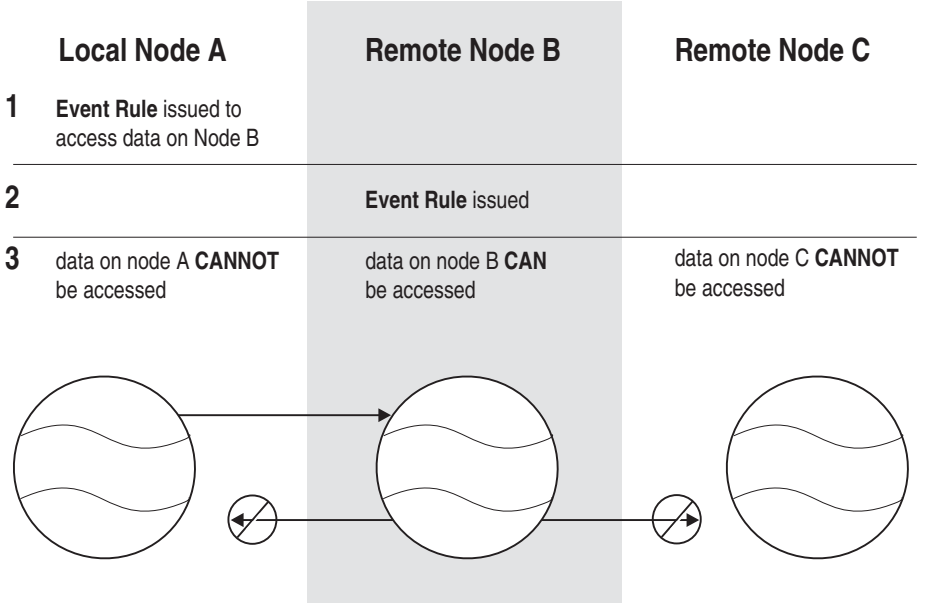
### Sample Validation Rule

The second rule, VALIDEMPNO, determines if an employee number entered in as data to the table is below an accepted numeric limit. It returns a message to the screen if the supplied value is invalid.

```
VALIDEMPNO;
_
_-------------------------------------------------------------------------
_ @EMPLOYEES.EMPNO <= 9999;                                  | Y  N
_ ---------------------------------------------------------+--------------
_ RETURN('The value for EMPNO must be greater than 9999');   | 1
_ RETURN('Y');                                               |    1
_ ---------------------------------------------------------------------------
```

## Event Rule Processing Across Nodes

An event rule running on a local node can access data on a remote node. An event rule running on a remote node as the result of a remote data access can only access data on the same node. The following illustration shows the valid and invalid accesses.



# Coding Rules to Determine Location

## Conditions that Apply

If you are using a rule to derive a value for the location of data, it:

- Must be a function
- Must have its arguments defined in the following order:
  a. An argument for the table name
  b. An argument for each data parameter
  c. An argument for the location parameter
- Should not contain COMMIT or ROLLBACK statements

## Search Path

The search path is determined by the current search path for the session. Because the Table Editor on the developer's workbench has a search path of S (for system library), when you are accessing the table from the workbench you must execute the shareable tools STEBROWSE or STE to browse or edit a table using a derived rule for location.

## Sample Source Rule Definition

The following example illustrates a sample definition for the **Src** and **Sourcename** fields for the NODENAME location parameter:

```
COMMAND==>                          TABLE DEFINITION

       Table: @EMPLOYEES         Type: TDS   Unit: DOCEXMPL            IDgen: N

   Parameter Name  ult     Src   Sourcename
   --------------- --------  -  ----------------
 _ REGION
 _ NODENAME                   D  FIND_LOCATION
```

## Sample Set of Source Rules

The following is an example of a set of rules used to return the value of a location. The rules use the table NODENAMES to determine the location. The two arguments for the first rule refer to the table name, and the two parameters of the @EMPLOYEES table, REGION and NODENAME.

```
    RULE EDITOR ===>                                                SCROLL: P
FIND_LOCATION(TABLENAME, DATAPARM, LOCPARM);
_
_ ------------------------------------------------------------------------------
_ DATAPARM = NULL;                                              | Y N
_ --------------------------------------------------------------+--------------
_ GET NODENAMES WHERE NAME = 'MASTER';                          | 1
_ RETURN(NODENAMES.DEFAULT);                                    | 2
_ GET NODENAMES WHERE NAME = DATAPARM;                          |   1
_ RETURN(FIND_LOCATION2(DATAPARM));                             |   2
_ ------------------------------------------------------------------------------
_ ON GETFAIL NODENAMES :
_    RETURN(DATAPARM);
```

If the value for the data parameter is not null, the following rule is used to get the value for the location.

```
    RULE EDITOR ===>                                                SCROLL: P
FIND_LOCATION2(DATAPARM);
_
_ ------------------------------------------------------------------------------
_ NODENAMES.PREFIX=HEADSTRING(P1, 3);                           | Y N
_ --------------------------------------------------------------+--------------
_ RETURN(NODENAMES.DEPARTMENT);                                 | 1
_ RETURN(NODENAMES.DEFAULT);                                    |   1
_ ------------------------------------------------------------------------------
```

## Modifying the Default Remote Location for a Session

The default remote location for your current session can be modified using the tools SETREMOTELOC(*remoteloc*) and REMOTELOCATION:

| | |
|---|---|
| SETREMOTELOC | Sets the value for your default remote location |
| REMOTELOCATION | Returns the current value for your default remote location |

### Setting the Peer Server

Using the shareable tool @PEERSERVERID, you can also specify which peer server you should be using.

### Sample Rule to Change the Default Location

The following rule changes the default remote location, if it is not already the value required.

```
    RULE EDITOR ===>                                                SCROLL: P
CHANGE_LOCATION(VALUE);
_
_ ------------------------------------------------------------------------------
_ VALUE = REMOTELOCATION;                                       | Y N
_ ------------------------------------------------------------+--------------
_ CALL SETREMOTELOC(VALUE);                                     |   1
_ CALL ENDMSG('THE LOCATION IS ' || VALUE);                     | 1 2
_ ------------------------------------------------------------------------------
```

See Also *TIBCO Object Service Broker Shareable Tools* for information about the tools.

# Coding Rules to Derive Values

## Conditions that Apply

If you are using a rule to derive a value for a field, the rule must be a function.

## Search Path

The search path is determined by the current search path for the session. Because the Table Editor on the developer's workbench has a search path of S (system library), when you are accessing the table from the workbench you must execute STEBROWSE to browse, or STE or edit, a table using a derived rule for location.

### Sample Definition

The following example illustrates a sample definition for the **Src** and **Sourcename** fields in the @EMPLOYEES_SUB table. In this definition, the HIREDATE field has the MODIFY_DISPLAY rule defined to it.

```
                        Synt    Dec   Order
     Field Name    Type   Len    Key    Rqd     Default      Src  Source Name
   ----------------  - - ----  -- -   -  -  ---------------- - ----------------
_ EMPNO                    0   0 P      N
_ LNAME                    0   0        N
_ POSITION                 0   0        N
_ MGR#                     0   0        N
_ DEPTNO                   0   0        N
_ HIREDATE           S C   10   0        N                    D MODIFY_DISPLAY
```

### Sample Source Rule

The following example illustrates the sample source rule. The rule uses the shareable tool $DATE_PIC to modify the display of the employee's hire date.

```
MODIFY_DISPLAY;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ RETURN($DATE_PIC('YY MMM DD', EMPLOYEES.HIREDATE));          | 1
_ ------------------------------------------------------------------------------
```

See Also *TIBCO Object Service Broker Shareable Tools* for information about these tools.

# Coding Rules for Remote Table Access

## Remote Table Access

The following rule illustrates replacing data by remote table access.

```
    RULE EDITOR ===>                                                SCROLL: P
REMOTE_ACCESS(LOCATION_PARM);
_ LOCAL FLD2;
_ ------------------------------------------------------------------------
_ ------------------------------------------------------------+-------------
_ FORALL TABLEA WHERE LOCATION = LOCATION_PARM & FIELD2 =      | 1
_       'A' :                                                 |
_    TABLEA.FIELD3 = 'B';                                     |
_    FLD2 = TABLEA.FIELD2;                                    |
_    REPLACE TABLEA WHERE LOCATION = LOCATION_PARM;           |
_    END;                                                     |
_ ------------------------------------------------------------------------
```

## Peer-to-peer Access

In peer-to-peer access, a remote FORALL generally performs better than a remote INSERT or REPLACE. Example 1 provides better performance than Example 2 because in example 1 the system is able to buffer multiple occurrences from the remote FORALL and return them in a single peer-to-peer operation; whereas, in example 2, each occurrence requires an individual peer-to-peer data movement.

### Example 1: A Single Peer-to-peer Operation

The following example illustrates a single peer-to-peer operation in a remote FORALL:

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

### Example 2: Individual Peer-to-peer Operations

The following example illustrates individual peer-to-peer operations in a local FORALL:

```
RULE EDITOR ===>                                                    SCROLL: P
PEER_ACCESS2;
_
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ FORALL LOCAL_TABLE :                                        | 1
_    REMOTE_TABLE.* = LOCAL_TABLE.*;                          |
_    INSERT REMOTE_TABLE;                                     |
_    END;                                                     |
_ ------------------------------------------------------------------------------
```

See Also *TIBCO Object Service Broker Shareable Tools* for information on tools for remote data accessing.

# Chapter 8 Managing TIBCO Object Service Broker MAP Data Definitions

This chapter describes how you use MAP tables to retrieve and manipulate external data in main storage using the TIBCO Object Service Broker table model.

## Topics

# MAP Tables

## What is a MAP Table?

A MAP table is a TIBCO Object Service Broker analog of a C or PL/I structure, an assembler language DSECT, or COBOL file definition. Using the Table Definer, you can define a MAP table so that its fields map an area of main storage. External data can be in various storage locations such as: CICS COMMAREA, IMS SPA, or COBOL structure.

Each field has an offset associated with it. The offset allows an external data type to begin on any byte boundary and possibly to overlap the other fields in the same table definition.

You use the Table Definer to define a MAP table. A pointer to the storage area is provided by the external environment or program that calls TIBCO Object Service Broker. TIBCO Object Service Broker itself also provides facilities for allocating storage of several classes for use with MAP tables. Pointers to these areas can in turn be passed to external programs called by TIBCO Object Service Broker.

### Data Types Supported

Both external and internal (TIBCO Object Service Broker) data formats can be specified. Since not all external data formats exactly match a TIBCO Object Service Broker data format, conversion is provided between internal and external formats as required.

## Main Storage Area

The main storage area used by a MAP table is defined by the following three values:

- A pointer to the beginning of the area
- The length of each row of the table
- The number of rows traversed

TIBCO Object Service Broker enforces certain security rules for storage access and provides a mechanism for registering storage for use with MAP tables.

## Who Should Use MAP Tables?

MAP tables are intended for use by experienced developers who are thoroughly familiar with both TIBCO Object Service Broker and the environment where their application executes. Incorrect or careless use of MAP tables could cause errors or data corruption that extends beyond the application using these tables. Therefore, TIBCO Object Service Broker provides restrictive default security controls for MAP table use.

These controls should be overridden only to the extent necessary and all MAP table applications should be carefully reviewed to ensure their correctness.

## How to Use MAP Tables

To use MAP tables, complete the following tasks:

| | Task | Refer to |
|---|---|---|
| A | Define a table of type MAP. | Accessing Storage Data from TIBCO Object Service Broker, page 107. |
| B | Acquire and register the memory space to be mapped to and accessed by the MAP table. | @MAP in *TIBCO Object Service Broker Shareable Tools* and Chapter 9, Manipulating Storage Data Using TIBCO Object Service Broker MAP Tables, page 119. |
| C | Perform the required accesses to the MAP table. | Chapter 9, Manipulating Storage Data Using TIBCO Object Service Broker MAP Tables, page 119. |

See Also *TIBCO Object Service Broker Shareable Tools* for information on the @MAP tool

# 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 MAP and press Enter; the appropriate Table Definition screen appears.

| To define a table of type… | Refer to… |
|---|---|
| MAP | This Chapter. |

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

# Using Data Discovery

When you create a MAP 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 on helping define MAP tables with Data Discovery.

# Accessing Storage Data from TIBCO Object Service Broker

To access storage data directly from TIBCO Object Service Broker, you must define a TIBCO Object Service Broker table of type MAP. A MAP table has three required parts: an address parameter, a primary key field, and one data field. It can have additional data fields and an optional count and/or location parameter.

### Table Definer Screen for a MAP Table

```
 COMMAND==>                          TABLE DEFINITION

      Table: MAP_TABLE         Type: MAP   Unit: USR40              IDgen: Y

   Parameter Name Typ Syn Len Dc Cls Reference        '      Event Rule   Typ Acc
   ---------------- -  -- --- --  - ----------------- '   ---------------- -   -
 _ ADDRESS             B    4  0  A                   ' _
 _ LOCATION          I  C   16  0  L                   ' _
 _                                                     ' _
                   | ----- EXTERNAL ----|----------- MetaStor --------------   -
      Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
   ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
 _ KEY               B       4   0          P   I   B     4  0
 _
 _
 _
 _
 _
 _
 _
 _
 _
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 105.

## Steps Required to Define a MAP Table

After invoking the Table Definer (refer to Initial Step for Defining Tables on page 104 for information on invoking the Table Definer), complete the following tasks to define a MAP table:

| | Task | Required | Refer to page |
|---|---|---|---|
| A | Identify the table. | Y | 109 |
| B | Specify address, count, and location parameters. | Y | 110 |
| C | Specify event rules. | N | 112 |
| D | Define fields. | Y | 113 |

# Task A: Identify the Table

## Purpose of this Task

This task 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: MAP_ONE        Type: MAP   Unit: USR40          IDgen: Y
```

## 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**, **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 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 MAP, which you changed in Initial Step for Defining Tables on page 104. |
| **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 must be set to Y for MAP tables. |

# Task B: Specify Address, Count, and Location Parameters

## Purpose of this Step

You can use this step to specify three types of parameters:

- Address
- Count
- Location

### Parameter Segment

The following example illustrates the fields used to specify address, count, and location parameters. To view additional fields, use PF11.

```
   Parameter Name Typ Syn Len Dc Cls Reference
   ---------------- -  -- --- --  - ----------------
_ ADDRESS             B    4  0  A
_ COUNT               B    4     C
_ LOCATION          I C   16  0  L
```

## Address Parameter

You require an address parameter to access storage data. This parameter must have syntax B, a length of 4, and class A (**Cls** field).

## Count Parameter

You can use an optional count parameter to limit the number of occurrences eligible for selection by a GET or FORALL statement. This does not mean the number of occurrences that are retrieved; the number of occurrences selected is equal to or lower than the count value, and could be zero. This parameter must have syntax B, a length of 4, and class C (**Cls** field). If you do not specify this parameter, a default value of infinity is used.

## Location Parameter

You can use an optional 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.

If you use a MAP table remotely, all storage references must be to addresses valid on the remote system.

### 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 name of the remote node where the full definition is located must be supplied in the **Default** field, **Src** field, or **Src** and **Sourcename** field.

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.

Define address and count parameters on the full definition, not a minimal definition.

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

# Task C: Specify Event Rules

## Purpose of this Task

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 access to MAP 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 D: Define Fields

## Purpose of this Task

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

### Field Definition Segment

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

```
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
   ---------------- ---- ----  --  ------  -   -   -- ---- --  - ----------------
_ KEY               B       4   0          P   I   B     4  0
_
_
_
_
```

## Considerations

Note the following considerations:

- When defining fields, you can type in external 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 MAP Attributes

The following fields are used to specify the external MAP attributes. For valid values, use PF1.

| | |
|---|---|
| **Field Name** | This field contains the name of the primary key or data field you are creating. This name must be unique within the table. You can use a name already used for as a field in any other table; if you are moving data between this table and another table, giving fields the same name simplifies the process. |
| **Xsyn** | This field contains the external syntax for the field. The external syntaxes E and J are not valid for OSB for z/OS. For more information on external syntaxes, refer to Mapping Data Types for MAP Table Definitions on page 148. |
| **Xlen** | This field contains the external length for the field. |
| **Xdec** | This field contains the external number of decimal places for the field. |
| **Offset** | This field contains the external offset of the field based on the length of the field. The offset is calculated from the start of the row (field) the cursor is on to the end of the defined fields in the TIBCO Object Service Broker MAP table definition. Overlaps of fields are allowed. 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>The key field does not participate in offset calculations; the first non-key field has offset 0 by default. |

## Specifying Internal TIBCO Object Service Broker Attributes

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

| | |
|---|---|
| **Key** | This field indicates if the MAP fields are to be used as a primary key. This field must be the first one in the definition and must have syntax B and length 4. Only a single key field can be specified. |
| **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. Valid combinations are described in *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. Valid combinations are described in *TIBCO Object Service Broker Programming in Rules*. If not specified, this field defaults to an appropriate syntax based on the external syntax and length (**Xsyn** and **Xlen** fields). |
| **Len** | This field contains the length of the field. The default is 0. If not specified, this field defaults to an appropriate length based on external syntax and length (Xsyn and Xlen fields). |
| **Dec** | This field contains the number of digits to appear to the right of the decimal point. The default is 0. The data is padded or truncated as necessary. Depending on the syntax specified in the **Syn** field, define this field as follows:<br>• For syntax P, the number of decimal places must be smaller than twice the length of the entire field.<br>• For syntaxes B, C, F, RD, UN, and V, the number of decimal places must be 0. |
| **Rqd** | This field contains a value that determines whether a user is required to provide a value in this field for each occurrence in the table (for example, a primary key). 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 is used for an occurrence that has no value assigned to it.<br><br>Default values are not permitted for fields of syntax F (float), RD (raw data) or UN (Unicode). |
| **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 Parameters* for more information about the CTABLESIZE Data Object Broker parameter.

# Sample Definitions

## MAP_ONE Table Illustrated

The following illustrates a sample table definition for the MAP_ONE table. This table definition illustrates the external attribute defaults supplied when you enter the TIBCO Object Service Broker attributes for the fields.

```
COMMAND==>                        TABLE DEFINITION

     Table: MAP_ONE           Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference       '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   --------------- -   -
_ ADDRESS             B    4  0  A                 ' _
_ COUNT               B    4  0  C                 ' _
_ LOCATION          I  C   16  0  L                 ' _
                   | ----- EXTERNAL ----|---------- MetaStor -------------   -
    Field Name     Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
_ KEY               B       4   0       0  P   I   B     4  0
_ FIELD1            V       8   0       0          V     8  0
_ FIELD2            B       3   0       9          B     4  0
_ FIELD3            V      32   0      12          V    32  0
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## MAP_TWO Table Illustrated

The following illustrates a sample table definition for the MAP_TWO table. This table definition illustrates the external attribute defaults supplied when you enter the TIBCO Object Service Broker attributes for the fields.

```
COMMAND==>                        TABLE DEFINITION

     Table: MAP_TWO           Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference       '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   --------------- -   -
_ ADDRESS             B    4  0  A                 ' _
_                                                  ' _
_                                                  ' _
```

```
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  ------  -   -   -- ---- --  - ----------------
_ KEY              B       4   0       0  P   I   B      4  0
_ FIELD1           B       4   0       0          B      4  0
_ FIELD2           C       8   0       4          C      8  0
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

Chapter 9

# Manipulating Storage Data Using TIBCO Object Service Broker MAP Tables

This chapter describes how to manipulate storage data using TIBCO Object Service Broker MAP tables.

## Topics

# Accessing TIBCO Object Service Broker MAP Tables

## Retrieving Meaningful Data

MAP tables are unlike other TIBCO Object Service Broker tables because the system has no inherent knowledge of how many occurrences exist in the table. For this reason, it is your responsibility to ensure that GET and FORALL statements do not sweep through storage unconstrained. A GET or FORALL statement that attempts to access unregistered storage causes the DATAREFERENCE exception to be raised; however, it is your responsibility to ensure that registered storage contains meaningful data in a format suitable for the syntax of the fields in the MAP table definition.

### Recommendations

For FORALL statements, we strongly recommend using the count parameter to limit the number of occurrences in storage that TIBCO Object Service Broker attempts to process on a MAP table access request. Other approaches include specifying suitable key ranges and condition testing using the UNTIL clause on the FORALL statement.

## Accessing Storage Data

You can access storage data by using:

- The Table Browser or Table Editor
- Rules

## Using the Table Browser or Table Editor

You can browse or edit a MAP table in the same way you would browse any other TIBCO Object Service Broker table with the following exceptions:

- Do not use the ORDERED primary command with the Table Browser or Table Editor since ordering is not supported on MAP tables.
- Data in transaction storage cannot be browsed or edited because the Table Browser and Table Editor run in their own transaction.

## Using Rules

Accessing storage data using the rules language is similar to accessing data. Refer to Using Rules to Access Storage Data on page 122.

When using rules to access storage data:

- If you use the default parameter value, you can access at least 16 MAP tables per transaction; more, depending on the size of the MAP table definitions, because 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.
- The INSERT and DELETE statements cannot be used with MAP tables.

See Also *TIBCO Object Service Broker Programming in Rules* for information on writing rules and transactions.

# Using Rules to Access Storage Data

The following sections outline the differences encountered while using rules and also point out normal rules behavior that you must consider when building applications. The following rules statements are discussed:

- GET
- FORALL
- REPLACE

These statements are all used to move data between the table buffer and an area in main storage.

### Examples Used

It is assumed that the storage area to be accessed is at decimal location 1000. Although it is unlikely that the storage address would be known before runtime, this simplifies the examples. All example storage locations are in decimal format.

## GET Statement

A GET statement returns a single occurrence from main storage. If no occurrence matches the selection criteria specified, the GETFAIL exception is raised.

### Evaluation Process

TIBCO Object Service Broker uses the value of the address parameter as the address of the first row in main storage to be examined. This occurrence is assigned the key value of 1. If the occurrence satisfies the selection criteria, it is returned; otherwise, TIBCO Object Service Broker computes the address of the next occurrence in storage by adding the length of a table occurrence to the current address and increasing the key value by 1.

This process continues until either:

- A satisfactory occurrence is found.
- The value of the count parameter is exhausted.
- A selection on the primary key makes it impossible that a suitable occurrence is found beyond this point.

## Examples of GET Statements

The following table illustrates examples of GET statements. The table definitions discussed in the examples are illustrated in Chapter 8, Managing TIBCO Object Service Broker MAP Data Definitions, on page 101.

| | |
|---|---|
| `GET MAP_TWO(1000);` | Retrieves the first row of table MAP_TWO starting at address 1000. The system assigns the field **KEY** the value 1. |
| `GET MAP_TWO(1000)`<br>`WHERE KEY=4;` | Retrieves the fourth row of table MAP_TWO, starting at address 1036. |
| `GET MAP_TWO(1036);` | Retrieves the same data from address 1036 as the previous example; however, the row number returned in KEY is 1. From this it can be seen that row numbers are not persistently related to storage addresses but are relative to the base address in the address parameter. |
| `GET MAP_TWO(1000)`<br>`WHERE`<br>`FIELD2='ROBINSON';` | Returns the first row encountered where the value of **FIELD2** is ROBINSON. If this value is not found in storage, the GET statement sweeps through main storage without limit, until TIBCO Object Service Broker or the operating system interrupts it. Therefore, you should limit the search; for example, if it is known that the row required should be within the first 100 records in storage, use the following example. |
| `GET MAP_TWO(1000)`<br>`WHERE`<br>`FIELD2='ROBINSON'`<br>`& KEY <=100;` | Limits the search to the first 100 records. Avoid specifying a condition that appears to limit the search range but does not do so. Using a count parameter is the safest way of ensuring that the search always terminates predictably. |

## FORALL Statement

A FORALL statement is a looping construct that processes a set of occurrences. The body of the loop consists of the statements to be executed for each occurrence satisfying the selection criteria. FORALL statements can be nested, provided they refer to different table names.

A FORALL statement returns multiple occurrences from main storage. If no occurrences match the selection criteria specified, the FORALL ends without error.

A FORALL statement that attempts to reference unregistered storage causes the DATAREFERENCE exception to be raised. Occurrences are retrieved and examined as for a GET statement.

## Examples of FORALL Statements

The following table illustrates examples of FORALL statements. The table definitions discussed in the examples are illustrated in Sample Definitions on page 117.

| | |
|---|---|
| `FORALL MAP_TWO(1000)`<br>`WHERE KEY>=2 & KEY < 10:`<br>`...`<br>`END;` | Returns rows 2 through 9. |
| `FORALL MAP_ONE(1000,5)`<br>`WHERE FIELD3 LIKE 'EMPL*':`<br>`...`<br>`END;` | Returns any rows where **FIELD3** begins with the string EMPL. Only the first five rows in storage are searched. |
| `FORALL MAP_ONE(1000,5)`<br>`WHERE FIELD3 LIKE 'EMPL*' & KEY>3:`<br>`...`<br>`END;` | Considers only the first 5 rows to be eligible for searching for the EMPL* string; however, the key selection further restricts the search to those rows numbered higher than 3. In effect, only rows 4 and 5 are searched. |
| `FORALL MAP_TWO(1000) UNTIL FINISHED:`<br>`CALL PROCESS_SAMPLE_DATA;`<br>`END;` | Returns records starting at location 1000. If the PROCESS_SAMPLE_DATA rule raises the FINISHED exception, the FORALL ends. If the FINISHED exception is not raised, the FORALL sweeps through registered storage unconstrained. |

## REPLACE Statement

The REPLACE statement copies a single row from the table buffer to main storage. The address parameter is used as the logical address of row 1 of the table.

### Evaluation Process

The value of the key field of the table is used to identify the row to be replaced. TIBCO Object Service Broker computes the actual address by multiplying the key value minus 1 by the row length and adding the result to the address of logical row 1.

## Examples of REPLACE Statements

The following table illustrates examples of REPLACE statements. The table definitions discussed in the examples are illustrated in Sample Definitions on page 117.

| | |
|---|---|
| `MAP_TWO.KEY=1;`<br>`MAP_TWO.FIELD1=17;`<br>`MAP_TWO.FIELD2='SMITH';`<br>`REPLACE MAP_TWO(1000);` | Replaces the content of storage from location 1000 to location 1011 with the value of the fields **FIELD1** and **FIELD2** of MAP_TWO. |
| `MAP_TWO.KEY=3;`<br>`MAP_TWO.FIELD1=17;`<br>`MAP_TWO.FIELD2='SMITH';`<br>`REPLACE MAP_TWO(1000);` | Replaces the content of storage from location 1024 to location 1035 with the value of the fields **FIELD1** and **FIELD2** of MAP_TWO. |

See Also *TIBCO Object Service Broker Programming in Rules* for information on using table access statements.

# Using TIBCO Object Service Broker MAP Tables with COMMAREAS and Other External Data Areas

In some TIBCO Object Service Broker environments (CICS, TSO, z/OS batch), the concept of a communications area or COMMAREA is provided. This is a block of storage whose address is provided by the external environment and passed to the TIBCO Object Service Broker application program. Alternatively, the TIBCO Object Service Broker application program can obtain storage for use as a COMMAREA and pass a COMMAREA pointer to an external program. Multiple storage areas can be obtained for use as COMMAREAs; however, only one COMMAREA is active at any one time.

## @SESSION Table

The System Interpreted Table @SESSION is used to obtain and manipulate COMMAREA pointers (sometimes called handles). If a COMMAREA is provided by the calling external environment, the value of @SESSION.COMMHANDLE is its address and @SESSION.COMMLENGTH is its length.

## IMS Environment

In the IMS environment three input and three output segments are provided and can be accessed using the pointer in @SESSION.SEG*n*INHANDLE and @SESSION.SEG*n*OUTHANDLE where *n* is the segment number 0, 1, or 2.

## Call Level Interface Environment

In the Call Level Interface environment any number of input and output COMMAREAs can be passed in by the calling program and the list of pointers can be accessed using @SESSION.APIINHANDLE and @SESSION.APIOUTHANDLE.

See Also *TIBCO Object Service Broker for z/OS External Environments* for information on the formats of COMMAREAS in the Call Level Interface environment.

# Handling TIBCO Object Service Broker Requests

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

- Synchronization and recovery
- Error handling

## Synchronization and Recovery

COMMIT and ROLLBACK statements have no effect on MAP tables; data in storage is read or written directly by the GET, REPLACE, and FORALL statements.

## 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 through the MAP server are trapped under one of the following exceptions:

- ERROR
- ACCESSFAIL
- INTEGRITYFAIL
- RULEFAIL

## 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** | No occurrence satisfies the selection criteria |
| **REPLACEFAIL** | 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 an error is detected in the MAP table definition |
| **SECURITYFAIL** | Indicates that permission for the requested action on the object is denied |

## RULEFAIL

A RULEFAIL exception indicates that an error is provoked by incorrect rules language coding. The following exceptions are valid under a RULEFAIL exception:

| | |
|---|---|
| **CONVERSIONFAIL** | TIBCO Object Service Broker is unable to convert between the internal and external data |
| **OVERFLOW** | An address, count, or key value is out of range |
| **DATAREFERENCE** | An attempt is being made to access unregistered storage, that is, there is no row in @MAP to match the address referenced |

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

# Understanding Security with TIBCO Object Service Broker MAP Tables

Due to the nature of MAP tables, normal TIBCO Object Service Broker security controls are largely inapplicable.

### Behavior of Persistent Table Types

With ordinary persistent tables (for example, TDS), access to the table definition controls access to the data contained in the table, that is, the definition of a table uniquely identifies the data. Therefore, suitable setting of user and group access to the definitions controls who can read and write the data.

### Behavior of Non-Persistent Table Types

Ordinary non-persistent tables (EES, TEM and SES) have global definitions but local data. Even if multiple users have access to the table definition, the data is always local to the transaction or session and there is no possibility of unauthorized access.

## MAP Table Behavior

MAP tables are significantly different because the table definition plays no part in controlling access to the data. The definition specifies the layout of the data in storage and its mapping to fields but the location of the data in storage is uniquely specified by the value of the address parameter of the table.

MAP table data access is controlled indirectly using the system interpreted table @MAP. @MAP is used to allocate and register storage for use by MAP tables. Without this control, a user who is denied access to a MAP table definition could define a new MAP table and then at runtime supply the storage address of the desired data via the address parameter.

## Accessing Data at a Particular Address with a MAP Table

To access data at a particular address with a MAP table, an occurrence representing that address must exist in the @MAP table. If no such occurrence exists, the table access fails and the DATAREFERENCE exception is raised. The **ADDRESS** field of @MAP can be thought of as a reference field for the address

parameter of the MAP table. It is not necessary for the address of the occurrence in @MAP to exactly match the address parameter of the MAP table; access succeeds as long as the storage mapped by the MAP table does not extend outside the boundaries described by the occurrence in @MAP.

In addition, occurrences in @MAP contain an implicit permission for read-only access (GET and FORALL) or for read/write access (REPLACE). This permission is based on the parameter set of @MAP where the occurrence appears. All parameter sets except EXTERNALRO allow both read and write access. EXTERNALRO allows read access only.

See Also *TIBCO Object Service Broker Shareable Tools* for more information on @MAP tool.

# Chapter 10 Sample Application Using TIBCO Object Service Broker MAP Tables

This chapter provides a sample application using TIBCO Object Service Broker MAP tables.

## Topics

# Sample Application

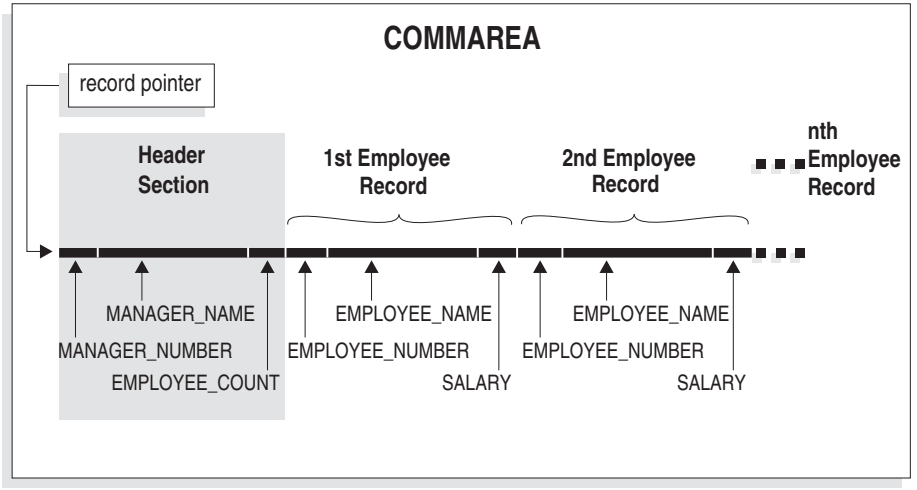
## What does this Application do?

This sample application receives a COMMAREA from its caller. The COMMAREA contains pointers to information on a manager and the manager's employees. The application calls an external CICS program passing it a record for each employee whose salary is $1000.00 or more.

This example runs in the CICS environment but the techniques used are applicable to other TIBCO Object Service Broker environments. Many irrelevant details are omitted that would be present in a real application.

## Pictorial Representation of the Sample Storage Areas

The following is a pictorial representation of the sample storage areas:




## The Application

The application consists of the MAIN rule (illustrated in MAIN Sample Rule on page 134) and the MAP tables illustrated inSample MAP Tables on page 137. In this example, all the data passed to the MAIN rule is in the COMMAREA and therefore can be accessed using MAP tables without special arrangements.

### Access to Data Outside of the COMMAREA

If the pointer provided in the COMMAREA contained the address of an area outside the COMMAREA, TIBCO Object Service Broker does not permit access to that area. To access such external data the area must first be registered using the EXTERNALRO or EXTERNALRW parameter of the @MAP table.

# MAIN Sample Rule

## MAIN Rule Illustrated

```
   RULE EDITOR ===>                                          SCROLL: P
MAIN;
_ LOCAL COMMAREA_PTR, COMMAREA_LEN, NEWCOMMAREA_PTR, EMPLOYEE_PTR;
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------+---------------
_ GET @SESSION(0);                                            | 1
_ COMMAREA_PTR = @SESSION.COMMHANDLE;                         | 2
_ COMMAREA_LEN = @SESSION.COMMLENGTH;                         | 3
_ @MAP.ADDRESS = 0;                                           | 4
_ @MAP.SIZE = 32;                                             | 5
_ INSERT @MAP('ENVIRONMENT');                                 | 6
_ NEWCOMMAREA_PTR = @MAP.ADDRESS;                             | 7
_ @SESSION.COMMHANDLE = @MAP.ADDRESS;                         | 8
_ @SESSION.COMMLENGTH = @MAP.SIZE;                            | 9
_ REPLACE @SESSION(0);                                        | A
_ GET COMM_HEADER(COMMAREA_PTR);                              | B
_ GET INPUT_HEADER(COMM_HEADER.POINTER);                      | C
_ EMPLOYEE_PTR = COMM_HEADER.POINTER + 40;                    | D
_ FORALL EMPLOYEE_RECORD(EMPLOYEE_PTR,                        | E
_      INPUT_HEADER.EMPLOYEE_COUNT) WHERE SALARY >= 1000 :    |
_    EMPLOYEE_SUMMARY.* = EMPLOYEE_RECORD.*;                  |
_    EMPLOYEE_SUMMARY.KEY = 1;                                |
_    REPLACE EMPLOYEE_SUMMARY(NEWCOMMAREA_PTR);               |
_    CALL CICS_ROUTINE;                                       |
_    END;                                                     |
_ @SESSION.COMMHANDLE = COMMAREA_PTR;                         | F
_ @SESSION.COMMLENGTH = COMMAREA_LEN;                         | G
_ REPLACE @SESSION(0);                                        | H
_ ------------------------------------------------------------------------------
```

## MAIN Rule Explained

The following table contains line by line comments on the MAIN rule:

| Line Number | Purpose of the Statement |
|---|---|
| 1 | Obtains data from the system interpreted table @SESSION. |
| 2 & 3 | Saves the address and length of the incoming COMMAREA in the local variables COMMAREA_PTR and COMMAREA_LEN. |

| Line Number | Purpose of the Statement |
|---|---|
| 4 | Sets @MAP.ADDRESS to zero as required to allocate ENVIRONMENT storage. |
| 5 | Sets @MAP.SIZE to the size of a single row of the MAP table EMPLOYEE_SUMMARY (shown in Sample MAP Tables on page 137). |
| 6 | Obtains ENVIRONMENT storage using the system interpreted table @MAP. |
| 7 | Saves the address of the newly allocated space in the local variable NEWCOMMAREA_PTR. |
| 8, 9, and A | Sets the address and length of the newly allocated space in the table @SESSION, thus making the new block of storage the active COMMAREA. |
| B | Uses the MAP table COMM_HEADER (shown in Sample MAP Tables on page 137) to map the area addressed by the original COMMAREA pointer. |
| C | Uses the MAP table INPUT_HEADER (shown in INPUT_HEADER Table on page 137) to map the area addressed by the field **COMM_HEADER.POINTER.** |
| D | Calculates the address of the start of the EMPLOYEE_RECORD data areas. |

| Line Number | Purpose of the Statement |
|---|---|
| E | The FORALL statement retrieves records starting at address EMPLOYEE_PTR whose SALARY value is greater than or equal to 1000. The record count in INPUT_HEADER.EMPLOYEE_COUNT is used to limit the number of occurrences in storage traversed by the FORALL. Fields in the summary table EMPLOYEE_SUMMARY (shown in EMPLOYEE_RECORD Table on page 138) are copied from the MAP table EMPLOYEE_RECORD (shown in EMPLOYEE_RECORD Table on page 138). The key field is set to 1 so that the first logical occurrence of the EMPLOYEE_SUMMARY table is the one written.<br><br>The REPLACE statement writes the EMPLOYEE_SUMMARY data to the storage area pointed to by NEWCOMMAREA_PTR. The external routine CICS_ROUTINE (which must be listed in the ROUTINES table) is called. Since the EMPLOYEE_SUMMARY record was written to the storage area pointed to by @SESSION.COMMAREA, the external CICS routine has access to it. |
| F, G, and H | Returns the COMMAREA pointer and length to their original values for the original caller to use (for example, for another call to this routine). |

See Also *TIBCO Object Service Broker for z/OS External Environments* for more information on the routines.

# Sample MAP Tables

## COMM_HEADER Table

The following illustrates the COMM_HEADER table, which is pictorially represented in the Sample Application on page 132:

```
COMMAND==>                          TABLE DEFINITION

      Table: COMM_HEADER      Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference        '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   ---------------- -   -
_ ADDRESS             B    4  0  A                  ' _
_ LOCATION          I  C   16  0  L                  ' _
_                                                   ' _
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
_ KEY               B       4   0       0  P   I   B     4  0
_ POINTER           B       4   0       0          B     4  0
_
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## INPUT_HEADER Table

The following example illustrates the INPUT_HEADER table:

```
COMMAND==>                          TABLE DEFINITION

      Table: INPUT_HEADER     Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference        '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   ---------------- -   -
_ ADDRESS             B    4  0  A                  ' _
_                                                   ' _
_                                                   ' _
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
_ KEY               B       4   0       0  P   I   B     4  0
```

```
_ MANAGER_NUMBER    B        4   0        0          B      4  0
_ MANAGER_NAME      C       32   0        4          C     32  0
_ EMPLOYEE_COUNT    B        4   0       36          B      4  0
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## EMPLOYEE_RECORD Table

The following example illustrates the EMPLOYEE_RECORD table:

```
COMMAND==>                        TABLE DEFINITION

      Table: EMPLOYEE_RECORD  Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference       '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   ---------------- -   -
_ ADDRESS             B    4  0  A                 ' _
_ COUNT               B    4  0  C                 ' _
_                                                  ' _
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
  ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
_ KEY               B       4   0       0  P   I   B     4  0
_ EMPLOYEE_NUMBER   B       4   0       0          B     4  0
_ NAME              C      32   0       4          C    32  0
_ SALARY            P       7   2      36      Q   P     7  2
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

## EMPLOYEE_SUMMARY Table

```
COMMAND==>                        TABLE DEFINITION

      Table: EMPLOYEE_SUMMARY Type: MAP   Unit: USR40              IDgen: Y

  Parameter Name Typ Syn Len Dc Cls Reference       '      Event Rule   Typ Acc
  -------------- -  -- --- --  - ---------------- '   ---------------- -   -
_ ADDRESS             B    4  0  A                 ' _
_ LOCATION          I  C   16  0  L                 ' _
```

```
_                                                        , _
                  | ----- EXTERNAL ----|----------- MetaStor --------------   -
     Field Name    Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Rqd Default
   ---------------- ---- ----  --  ------  -   -   -- ---- --  - ---------------
_  KEY              B       4   0       0  P   I   B     4  0
_  EMPLOYEE_NUMBER  P       3   0       0      I   P     3  0
_  NAME             C      32   0       0      S   C    32  0
_
_
_
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

# Appendix A Primary Command Syntax Reference

This appendix describes the syntax of the primary commands that allow operands and choices in the Table Browser and Table Editor.

## Topics

# Overview

## Purpose of this Appendix

This appendix describes the syntax of the primary commands that allow operands and choices in the Table Browser and Table Editor. Unless otherwise indicated, commands are available to both the Table Browser and Table Editor.

## Notation

Note the following points about notation:

- Square brackets indicate optional items. For example, square brackets show that an **ORDERED** clause in a **SELECT** command is optional:

  ```
  [ ORDERED <OrderList> ]
  ```

- Curly brackets indicate items that can be repeated zero or more times. For example, curly brackets show that in a selection expression a selection term can be followed by zero or more items composed of a logical operator and a selection term:

  ```
  <SelExpr> ::= { <logicalOp> <SelTerm> }
  ```

- Round brackets indicate explicit grouping, in which case the round brackets must appear literally. For example, a negated selection expression that contains logical operators must be enclosed in parentheses to show that the whole expression, not just its first term, is negated:

  ```
  [NOT] ( <SelExpr> )
  ```

# Primary Command Syntax

## SELECT Command

This section describes the syntax for the **SELECT** command:

SEL[ECT] [<Sel Predicate> {<OrderList>}]

| | |
|---|---|
| **<Sel Predicate>** | ::= <Sel NRelation>{<Logop><Sel NRelation>} |
| **<Sel NRelation>** | ::= [<Not>} <Sel Relation> |
| **<Sel Relation>** | ::= <Field Name> <Relop> <Expression><br><Sel Predicate>) |
| **<Expression>** | ::= {<Unary op>}<Expr Term><br>{<Addop><Expr Term>} |
| **<Expr Term>** | ::= <Expr Factor>{<Multop><br><Expr Factor>} |
| **<Expr Factor>** | ::= <Expr Primary>{**<br><Expr Primary>} |
| **<Expr Primary>** | ::= <Field Name><br><Constant><br>(<Expression>) |
| **<Logop>** | ::= <AND><br><OR> |
| **<Relop>** | ::= =<br>=<br>><br>>=<br><<br><=<br>LIKE |
| **<Addop>** | ::= +<br>-<br>\|\| |

| | |
|---|---|
| **<Multop>** | ::= *<br>/ |
| **<Unary Op>** | ::= -<br>+ |
| **<And>** | ::= AND<br>& |
| **<Or>** | ::= OR<br>\| |
| **<Not>** | ::= NOT<br>¬ |
| **<Constant>** | ::= <Quoted String><br><Number> |
| **<OrderList>** | ::= <OrderTerm>{<And><br><OrderTerm>} |
| **<OrderTerm>** | ::= ORD[ERED] [<Ordering>]<br><FieldName> |
| **<Ordering>** | ::= DESC[ENDING]<br>ASC[ENDING] |
| **<FieldName>** | ::= A field name of the table |
| **<Quoted String>** | ::= A sequence of characters enclosed in quotes |
| **<Number>** | ::= A sequence of numeric characters (or digits) |

## ORDERED Command

This section describes the syntax for the **ORDERED** command:

ORD[ERED] [<Ordering>] <FieldName> {<And> <OrderTerm>}

## FIND Command

This section describes the syntax for the **FIND** command:

F[IND] <FieldName> <Relop> <Constant> {<And> <FieldName> <Relop> <Contant>}

## CHANGE Command (Table Editor Only)

This section describes the syntax for the **CHANGE** command:

CH[ANGE] <FieldName> = <Constant>

<Constant> [<Change Option>]

| | |
|---|---|
| **<Change Option>** | ::= <PAGE><br><REST><br><ALL> |
| **<PAGE>** | ::= Current screen |
| **<REST>** | ::= Current screen and down |
| **<ALL>** | ::= From top to bottom |

## EDIT Command (Table Editor Only)

This section describes the syntax for the **EDIT** command:

ED[IT] <TableSpec> [WHERE <Sel Predicate>] [<OrderList>]

| | |
|---|---|
| **<TableSpec>** | ::= <Table Name> [(<Constant> {, <Constant>})] |

## XEDIT Command (Table Editor Only)

This section describes the syntax for the **XEDIT** command:

XE[DIT] <TableSpec> [WHERE <Sel Predicate>] [<OrderList>]

## BROWSE Command

This section describes the syntax for the **BROWSE** command:

BR[OWSE] <TableName> [WHERE <Sel Predicate>] [<OrderList>]

## XBROWSE Command (Table Editor Only)

This section describes the syntax for the **XBROWSE** command:

XB[ROWSE] <TableName> [WHERE <Sel Predicate>] [<OrderList>]

## EXCLUDE Command

This section describes the syntax for the **EXCLUDE** command:

EXC[LUDE] <FieldName> {<FieldName>}

## EXPAND Command

This section describes the syntax for the **EXPAND** command:

EXP[AND] <ObjectName> [<ObjectType>]

| | |
|---|---|
| **<ObjectName>** | ::= <field_name><br><report_name><br><screen_name><br><table_name><br><rule_name><br><builtin_name> |
| **[<ObjectType>]** | ::= FIELD<br>FLD<br>REPORT<br>RPT<br>SCREEN<br>SCR<br>TABLE<br>TBL<br>RULE |

# Appendix B Mapping Data Types

This appendix describes how to map data types for MAP Table Definitions.

## Topics

# Mapping Data Types for MAP Table Definitions

## Translation of External Data

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 | Alphabetic (uppercase). | 1 – 31742 | 0 | C | Xlen[a] |
| B [b, c] | Binary, signed. | 1 & Xdec=0 | 0 | B | 2 |
| | | 2 – 8 & 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 |

| 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) |
|---|---|---|---|---|---|
| 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 | Fixed length character string, with trailing blanks ignored. | 1 – 31742 | 0 | C | Xlen[a] |
| E [c, i] | Little-endian binary, signed. | 1 & Xdec=0<br>2 – 8 & Xdec=0<br>1 & Xdec>0<br>2 & Xdec>0<br>3 & Xdec>0<br>4 & Xdec>0<br>5 & Xdec>0<br>6 & Xdec>0<br>7 & Xdec>0<br>8 & Xdec>0 | 0<br>0<br>3<br>5<br>9<br>11<br>13<br>15<br>17<br>19 | B<br>B<br>P<br>P<br>P<br>P<br>P<br>P<br>P<br>P | 2<br>Same as Xlen<br>2<br>3<br>5<br>6<br>7<br>8<br>9<br>10 |
| F | Floating point (short, long, or extended). | 4<br>8<br>16 | 0 | F | 4<br>8<br>16 |
| G | Packed, neutral (X'0F') sign when positive. | 1 – 16 | Xlen * 2 – 1 | P | Xlen |
| H | Hexadecimal. | 1 – 31742 | 0 | RD | 4 + 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) |
|---|---|---|---|---|---|
| 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] |
| K [k, l] | Binary, unsigned. | 1 & Xdec=0 | 0 | B | 2 |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2 – 7 & 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 |
| | | 8 & Xdec=0 | 0 | P | 11 |
| L | Long packed, signed (up to 31 digits). | 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 [m] | 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 [m] | 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). | 1<br>2 & Xdec=0<br>2 & Xdec>0<br>3<br>4 & Xdec=0<br>4 & Xdec>0<br>5 – 7<br>8 | Xlen * 2 | P | Xlen + 1 |
| P [n] | Packed, signed (up to 31 digits). | 1 – 4<br>5 & Xdec=0<br>5 & Xdec>0<br>6 – 8 | Xlen * 2 – 1 | P | Xlen |
| Q | Quoted character string. | 3 – 31742 | 0 | V | Xlen-2[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) |
|---|---|---|---|---|---|
| R [i, l] | Little-endian binary, unsigned. | 1 & Xdec=0 | 0 | B | 2 |
| | | 1 & Xdec>0 | 3 | P | 2 |
| | | 2-7 & 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 |
| | | 8 & Xdec=0 | 0 | P | 11 |
| RD | Raw data. | 5 – 31742 | 0 | RD | Xlen[a] |
| T | Text numeric. | 2 – 17 | Xlen – 2 | P | Xlen/2 (round down) |
| U [n] | Packed, unsigned (up to 31 digits). | 1 – 16 | Xlen * 2 – 1 | P | 1 – 16 |
| 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] | Variable character. | 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) |
|---|---|---|---|---|---|
| W | Double- & single-byte character string. | 4 – 31742 | 0 | W | Xlen[a] |
| X[j] | Fixed length, mixed case character string. | 1 – 31742 | 0 | V | Xlen[a] |
| XCnn[o] | 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] |
| Z[j] | 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 the *TIBCO Object Service Broker for z/OS Installing and Operating* or *TIBCO Object Service Broker for Open Systems Installing and Operating*.

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.

# Index

## Symbols



## A



## B



## C

## D

## E

## F



## G

## H



## I



## J



## K



## L

## M

## N



## O

## P



## Q

## R



## S

## T

## U



## V



## W

## X



## Z