TIBCO Service Gateway™ for Files

SDK User's Guide

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 the SDK for the Service Gateway for Files for building applications to access Adabas, CA Datacom, and VSAM LDS data.

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 *OSB_HOME*	By default, all TIBCO products are installed into a folder referenced in the documentation as *TIBCO_HOME*. 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`. 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: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **`admin`**. • In large code samples, to indicate the parts of the sample that are of particular interest. • 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: • To indicate a document title. For example: See *TIBCO ActiveMatrix BusinessWorks Concepts*. • To introduce new terms For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal. • 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. 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. For example: `MyCommand [optional_parameter] required_parameter`
`|`	A logical `OR` that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: `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.
	
	For example, the following command requires two parameters, which can be either the pair `param1` and `param2`, or the pair `param3` and `param4`.
	
	`MyCommand {param1 param2} | {param3 param4}`
	
	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`:
	
	`MyCommand {param1 | param2} {param3 | param4}`
	
	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`.
	
	`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 Introduction

This chapter introduces the SDK for Service Gateway for Files.

Topics

Overview

The SDK for Service Gateway for Files is a software development kit that you can use to build applications to query and update Adabas®, CA Datacom®, and VSAM LDS data from within TIBCO Object Service Broker. It ensures that the data is presented in a manner consistent with TIBCO Object Service Broker behavior.

The SDK is included with the Service Gateway for Files, which is a separately licensed add-on for TIBCO Object Service Broker.

Prerequisites for Use

Use of the SDK for the purposes of accessing Adabas data assumes you are familiar with TIBCO Object Service Broker application development principles and administration, and Adabas from Software AG.

Use of the SDK for the purposes of accessing CA Datacom data assumes you are familiar with TIBCO Object Service Broker application development principles and administration, and CA Datacom from CA.

Use of the SDK for the purposes of accessing VSAM LDS data assumes you are familiar with TIBCO Object Service Broker application development principles and administration, the Service Gateway for Files, other VSAM accesses, and VSAM LDS.

See Also *TIBCO Object Service Broker Managing External Data* for information about the Service Gateway for Files and TIBCO Object Service Broker accesses to other forms of VSAM data.

Outline of This Manual

Setup, operations, and processing of the Adabas accesses are described in Chapters 2-5 of this manual.

Setup, operations, and processing of the CA Datacom accesses are described in Chapters 6-9 of this manual.

Setup and processing of the VSAM LDS accesses are described in Chapter 10 of this manual.

Installation of the SDK

Installation instructions for the Service Gateway for Files, which includes the SDK, are in the *TIBCO Object Service Broker Managing External Data* manual.

Support for the SDK

TIBCO will handle SDK associated service requests submitted via the normal TIBCO Support mechanism. No fixes will be provided between releases. See How to Contact TIBCO Support on page xx for information about contacting Support. TIBCO does not support the applications you build using the SDK.

Chapter 2 Configuring the Accesses to Adabas Data

This chapter provides information on configuring the accesses to Adabas data, starting and stopping the server, and how to connect the server to a Data Object Broker that resides on Windows or Solaris.

Topics

Access of Adabas Data

Access to Adabas data is only supported via the Service Gateway for Files SDK. Once the SDK has been installed, you need to define the appropriate tables to facilitate the manipulation of the Adabas data, then write the rules or use workbench tools to process the data. Your applications used to access the Adabas data are comprised of table definitions and rules.

Once your Adabas server environment is set up, as described in the following sections, you access Adabas data by:

1. Extracting the definition of the Adabas table to the TIBCO Object Service Broker environment (optional).
2. Defining a table of type ADA based on an extracted Adabas table definition or by manually entering known field definitions.
3. Using TIBCO Object Service Broker table access statements or workbench tools to access the Adabas data.

The access uses the Adabas ADALNK call interface to connect to the Adabas environment. This call interface is supplied with Adabas.

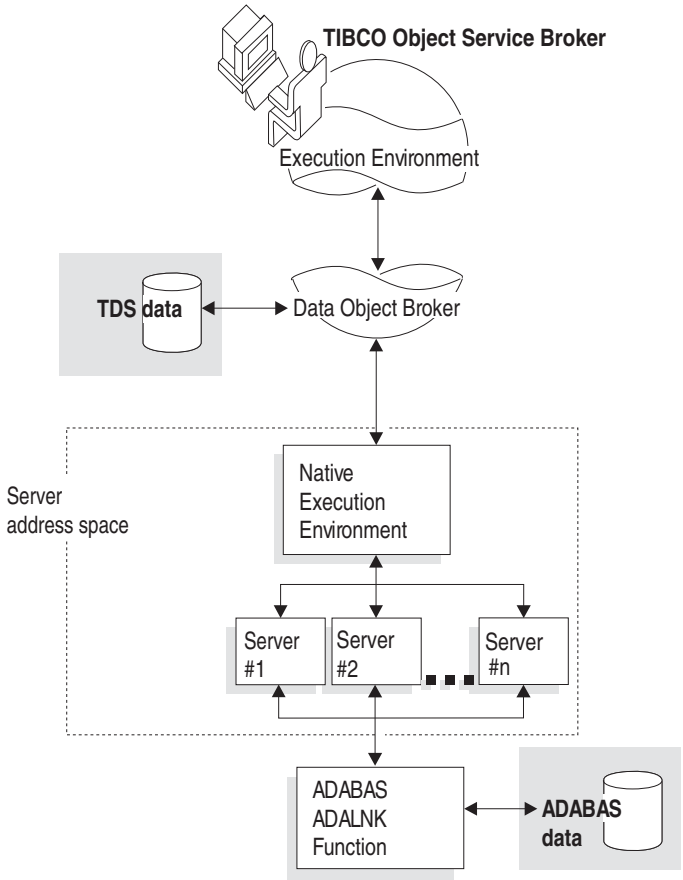
You use the components below to access Adabas data from TIBCO Object Service Broker. Once your data is defined, you build your applications to process the data using tools supplied with TIBCO Object Service Broker and its rules language.

Component	Function
Table Definer	Define TIBCO Object Service Broker ADA tables.
Server	Access Adabas data when TIBCO Object Service Broker data access is requested for an ADA table.
Extract utility	Decode Adabas file definitions and provide Adabas fields to assist in the definition of an ADA table (optional).
ADALNK call interface	Connect to the Adabas environment.

Supported Configurations

The Data Object Broker and the server can be configured to reside on the same, or different domains and operating systems (z/OS, Windows or Solaris). The server must be in the same domain as the Adabas database system. Once your environment is set up, you can access Adabas data while having equal access to TDS data, which is TIBCO Object Service Broker's native data type.

Refer to Connection to a Windows or Solaris Data Object Broker on page 26 for information about configuring on different operating systems.



Preparations for Installation

This section discusses the preliminary steps for installation.

Installation of the TIBCO Object Service Broker Base Component

You must install the TIBCO Object Service Broker base component before configuring the server. The base component can reside on z/OS, Windows or Solaris. Installation instructions for all platforms are located in *TIBCO Object Service Broker Installation and Operations*.

Installation of the Server

Installation instructions for the Service Gateway for Files, which includes the SDK, are located in *TIBCO Object Service Broker Managing External Data*. The Adabas server is installed as part of the SDK.

Implementation of TIBCO Object Service Broker Security

You can implement security for the server using standard TIBCO Object Service Broker security:

- To restrict the ability to define ADA tables from within TIBCO Object Service Broker, restrict read and update access to the @ADAFIELDS table.
- To restrict access to TIBCO Object Service Broker ADA tables after defining them, proceed with security as for any other TIBCO Object Service Broker table.

Adabas Security Considerations

Because Adabas does not provide an interface in ADALNK to pass security information to Adabas, the server threads get their security permissions and user identification from the server address space. It is possible, however, to pass some information to Adabas through the ADALNK user exits. For information and assistance, contact TIBCO Support.

Fail Safe Processing Considerations

To guarantee consistency when updating both TIBCO Object Service Broker TDS and Adabas data from a single instance of the server in a single transaction, you must use Fail Safe level-1 processing.

For more information, refer to Implementing Fail Safe Processing on page 40.

Communications Requirements

If all components reside in the same domain and in authorized libraries, Cross Memory Services is used for communications. In all other cases, TCP/IP is used for communications.

See Also

TIBCO Object Service Broker *for z/OS Installation and Operations* for detailed information about TIBCO Object Service Broker communications.

TIBCO Object Service Broker *Messages with Identifiers* for information on messages produced by the server.

TIBCO Object Service Broker *Security* for information on security for tables.

Startup Process

The TIBCO Object Service Broker Execution Environment receives several parameters at startup. The parameters are described in Startup Parameters on page 11. The SERVERS parameter instructs this Execution Environment to attach a specified number of instances of the server used to access Adabas, thus accommodating multiple servers in a single address space.

Each server connects to Adabas with a unique Adabas session identifier. The Adabas session is disconnected from Adabas only when the thread is disconnected or the address space is terminated.

At Transaction Start

At the beginning of a TIBCO Object Service Broker transaction, the first request for data from Adabas results in an Adabas **OP** (Open) command being sent to Adabas. A TIBCO Object Service Broker transaction running in browse mode causes the file to be opened with the ACCess argument. Transactions running in update mode are opened with the UPDate argument.

At Transaction End

When the transaction ends, the Adabas session terminates. If the transaction completes successfully, only an Adabas **CL** (Close) command is issued. This close command implies an Adabas **ET** (End transaction), which is equivalent to a TIBCO Object Service Broker commit point. However, if the transaction fails, an Adabas **BT** (Backout Transaction) is sent before the **CL** (Close) command.

See Also *TIBCO Object Service Broker External Environments for z/OS* about Execution Environments.

Startup Parameters

This section describes the startup parameters.

Required and Optional Parameters

The table below lists all server parameters with their default values (if any). Those you must set at installation time are shown as Required=Y, and those you can add or modify at a later time as Required=N. All these parameters are described in Supplying the Startup Parameters on page 35.

Parameter	Default	Required
IDPREFIX	ADA	Y
MDL	OSB9999	Y
SERVERID	none	Y
SERVERTYPE	ADA	Y
TDS	none	Y
FSLEVEL	0	N
FSTABLENAME	@ADAFSTRXDB	N
SERVERS	1	N

Specify the server startup parameters on the EXEC statement in the startup JCL, in a data set, or both.

If you specify parameters in both the EXEC statement and a data set, EXEC statement parameters override data set parameters.

Startup Parameters in the EXEC Statement

Include parameters in any order, one per line, ending with a blank or a comma (up to 100 bytes). The example below shows parameters specified on the EXEC statement in the startup JCL.

```
//ADAGTW EXEC PGM=S6BDR000,
//         REGION=4096K,
//         PARM=('TDS=$TDS$',
//         'SERVERS=6',
//         'MDL=$MDL$',
//         'SERVERTYPE=ADA',
//         'SERVERID=HRNADA',
//         'FSLEVEL=1',
//         'FSTABLENAME=@ADAFSTRXDB',
//         'IDPREFIX=ADA')
```

Startup Parameters in a Data Set

Include parameters in any order, one per record, beginning in column one, and ending with a blank or a comma. An asterisk (*) in column one indicates a comment record. The data set must be defined as follows:

- DDname HRNIN
- Allocated FB LRECL=80

The example below shows parameters specified in a data set or inline JCL.

```
//HRNIN DD *
SERVERS=3
SERVERTYPE=ADA
SERVERID=HRNADA
FSLEVEL=0
FSTABLENAME=@ADAFSTRXDB
IDPREFIX=HRNDC
```

Configuration Parameters

This section describes the configuration parameters.

@CONFIGURESERVER Tool

Use the @CONFIGURESERVER tool to set and modify the configuration parameters. Execute the following from the workbench:

```
EX @CONFIGURESERVER(ADA)
```

A screen similar to the one shown below appears.:

```
Command ==>                                                    Scroll P

NUMBER       SERVERTYPE SERVERID
------------ ---------- --------
_          1 ADA        ANDY
_          2 ADA        EXTRACT
_          3 ADA        IMZADA
_          4 ADA        SUB




 A-ADD  D-DELETE  S-SELECT
 PFKEYS: 12=EXIT 13=PRINT 3=END 5=FIND NEXT 9=RECALL
```

Configuration Parameters for a New Server ID

To set the configuration parameters for a new server ID, complete the following steps:

1. Type **A** beside an existing entry on the screen and use Enter.

   A screen similar to the one shown below appears, prompting you to enter a value into the **SERVERTYPE** and **SERVERID** fields.

```
To complete this command:
      NUMBER    SERVERTYPE SERVERID
    ----------- ---------- --------
  A           1 ADA        ADA001
 Enter parameter(s):


  SERVERTYPE          ===> ADA

  SERVERID            ===> EXTRACT









 PFKEYS: ENTER=PROCESS 3=PROCESS 12=CANCEL
```

2. Enter the SERVERTYPE ADA.
3. Type the server ID for which you want to set the configuration parameters and press Enter. A default configuration settings screen appears similar to the one shown below.

```
                External Server Configuration Utility
COMMAND ==>
Server Type: ADA   Server ID: EXTRACT
                                                                Recommended/
Name              Value                                        Allowed Values
----------------- -------------------------------------------- ------------------
DEBUGLEVEL        0                                            '0...3'
DUMP              N                                            Y, N
DUMPLIMIT         512                                          '0...2147483647'
EXTERNALROUTINE   ADALNK
KEEPLOG           N                                            Y, N
LOGMEDIA          TBL                                          TBL, SCR, PRT
PROGRAMLIBRARY    EXL                                          EXL, SMG
RUNAWAY           0                                            '0...2147483647'
SERVERSTATISTICS  N                                            Y, N
TRACE             N                                            Y, N


 FCNKEYS: ENTER=VALIDATE 3=SAVE & EXIT 12=EXIT
```

4. Modify any default settings for the parameters in the Value field.

   Valid values are shown in the **`Recommended/Allowed Values`** field. Refer to Available Configuration Parameters on page 15 for parameter descriptions.

5. Press PF3 to save the settings and return to the workbench.

If the server you are modifying is active, it must be recycled for the new values to be applied. Refer to *TIBCO Object Service Broker for z/OS Installing and Operating* for instructions on recycling.

Customization of the Load Point and the Adabas Interface Module

Use the @CONFIGURESERVER tool for customization according to the server ID. To modify the load point and the name of the Adabas interface module, add an entry for a server type of ADA and a server ID that identifies the server to be customized, and then set the appropriate values for the module.

For example, you can customize the Adabas extract utility by using @CONFIGURESERVER to add an entry for a server type of ADA and a server ID of EXTRACT, and then set the values appropriate for the interface module to be used to issue Adabas **`LF`** commands.

Refer to Configuration Parameters on page 13 for more information on using the @CONFIGURESERVER tool.

Available Configuration Parameters

The following table describes the configuration parameters that are available.

DEBUGLEVEL	Used during problem analysis to determine which portion of code is in error. Valid values are listed below. The default is **`0`**. **`0`** – Used during normal processing. **`1`** – Produces TRACE and/or DUMP information if they are selected. DUMP information shows the Adabas control blocks after the call to ADALNK. **`2`** – Produces TRACE and/or DUMP information if they are selected. DUMP information shows the Adabas control blocks before and after the call to ADALNK. **`3`** – Produces TRACE and/or DUMP information if they are selected. DUMP information shows the Adabas control blocks before and after the call to ADALNK; ESTAE and ADALNK are not called.

DUMP	Specifies whether Adabas messages and control blocks should be logged (in the location specified in the LOGMEDIA configuration parameter). Valid values are listed below. The default is **N**. **Y** – Both TIBCO Object Service Broker and Adabas messages and control blocks are logged. **N** – No logging is in effect.
DUMPLIMIT	Specifies the amount (in bytes) of TIBCO Object Service Broker and Adabas messages and control blocks that can be logged. The default is 512 bytes. Valid values are between 0 and 2147483647 bytes.
EXTERNALROUTINE	Specifies the name of the load module that provides the ADALNK functionality. The default is ADALNK.
KEEPLOG	Specifies whether to keep the previous server login startup. This applies only if the LOGMEDIA configuration parameter is set to TBL. Valid values are listed below. The default is **N**. **Y** – The log that matches the current server type, server ID, and server user ID is deleted on startup. **N** – The log that matches the current server type, server ID, and server user ID is kept on startup. Subsequent logs are appended.

LOGMEDIA	Specifies where to store DUMP, TRACE, and error message information. Valid values are listed below. The default is **TBL**. **PRT** – DUMP, TRACE, and error message information is sent to the default print file for the TIBCO Object Service Broker user ID. Set the desired print destination with the UP user profile option on the workbench. **SCR** – DUMP, TRACE, and error message information appears in the *session.log* file. This can be viewed only after the server is shut down. **TBL** – DUMP, TRACE, and error message information is inserted into the @SERVERLOG TDS table, which is parameterized by the servertype, server ID, and server user ID. You can browse this table while the server is running. • The server type is always ADA for Adabas data, and is specified by the SERVERTYPE startup parameter. • The server ID is specified by the SERVERID startup parameter. • The server user ID is the thread ID used by the transaction specified by the IDPREFIX startup parameter. It is generated and consists of the IDPREFIX concatenated with four characters derived from the server number. The Data Object Broker log displays the servertype (ADA), server ID (ADANT), and server user ID (ADAB0001) in the S6BUA023I message as follows: 06/01/08 15:07:55 3402 S6BUA023I ADA server ADANT(ADAB0001) logged on from'@SOY5C01'
PROGRAMLIBRARY	Specifies where the routine named by EXTERNALROUTINE is loaded from. Valid values are listed below. The default is **EXL**. **EXL** – DD statement HRNEXTR **SMG** – Standard z/OS load library search path conventions STEPLIB, JOBLIB, LINKLIB, and LPA

RUNAWAY	Specifies the maximum number of calls to Adabas that can be issued to satisfy one TIBCO Object Service Broker request. Valid values are between 0 and 2147483647 bytes. The default is 0. Use this feature in development to test access statements in the application code. When the limit is exceeded, the request is terminated with a SERVERERROR exception passing back an appropriate error message, and a message is written to the @SERVERLOG table. A value of 0 disables this parameter.
SERVERSTATISTICS	Controls how the server transaction is managed. Valid values are listed below. The default is **N**. **Y** – Transactions are recycled at the end of the TIBCO Object Service Broker transaction. **N** – An endless TIBCO Object Service Broker transaction runs. For best performance, do not change the default of N, since this transaction is independent of the client transaction.
TRACE	Specifies whether TIBCO Object Service Broker requests and Adabas statements should be logged (in the location specified in the LOGMEDIA configuration parameter). Valid values are listed below. The default is **N**. **Y** – Both TIBCO Object Service Broker requests and Adabas statements are logged. **N** – No logging is in effect.

Startup Prerequisites

Before you can start the server, you must do either of the following:

- If the Data Object Broker is on z/OS and Dynamic Resource Creation is not permitted (Data Object Broker Parameter `DYNAMICRESOURCE = N`), the Gateway must be identified to the Data Object Broker in a permanent resource. To do this, define Gateway resources to the Data Object Broker's resource management repository file.

  If the Data Object Broker is on z/OS and Dynamic Resource Creation is permitted (Data Object Broker Parameter `DYNAMICRESOURCE = Y`) and there is no matching permanent resource a dynamic resource entry matching, the Gateways requirements will be created when the Gateway connects to the Data Object Broker. If there is a permanent entry for the Gateway it must match the requirements for the Gateway. It is recommended that all permanent entries for File, ADA and DAT Gateways are deleted from the repository when dynamic resource creation is permitted.

- If the Data Object Broker is on Windows or Solaris, set up National Language Support, if necessary. Refer to *TIBCO Object Service Broker National Language Support* for setup and configuration information.

Default Resource Settings (z/OS only)

The table below lists the base defaults you must specify in the Resource Detail (PF2) and the Resource Schedule (PF10) screens through the Resource Management facility available from the Administration menu. Resource Management is option 3 on the Administration menu. This option brings you to the Resource Type list, on which you must supply a TYPE of ADA and a GROUP corresponding to the server ID of your Adabas server.

Resource Details				Resource Schedule
Intermediate Rollbk	Early Release	Last User Reuse	Commit Level	Online Only
N	Y	N	0	N

If the FSLEVEL startup parameter equals 1, the COMMIT LEVEL value on the Resource Detail screen must also be set to 1.

Refer to Supplying the Startup Parameters on page 35, for the description of the FSLEVEL parameter.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for more information on defining and managing TIBCO Object Service Broker resources using the Resource Management option, and for information about the Administration menu.

Customization of the Startup Batch JCL

Before starting the server, use the OSEMOD macro to customize the JCL for your installation. Sample batch JCL is shipped as member S6BSJCL in the JCLSAMP data set. Customize it as required.

By default, TIBCO Object Service Broker loads the Adabas interface module in the HRNEXTR DD statement. This can be changed to load from the standard load library search path STEPLIB, JOBLIB, LINKLIB, and LPA (Link Pack Area) using the @CONFIGURESERVER tool. If the Adabas load library is part of the STEPLIB, both the Adabas and TIBCO Object Service Broker load libraries should be authorized.

Refer to Communications Requirements on page 9 for more information about the OSEMOD macro. Refer to Configuration Parameters on page 13 for information about @CONFIGURESERVER.

Startup of the Server

To start the server, bring up the Native Execution Environment using the startup parameters described in Supplying the Startup Parameters on page 35.

When you start the server, you are connected to both Adabas and TIBCO Object Service Broker. When the Execution Environment requests an access to Adabas data, a thread to Adabas is established.

Because a separate Execution Environment is not required for each server, you can combine the parameters for the server with other parameters.

Using the MODIFY Operator Command

When you have an instance running, use the **MODIFY** operator command from the z/OS operator console to dynamically:

- Increase the number of server tasks.
- Set the maximum number of server tasks.

Increasing the Number of Server Tasks

If the number of server tasks is insufficient to process transaction requests, unsatisfied requests raise a SERVERBUSY exception and are queued until a free server is available. You can dynamically increase the number of tasks without restarting the Execution Environment by using a **MODIFY** operator command of the form:

MODIFY *ee_jobname*,**STARTNUMSERVER**=*nn*,**TYPE**=ADA

where

ee_jobname	Name of the batch job under which the Execution Environment runs.
nn	The number of new server tasks to start. This number can be from 1 to a value less than or equal to the value set in the Maximum Connection Count field in your Network Configuration.
ADA	The type of server to start.

Setting the Maximum Number of Tasks

Use the following **MODIFY** operator command to dynamically set the maximum number of tasks available in a particular Execution Environment:

MODIFY *ee_jobname*,**SETNUMSERVER=***nn*,**TYPE=ADA**

where

ee_jobname	Name of the batch job under which the Execution Environment runs.
nn	The maximum number of tasks available for a particular Execution Environment. This number can be from 1 to a value less than or equal to the value set in the Maximum Connection Count field in your Network Configuration.
ADA	The type of server tasks to be made available.

Shutdown of the Server

This section describes how to shut down the server.

Shutdown Order

Do a shutdown in the following order:

1. Shut down the server.
2. Shut down the Native Execution Environment.

   For more information on shutting down the Native Execution Environment, refer to Shutdown of the Native Execution Environment on page 25.

Shutdown Methods

Shut down the server in one of two ways:

- Use the **MODIFY** operator command from the z/OS operator console to shut down a group of instances of the server.
- Use the RESOURCE MANAGEMENT option from the Administration menu.

Using the MODIFY Operator Command

The format of the **MODIFY** command is:

`MODIFY` *dob_jobname*`,STOPSERVER=`*serveruserid*

where

dob_jobname	The name of the batch job under which the Data Object Broker runs.
serveruserid	Unique name of the server. This name is created by appending a three-digit number to the IDPREFIX parameter specified in the server startup JCL. The default is ADA. To see the unique name assigned to existing instances of the server, select the RESOURCE MANAGEMENT option from the Administration menu.

Use one the **MODIFY** operator command to do the following:

- Shut down all instances as follows:

MODIFY *dob_jobname*,**STOPSERVER=ALLADA**

- Shut down all instances of the server with a common IDPREFIX as follows:

  MODIFY *dob_jobname*,**STOPSERVER=***idprefix*

- Shut down all instances with a common server ID as follows:

  MODIFY *dob_jobname*,**STOPSERVER=SRVID***serverid*

- Dynamically shut down one or more of the tasks without shutting down the Execution Environment.

  MODIFY *ee_jobname*,**STOPNUMSERVER=***nn*,**TYPE=ADA**

where

dob_jobname	The name of the batch job under which the Data Object Broker runs.
ee_jobname	The name of the batch job under which the Execution Environment runs.
idprefix	The value of the IDPREFIX startup parameter. Used to construct a unique name for each server. Must be a valid TIBCO Object Service Broker user ID.
serverid	The value of the SERVERID startup parameter. Identifies a pool of servers with common characteristics.
nn	The number of new tasks to stop. This number can be from 1 to a value less than or equal to the value set in the Maximum Connection Count field in your Network Configuration. For more, refer to TIBCO Object Service Broker *for z/OS Installing and Operating*.
ADA	The type of server to stop.

Using the RESOURCE MANAGEMENT Option

Use the RESOURCE MANAGEMENT option from the Administration menu to shut down one or more servers.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for more information on defining and managing TIBCO Object Service Broker resources.

Shutdown of the Native Execution Environment

To shut down a Native Execution Environment, ensure that all users are logged out and then issue one of the following commands from the z/OS operator console:

- **P** *ee_jobname*
- **MODIFY** *ee_jobname*,**SHUTDOWN**

  If you do not want to stop all the threads or have all users log out, use the following command:

- **MODIFY** *ee_jobname*,**SHUTIMMED**

Use these values as follows:

P	z/OS operator command (Stop).
MODIFY	z/OS operator command (can be abbreviated to F).
ee_jobname	The name of the batch job or started task used to start the Native Execution Environment.

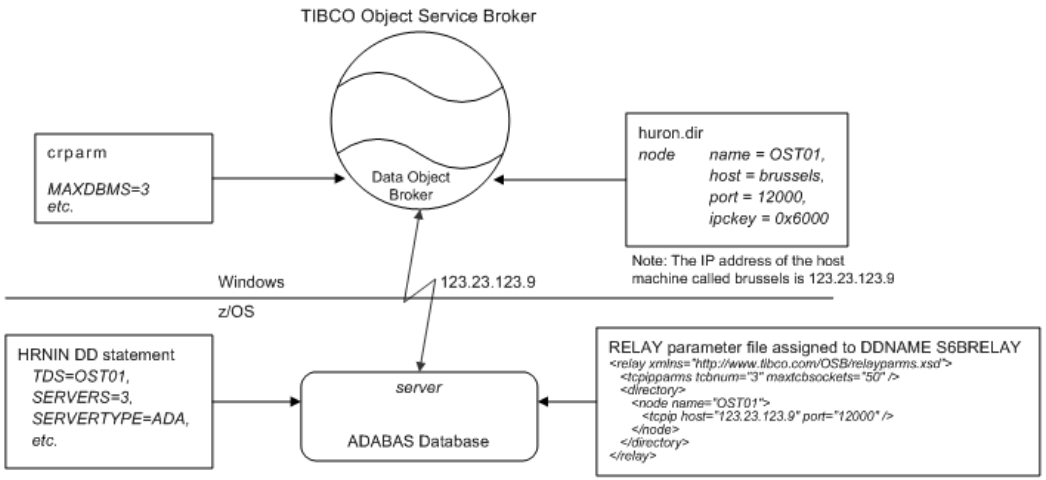
Connection to a Windows or Solaris Data Object Broker

You can configure the Data Object Broker and the server to reside on different domains and operating systems (z/OS, Windows, or Solaris). The server must be in the same domain as the Adabas database system.

The following configuration steps are required to access a Data Object Broker from a different operating environment than your server:

- Configure the TCP/IP connection on the z/OS system where your server and Adabas database reside.
- Configure the TCP/IP connection on the machine where your TIBCO Object Service Broker resides.
- Specify the number of instances of the server that can connect to the Data Object Broker.
- Specify the appropriate server parameters.

The following diagram shows a sample configuration.



Configuring the TCP/IP Connection on z/OS

Prepare the TIBCO Object Service Broker relay file – RELAYCFG member in the CNTL data set. This file associates the TIBCO Object Service Broker communications identifier with the TCP/IP application addressing information.

Following is a sample relay file assigned to DDNAME S6BRELAY:

```
<relay xmlns="http://www.tibco.com/OSB/relayparms.xsd">
    <tcpipparms tcbnum="3" maxtcbsockets="50" />
    <directory>
     <node name="OST01">
      <tcpip host="123.23.123.9" port="12000" />
     </node>
    </directory>
</relay>
```

The element and attribute names in the relay file are case sensitive.

See Also

TIBCO Object Service Broker for z/OS Installing and Operating for detailed information about preparing the TIBCO Object Service Broker relay file.

TIBCO Object Service Broker Parameters for details about the parameters and how to specify them.

Configuring the TIBCO Object Service Broker TCP/IP Environment

Add the following parameters for the TCP/IP connection to the Data Object Broker directory file, *huron.dir*:

name	This must be the same value as the node name set in the relay file described in Configuring the TCP/IP Connection on z/OS on page 26.
host	The name of the host machine where the TIBCO Object Service Broker monitor process listens for connections.
port	The number of the TIBCO Object Service Broker monitor socket port.
ipckey	The value of the IPC key.

Specifying the Number of Instances Connecting to the Data Object Broker

Specify the following value for the MAXDBMS parameter in the *crparm* file for your Data Object Broker.

MAXDBMS	This must be equal to or greater than the value specified in the SERVERS=parameter HRNIN DD statement for startup.

Specifying the Server Parameter

Specify the following value for the TDS=parameter HRNIN DD statement for startup. Refer to Startup Parameters on page 11 for more details

Chapter 3 Operational Requirements for Adabas Access

This chapter details the operational requirements for accessing the Adabas data from TIBCO Object Service Broker.

Topics

Extracting Adabas Table Information

You must define a table of type ADA from within TIBCO Object Service Broker to access the Adabas data. To assist in the definition of ADA tables, Adabas file definitions can be extracted so that their table information is available for the definition process. The extracted Adabas file definition information is stored in a TIBCO Object Service Broker TDS table. Because the extracted data is static, you must re-extract the table information whenever changes are made to new or existing Adabas table definitions.

Prerequisites

Before extracting Adabas file definition information:

- The Adabas system must be running.
- The Execution Environment performing the extract must have the Adabas interface program ADALNK available. The load name and load DD can be customized with the @CONFIGURESERVER tool.

  Refer to Configuration Parameters on page 13 for information about using @CONFIGURESERVER.

Extraction of Table Information

To extract the table information, complete the following steps:

1. Execute the @ADAEXTRACT rule using the **EX** execute rule option on the workbench.

The Adabas Extract Utilities Screen appears (shown here).

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>

Select Extract Functionality

   Extract one definition                                : _

   List all extracted definitions                       : _

   List all possible definitions                        : _

   Extract all possible definitions                     : _

   Extract all non-extracted definitions                : _




PFKEYS: 2=LOGS 3=EXIT 12=EXIT
```

2. On the displayed screen, position your cursor on one of the following menu options and press Enter:

Menu Option	Description
Extract one definition	You can use this to specify the Database ID and the File Number of a file to be extracted in isolation.
List all extracted definitions	Displays a list of all Adabas files whose definitions were extracted.
List all possible definitions	Displays a list of all extracted definitions including Adabas files not yet extracted.
Extract all possible definitions	Extracts definitions from all the Database IDs and File Numbers defined to Adabas. The process also deletes existing definitions before running the new extract.
Extract all non-extracted definitions	Performs the same processing as above except for deleting or re-extracting previously extracted file definitions.

Extracting or listing multiple definitions could take a significant time to complete, depending on the number of Adabas databases and files in your configuration. If you select List all extracted definitions, an Extract Screen listing similar to the one shown below appears:

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>


     Database  File   Extract
        ID      No     Date      Description
       ---     ---  2007-01-19  employees
  _      1       2  2007-01-19  vehicles
  _      1       3  2007-01-19  miscellaneous
  _      1       4  2007-01-19  agent info
  _      1      21  2007-01-19
  _      1      22  2007-01-19
  _      1      23  2007-01-19
  _      1      24  2007-01-19
  _      1      25  2007-01-19
  _      1      26  2007-01-19
  _      1      27  2007-01-19
  _      1      66  2007-01-19
  _      1      71  2007-01-19
  _      1      99  2007-01-19

S=Edit Description  D=Delete  X=Extract
PFKEYS: 3=EXIT 12=CANCEL
```

From this screen, the following actions are available:

S	Presents a screen for modifying the description.
D	Deletes the extracted data from the TIBCO Object Service Broker definition database.
X	Extracts or re-extracts the selected definition.

3. Select an Adabas file definition to delete, extract, or edit the description.

When you issue the **S** line command to edit the description, a screen similar to the one shown below appears, enabling you to edit extracted Adabas file descriptions.

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>


  Database Id:  1         File Id:  2

    Description: vehicles









  PFKEYS: 3=EXTRACT & SAVE 12=CANCEL
```

Binding TIBCO Object Service Broker ADA Table Definitions

You can bind the TIBCO Object Service Broker ADA table definition but not its data. ADA tables for which you requested binding are bound to both the Execution Environment and the server used to access Adabas, when they are accessed from a TIBCO Object Service Broker rule.

Before you request binding on an ADA table definition, ensure that the table definition does not require changes, since it is bound to the server for the life of the server.

If you want to change a definition, you can dynamically request a re-bind for the Execution Environment and you must re-cycle the server for the change to be recognized.

See Also *TIBCO Object Service Broker Application Administration* for more information about binding tables.

Supplying the Startup Parameters

You can specify the server parameters on the EXEC statement in the startup JCL, in a data set or both. For more information on how to specify the server parameters, refer to Startup Parameters on page 11.

The SERVERID parameter can be overridden at runtime. Refer to Dynamically Changing the Parameters on page 38 for more information.

Available Parameters

The available parameters are as follows:

FSLEVEL (FSL)	Use to specify the level of Fail Safe processing. The default value is zero. Valid values: **1** – Activate Fail Safe Level 1. The server informs the Data Object Broker that it can support Fail Safe level-1 processing. If the server is to attach to a z/OS Data Object Broker, the Data Object Broker's connection attribute setting "commit level" must be set to 1. If not, the connection is rejected. Refer to Implementing Fail Safe Processing on page 40 for more information. You must specify the TRXDB, FSTABLENAME, RECOVERYID, and RECOVERYPASSWORD parameters. **0** – De-activate Fail Safe processing. If attaching to a z/OS Data Object Broker, the Data Object Broker's connection attribute setting "commit level" must be set to 0. If not, the connection is rejected. Refer to Implementing Fail Safe Processing on page 40 for more information.
FSTABLENAME	The ADA table name of the Adabas transaction database. Required only if FSLEVEL=1. A sample ADA table definition is supplied with the name @ADAFSTRXDB (default). For more information, refer to Implementing Fail Safe Processing on page 40.
IDPREFIX	Prefix used to construct a unique name for each instance of the server. Four decimal digits are appended to this prefix and the resulting value is used to log in to TIBCO Object Service Broker. It uniquely identifies each instance to TIBCO Object Service Broker. Each instance must have a unique IDPREFIX. This parameter must be a valid TIBCO Object Service Broker user ID.

MDL	The pattern for selecting the VTAM ACB name that is used for communications. If not specified, the TDS parameter is used as the pattern. If you do not specify an MDL parameter, ensure that the TDS parameter is always a valid VTAM ACB model.
SERVERID	Identifies a pool of instances of the server with common characteristics. SERVERID can have up to eight characters and you must specify one when you are defining an ADA table. The SERVERID parameter can be overridden at runtime. Refer to Dynamically Changing the Parameters on page 38 for more information.
SERVERS	The number of instances of the server that the initializer program should attach to the server address space at startup. This number can be from 1 to a value less than or equal to the value set in the **Maximum Connection Count** field in your network configuration. The default value is 1. For more information, refer to *TIBCO Object Service Broker for z/OS Installing and Operating*.
SERVERTYPE	The value ADA must be specified.
TDS	Supplies the Communications Identifier of the Data Object Broker with which the server communicates.

If you have multiple server initializer programs with the same SERVERID value, ensure that the FSLEVEL and FSTABLENAME Fail Safe server parameters are assigned the same values for each server address space.

Estimation of the CTABLESIZE Parameter

The number of fields in an ADA table definition is dependent upon the Data Object Broker CTABLESIZE parameter. To estimate the number of bytes required to support a specified number of fields, run the ESTIMATEBLDFN tool:

```
EX ESTIMATETBLDFN(num_fields)
```

You must supply a value for the argument *num_fields*, which is the maximum number of fields accessed by any ADA table in your system. This tool returns an estimate of the maximum CTABLESIZE that is required to support the number of fields you specify.

Following is a sample result of executing ESTIMATETBLDFN for 50 fields:

```
------------------------------ INFORMATION LOG ---------------------------------:
COMMAND ===>                                                    SCROLL ===> P
DATE: Mar 28,2007      REPORT ON ESTIMATE CTABLESIZE
                         FOR "50" FIELDS

Table Type       CTablesize(K)      XTablesize(K)
----------       -------------      -------------

ADA                 5
DAT                 7
DB2                 5
IDM                 6
IMS                 6                  3
MAP                 4
SLK                 4
204                 6
TDS                 3


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

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

Dynamically Changing the Parameters

At runtime, you can dynamically modify the SERVERID startup parameter with the SETXPARM and RESETXPARM tools. This reduces the number of table definitions required to define the external data.

The changes to the server parameters are stored in either of two session tables:

@SRVRPRMS_TYP	Manages global changes to all tables of a specified table type, for example, all ADA tables.
@SRVRPRMS_TBL	Manages specific changes to a single named table.

The changes are in effect for the duration of the session, until SETXPARM is invoked again, or the overrides are reset.

TIBCO Object Service Broker does not check the parameter values set or changed at runtime by SETXPARM and RESETXPARM. Your application must ensure appropriate values are set.

Parameters That Can Be Overridden at Runtime

The SERVERID startup parameter is the only parameter that can be dynamically changed with SETXPARM and RESETXPARM.

Examples

The following example sets the SERVERID startup parameter for all ADA tables to TORONTO:

```
CALL SETXPARM('TABLETYPE', 'ADA', 'SERVERID', 'TORONTO', '');
```

This example resets the SERVERID startup parameter for ADA tables to the Table Definer default value:

```
CALL RESETXPARM ('TABLETYPE', 'ADA', 'SERVERID', '');
```

See Also *TIBCO Object Service Broker Shareable Tools* for detailed descriptions of the SETXPARM and RESETXPARM tools.

Adding Threads

When you start the server you are connected to both Adabas and TIBCO Object Service Broker. When the Execution Environment requests an access to Adabas data, a thread to Adabas is established.

The number of instances of the server attached to the address space is specified in the startup JCL. If you require additional instances, do one of the following:

- Shut down the server and start it again with an increased number of instances, using the SERVERS startup parameter.
- Start another instance of the server with the same SERVERID value and a different IDPREFIX.
- Use the **`MODIFY`** operator command to dynamically add instances of the server to an existing TIBCO Object Service Broker Execution Environment. For more information, refer to Using the MODIFY Operator Command on page 23.

For detailed instructions on starting the server, refer to Startup Prerequisites on page 19.

Implementing Fail Safe Processing

Within a single instance of the server, Fail Safe level-1 processing ensures data integrity for a TIBCO Object Service Broker transaction that updates both TDS data and one Adabas database.

The base Adabas architecture does not ensure integrity of data across Adabas databases, therefore placement of a TIBCO Object Service Broker Adabas transaction database should be made with care. Fail Safe level-1 data integrity can only be ensured to the Adabas database containing the transaction database.

To ensure data integrity, configure the server for each Adabas database, each with its own transaction database. This prevents one TIBCO Object Service Broker transaction from updating files from more than one Adabas database.

Transaction Processing

At the successful end of a TIBCO Object Service Broker transaction, the Data Object Broker requests that outstanding updates be committed. During this process, the transaction database is updated.

The update to the transaction database and the Adabas data updates are committed together by one Adabas **CL** (Close) command. When the server responds with a successful completion to the Data Object Broker, the TIBCO Object Service Broker data is committed.

In-doubt Transactions

If the Data Object Broker does not receive a response from the server, the TIBCO Object Service Broker transaction is placed in-doubt. All outstanding TIBCO Object Service Broker locks are held until the in-doubt condition is resolved.

When a connection is re-established with the same instance of the server, the Data Object Broker asks the server to determine if the Adabas data for the in-doubt transaction was committed. The server reads the transaction database to determine the state of the Adabas data. If the Adabas data was updated, the Data Object Broker rolls forward the in-doubt transaction by committing TIBCO Object Service Broker data and releasing locks. If the Adabas data was not updated, the Data Object Broker rolls back the in-doubt transaction by discarding the intent list and releasing locks.

Definition of a Transaction Database

The transaction database is an Adabas file on an Adabas database. It is used by the server to ensure data integrity during Fail Safe level-1 processing. You need to define an ADA table that points to this transaction database. A sample ADA table definition is supplied with the name @ADAFSTRXDB. Modify this definition to choose the **Name**, **DBID**, and **FILE No** for your transaction database. Any valid name can be used.

The FSTABLENAME startup parameter must equal the name of this ADA table, for example:

FSTABLENAME=@ADAFSTRXDB

The transaction database can be managed in TIBCO Object Service Broker like any other ADA table. For example, you can write a rule to clean up the Fail Safe database on shutting down the server.

Following are sample definitions for a transaction database:

```
ADACMP FNDEF='01,AA,008,A,DE'
ADACMP FNDEF='01,AB,008,A,DE'
ADACMP FNDEF='01,AC,004,B'
ADACMP FNDEF='01,AD,004,B'
ADACMP FNDEF='01,AE,004,B'
ADACMP FNDEF='01,AF,025,A'
```

See Also *TIBCO Object Service Broker for z/OS Managing Backup and Recovery* for more information on Fail Safe processing.

Understanding Other Operational Procedures

This section describes other operational procedures: distribution access, status display, debugging, and problem reporting.

Distributed Access

Distributed access between TIBCO Object Service Broker and Adabas is permitted and is subject to the requirements of all distributed access. Refer to *TIBCO Object Service Broker Application Administration* and *TIBCO Object Service Broker for z/OS Installing and Operating* for more information on distributed access.

Refer to Connection to a Windows or Solaris Data Object Broker on page 26 for information about connecting to a Data Object Broker on Open Systems.

Status Display

Use the RESOURCE MANAGEMENT option from the Administration menu to display the status of the instances of the server. The sample screen below shows an example of the type of information displayed for resource type ADA.

```
S6BADM33  HTSTSRV RESOURCE DETAIL FOR ADA ADASRV  2007JAN02 16:40:52
INTERMEDIATE ROLLBK  N EARLY RELEASE     Y LAST USER REUSE    N COMMIT LEVEL 0
RETRY INTERVAL       0 TP NAME            USER ID PREFIX       FAILURES      0
NODE  MMMSYSTEMA01     INDOUBTS          N                    DELETE

                       CONNECTIONS      IN-USE         TRX          MESSAGE
                      CUR  MAX  LMT    CUR  MAX       COUNT          COUNT
           ONLINE       0    0    0     0    0           0              0
           COMMON       0    0   25     0    0           0              0

 SCHEDULE NAME IMSIMSDRASV

          APPLICABLE DAYS          EXCEPTION  START  ONLINE  CONNECTIONS
    MON TUE WED THR FRI SAT SUN    DATE        TIME   ONLY       MAX
     Y   Y   Y   Y   Y   Y                    00:00    N          25




  ENTER-PATHS PF2-TYPE PF4-GROUP PF5-PEER PF9-START PF10-SCHEDULES PF11-UPDATE
 THERE ARE NO ACTIVE PATHS TO BE DISPLAYED, REQUEST IGNORED
```

Debugging

Use the TIBCO Object Service Broker Rule Debugger to identify and fix errors in your TIBCO Object Service Broker application. You can also make and test changes to your rules. The Debugger stops the rule execution at events that you specified from within the Debug screen.

Use the @CONFIGURESERVER tool to trace all Adabas calls that are made to the Adabas nucleus and the control blocks.

Set the TRACE and DUMP configuration parameters to Y, and the LOGMEDIA and DEBUGLEVEL configuration parameters as required to capture the log information necessary for diagnosing the problem. Setting TRACE and DUMP to Y logs all accesses and dumps the Adabas control blocks before and after the call to ADALNK. Setting the DEBUGLEVEL value to 1, 2, or 3 enables the logging of this data, and determines at what point the Adabas control blocks are dumped:

1	The Adabas control blocks are dumped after the call to the Adabas interface program ADALNK.
2	The Adabas control blocks are dumped both before and after the call to ADALNK.
3	The Adabas control blocks are dumped both before and after the call to ADALNK. ESTAE and ADALNK are not called.

The LOGMEDIA configuration parameter determines where the output from the trace goes:

PRT	The default print destination for your TIBCO Object Service Broker user ID. Set the desired print file with the UP user profile option on the workbench.
SCR	The HRNPRINT DDname.
TBL	The @SERVERLOG table. The table is parameterized by the server type (ADA for Adabas), the server ID specified by the SERVERID startup parameter, and the thread ID used by the transaction, specified by the IDPREFIX startup parameter.

For more information on the @CONFIGURESERVER tool, refer to Configuration Parameters on page 13.

For more information on the TRACE, DUMP, DEBUGLEVEL, and LOGMEDIA parameters, refer to Available Configuration Parameters on page 15.

For more information on the SERVERID and IDPREFIX parameters, refer to Supplying the Startup Parameters on page 35.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for more information on the RESOURCE MANAGEMENT option of the Administration menu.

TIBCO Object Service Broker Programming in Rules for more information on using the Rule Debugger.

TIBCO Object Service Broker Parameters for more information about parameters.

Problem Reporting

Refer to How to Contact TIBCO Support on page xx for information about reporting problems with the server to TIBCO Support.

Have the following information available when reporting server related problems to TIBCO Support:

- The ADA table definitions and sample data.
- The server job log and control cards.
- Output from a server trace, if applicable. For more information on the trace options, refer to Debugging on page 43.

Chapter 4 Defining the Accesses to Adabas

This chapter shows you how to define the TIBCO Object Service Broker ADA tables for accessing Adabas data.

Topics

Overview

To access Adabas data from TIBCO Object Service Broker, you must define a TIBCO Object Service Broker table of type ADA. An ADA table can have one or more Adabas fields as TIBCO Object Service Broker fields and an optional location parameter.

To define an ADA table, complete the following tasks:

1. Task A: Extract the Adabas File Definition on page 47
2. Task B: Invoke the Table Definer on page 51
3. Task C: Specify Header Information on page 55
4. Task D: Define Fields for the ADA Table on page 61
5. Task E: Select Extracted Fields on page 64
6. Task F: Map Adabas External Data to TIBCO Object Service Broker Types on page 66

Task A: Extract the Adabas File Definition

To assist in the definition of ADA tables, Adabas file definitions can be extracted so that their table information is available for the definition process. Your system administrator extracts the Adabas file definition information and stores it in TIBCO Object Service Broker.

After extracting the Adabas file definition into TIBCO Object Service Broker, you can use it to create a TIBCO Object Service Broker ADA table. If this information does not exist or was not extracted, the definition is based on the known structure of the Adabas file structure. If there is no extracted data, the Table Definer is unable to verify that the structure is correct and issues a warning message before the definition is saved.

The extracted Adabas file definition information is stored in a TIBCO Object Service Broker TDS table. Because the extracted data is static, you must re-extract the table information whenever changes are made to new or existing Adabas table definitions.

Understanding the Prerequisites

Before extracting Adabas file definition information:

- The Adabas system must be running.
- The Execution Environment performing the extract must have the Adabas interface program ADALNK available. The load name and load DD can be customized with the @CONFIGURESERVER tool.

  Refer to Configuration Parameters on page 13 for information about using @CONFIGURESERVER.

Extracting Table Information

To extract the table information, complete the following steps:

1. Execute the @ADAEXTRACT rule using the **EX** execute rule option on the workbench.

The Adabas Extract Utilities Screen appears (shown here).

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>

Select Extract Functionality

   Extract one definition                                : _

   List all extracted definitions                       : _

   List all possible definitions                        : _

   Extract all possible definitions                     : _

   Extract all non-extracted definitions                : _




PFKEYS: 2=LOGS 3=EXIT 12=EXIT
```

2. On the displayed screen, position your cursor on one of the following menu options and press Enter:

Menu Option	Description
Extract one definition	You can use this to specify the Database ID and the File Number of a file to be extracted in isolation.
List all extracted definitions	Displays a list of all Adabas files whose definitions were extracted.
List all possible definitions	Displays a list of all extracted definitions including Adabas files not yet extracted.
Extract all possible definitions	Extracts definitions from all the Database IDs and File Numbers defined to Adabas. The process also deletes existing definitions before running the new extract.
Extract all non-extracted definitions	Performs the same processing as above except for deleting or re-extracting previously extracted file definitions.

Extracting or listing multiple definitions could take a significant time to complete, depending on the number of Adabas databases and files in your configuration. If you select List all extracted definitions, an Extract Screen listing similar to the one shown below appears:

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>


     Database  File   Extract
        ID      No     Date     Description
       ---     ---  2007-01-19  employees
  _      1       2  2007-01-19  vehicles
  _      1       3  2007-01-19  miscellaneous
  _      1       4  2007-01-19  agent info
  _      1      21  2007-01-19
  _      1      22  2007-01-19
  _      1      23  2007-01-19
  _      1      24  2007-01-19
  _      1      25  2007-01-19
  _      1      26  2007-01-19
  _      1      27  2007-01-19
  _      1      66  2007-01-19
  _      1      71  2007-01-19
  _      1      99  2007-01-19

S=Edit Description  D=Delete  X=Extract
PFKEYS: 3=EXIT 12=CANCEL
```

From this screen, the following actions are available:

S	Presents a screen for modifying the description.
D	Deletes the extracted data from the TIBCO Object Service Broker definition database.
X	Extracts or re-extracts the selected definition.

3. Select an Adabas file definition to delete, extract, or edit the description.

When you issue the **S** line command to edit the description, a screen similar to the one shown below appears, enabling you to edit extracted Adabas file descriptions.

```
2007-01-20            ADABAS EXTRACT UTILITIES                  USR40
Cmd ==>


  Database Id:  1         File Id:  2

    Description: vehicles









  PFKEYS: 3=EXTRACT & SAVE 12=CANCEL
```

Task B: 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 ADA table.

Accessing Existing Tables

Display the definition of an existing ADA table from the workbench in one of three ways:

- Use the DT define table option with the name of an existing table.
- Use the DT primary command with the name of an existing table, for example:

  ```
  DT EMPLOYEE<Enter>
  ```

- Use the DT define table option with no table name. This displays the Object Manager screen, which lists existing tables in your TIBCO Object Service Broker database.

  Scroll through this list to see which table you require. Use the SELECT command to filter the list to show only tables of type ADA:

  ```
  SELECT TYPE='ADA'
  ```

  To select a table, type **S** in the line command field and press Enter.

Defining a New Table

To define a new table, complete the following steps:

1. Type the name of a new ADA table beside the DT define table option or in the primary command field.

   This displays a TDS definition template.

2. Change the Type field at the top of the screen to ADA and press Enter.

   A Table Definition screen similar to the following appears.

```
COMMAND==>                          TABLE DEFINITION

      Table: ADAEXAMPLE       Type: ADA   Unit: USR40


  Serverid:           DBID:      FILE No.:      Server->EE Block: 0
  ISN Assigner: SYSTEM (System/User)  Use GF ID:N  Name of countfield:
  Repeat Type:    (PE/MU) MU/Special Fld in Grp: Y  Occurs/Read: 5

  Location Parm    Typ Syn Len Dec Default          '   Event Rule     Typ Acc
  ---------------- --- --- --- -- ----------------  '   ---------------- -  -
  LOCATION          I   C  16  0                    ' _
                                                    ' _
   | -------- External Field -------- | ---------------- Metadata Field -------
   |Ext Name Syn  Len Dec Upd Fmt Des|    Name          Typ Syn  Len  Dec Ord Def
     --------  -  ----- --  -   -   -  --------------- -  --- -----  --  -  ---
  _
  _
  _
  _
  _
(K=Key  D=Delete  I=Insert  R=Replicate)
PFKEYS: 3=SAVE 12=CANCEL 22=DELETE 13=PRINT 2=DOC 5=COLUMNS
New table definition
```

Following are the TIBCO Object Service Broker ADA Table Definition Screen Segments

Header	Where you specify the Adabas table on which to base your new ADA table and address access control information. Refer to Task C: Specify Header Information on page 55 for more information.
Location parameter	Where you specify an optional location parameter for the new ADA table. Refer to Specifying Optional Location Parameter Information on page 58 for more information.

Event rule	Where you specify one or more optional event rules for the new ADA table. Refer to Specifying Optional Event Rule Information on page 59 for more information.
External field and Metadata field	Displays the columns of the Adabas table on which your new ADA table is based and the TIBCO Object Service Broker fields that you chose. Refer to Task E: Select Extracted Fields on page 64 for more information.
PF keys	Displays the PF keys available from this screen. Refer to Using Table Definer PF Keys on page 53 for more information.

Using Table Definer PF Keys

The following table describes the PF keys that are available from the Table Definer screen and their corresponding primary commands or abbreviations:

PF Key	Primary Command	Description
-	**COPY**	Copies the definition of an existing ADA table into the current definition.
1	**HELP**	Displays help for the field or screen where your cursor is placed.
2	**DOCUMENT**	Displays the screen for documenting the table definition. Refer to Task G: Document ADA Tables on page 68 for more information.
3	**END**	Saves changes to the existing definition and returns you to the workbench.
5	**COLUMNS**	Displays the external fields list from which you can select Adabas fields. To obtain information from the extracted Adabas file definition, the fields **DBID** and **FILE No** must be entered for PF5 to become active. When these fields are supplied, use PF5 to list the fields that were in the Adabas definition when the extract was performed. All relevant fields are set when you return from Adabas field selection screen.
12	**CANCEL**	Cancels the changes to the definition and returns you to the workbench.

PF Key	Primary Command	Description
13	**PRINT**	Prints the definition of the table. You remain in the Table Definer.
22	**DELETE**	Deletes the definition of the ADA table. You are prompted to confirm the deletion.

Task C: Specify Header Information

An Adabas file has a specific structure that must be mapped to one or more ADA tables. The primary key is made up from a field representing the record's Internal Sequence Number (ISN), which is a unique Adabas identifier and, optionally, a second key representing the occurrence number for a periodic group (PE) or a multiple-value field (MU). A third key is required when mapping a multiple-value field within a periodic group.

Representing Adabas Data in TIBCO Object Service Broker Tables

In this example, the Adabas file layout is represented by pseudo-COBOL storage definitions.

For example:

```
01 AA
01 AB,PE             (Periodic Group)
   03 AC,PE
   03 AD,PE
      05 AE,MU,PE    (Multiple-Value Field)
01 AF
01 AG,MU
```

This structure must be mapped to four TIBCO Object Service Broker tables, each representing a different set of fields:

1. The non-periodic groups and multiple-value fields (**AA** and **AF**):

ISN
AA
AF

2. The fields of the periodic group fields **AC** and **AD**:

ISN
CNT1
AC
AD

3. The multiple-value field **AE** within a periodic group:

ISN
CNT1
CNT2
AE

4. The multiple-value field **AG**:

ISN
CNT1
AG

The **ISN** and count (**CNT**) fields are maintained by TIBCO Object Service Broker and do not exist in the Adabas data as defined fields. They are used in this example to identify the current Adabas record (ISN) and the relative occurrence number of a repeating group and or a multiple value field, so that the data can be represented as a relational view. The field names specified in the example are the default values assigned by the Table Definer but these can be altered. These fields can be specified in search arguments to retrieve a specific Adabas occurrence.

Specifying ADA Table Header Fields

The following fields are in the header segment of the ADA table definition screen:

Table	Displays the table name specified when you invoked the Table Definer. Type a new name to save the definition of the current table under a new name. For more information on how to copy TIBCO Object Service Broker objects, refer to TIBCO Object Service Broker *Shareable Tools*. Valid entries: A character string of up to 16 characters beginning with a letter (A - Z) or a special character ($ or #), and continuing with more letters, special characters, digits (0 - 9), or underscore characters (_). A table name starting with an @ symbol denotes a table supplied by TIBCO Object Service Broker.

Type	Displays the table type ADA, which you changed in Task B: Invoke the Table Definer on page 51.
Unit	Displays the user unit associated with the table. The unit marks a table as belonging to a particular application or to a logical unit, for example, utilities, accounting, or network control. The default unit for your user ID is specified in your TIBCO Object Service Broker user profile. Valid entries: A character string of a maximum of 8 characters. These are typically provided by your system administrator, for example, ACC.
Server ID	Identifies a group of instances of the server with common characteristics, and must match the SERVERID startup parameter specified in the server JCL. Type the ID for the server or group of servers to use when accessing the table you are defining. Refer to Supplying the Startup Parameters on page 35 for more information. The SERVERID parameter can be overridden at runtime. Refer to Dynamically Changing the Parameters on page 38 for more information. Valid entries: A character string up to eight characters.
DBID	Specifies the Adabas database number in which the Adabas file resides.
FILE No.	Specifies the Adabas file number of the file for which the table is being defined.
Server->EE Block	Specifies the number of rows passed to the Execution Environment in any one interaction with the server. If set to 0, a full 31 KB buffer is used. If set to the number of rows that satisfy the request, the buffer is less than 31 KB. Setting a large value for this field means that a 31 KB buffer is set when required. Refer to Taking Advantage of Adabas Features on page 79 for examples of how to use this field.
ISN Assigner	Specifies whether the Internal Sequence Number (ISN) of the Adabas file is assigned by Adabas or the user application. Valid entries: **SYSTEM** or **USER**.
Use GFID	Specifies that when the table is bound, a generated Global Format ID is to be used when accessing the table.

Name of countfield	Determines how many occurrences are in a periodic group or multiple-value field. The field is used to turn on the count function. If a definition contains fields from more than one periodic group, choose the value for this field carefully.
Repeat Type	Determines the type of repeating construct to be mapped for tables that map a multiple field or a periodic group. Valid entries: **MU** or **PE**.
Mu/Special field in Grp	When set to **N**, the access to the Adabas file can be optimized. This field has two meanings depending on whether a periodic group (PE) or multiple field (MU) is being mapped. For a periodic group, this field can be set to **N** if: • The definition contains all the fields that are defined in the Adabas group • The fields are in the same order as that defined to Adabas • The group does not contain a multiple value field Otherwise, set it to **Y**. For a multiple value field, the field must be set to **N** if the MU field being mapped is not part of a PE group. If it is an MU within a PE group the field must be set to **Y**.
Occurs/Read	Sets the number of occurrences of a PE group or MU field that are retrieved from Adabas for every Adabas call (occurrences per read). By selecting a value based on the number of occurrences within your own data structures, you can optimize the Adabas access that is performed. Set this field to the average number of occurrences expected to be read for each row or ISN.

Specifying Optional Location Parameter Information

The location parameter segment of the screen is where you define a location parameter for the ADA table. A location parameter is required only if you want to access Adabas data through a peer server associated with a different Data Object Broker (remote node). If you do not require a location parameter, position your cursor in this segment, and then press PF4 and use the **D** line command to delete the parameter. If you always access the Adabas table remotely, the node from which you request the access can have either a minimal or a full definition.

Minimal Definition

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 through the use of the **`Default`** field. Data parameters are defined on the full definition, not a minimal definition.

A minimal definition with a location parameter means you always access data at a remote node. The table type specified in a minimal definition need not match the table type of the full definition on the remote node.

Full Definition

A full table definition with a location parameter indicates you can access data at either the local node or a remote node.

The table type specified in a full definition must match the data on the local node. For example, a full definition of type TDS used to access TDS data on the local node can also be used to access an ADA table with the same name on a remote node.

Specifying Optional Event Rule Information

Use the event rule segment of the Table Definition screen to provide additional controls over the access to a table, and then define event rules based on these accesses.

Event rules are always called when the table is accessed in the access type specified. All the rules that apply to a specific access are executed in the order in which they are entered in the event rule segment. They cannot access tables on a remote node.

Types of Event Rules

You can define two types of event rules, as follows:

Validation	Verify the value of an occurrence when the table is being modified, such as checking the validity of a field value.
Trigger	Cause additional processing to take place when a table is accessed. For example, a trigger rule can be used to create an audit trail or update other tables.

Field Definitions

The event rule information is entered in the scrollable event rule segment. To define event rules, position your cursor in this segment and press PF4 to bring up a definition screen. Complete the following fields:

Line#	Type in a line number, starting at 1 for the first line, with one event rule per line. The line numbers must be numbered consecutively.
Typ	The type of event rule. Valid entries: **V** – Validation Rule. No database updates are allowed during the validation process. The rule must be a function that returns Y (yes), the validation was successful, N (no), the validation was not successful, or a message explaining why it was not successful. **T** – Trigger Rule. A trigger rule cannot be a function or change the contents of the triggering row, and cannot use the TRANSFERCALL statement. Nested triggers are permitted.
Acc	The type of access (or manipulation) to be performed on the data, causing the event to be executed. Valid entries: Validation and Trigger Rules: **W** - Any write (insert, replace, delete) **I** - Only insert **R** - Only replace **D** - Only delete Trigger Rules only: **G** - Any retrieval

See Also

- *TIBCO Object Service Broker Managing Data* for more information on location parameters, event rules and minimal table definitions.
- *TIBCO Object Service Broker Programming in Rules* for more information on the TRANSFERCALL statement.

Task D: Define Fields for the ADA Table

Both the Adabas and TIBCO Object Service Broker definitions are required for a complete field definition because the TIBCO Object Service Broker field definition is extended by the Adabas field definition.

Some Adabas fields cannot easily be mapped to corresponding TIBCO Object Service Broker definitions. For these fields it is possible to tell Adabas to pass the field to TIBCO Object Service Broker in a specific format. Where the Adabas definition maps directly to a TIBCO Object Service Broker definition, it is best to use an identical mapping since no conversion is needed.

If you enter only Adabas field information, the Table Definer fills in the TIBCO Object Service Broker field information based on the Adabas definition.

The field segment of the Table Definer screen is divided into two areas. The left side is the Adabas (External Field) definition area, which must map exactly to the original Adabas field definition. The right side is the TIBCO Object Service Broker (Metadata Definition) definition area, which maps how the data is to be presented to the application. These two definitions do not have to be the same.

Fields in the External Field Definition Area

These fields are found in the External Field (Adabas field) definition area:

Ext Name	Specifies the two-character Adabas short name from the Adabas definition. This name is ignored in the generated key fields (**`ISN`**, **`CNT1`**, and **`CNT2`**) but it still must be entered.
Syn	Specifies the external syntax of the field in the Adabas definition. Refer to *TIBCO Object Service Broker Managing External Data* for a complete listing of external syntaxes. Where the trailing spaces are not significant, the external field should be defined as X if it is mixed case or C if it contains only uppercase. The external syntax affects the way that searches are resolved. For instance, because Adabas always stores data space filled, syntax V is always returned at the maximum length of the field. Therefore, to find the string `ABC` in a six byte syntax V field, the predicate must be "`ABC  `" (the string `ABC` followed by three spaces).
Len	Specifies the length of the Adabas field, in bytes.

Dec	Specifies the number of digits to the right of the decimal point. Valid entries: Syntax P – Number of decimal places must be smaller than twice the length of the entire field. Syntax B – Maximum is 15 for 8 bytes, 11 for 4 bytes and 5 for 2 bytes. All other syntax – 0.
Upd	Specifies whether the field can be updated. Various Adabas fields, subdescriptors and superdescriptors, for example, can be read, but not updated. A non-updateable field in a definition must be specified immediately following the key fields.
Fmt	Specifies whether this Adabas field is to be requested from Adabas in a format different from the format in which it is stored in Adabas. Valid entries: **Y** – The field is requested in the format indicated by the external syntax and length. **N** – The external syntax describes the format of the field as defined in Adabas.
Des	Indicates that the field is a descriptor and can be accessed via the descriptor index.

Fields in the Metadata Definition Area

These fields are found in the Metadata definition area:

Name	Specifies the field name. This name must be unique within the table and must be the name referenced in the application. Valid entries: a character string of up to 16 characters beginning with a letter (A - Z) or a special character ($ or #), and *continuing* with more letters, special characters, digits (0 - 9), or underscore characters (_); for example, `DEPTNO`.
Typ	Specifies the TIBCO Object Service Broker semantic data type of the field. Valid entries: **C** – Count; **D** – Date; **I** – Identifier; **L** – Logical; **Q** – Quantity; **S** – String. Refer to Task F: Map Adabas External Data to TIBCO Object Service Broker Types on page 66 for information on the default mapping of Adabas data types to TIBCO Object Service Broker semantic data types, syntax, and lengths.

Syn	Specifies the TIBCO Object Service Broker syntax of the field. Valid entries are B, C, F, P, and V. For details, refer to *TIBCO Object Service Broker Programming in Rules*. Refer to Task F: Map Adabas External Data to TIBCO Object Service Broker Types on page 66 for information on the default mapping of Adabas data types to TIBCO Object Service Broker semantic data types, syntax, and lengths.
Len	Specifies the TIBCO Object Service Broker length of the field, in bytes. For valid entries, refer to *TIBCO Object Service Broker Programming in Rules*. Refer to Task F: Map Adabas External Data to TIBCO Object Service Broker Types on page 66 for the default mapping of Adabas data types to TIBCO Object Service Broker semantic data types, syntax, and lengths.
Dec	Specifies the number of digits to the right of the decimal point. Relevant only for syntax P. Valid entries: Syntax P – Value must be smaller than twice the length of the entire field. Syntax B, C, F, V, and W – 0. Data type C – 0.
Ord	Specifies that data retrieved from this table is sorted on this field. Valid entries: **A** – Ascending sequence; **D** – Descending sequence.
Default	Specifies a default value to be assigned to the field if the field is null at insert time.

Task E: Select Extracted Fields

To select extracted Adabas fields, complete the following steps:

1. From the Table Definer screen, press PF5 to list extracted fields.

   A screen similar to the one shown below appears, listing extracted fields. For information on using the Table Definer, refer to Defining a New Table on page 52.

```
                    Adabas Field Selection

Non Repeating Fields

     Name     Attributes            Type  Syntax  Length  Dictionary Name
_     AA      DE,UQ                  S       V        8
_     AC      NU                     S       V       20
_     AE      DE,PD,SD               S       V       20
_     AD      NU                     S       V       20
_     PH      PHON                   S       V       20
_     AF      FI                     S       V        1
_     AG      FI                     S       V        1
_     AH      DE                     C       V        6
_     AJ      DE,NU                  S       V       20

Repeating Fields

   Group Name    Name  Attributes          Type  Syntax  Length  Dictionary Name
_      AI          AI   MU,NU               S       V       20
_      AQ               PE
                _  AR   NU,PE,SD            S       V        3
                _  AS   NU,PE,SD                    P        5
                _  AT   MU,NU,PE                    P        5
                _  S3   SUP,DE,NU,PE        S       V       12
_      AW               PE
                _  AX   NU,PE               C       V        6
                _  AY   NU,PE               C       V        6
_      AZ          AZ   DE,MU,NU           S       V        3


PFKEYS: 3=SAVE 12=CANCEL
```

2. Type an **S** next to each of the fields you require.
3. Press PF3 to save the selection and return to the Table Definer screen or press PF12 to cancel and return without selecting any fields.

Note the following when selecting Adabas fields for an ADA table definition:

- The screen is divided into two sections: the top half for non-repeating fields, the bottom for repeating structures. You can select fields in either the non-repeating or the repeating area of the screen, not both.
- In the repeating fields section, select either by group name or by individual field names.
- Only one group-level repeating field can be selected. At this time no other fields can be selected.
- When individual repeating fields are selected, an **`MU`** field cannot be mixed with other fields.

Task F: Map Adabas External Data to TIBCO Object Service Broker Types

This section describes how to map Adabas external data to TIBCO Object Service Broker types.

Understanding the Default Mapping of Adabas External Data Types

The following table shows the default mapping of Adabas external data types to TIBCO Object Service Broker semantic data types, syntax, and lengths. Changing the Defaults describes the Adabas data types that can be converted to TIBCO Object Service Broker syntax. The supported TIBCO Object Service Broker semantic data types, syntax, and lengths are described in *TIBCO Object Service Broker Programming in Rules*.

Adabas Definition				Adabas External Definition		TIBCO Object Service Broker Internal Definition	
Field Type		Sign	Length	Length	Syntax	Length	Syntax
A	Alphanumeric, left-justified	N	1-253	1-253	V, C or X	1-253	V or C
B	Binary, right-justified	N	1-126	1-126	H	5-130	RD
F	Fixed-point, right-justified	Y	4	4	B	8	P
G	Floating-point, right-justified	Y	8	8	F	8	F
P	Packed decimal	Y	1-15	1-15	P	1-15	P
U	Unpacked decimal, Fixed-point, space padded left and right	Y	1-29	1-29	N	1-15	P
W	Unicode	N	1-253	2-126	UN	2-126	UN

Changing the Defaults

You can modify attributes in the TIBCO Object Service Broker field definition area using the values described in Fields in the Metadata Definition Area on page 62. The following options are available:

- Change the entry in the **`Name`** field to a new name to uniquely identify the field within the ADA table. You can name a field the same as a field in another

table; if you are moving data between tables, giving the fields the same name simplifies the process.

- Change the TIBCO Object Service Broker semantic data type (**`Typ`** field) and syntax (**`Syn`** field) of the field. You can use any valid TIBCO Object Service Broker semantic data type (except date) and syntax (except floating point), provided the combination is valid. *TIBCO Object Service Broker Programming in Rules* lists valid combinations.

Changing the TIBCO Object Service Broker field syntax can cause a conversion error, since each affected field of each row to the new syntax as defined in the ADA table definition must be converted.

If data contains only uppercase characters, use TIBCO Object Service Broker syntax C.

Requesting Adabas Data Conversion

TIBCO Object Service Broker can request Adabas data in a different format than the one in which the data is stored. Do the following:

1. Change the Fmt field from N to Y.
2. Change the external syntax and length to the value in which Adabas should return it.

Having TIBCO Object Service Broker request the data from Adabas forces Adabas to perform the conversion, which is more efficient.

TIBCO Object Service Broker can potentially convert this data again to the format described by the Metadata half of the field definition. To avoid the overhead of performing two data conversions, make these two formats the same.

The required syntax is specified as an TIBCO Object Service Broker external syntax not an Adabas syntax. It is then converted to an Adabas syntax.

Task G: Document ADA Tables

Each table definition in TIBCO Object Service Broker has a Documentation screen associated with it. You use this screen to create or modify documentation for the table. To display the Documentation screen for an ADA table, press PF2 from the Table Definer. Following is a sample:

```
DESCRIPTION OF TABLE            PERSONNELD                  UNIT: USR40

 MODIFIED ON 20 JAN 2007 BY ACC            CREATED ON 15 JAN 2007 BY USR40

 KEYWORDS: PERSONNEL
 SUMMARY : TABLE CONTAINING EMPLOYEE DATA

                                DESCRIPTION

 _ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
 _ This table contains all information on current employees.









 PFKEYS: 3=END 5=VIEW DOCUMENT 13=PRINT 12=EXIT
```

Defining Field Values

The Table Definer updates some of the fields on this screen, but you must maintain the **KEYWORDS**, **SUMMARY**, and **DESCRIPTION** fields. Complete these fields as follows:

KEYWORDS	Type individual words that briefly describe the table. These words are used by the Keyword Search facility in TIBCO Object Service Broker. This field is one-line long and can contain multiple entries, separated by commas or blanks.
SUMMARY	Type a one-line summary of the **DESCRIPTION** field.

DESCRIPTION	Type information about the table (for example, what its role is, what it does, and how it works) using TIBCO Object Service Broker **`SCRIPT`** commands. There is no limit to the amount of information you can type in this field.

Using the PF Keys

The following function keys are supported in the Documentation screen:

1	Displays corresponding help for the field or screen where your cursor is placed.
3	Saves changes and returns you to the Table Definer.
5	Toggles between browse and edit modes.
12	Cancels changes and returns you to the Table Definer.
13	Prints the version of the documentation that you are viewing.

See Also *TIBCO Object Service Broker Shareable Tools* for more information the SCRIPT tool.

Chapter 5 Using TIBCO Object Service Broker to Process Adabas Data

This chapter describes how to access and process data in TIBCO Object Service Broker ADA tables and how to take advantage of the Adabas features in TIBCO Object Service Broker.

Topics

Processing the Data

When accessing Adabas data to process it from TIBCO Object Service Broker, the following occurs:

- Requests are translated into native Adabas commands.
- Adabas field data types are translated to the TIBCO Object Service Broker field types defined in the ADA table.

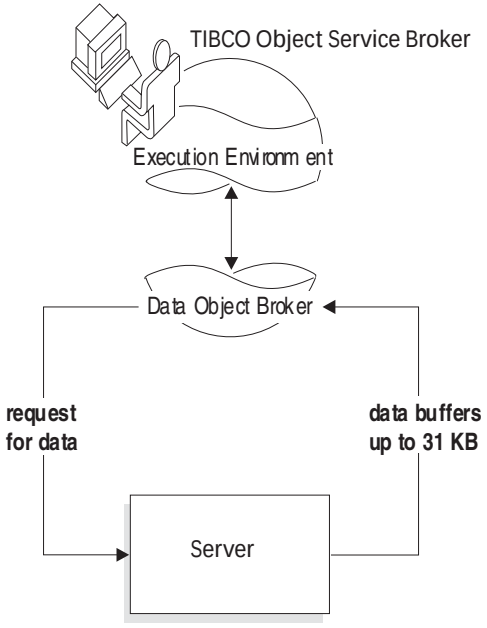
You can access the data using:

- Various workbench tools, e.g., Table Browser and Table Editor
- The rule language

The following sections describe the mechanisms available to process the data, how to take advantage of Adabas functionality, and how to handle errors.

Transaction Process

When Adabas data is requested, a server starts a FORALL. Data is sent to the Data Object Broker in variable length buffers up to a maximum of 31 KB, as shown in the following figure. If a single request requires more than 31 KB of data, multiple 31 KB buffers are sent and processed until the request is complete.



Restrictions on Adabas Processing

Processing Adabas data through TIBCO Object Service Broker is subject to the following limitations:

- TIBCO Object Service Broker Adabas must be initialized by a standard ADALNK module that is identical to that provided by Software AG.
- TIBCO Object Service Broker has a maximum occurrence size of 3.9 KB.
- TIBCO Object Service Broker has no direct equivalent to the Natural statements HISTOGRAM or FIND UNIQUE. Therefore TIBCO Object Service Broker does not issue the Adabas commands **`L9`**, **`S1`**, or **`RC`**. Using TIBCO Object Service Broker rules to manipulate the returned data can enable the same functionality.
- TIBCO Object Service Broker table definitions for Adabas fields are based solely on the FDT. Therefore Predict and Natural long field names as defined in the DDM must be entered manually.

Using the Table Browser and Table Editor

You can browse and edit an ADA table in the same way you would browse or edit another TIBCO Object Service Broker table. Use the Table Browser to browse a defined ADA table by typing the table name next to the BR browse table option and pressing Enter. This displays a screen similar to the following example.

```
 BROWSING TABLE   :  ADDRESSD
COMMAND ==>                                                      SCROLL: P
  ISN        KEY1          CLIENT_NUMBER       SURNAME          INITIALS POSTCODE
_ -------- ----------- ----------------  ------------------   ---     --------
_         2 HARGREAVES  NR CR0    1       HARGREAVES  (HQ)     NR      CR0 1AL
_         3 O'LANE      JE SK13   2       O'LANE      (HQ)     JE      SK139XF
_         6 HARGREAVES  NJ CR0    3       HARGREAVES  (HQ)     NJ      CR0 1AL
_         1 O'LANE      L  SK13   4       O'LANE      (HQ)     L       SK139XF
_        17 POWELL      FJ L23    5       POWELL      (HQ)     FJ      L23 9ST
_        18 KNIGHTS     GC CR7    6       KNIGHTS     (HQ)     GC      CR7 5DQ
_        19 PEARCE      V  KT8    7       PEARCE      (HQ)     V       KT8 0DQ
_        20 TARZAN      L  KT1    8       TARZAN      (HQ)     L       KT1 1PT
_        21 BROWN       TT CR0    99      BROWN       (HQ)     TT      CR0 67RT
_        22 HILARY      HH TT56   98      HILARY      (HQ)     HH      TT56 6TY
_        27 O'REILLY    B  DU9    99      O'REILLY    (HQ)     B       DU9  1TY
_        31 BROWN       X  DU8    1004    BROWN       (HQ)     X       DU8
_        32 BLENKS      BBBSK13   9999    BLENKS      (HQ)     BBB     SK13 9XF



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

See Also *TIBCO Object Service Broker Managing Data* for more information about browsing and editing tables in TIBCO Object Service Broker.

Using Rules

Accessing Adabas data using the rules language is similar to accessing TIBCO Object Service Broker data. The main difference is in the way Adabas interprets the request.

The following sections outline differences encountered while using rules and also point out normal TIBCO Object Service Broker rules behavior that you must consider when building applications.

Transaction Streams

If you issue a TIBCO Object Service Broker EXECUTE statement within a main (parent) transaction, it creates another transaction stream (child transaction), to a maximum of nine streams. The number of streams allowed in a TIBCO Object Service Broker transaction depends on the TRANMAXNUM Execution Environment parameter, which has a default of nine streams. Each transaction stream in TIBCO Object Service Broker that accesses Adabas data requires its own thread.

Ensure that your system administrator is aware of the number of the threads required to accommodate all transaction streams accessing Adabas data in a single transaction.

Using TRANSFERCALL or DISPLAY & TRANSFERCALL statements in your rules minimizes the threads and reduces the possibility of Adabas locking contention.

Table Access Dependencies

The number of ADA tables you can access per transaction depends on the POOLSIZE parameter. Refer to Supplying the Startup Parameters on page 35.

If you use the default startup and configuration parameter values, you can access at least sixteen ADA tables in one transaction. Your ability to access more tables depends on the size of the ADA table definitions. Refer to Estimation of the CTABLESIZE Parameter on page 37 for more information.

Retrieval Processing

The Adabas **`Lx`** and **`Sx`** commands are used to retrieve data from Adabas for each retrieval statement (GET or FORALL) in your rule. The command you use depends on the request and the mode of the transaction.

When TIBCO Object Service Broker runs in browse mode, no locks are taken in Adabas. When TIBCO Object Service Broker runs in update mode, Adabas takes an exclusive lock on every occurrence that is read. When an exclusive lock cannot be obtained by Adabas, a LOCKFAIL exception is returned.

If the number of occurrences locked exceeds the maximum value for the NISNHQ Adabas parameter, a SERVERERROR exception is returned to the requesting rule along with a message documenting that this condition occurred.

Refer to Exceptions on page 87 for more information about LOCKFAIL and SERVERERROR.

Some expressions cause the retrieval processing to do a complete sweep of the Adabas file. Wherever possible, specify at least one equality operator to restrict the amount of data retrieved. Refer to Taking Advantage of Adabas Features on page 79 for more information on using rules efficiently.

GET Statement

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

A TIBCO Object Service Broker GET is equivalent to an Adabas FIND FIRST statement.

FORALL Statement

A FORALL statement returns occurrences to TIBCO Object Service Broker in the order in which Adabas passes them. If you require a different order, you must include an ORDERED clause in your FORALL statement, or specify ordering in the table definition.

If you know that you require only one record, use a GET statement rather than FORALL.

The TIBCO Object Service Broker FORALL is equivalent to an Adabas READ or FIND statement.

When running in update mode, the equivalent of an exclusive lock is applied to the occurrence being updated. To accommodate the largest number of occurrences that can be updated within a single transaction, increase the size of the NISNHQ parameter in the Adabas configuration as needed.

Replace (Update) Processing

A REPLACE statement that contains an operator is allowed for non-Adabas data only. This is not supported for ADA tables; the occurrence to be replaced must first be retrieved by a GET or a FORALL.

Sample REPLACE Statement Wrapped in FORALL

```
FORALL TABLEA WHERE FIELD1 = ABCD:
   TABLEA.FIELDn='A';
   REPLACE TABLEA;
   END;
```

Delete Processing

Deleting an occurrence of a repeating structure whether it is a repeating group or a multiple value field is not supported. However, if you want to remove data from a repeating structure, use the same technique in TIBCO Object Service Broker as you would in Adabas, which is to nullify all the fields in the occurrence and replace this empty occurrence. For alphanumerics, set the field to spaces and for numerics to zeros.

A DELETE statement that contains an operator is allowed for non-Adabas data only. This is not supported for ADA tables; the occurrence to be DELETED must first be retrieved by a GET or a FORALL.

Here is a sample DELETE statement:

```
GET TABLEA WHERE FIELD1 = ABCD;
DELETE TABLEA;
```

Insert Processing

Adabas supports two kinds of insert processing according to the Internal Sequence Number (ISN). The ISN is either assigned by the system or the user specifies the ISN at insert time. The kind of processing to be used is determined in the table definition **`ISN Assigner`** field.

An ISN value must always be specified in the INSERT statement, even if it is assigned by the system. When the **`ISN Assigner`** field is set to SYSTEM, the supplied ISN value is ignored. This is because the ISN is a key field, which is a required field in the TIBCO Object Service Broker rules language.

See Also *TIBCO Object Service Broker Programming in Rules* for general information about using the TIBCO Object Service Broker rules language and about TIBCO Object Service Broker transaction processing.

Taking Advantage of Adabas Features

Two considerations take place when accessing data from Adabas:

- Maintaining a consistent interface across all external DBMSs and the TIBCO Object Service Broker TDS database
- Providing an efficient access path to the external DBMS

You can tune several aspects of your application rules and tables to make accessing Adabas data more efficient. This section provides some suggested techniques for building efficient applications.

Understanding Descriptor Indexes

A descriptor index is built within the Adabas database to aid efficient access to data in Adabas. It enables access to the data in an order determined by a particular field, sub-component of a field, or a combination of multiple fields (descriptor, subdescriptor and superdescriptor).

Using the Adabas direct call interface, you can choose the access path that Adabas uses. In Natural, for example, you can tell Adabas to access by descriptor. You can also do this with this SDK; unfortunately this breaks the consistent interface to data that TIBCO Object Service Broker tries to provide across all data sources. An application written using these techniques can lose some of its portability across different data sources without code changes. That is, you can lose the ability to move your data from Adabas into TDS or another DBMS without affecting the application. Therefore, use this feature carefully.

If the data is moved to an RDMS from Adabas, it is unlikely that the logical or physical design of the database mirrors that of the Adabas file structure.

Data Reading

To access by descriptor index, the descriptor field in the table definition must be marked as a descriptor by setting the **Des** field to Y. The access statement in the TIBCO Object Service Broker rules should be in the form:

access_statement tablename `WHERE` *descriptor_field* `>=` '*value*'

where:

access_statement	Is either GET or FORALL.
tablename	Is an ADA table.
descriptor_field	Is a field marked as a descriptor.
value	Is the starting value in the index.

This statement uses the greater than or equal to (>=) operator instead of just equal to (=). Although using equal to (=) is correct, greater than or equal to (>=) is more efficient since access to the Adabas table is with a direct Adabas **L3** or **L6** command, which reads data via the descriptor index that makes up the *descriptor_field* in the table definition. This requires the reading of only part of the index instead of the entire index. For this to occur, set the **Des** column for the descriptor field to a value of Y.

If you require more than one record, ensure that your rule has a means of ending the loop as soon as all required records have been returned. This can be done by calling a subsequent rule to do a check that issues a SIGNAL *signal* when all qualifying records have been received.

Drawbacks

There are two drawbacks to using descriptor indexes to read data:

- The data is returned to the rule in the same order that it is retrieved from Adabas, which is different from the TIBCO Object Service Broker TDS, which retrieves the data in primary key sequence.
- This is an open-ended request which implies that the processing should start at the point indicated by the WHERE statement, and end only at the end of the file. If this is the desired found set, this is not a problem. However, it is more likely that only part of this potential found set is actually required and processing should really end when a particular value is reached.

The higher-level rule containing the FORALL should either have an ON signal... exception or the access statement should take the form FORALL.....UNTIL *signal*. The examples below show a combination of rules for narrowing access by descriptor.

When the value of `FIELD1` changes, the loop terminates and processing continues at the next rules statement.

```
RULE EDITOR ===>
 ADATEST
 _
 _ ------------------------------------------------------------
 _ ------------------------------------------------------------
 _ FORALL TABLEA WHERE FIELD1 >= ABCD UNTIL END_OF_DATA:
 _       CALL TEST_FOR_END;
 _       CALL PROCESS_TABLEA;
 _ END;
 _ ------------------------------------------------------------
```

```
     RULE EDITOR ===>
 TEST_FOR_END;
 _
 _ ------------------------------------------------------------
 _ TABLEA.FIELD1 > 'ABCD';                          | Y N
 _ ---------------------------------------------------+----
 _ SIGNAL END_OF_DATA;                              | 1
```

Compensation for Performance Degradation

Using descriptor indexes can slow the performance since TIBCO Object Service Broker tries to balance two factors when getting the external data. TIBCO Object Service Broker compensates by reducing either of the following:

- The amount of work the external database is asked to do
- The communications overhead

This reduction is achieved by sending a request for data with the response being a block of data from the external database. The maximum size of the data in this block is approximately 3.9 KB. As a result, one message in response to the TIBCO Object Service Broker rules code can contain many rows of data.

In the previous example, more rows could have been read from Adabas than required. To be efficient as possible, you are provided with the option of predefining the number of rows that are returned in response to TIBCO Object Service Broker rules code.

Using the **SERVER=>EE Block** field to limit the number of occurrences accessed in one case can have a detrimental effect on other accesses to that ADA table by increasing the overhead of the message traffic. Refer to Specifying ADA Table Header Fields on page 56 for the description of the **SERVER=>EE Block** field.

Defining Effective ADA Tables

Follow these recommendations to create well designed ADA tables:

- If you select all the fields in a periodic group and none of them are descriptors, set the **MU/Special Field in Grp** field to N. This optimizes the processing of the Adabas commands issued.
- For all Adabas fields of format A (Alphanumeric) that contain only uppercase data and no significant trailing blanks, set the **Syn** field for both external field syntax (default X) and the TIBCO Object Service Broker (Metadata Definition) syntax (default V) for the field to C (character).

  This simplifies data access in coding the rules and is also useful for descriptors, superdescriptors, or subdescriptors.
- Use TIBCO Object Service Broker syntax P rather than B for Adabas fields defined externally as syntax P. Also ensure that the lengths are the same.
- Although TIBCO Object Service Broker can store an Adabas P5 data value to a TIBCO Object Service Broker B4 field, you can avoid the unnecessary conversion overhead on each record retrieved by using identical syntax on both sides of the field definition.
- Set the **Occurs/Read** (occurrences per read) field (default 5) to the most appropriate value for your data.

  This field determines how many occurrences of a repeating data structure are asked of Adabas for in a single access. If you know that there are always 12 occurrences, set the value to 12, if always 3, set it to 3. If you are uncertain whether there are more often 12 or 10, set it to 12. It is better to overestimate, in order to minimize the number of accesses to the file.

Coding Efficient Adabas Accesses

In general, an efficient data-access design via Natural is also efficient in TIBCO Object Service Broker, since both generate direct calls and send them to Adabas for resolution. Although both can optionally do some additional evaluation of the found set returned from Adabas, it is more efficient in both cases for Adabas to do the work wherever possible. This is especially true for sorting returned items and is true of virtually all databases.

Optimized access for a TDS table can exhibit different behavior from optimized access for an Adabas table.

Remember that in general:

- The TIBCO Object Service Broker GET is equivalent to Adabas FIND FIRST.
- The TIBCO Object Service Broker FORALL is equivalent to Adabas READ or FIND.

Follow these recommendations for coding efficient TIBCO Object Service Broker Adabas accesses:

- Ensure that all accesses to Adabas are done in browse mode, unless you want to update data.
- This ensures that no locks are taken by Adabas, so that the data is not requested in update mode.
- Use the UNTIL statement to end searches.

Except for accesses using the equality (=) operator, a search (for example greater than or equal to (>=)) can trigger a long running request. To avoid this, you should always consider including an UNTIL clause in the FORALL statement to end the search and stop issuing requests to Adabas.

Preserving Data Sequence in FORALL Statements

When data sequence is important, code FORALL statements appropriately. It is possible that the same FORALL statement, retrieving the same number of similarly structured rows from either Adabas or TDS using the same table definition, returns the data in a different sequence from each database. This occurs because Adabas returns data in descriptor index sequence, if the search expression uses one, whereas the same search on TDS returns data in primary-key sequence, which is the ISN in the case of Adabas.

This usually does not matter, as long as the same set of records is returned; normally the same operations are carried out on all records retrieved. However, if the record sequence is important, there are two possible solutions:

- Insert a condition at the top of the rule to determine (based on the value of the LOCATION parameter for the table) whether for each execution of the rule the access should be to local TDS or remote Adabas data. Replicate the FORALL loop and add an ORDERED clause to the one to be used to access TDS.
- Define the field to be used for access as a secondary index on the TDS table, so that the behavior matches that found in the Adabas access.

Using LIKE and NOT EQUAL with Other Operators

Use the TIBCO Object Service Broker Partial Match LIKE operator and the NOT EQUAL (¬= *or* NOT =) operator only in combination with other operators. There is no direct command equivalent to these operators in Adabas, so using them alone forces a full sweep of all records in the file to satisfy the request. Using them with additional arguments enables Adabas to return a much smaller found set to TIBCO Object Service Broker.

For example:

```
FORALL CLIENTS WHERE LNAME NOT EQUAL 'Smith' AND AreaCode='905'
```

or

```
FORALL CLIENTS WHERE LNAME LIKE 'Mac*' AND AreaCode='905'
```

The equality (=) operator is applied first to retrieve each occurrence with `AreaCode` equal to `905` and *then* the NOT operator is applied before the data is passed on to the Execution Environment.

Reducing CPU Consumption

The only area in which consumption of CPU can be affected is in the table definition. Use the following techniques in your TIBCO Object Service Broker tables to optimize system performance:

- Bind the table definition in a production environment.
- Avoid defining a single table consisting of a large number of fields. It is better to define a number of small tables rather than one larger one. This can help avoid unnecessary processing overhead and additional fields can easily be added at any time as required.
- Define only the fields that are required in the ADA table definition. This reduces the work to be done to access, process, and convert the required fields

and increase the number of occurrences contained in a given message, reducing the communications overhead.

- Ensure that the external field definitions and the TIBCO Object Service Broker field definitions specify identical syntax. It is better to have Adabas perform data conversion, since it must convert from its internal storage format to the predefined external format anyway.

Understanding Adabas Direct Calls Generated from TIBCO Object Service Broker

The TIBCO Object Service Broker Statement...	Is Equivalent to Adabas...
`GET TABLE WHERE FIELD = value`	`FIND FIRST (S1)`
`FORALL TABLE WHERE FIELD = value`	`FIND (S1 + L1's)`
`FORALL TABLE WHERE FIELD >= value`	`READ (L3's from a start value until EOF)`
`FORALL TABLE WHERE FIELD >= value AND FIELD <= value`	`FIND (S1 + L1's)`
`FORALL TABLE UNTIL signal`	`READ PHYSICAL (L2's) of whole file`
`GET TABLE WHERE ISN = value`	`GET ISN (L1)`
`GET TABLE WHERE FIELD = value`	`FIND FIRST (S1)`

Handling of Errors

This section explains how TIBCO Object Service Broker requests are handled with respect to the following:

- Synchronization and recovery
- Exception handling

Synchronization and Recovery

Synchronization and recovery are governed by the following:

- Adabas locks are not requested when the TIBCO Object Service Broker transaction is running in browse mode and updates on Adabas data cannot be performed.
- When a TIBCO Object Service Broker transaction ends successfully, an Adabas **CL** (Close) command is issued, causing Adabas locks to be released. An Adabas **CL** implies an **ET** (End Transaction) command. **ET** is equivalent to a TIBCO Object Service Broker commit point.
- When a TIBCO Object Service Broker transaction ends unsuccessfully, an Adabas **BT** (Backout Transaction) command is issued, immediately followed by a **CL**, causing the Adabas updates to be backed out and the locks to be released.
- Intermediate COMMIT and ROLLBACK requests are not supported because the locking architecture of TIBCO Object Service Broker and Adabas is fundamentally different. While TIBCO Object Service Broker frees locks at the end of a transaction, Adabas frees locks at COMMIT (**ET**) or ROLLBACK (**BT**).

COMMITLIMIT exception does not apply to ADA tables. Adabas update requests are passed to Adabas as they are encountered. They are not held in the TIBCO Object Service Broker intent list.

Data Integrity

Fail Safe level-1 processing is supported to ensure data integrity when updating both TIBCO Object Service Broker and a single Adabas database from within one TIBCO Object Service Broker transaction.

If you did not request Fail Safe processing, transactions that update both Adabas and TIBCO Object Service Broker data can result in discrepancies if abnormal termination occurs during the transaction end processing. Refer to Implementing Fail Safe Processing on page 40 for more information.

Data integrity cannot be assured when updating TIBCO Object Service Broker and multiple Adabas databases, because Adabas does not support updates to multiple databases with integrity.

To ensure data integrity, only one Adabas database should be updated within a single TIBCO Object Service Broker transaction.

Exceptions

The TIBCO Object Service Broker runtime environment signals system exceptions to enable 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 catch all exception. Each exception traps the exceptions that appear below it in the hierarchy.

All errors encountered when accessing Adabas data are trapped under one of the following exceptions:

ERROR	An error is detected and no lower-level exception handler is coded in the application.
ACCESSFAIL	A table access error is detected. **GETFAIL** – No occurrence satisfies the selection criteria. The Adabas error code 14 raises this exception. **INSERTFAIL** – The primary key provided for an INSERT statement already exists. The Adabas error code 10 raises this exception.
INTEGRITYFAIL	An attempt to violate data integrity is detected.
LOCKFAIL	An attempt was made to read more occurrences with Adabas's primary exclusive lock. Restructure your application to take fewer locks in a transaction, run in browse mode so locks are not taken, or increase the NISNHQ parameter.
SERVERBUSY	A new transaction requested an instance of the server processing the request, but none is available. Control is passed back to the rule, giving the rule the opportunity to try the transaction again. If this exception is raised too often, consider requesting more server instances or reviewing the amount of work being done in your transactions.

SERVERERROR	A request was made to Adabas and Adabas returned an error code that does not map to one of the specific TIBCO Object Service Broker exceptions. The ON SERVERERROR handler should call @SERVERERROR to parse the error message (contained in ENDMSG). Refer to @SERVERERROR on page 88 for more information.
SERVERFAIL	A transaction was in progress when the connection to the server servicing the request was broken, or the server failed. Control is passed back to the rule, giving the rule responsibility for transaction cleanup.

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

@SERVERERROR

You must pass @SERVERERROR the contents of RETURN_MESSAGE, which has the following format:

`S6BAD`*nnnx serverid serveruserid source*`:` *Message*

The following table describes the variables necessary to pass the RETURN_MESSAGE contents to @SERVERERROR:

nnn	The Adabas external message number.
x	The message severity (E for error, W for warning, and I for information).
serverid	The ID of the server servicing the request.
serveruserid	The user ID (IDPREFIX + ###) of the server.
source	The code portion that trapped the error and returned the message (for example, CSECT, rule, or function).
Message	The actual error message text.

If a specific message has some information that is required to process the error, the table driven approach to the execution of @SERVERERROR causes a rule (specified for that error by the developer using @SERVERERROR) to execute. The error message is interpreted in the @SERVERERROR processing and put into a temporary table until required.

Customization of Error Handling

To customize error handling, you must update data in the @SERVERMSGCNTL control table. The definition of this table is owned by TIBCO Object Service Broker and must not be modified. The data is owned by the users.

Processing of Table

Here is how the table is processed when the SERVERERROR exception is raised and the @SERVERERROR rule is called by your application:

1. @SERVERERROR reads the @SERVERMSGCNTL table and looks up the specific message identifier handlers.
2. The appropriate message handler looks up the external error codes in the correct control tables.
3. If any codes are found, they call the associated user-written handler.
4. The user-written handler can use other functions and data stored in specific tables to handle any specific external error/status code.

@SERVER ERROR can be called at any time, although it is useful only for parsing TIBCO Object Service Broker messages generated due to external Adabas errors. The original message can always be retrieved using the @SE_MSG rule after @SERVERERROR is called. The information parsed by @SERVERERROR has transaction scope.

You can add your own instances in the @SERVERMSGCNTL table, provided that the OWNER specified begins with letters A to Z, and the key values in their instance are message identifiers in the form S6BAD*nnnx* mentioned above.

@SERVERERRORADA

Use the @SERVERERRORADA rule to reformat the message returned from a SERVERERROR exception.

The @SERVERERRORADA rule populates the fields of a temporary table @SERVERERRORADA by breaking up the returned message string into individual fields. For example, it places the message number into field **`MSG_ID`** and the thread ID into the field **`MSG_THREAD`**.

See Also *TIBCO Object Service Broker Shareable Tools* for more information on the @SERVERERROR and RETURN_MESSAGE tools.

Chapter 6 Configuring Accesses to CA Datacom Data

This chapter describes how to configure the accesses to CA Datacom data, how to connect the server to a Data Object Broker, and how to start and stop the server.

Topics

Accessing CA Datacom Data

Access to CA Datacom data is only supported via the Service Gateway for Files SDK. After the SDK has been installed, you need to define the appropriate tables to facilitate the manipulation of the CA Datacom data, then write the rules or use tools to process the data. Your applications used to access the CA Datacom data are comprised of table definitions and rules.

The access uses the CA Datacom User Requirement Tables (URT) to connect to the CA Datacom environment. All CA Datacom I/O uses the Multi-User Facility.

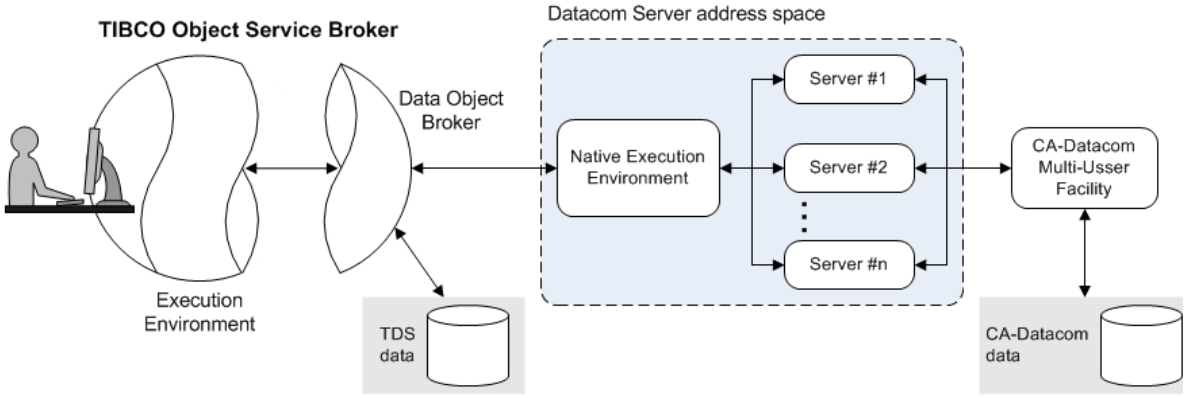
You use the components below to access CA Datacom data from TIBCO Object Service Broker:

Component	Function
DATACOM shareable tool	To extract CA Datacom metadata and generate or manage TIBCO Object Service Broker DAT table definitions.
Workbench DAT Table Definer	To create TIBCO Object Service Broker DAT table definitions on the basis of the metadata extracted from CA Datacom Data Dictionary through the DATACOM shareable tool. Also, to view or edit all existing DAT table definitions.
UI DAT Table Editor	To create TIBCO Object Service Broker DAT table definitions on the basis of a COBOL copybook. Also, to view or edit all existing DAT table definitions.
DAT Server	To access CA Datacom data when TIBCO Object Service Broker data access is requested for a DAT table.
CA Datacom User Requirement Table (URT)	To connect to the CA Datacom environment.

Supported Configurations

The Data Object Broker and the server can be configured to reside on the same or different domains and operating systems (z/OS, Windows, and Solaris). The server must be on the same domain as the CA Datacom database system.

Once your environment is set up you can access CA Datacom data while having equal access to TIBCO Object Service Broker's tables of other types.



Preparations for Installation

This section describes the steps to take in preparation for installation.

Installation of the TIBCO Object Service Broker Base Component

You must install the TIBCO Object Service Broker base component before configuring the server. The base component can reside on z/OS, Windows, or Solaris. Installation instructions are in the TIBCO Object Service Broker *for z/OS Installation and Operations* and *TIBCO Object Service Broker for Open Systems Installing and Operating* manuals.

Installation of the Server

Installation instructions for the Service Gateway for Files, which includes the SDK, are located in TIBCO Object Service Broker *Managing External Data.* The Datacom server is shipped as part of the SDK.

Implementation of TIBCO Object Service Broker Security

You can implement security for the server using standard TIBCO Object Service Broker *security:* To restrict access to DAT tables after they are defined, proceed with security as for any TIBCO Object Service Broker table.

CA Datacom Security Considerations

The server places the user ID in positions 4 through 11 of the User Information Block (UIB). When the X option is specified in the DEBUGPARMS field in the @SERVERDEBUG(DAT) table, the literal PXX is placed in positions 1 through 3 of the UIB. Additionally, user impersonation takes place if the startup parameter SECLEVEL is set to 1.

Refer to Specifying the Startup Parameters on page 119.

Fail Safe Processing Considerations

To guarantee consistency when updating both TIBCO Object Service Broker TDS data and CA Datacom data in a single transaction and through a single instance of the server, you can use Fail Safe Level 1 processing.

For more information, refer to Implementing Fail Safe Processing on page 126.

Communications Requirements

If all components reside in the same domain and in authorized libraries, Cross Memory Services is used for communications. In all other cases, TCP/IP is used for communications.

See Also

TIBCO Object Service Broker *for z/OS Installation and Operations* for detailed information about TIBCO Object Service Broker communications.

TIBCO Object Service Broker *Messages with Identifiers* for information on messages produced by the server.

TIBCO Object Service Broker *Security* for information on security for tables.

Prerequisites for the CA Datacom Environment

You must complete the following tasks for your CA Datacom environment before you can access the CA Datacom data:

- Generate the CA Datacom User Requirements Tables.
- Optionally, populate the Data Dictionary tables.
- Define the CA Datacom environment.

For details, see the following sections.

Generating the CA Datacom User Requirements Table

The server can use multiple URTs to access your data. Those URTs are loaded at runtime and must, therefore, reside in one of the load libraries available to the server. You must ensure that any URTs that you provide have DBNTRY as the main entry point.

Populating Data Dictionary Tables

Before defining the CA Datacom environment, you might want to extract some metadata from the CA Data Dictionary into TIBCO Object Service Broker. Use the DATACOM interactive tool to extract the metadata of your choice.

Also, you can run the extraction as a separate job. Member DCOMDEXT in the JCL data set contains the JCL for extracting metadata from CA Datacom Data Dictionary tables into TIBCO Object Service Broker.

The interactive DATACOM tool can be used for extracting metadata from CA Datacom Data Dictionary only if your TIBCO Object Service Broker runs on z/OS. In other cases (Windows or Solaris), your only choice is running the DCOMDEXT JCL as a separate job on z/OS with the TDS parameter denoting your TIBCO Object Service Broker.

Defining the CA Datacom Environment

The CA Datacom Master List and the URT must be properly defined before the server can run.

Defining the Master List

The following parameter must be set to control the CA Datacom environment:

TASKS	Specifies the maximum number of tasks that can operate concurrently within CA Datacom. Each server that is running uses one task.

Defining the User Requirement Table

Set the following parameters for the URT:

Parameter	Macro	Description
OPEN=USER	DBURINF	Indicates that the server is responsible for the open and close processing of the URT.
URTABLE=ASM	DBURINF	Indicates that the CA Datacom macros are assembled as one object when the URT is generated.
TIMEMIN=0, TIMESEC=1	DBURSTR	Indicates that control is to be returned to the caller immediately upon a failed attempt to lock a row. This behavior is essential for the TIBCO Object Service Broker DAT Server's ability to signal the LOCKFAIL exception when appropriate; the CA Datacom default behavior is a "wait forever" condition.
TXNUNDO=YES	DBURSTR	Indicates that all tables described in this URT use logging when modified (see UPDATE=YES), so TIBCO Object Service Broker's ROLLBACK statement can be carried out against tables of type DAT mapped to them (CA Datacom will commit data if TXNUNDO=YES is not specified).
UPDATE=YES	DBURTBL	Indicates that the caller can modify the rows in this table. Also, callers that issue CA Datacom's SELxx commands (TIBCO Object Service Broker DAT Server falls into this category) require UPDATE=YES. See the CA Datacom documentation for details.

Your URT must have DBNTRY as the primary entry point (use the linkage editor's ENTRY DBNTRY control statement). Also, you can provide multiple URTs. For details, see Adding URT Names on page 117.

Startup Parameters

Once your environment is ready to access the CA Datacom data, you need to specify the server startup parameters. You specify these parameters on the EXEC statement in the startup JCL for the server, in a data set, or both.

If you specify parameters in both the EXEC statement and a data set, EXEC statement parameters override data set parameters.

Startup Parameters in the EXEC Statement

Include parameters in any order (up to 100 bytes). The following example shows sample parameters specified on the EXEC statement in the startup JCL:

```
//DCOMGTW EXEC PGM=S6BDR000,
//         REGION=0M,
//         PARM=('TDS=SAMPDOB')
```

Startup Parameters in a Data Set

Include parameters in any order, one per record or comma-separated, beginning in column one, and ending with a blank or a comma. An asterisk (*) in column one indicates a comment record. The data set must be defined as follows:

- DDNAME HRNIN
- Allocated FB LRECL=80

The following example shows sample parameters specified in a data set:

```
//HRNIN DD *
SERVERS=3
SERVERTYPE=DAT
SERVERID=DATACOM1,IDPREFIX=DCOM
```

Startup Prerequisites

If the Data Object Broker is on z/OS and Dynamic Resource Creation is not permitted (Data Object Broker Parameter `DYNAMICRESOURCE = N`) the server must be identified to the Data Object Broker in a permanent resource. To do this, define server resources to the Data Object Broker's resource management repository file.

If the Data Object Broker is on z/OS and Dynamic Resource Creation is permitted (Data Object Broker Parameter `DYNAMICRESOURCE = Y`) and there is no matching permanent resource, a dynamic resource entry matching the servers requirements will be created when the server connects to the Data Object Broker. If there is a permanent entry for the server it must match the requirements for the server. It is recommended that all permanent entries for Datacom servers are deleted from the repository when dynamic resource creation is permitted.

Default Resource Settings (z/OS only)

Use the Resource Management option (option 3) available from the Administration control group of the TIBCO Object Service Broker Administration Menu. You need to use the Resource Details (PF5) and the Resource Schedules (PF10) screens to specify the connection attributes. To get to the Resource Detail screen you must first specify a type (SERVERTYPE) and group (SERVERID) on the Resource Type screen.

The following table illustrates the attributes for SERVERID=DEFAULT.

Resource Details				Resource Schedule
Intermediate Rollbk	Early Release	Last User Reuse	Commit Level	Online Only
Y	Y	N	0	N

If the FSLEVEL startup parameter equals 1 the COMMIT LEVEL value on the Resource Detail screen must also be set to 1.

See Also TIBCO Object Service Broker *for z/OS Installation and Operations* for more information on defining and managing resources, and the Administration Menu.

Customization of the Startup Batch JCL

The CA Datacom load libraries must be part of the STEPLIB. Both the CA Datacom and TIBCO Object Service Broker load libraries should be authorized. A sample JCL for starting the server is distributed with TIBCO Object Service Broker as member DCOMSJCL in the JCL data set.

Startup of the Server

To start the server, bring up a Native Execution Environment using the startup parameters described in Specifying the Startup Parameters on page 119.

When you start an instance of a server, you are connected to both CA Datacom and TIBCO Object Service Broker. When the Execution Environment requests access to CA Datacom data, a thread to CA Datacom is established.

See Also

- TIBCO Object Service Broker *Messages with Identifiers* for information on messages produced by the server
- TIBCO Object Service Broker *Parameters* for information on Execution Environment parameters

Dynamic Startup

If the number of server instances is insufficient to process transaction requests, unsatisfied requests receive a SERVERBUSY signal. You can dynamically increase the number of server instances without restarting the Execution Environment by using the **MODIFY** operator command from the z/OS operator console.

The format of the **MODIFY** command is:

MODIFY *ee_jobname*,**STARTNUMSERVER=***nn*,**TYPE=DAT**

ee_jobname	The name of the batch job under which the Execution Environment is running.
nn	The number of new server instances to start. Can be from one to a value less than or equal to the value set in the **Maximum Connection Count** field in your network configuration.
DAT	Type of server to start.

Maximum Number of Server Instances

You can also use the following **MODIFY** command to dynamically set the maximum number of server instances available in a particular Execution Environment. The command format is:

MODIFY *ee_jobname,SETNUMSERVER=nn* **,TYPE=DAT**

ee_jobname	The name of the batch job under which the Execution Environment is running.
nn	The maximum number of server instances available for this particular Execution Environment. This number can be from one to a value less than or equal to the value set in the **Maximum Connection Count** field in your network configuration.
DAT	Type of server instances to be made available.

See Also TIBCO Object Service Broker *for z/OS Installing and Operating* for more information on increasing the number of server instances.

Shutdown of the Server

This section describes how to shut down the server.

Shutdown of a Single Server

If the server and the Native Execution Environment where it is running must be shut down, shut down the server first, followed by the Native Execution Environment. For more information, refer to Shutdown of the Native Execution Environment on page 103.

You can shut down a server in one of two ways:

- Using the z/OS **MODIFY** command
- Using the RESOURCE MANAGEMENT option from the Administration menu (S6BTLADM utility) (z/OS only)

The MODIFY Command

The format of the **MODIFY** command is:

MODIFY *dob_jobname*,**STOPSERVER=***idprefix*

dob_jobname	Name of the batch job under which the Data Object Broker is running.
idprefix	The unique name of the server. The server creates this name by appending a three-digit number to the IDPREFIX parameter specified in the server startup JCL. The default is DAT. To view the unique name assigned to existing servers, select the RESOURCE MANAGEMENT option from the Administration menu.

The RESOURCE MANAGEMENT Option

Use the RESOURCE MANAGEMENT option from the Administration menu to shut down either one server or a group of instances of the server.

Shutdown of the Native Execution Environment

To shut down a Native Execution Environment, ensure that all users are logged out and then issue one of the following commands from the z/OS operator console:

- **P** *ee_jobname*
- **MODIFY** *ee_jobname*,**Shutdown**

where

P	z/OS operator command (**Stop**).
MODIFY	z/OS operator command (can be abbreviated to **F**).
ee_jobname	The name of the batch job or started task used to start the Native Execution Environment.

Closing and Opening of URTs

Specific URTs can be closed and re-opened via the console **MODIFY** command. The command syntax is:

MODIFY *dob_jobname*,**SERVERCOMMAND=***servertype,serverid,command*

Example 1

```
MODIFY PRODDOB,SERVERCOMMAND=DAT,DATACOM1,CLOSEURT=URT5000
```

This command instructs all the DAT server instances with *serverid* DATACOM1 to issue a CLOSE for URT5000. Access to URT5000 is blocked until an OPENURT=URT5000 is issued. If a server has a transaction active and using URT5000, the Data Object Broker completes the transaction before sending the command to the server.

Example 2

```
F PRODDOB,SERVERCOMMAND=DAT,DATACOM1,OPENURT=URT5000
```

This command instructs all the DAT server instances with *serverid* DATACOM1 group to mark URT5000 as available so it can be opened on the next database access.

Example 3

```
F PRODDOB,SERVERCOMMAND=DAT,*,URTSTATUS
```

This command instructs all the DAT server instances to display the status of all the URTs in each individual instance.

Shutdown of a Group of Instances of the Server

You can also use one of four variations of the **MODIFY** operator command to shut down a group of instances of the server:

- Shut down all the DAT server instances:

  MODIFY *dob_jobname*,**STOPSERVER=ALLDAT**

- Shut down all the server instances with a common IDPREFIX:

  MODIFY *dob_jobname*,**STOPSERVER=***idprefix**

- Shut down all the server instances with a common SERVERID:

  MODIFY *dob_jobname*,**STOPSERVER=SRVID***serverid*

- Shut down one or more DAT server instances without shutting down the Execution Environment:

  MODIFY *ee_jobname*,**STOPNUMSERVER=***nn*,**TYPE=DAT**

where

dob_jobname	The name of the batch job under which the Data Object Broker is running.
ee_jobname	The name of the job under which the Execution Environment is running.
nn	The number of server instances to stop. This number can be from one to a value less than or equal to the value set in the **Maximum Connection Count** field in your Resource Management network configuration. For more information, refer to TIBCO Object Service Broker *for z/OS Installing and Operating*.
DAT	The type of server to stop.

See Also TIBCO Object Service Broker *for z/OS Installing and Operating* for information on:

- Shutting down various types of Execution Environments
- The Administration menu RESOURCE MANAGEMENT option

Connection to a Windows or Solaris Data Object Broker

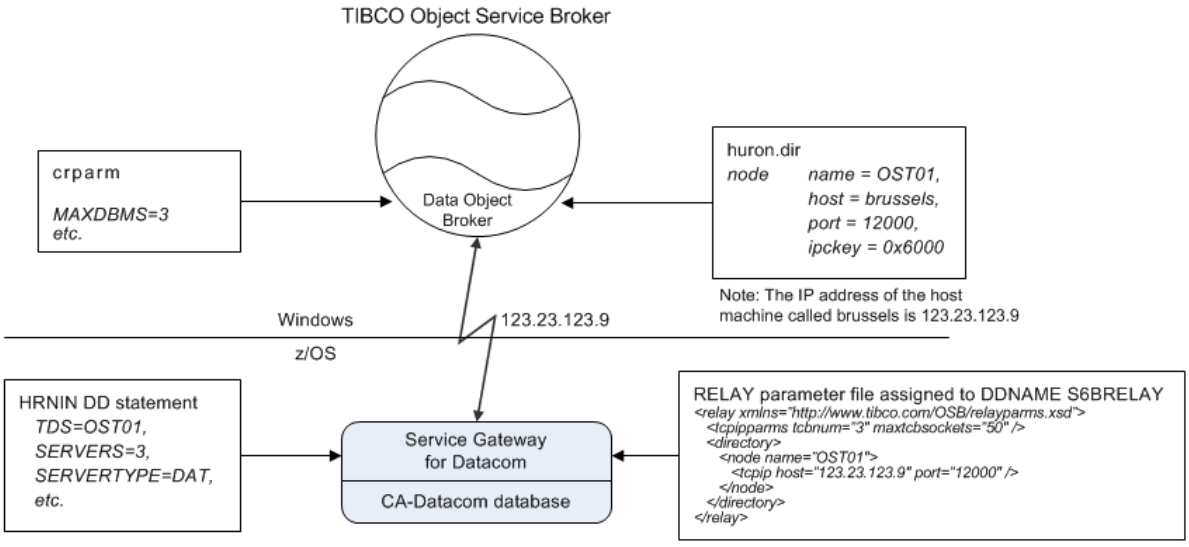
You can configure the Data Object Broker and the server to reside on different domains and operating systems (z/OS, Windows, or Solaris). The server must be in the same domain as the CA Datacom database system.

The following configuration steps are required to access a Data Object Broker from a different operating environment than your server:

- Configure the TCP/IP connection on the z/OS system where your server and CA Datacom database reside. **Note:** You must code the `keepalive` attribute as part of the description of your TIBCO Object Service Broker node.
- Configure the TCP/IP connection on the machine where your TIBCO Object Service Broker for Open Systems resides.
- Specify the number of servers that can connect to the Data Object Broker.
- Specify the appropriate server parameters.
- If the code page of the data in your CA Datacom database is different than IBM-037, refer to National Language Support for External Database Servers in the *TIBCO Object Service Broker National Language Support* manual.

Sample Configuration

The following diagram shows a sample configuration.



Configuration of the TCP/IP Connection on z/OS

Prepare the TIBCO Object Service Broker relay file–RELAYCFG member in the CNTL data set. This file associates the TIBCO Object Service Broker communications identifier with the TCP/IP application addressing information.

Following is an example of the input:

```
<relay xmlns="http://www.tibco.com/OSB/relayparms.xsd">
    <tcpipparms tcbnum="3" maxtcbsockets="50" />
    <directory>
     <node name="OST01">
      <tcpip host="123.23.123.9" port="12000" keepalive="600" />
     </node>
    </directory>
</relay>
```

The element and attribute names in the relay file are case sensitive.

See Also

- TIBCO Object Service Broker *for z/OS Installation and Operations* for detailed information about preparing the TIBCO Object Service Broker relay file.
- TIBCO Object Service Broker *Parameters* for details about the parameters and how to specify them.

Configuration of the TIBCO Object Service Broker TCP/IP Environment

Add the following parameters for the TCP/IP connection to the Data Object Broker directory file, huron.dir:

name	This must be the same value as the node name set in the relay file described in Configuration of the TCP/IP Connection on z/OS on page 107.
host	The name of the host machine where the TIBCO Object Service Broker monitor process listens for connections.
port	The number of the TIBCO Object Service Broker monitor socket port.

Number of Server Instances Connecting to the Data Object Broker

Specify the MAXDBMS parameter in the crparm file for your Data Object Broker:

MAXDBMS	This must be equal to or greater than the value specified in the SERVERS= parameter HRNIN DD statement for startup.

Server Parameter

Specify the TDS parameter in the HRNIN DD statement for startup. Refer to Specifying the Startup Parameters on page 119 for more details about specifying server parameters.

TDS	This must be the same value as the node name set in the relay file described in Configuration of the TCP/IP Connection on z/OS on page 107.

See Also TIBCO Object Service Broker *Installation and Operations* for the procedures on how to prepare the TIBCO Object Service Broker relay file and how to specify the parameters.

Chapter 7 Operational Requirements for CA Datacom Access

This chapter provides information about the operational requirements to access CA Datacom data from TIBCO Object Service Broker.

Topics

Extracting CA Datacom Table Information

Use the TIBCO Object Service Broker tool DATACOM to extract metadata from CA Datacom Data Dictionary and generate TIBCO Object Service Broker DAT definitions. Note that the extractor only handles one-element CA Datacom tables and cannot generate multiple-element DAT table definitions.

Use TIBCO Object Service Broker tools to create and manipulate the TIBCO Object Service Broker DAT table definition, as follows:

- The screen-based definer, available as part of the 3270 Workbench, enables you to generate TIBCO Object Service Broker table definitions of type DAT on the basis of the metadata extracted from CA Datacom Data Dictionary. You can inspect and modify definitions, including multiple-element ones, through the screen-based definer.
- The UI DAT table editor, available as part of the TIBCO Object Service Broker UI, enables the creation and manipulation of DAT table definitions, including multiple-element ones, on the basis of a COBOL copybook available as a text file at definition time. For details, refer to TIBCO Object Service Broker UI Online Help.

You can use the Workbench table definer and the UI table editor against a definition any number of times in any order.

Before defining TIBCO Object Service Broker DAT tables through the screen-based definer, you must extract the CA Datacom metadata to make it available for the definition process.

The extracted CA Datacom table definition information is stored in a TIBCO Object Service Broker TDS table. Because the data is static, you must re-extract the table information whenever changes are made to new or existing CA Datacom table definitions.

Prerequisites

Before you can extract CA Datacom information, the CA Datacom Multi-User Facility must be running, but the server does not need to be.

Only those CA Datacom tables that include an element spanning the entire row can be processed by the extractor.

To run the extractor as a job in a separate address space, first do the following:

1. Customize the contents of the @SCHEDULEMODEL(MVS,DCOMDEXT) table instance (the JOB name and the names of the libraries in the STEPLIB concatenation).
2. Ensure that you are running your TIBCO Object Service Broker Workbench session in a TSO Execution Environment.

At install time, the @SCHEDULEMODEL(MVS,DCOMDEXT) and @SCHEDULESAMPLE(MVS,@DCOMDEXT) table instances contain identical data. After modifying the former, you can use the latter for reference. If maintenance needs to be applied in the future, TIBCO Object Service Broker will deliver the changes to @SCHEDULESAMPLE(MVS,@DCOMDEXT) only.

To run the extractor in-process, ensure that the STEPLIB library used by the Execution Environment includes the CA Datacom LOAD library (CAAXLOAD or CABDLOAD) and the CA Datacom URT library (CUSLIB) in its concatenation.

Extraction of Table Information

To extract the table information:

1. Run the DATACOM tool with the execute rule option on the workbench.

   A Table Management Facility screen appears similar to the following:

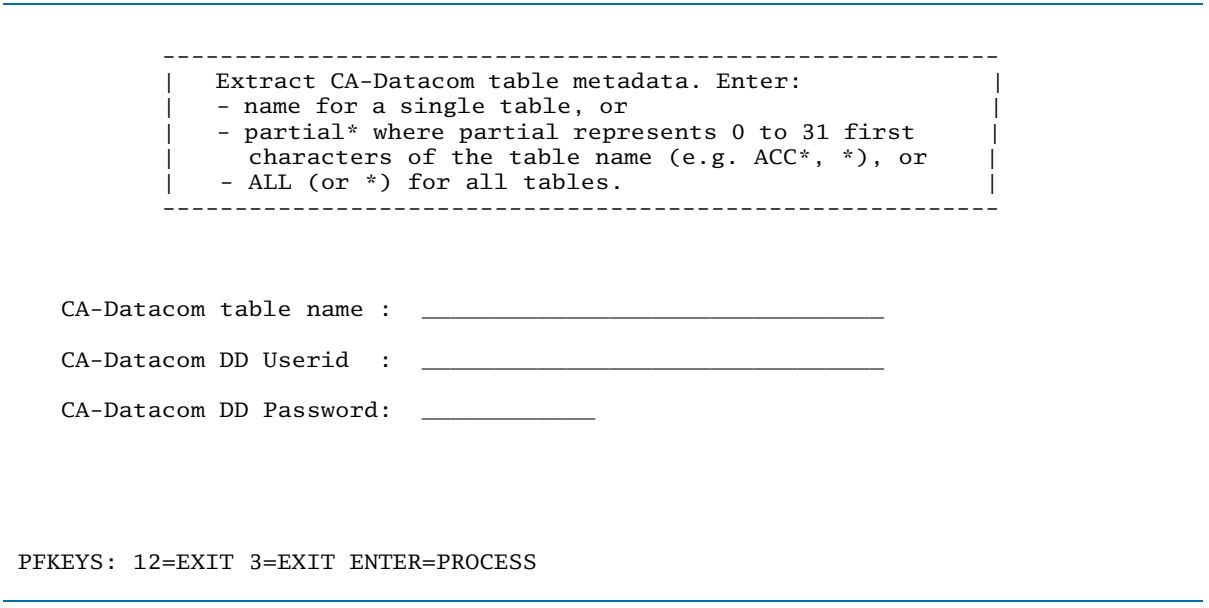
```
 TIBCO OSB Facility for DAT-type Table Definition Management


_ Manage TIBCO OSB DAT-type table definitions
_ Extract CA-Datacom metadata


PFKEYS: 2=LOGS 3=EXIT 12=EXIT
```

2. On the displayed screen, position your cursor beside the Extract CA-Datacom metadata menu option and press Enter.

An Extract screen appears similar to the example shown:



3. In the CA-Datacom table name field, type one of the following:
   - The dictionary name of a CA Datacom table
   - ALL for all available CA Datacom tables
   - *partial** for all the tables whose name starts with *partialname*, where *partial* is 0 to 31 characters long
4. In the CA-Datacom DD Userid field, type a valid CA-Datacom Data Dictionary user ID.
5. If you do not specify a user ID, the default user ID specified during installation of CA Datacom is used.
6. In the CA-Datacom DD Password field, type the CA-Datacom Data Dictionary password that corresponds to the user ID entered in the previous field.

   If you do not specify a password, the default password specified during installation of CA Datacom applies.
7. Press Enter.

   The extractor is invoked and produces an extract report similar to the one below.

```
2006-10-02                          GENERATED DATACOM TABLES FROM DATADICTIONARY                          USERA

_______ACTION TABLE_____________________________   _ID'S__ ELEMENT EXCEPTIONS_____________________________________
     SELECTED SYSAUTHOBJ                          00015 AOB   AOBRW DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSCOLAUTH                          00015 ACL   ACLRW DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSCONSTRDEP                        00015 CND   CNDEL DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
 NOT SELECTED SYSCONSTROBJ                        00015 CNO         NO ELEMENT SPANNING ENTIRE RECORD
     SELECTED SYSCONSTRSRC                        00015 CNS   CNSEL DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSDBAUTH                           00015 ADB   ADBRW DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSPLANAUTH                         00015 SPA   SQLEL DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSPRIVDEP                          00015 SPD   SQLEL DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
     SELECTED SYSTABAUTH                          00015 ATB   ATBRW DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
 NOT SELECTED SYSTEM                              00002 SYS         NO ELEMENT SPANNING ENTIRE RECORD
     SELECTED SYSVIEWDEP                          00015 VWD   VWDEL DUPLICATE MASTER KEY -  OPTIMIZED GENERATED
 NOT SELECTED TEXT                                00002 TXT         NO ELEMENT SPANNING ENTIRE RECORD
 NOT SELECTED TRIGGER                             00002 TRG         NO ELEMENT SPANNING ENTIRE RECORD
 NOT SELECTED TTM-TABLE                           00017 TTM         NO ELEMENT SPANNING ENTIRE RECORD
```

If you are running the extraction in a TSO session, the @SCHEDULEMODEL(MVS,DCOMDEXT) table instance, which you must customize for your site, spawns a separate job and thus determines where the extract report is sent. Otherwise, a child transaction is executed and the resulting log is available to you via PF2 upon termination.

Extract Report

The extract report shows the following information on the CA Datacom tables whose metadata has been extracted:

- The names of selected tables whose definitions are now stored within TIBCO Object Service Broker
- The names of the tables not selected and why they were not selected (for example, the CA Datacom table does not have an element name spanning the entire row)
- The names of the tables that enable duplication of the master key

Binding DAT Table Definitions

You can bind a DAT table definition, but *not* its data. Refer to Task C: Add Control Information on page 147 for information on binding. DAT tables for which you request binding are bound to both the Execution Environment and the server when they are accessed from a rule.

Rebinding

If you change a definition, it is automatically rebound in the server.

Understanding Space Requirements

You can specify the maximum amount of space available to hold all DAT table definitions by using the POOLSIZE server startup parameter. Refer to Specifying the Startup Parameters on page 119 for more information.

See Also TIBCO Object Service Broker *Application Administration* for more information on binding tables.

Implementing Security

Security can be implemented at both the CA Datacom and TIBCO Object Service Broker levels of access:

CA Datacom	• If the server Startup Parameter SECLEVEL is 0, the server passes information on the user in the User Information Block (UIB). The user ID is in positions 4 through 11 of the UIB. If you specify the X option in the DEBUGPARMS field in the @SERVERDEBUG(DAT) table, the literal PXX is inserted in positions 1 through 3 of the UIB to denote that additional statistics are to be collected. CA Datacom security can make use of those statistics. For details, see Specifying the Startup Parameters on page 119. • If the server Startup Parameter SECLEVEL is 1, the CA Datacom security is handled as follows: — Users of a rules-based application that will access Datacom tables must invoke the rule SET_DAT_SEC(*userid*, *password*) from their sessions, setting the user credentials to be passed to the server. — When first allocated to a particular transaction, the server instance accepts those credentials and creates or modifies a respective ACEE. All the subsequent CA Datacom calls within the boundaries of the current transaction are carried out on behalf of the user identified by those credentials. — As long as the caller does not invoke rule SET_DAT_SEC again, any server instance allocated to this caller uses the same user credentials when communicating with CA Datacom. — It is only at the transaction boundary that the server accepts new user credentials set by the caller by invoking rule SET_DAT_SEC.
TIBCO Object Service Broker	To restrict access to the DAT tables after they are defined, proceed with security as for any TIBCO Object Service Broker table. For details, see the TIBCO Object Service Broker *Managing Security* manual.

Adding URT Names

You can define multiple URT names for use in a server group identified by a server ID. These URTs are loaded into memory when the server is first initialized. You can also add a new URT name to a server without recycling it.

Populating the URT Table

In the @DATACOM_URTS TDS table, parameterized by a server ID, define the URT names for use in a server associated with this server ID. Populate the table with all possible CA Datacom DBIDs.

A URT name can be associated with one or more DBIDs; all URT modules named in the table must reside in the libraries referenced in either the STEPLIB or HRNEXTR DD statement. Refer to Closing and Opening of URTs on page 104.

If you need to associate a new URT name to a server, add the new entry to the @DATACOM_URTS(*serverid*) table. The new entry is automatically picked up by the server, which does not require recycling.

Example TDS Table

```
EDITING TABLE    :
@DATACOM_URTS (DATACOM1)

      DBID       URTNAME
-------------  ----------
            10  URT010
            11  URT010
          5000  URT5000
```

Determining the URT to Use

At startup, each server instance uses the multiple-URT method, that is, it attempts to load the URTs referred to in the @DATACOM_URTS(*serverid*) table. If the latter is empty, the server uses the single-URT method. When the single-URT method is activated, the server cannot use the multiple-URT method until the next session. When loading a URT, the server searches the STEPLIB concatenation of load libraries first and then the load library denoted by the HRNEXTR DD statement.

URT in the Table Definition Method

When a table is accessed that has a URT name associated with it, that URT will be loaded into the server address space and marked as open.

Example 1

```
+DATACOM1 Starting CA-DATACOM Server DCOM001
+DATACOM1 Starting CA-DATACOM Server DCOM001
+DATACOM1 Starting CA-DATACOM Server DCOM001
+DATACOM1 DCOM002 URT name URT010 successfully loaded
+DATACOM1 DCOM001 URT name URT010 successfully loaded
+DATACOM1 DCOM002 URT name URT5000 successfully loaded
+DATACOM1 DCOM000 URT name URT010 successfully loaded
+DATACOM1 DCOM001 URT name URT5000 successfully loaded
+DATACOM1 DCOM000 URT name URT5000 successfully loaded
+DATACOM1 DCOM001 URT=URT010 STATUS=AVAILABLE
+DATACOM1 DCOM000 URT=URT5000 STATUS=AVAILABLE
+DATACOM1 DCOM001 URT=URT5000 STATUS=AVAILABLE
+DATACOM1 DCOM000 URT=URT010 STATUS=AVAILABLE
+DATACOM1 DCOM002 URT=URT5000 STATUS=AVAILABLE
```

At the first database call, the server loads or locates among the already loaded URTs the corresponding URT and uses the same URT for the duration of the transaction. A CA Datacom OPEN is issued and the URT is marked as open and remains open, until it is specifically closed or the server is shut down.

Example 2

The URT5000 in server USR01002 is opened.

```
+DATACOM1 DCOM002 URT=URT010 STATUS=AVAILABLE
+DATACOM1 DCOM002 URT=URT5000 STATUS=OPEN
+DATACOM1 DCOM001 URT=URT010 STATUS=AVAILABLE
+DATACOM1 DCOM000 URT=URT5000 STATUS=AVAILABLE
+DATACOM1 DCOM001 URT=URT5000 STATUS=AVAILABLE
```

Multiple-URT Method

When the server starts up, all the URT names specified in the DATACOM_URTS(*serverid*) table are loaded into the server address space and marked as available. This indicates that the URT is loaded and left unopened.

Single-URT Method

Look for a URT named after the server ID. If one does not exist, look for a URT named S6BDATDB.

Specifying the Startup Parameters

You can specify parameters on the EXEC statement of the startup JCL, in a data set, or both.

Available Parameters

The available parameters are as follows:

IDPREFIX	**Required.** The server appends three decimal digits to this prefix and uses the result, which uniquely identifies each instance of the server, to log in to TIBCO Object Service Broker. An IDPREFIX can have up to, or will be truncated down to, five characters.
SERVERID	**Required.** This parameter identifies a pool of server instances with common characteristics. If you run multiple address spaces to start up servers with the same SERVERID, ensure that the FSLEVEL and TRXDB parameters are assigned the same values in each of these jobs. A SERVERID can have up to eight characters and you *must* specify one when you are defining a DAT table. The SERVERID specified in a DAT table definition allocates a suitable server instance to your transaction when accessing the CA Datacom data with this table definition. The SERVERID parameter can be overridden at runtime. Refer to Dynamically Changing the Parameters on page 123 for more information.
SERVERS	**Required.** The number of server instances to be attached to the server address space at startup. This number can be from one to a value less than or equal to the value set in the **`Maximum Connection Count`** field in your network configuration. The default value is 1.
SERVERTYPE	**Required.** The type of server to be initialized. For CA Datacom the value is DAT and it must be specified.
TDS	**Required.** The Communications Identifier of the Data Object Broker for the server to attach to.

FSLEVEL (FSL)	This parameter specifies the level of Fail Safe processing. The default value is 0. The valid values are as follows: **1** – Activate Fail Safe processing. The server informs the Data Object Broker that it can support Fail Safe level-1 processing. If the server is to attach to a z/OS Data Object Broker, the Data Object Broker's connection attribute setting "commit level" must be set to 1. If not, the server connection is rejected. Refer to Implementing Fail Safe Processing on page 126 for more information. You must specify the TRXDB parameter. **0** – Deactivate Fail Safe processing. The server informs the Data Object Broker that it does not support Fail Safe level-1 processing. If the server is to attach to a z/OS Data Object Broker, the Data Object Broker's connection attribute setting "commit level" must be set to 0. If not, the server connection is rejected. Refer to Implementing Fail Safe Processing on page 126 for more information.
POOLSIZE	This parameter sets the amount of space (in kilobytes) to hold DAT table definitions in the server. This parameter can be 4 KB through16384 KB. The default value is 256 KB. An estimate of the number of DAT tables that can be accessed in a single transaction is POOLSIZE divided by CTABLESIZE. Refer to Estimating the CTABLESIZE Parameter on page 121 for more information.
SECLEVEL	The valid values are 0 (default) and 1. If SECLEVEL=0 or 1, the server passes to CA Datacom the caller's TIBCO Object Service Broker user ID in positions 4 through 11 of the User Information Block (UIB). If SECLEVEL=1, the server "impersonates" a CA Datacom client, as follows: 1. The TIBCO Object Service Broker caller passes to the server the credentials (user ID and password) of the CA Datacom client. 2. The server verifies those credentials with the operating system, which either rejects them or creates an ACEE (system control block representing a user). 3. The server associates the ACEE with its own task control block and invokes CA Datacom to process the caller's request. 4. CA Datacom checks the ACEE-represented client's rights with respect to the resources required for the current request. If any of the above actions fails, the server raises the SECURITYFAIL exception.

SCOPE	The valid values are SESSION (default) and TRANSACTION. If SCOPE=TRANSACTION, all the URTs opened by a server instance during a user's transaction are closed at transaction end. Otherwise, all those open URTs stay open until explicitly closed by MODIFY … SERVERCOMMAND (see Closing and Opening of URTs on page 104) or until the end of the session. If CA Datacom MUF's number of tasks is limited, consider specifying SCOPE=TRANSACTION to release a task as quickly as possible. However, if you repeatedly execute the same transaction in the server, specifying SCOPE=SESSION might result in optimal performance.
TRXDB	This parameter is required only if FSLEVEL=1. Specify the database number of the CA-Datacom transaction database used to log Fail Safe records. The default value is 888.

Estimating the CTABLESIZE Parameter

When you select CA Datacom fields as TIBCO Object Service Broker DAT fields, the number of fields that you can access using a DAT table definition depends on the CTABLESIZE Data Object Broker parameter. To estimate the number of bytes required to support a specified number of fields, execute the following shareable tool:

ESTIMATETBLDFN(*num_fields*)

You must supply one argument, which is the maximum number of fields accessed by a DAT table in your system. The tool returns an estimate of the maximum CTABLESIZE required (for each TIBCO Object Service Broker table type) to support this number of fields.

Result of Executing ESTIMATETBLDFN for 50 fields.

```
----------------------------- INFORMATION LOG --------------------------------
COMMAND ===>                                                  SCROLL ===> P
DATE: Nov 28,2006      REPORT ON ESTIMATE CTABLESIZE
                         FOR "50" FIELDS

Table Type       CTablesize(K)
----------       -------------

ADA                 5
DAT                 7
DB2                 5
IDM                 6
IMS                 6
MAP                 4
SLK                 4
```

```
TDS                 3

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

See Also

- TIBCO Object Service Broker *Parameters* for information about parameters.
- TIBCO Object Service Broker *for z/OS Installing and Operating* for more information about network connections.

Dynamically Changing the Parameters

You can change the parameters dynamically, as described in this section.

Table Type Attributes

When a table is defined, attributes specific to external DBMS table types are held in the @SERVERPARMS control table, which is parameterized by table name. Regarding the external environment, each occurrence in the table specifies a value (such as SERVERID and SERVERTYPE) for the table.

Following is a sample @SERVERPARMS(PAYROL) control table:

```
TABLE    :  @SERVERPARMS(PAYROL)
 COMMAND ==>
                                                                    SCROLL:
     NUMBER          NAME        TYPE  SYNTAX  LENGTH  DECIMAL  DEFAULT USAGE
 _ ----------- ---------------  -     -     ------ ------     -       -
 _           1 SERVERID          I     C          8       0      .       C
 _           2 SERVERTYPE        S     C          3       0      .       T
 _           3 DATACOMNAME       S     C          3       0      .       D
 _           4 DBID              Q     B          2       0      .       I
 _           5 ELMNAME           S     C          5       0      .       E
 _           6 OPTIMIZE          L     C          1       0      .       O
 _           7 LENGTH            Q     B          2       0      .       L
 _           8 NAME              S     C         32       0      .       N
 _           9 VERSION           Q     V          8       0      .       V
 _          10 URTNAME           S     C          8       0      .       U

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

Use the SETXPARM and RESETXPARM tools to modify the SERVERID. Modifying this parameter reduces the number of table definitions required to define the external data. The changes are stored in either of the following tables:

@SRVRPRMS_TYP	Manages global changes to the table type.
@SRVRPRMS_TBL	Manages specific changes to an individual table.

The changes are in effect for the duration of the session until SETXPARM is invoked again or when the overrides are reset.

Examples Using SETXPARM and RESETXPARM

The following example sets the SERVERID for all DAT tables to TORONTO:

```
CALL SETXPARM('TABLETYPE', 'DAT', 'SERVERID', 'TORONTO', '');
```

The following example resets the SERVERID for DAT tables to the Table Definer default value:

```
CALL RESETXPARM ('TABLETYPE', 'DAT', 'SERVERID', '');
```

See Also TIBCO Object Service Broker *Shareable Tools* for detailed descriptions of the SETXPARM and RESETXPARM tools.

Adding Server Instances

The number of instances of the server attached to the server address space is specified in the server startup JCL. If you require additional instances of the server, do one of the following:

- Shut down the server and start it again with an increased number of instances, using the SERVERS startup parameter.
- Start another instance of the server with the same SERVERID but with a different IDPREFIX.
- Use the z/OS **`MODIFY`** command to dynamically add the server to an existing server Execution Environment. For more information, refer to Dynamic Startup on page 101.

Implementing Fail Safe Processing

Fail Safe level 1 processing guarantees consistency when you update both TIBCO Object Service Broker TDS and CA Datacom data from a single instance of the server in the same transaction.

Transaction Processing

At the end of a transaction, the Data Object Broker requests that the server commit any outstanding updates. As part of the CA Datacom commit processing, the server updates a CA Datacom transaction database to record the fact that the commit was successful. If the server does not respond to the Data Object Broker in a reasonable amount of time, the transaction is flagged as being in doubt. Locks held on TDS data remain in place until the problem is resolved.

When a connection is re-established between the Data Object Broker and an instance of the server with the same configuration as the one that failed, the Data Object Broker asks the server if the in-doubt transaction completed. The server checks the CA Datacom transaction database to determine this. If the update was completed in CA Datacom, the TDS updates are applied and the locks are released.

You can resolve in-doubt transactions only by starting an instance of the server with parameter settings that are exactly the same as the server in use at the time the transaction is placed in doubt.

Implementation Procedure

For each CA Datacom environment with at least one instance of the server running with Fail Safe processing, complete the following tasks:

1. Update CA Datacom Data Dictionary on page 127
2. Promote the CA Datacom Data Dictionary update on page 127
3. Update the CA Datacom CXX data set on page 127
4. Allocate and initialize the CA Datacom transaction database on page 127

These tasks are described in detail in the following sections.

Task A Update CA Datacom Data Dictionary

Member DCOMUDD in the JCL data set contains the JCL to update CA Datacom Data Dictionary with TIBCO Object Service Broker Fail Safe definitions. It reads member CNTL(XDCOMBTG), which contains data dictionary and database specifications. Complete the following step:

1. Submit DCOMUDD.

   It should end with RC=0.

Task B Promote the CA Datacom Data Dictionary update

Member DCOMUDD1 in the JCL data set contains the JCL to promote into production the CA Datacom Data Dictionary update performed by step DCOMUDD. Complete the following step:

1. Submit DCOMUDD1.

   It should end with RC=0.

Task C Update the CA Datacom CXX data set

You should backup your CXX data set before performing this update.

Member DCOMCXX in the JCL data set contains the JCL to update the CA Datacom CXX data set with TIBCO Object Service Broker Fail Safe definitions.

Complete the following steps:

1. Submit DCOMCXX.

   It should end with RC=0.

Task D Allocate and initialize the CA Datacom transaction database

To allocate and initialize the CA Datacom transaction database in the CA Datacom environment, you must run the appropriate CA Datacom utilities. Ensure that the transaction table is also referenced in the URT for the server.

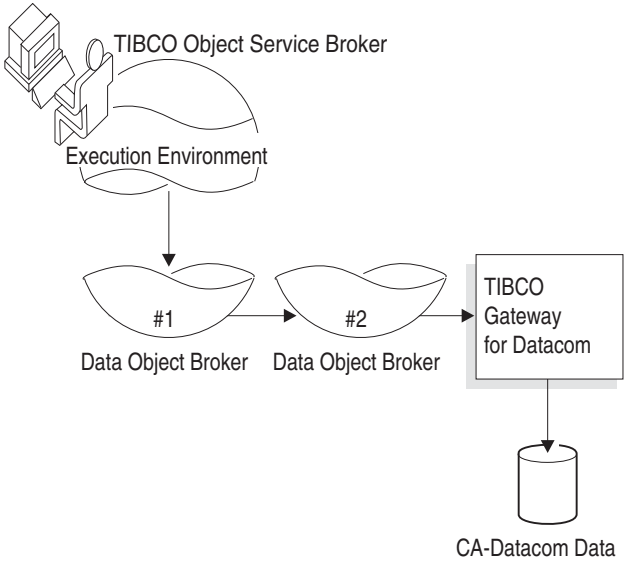
See Also TIBCO Object Service Broker *for z/OS Managing Backup and Recovery* for more information on Fail Safe processing

Performing Other Operational Procedures

This section describes other operational procedures.

Using Distributed Data with the Server

Distributed access between TIBCO Object Service Broker and CA Datacom is permitted subject to requirements of all distributed access. This illustration shows a sample TIBCO Object Service Broker CA Datacom distributed-data scenario:



Displaying the Status of the Server

Use the RESOURCE MANAGEMENT option from the Administration menu (S6BTLADM utility) (z/OS only) to display the status of your server. The sample screen below shows an example of the type of information displayed for resource type DAT.

Resource Detail

```
S6BADM33  DCOMSRV      RESOURCE DETAIL FOR DAT DEFAULT       2007MAR20 21:59:38

INTERMEDIATE ROLLBK  Y EARLY RELEASE    Y LAST USER REUSE    N COMMIT LEVEL  0
RETRY INTERVAL       0 TP NAME            USER ID PREFIX       FAILURES    0
NODE  MMMSYSTEMA01     INDOUBTS          N MONITOR/SMF        Y

                      CONNECTIONS      IN-USE        TRX          MESSAGE
                     CUR  MAX  LMT   CUR  MAX       COUNT          COUNT
          ONLINE       0    0    0     0    0          0              0
          COMMON       0    0  100     0    0          0              0

SCHEDULE NAME DEFAULT

        APPLICABLE DAYS          EXCEPTION  START  ONLINE  CONNECTIONS
   MON TUE WED THR FRI SAT SUN   DATE        TIME   ONLY       MAX
    Y   Y   Y   Y   Y   Y                   00:00   N         100




 ENTER-PATHS PF2-TYPE PF4-GROUP PF5-PEER PF9-START PF10-SCHEDULES PF11-UPDATE
 NO MORE DATA TO BE DISPLAYED
```

Debugging Rules and Applications

Using the TIBCO Object Service Broker Rule Debugger you can identify and fix errors within your application. You can also make and test certain ad hoc changes to your rules. The Debugger stops the rules execution at events that you specify in the Debug screen.

CBS Trace

To turn on the CBS Trace in CA Datacom, include the X option in the **DEBUGPARMS** field in the @SERVERDEBUG(DAT) table. SERVERID is the primary key value for this table. This records runtime information in the CA Datacom PXX.

SERVERID Parameter

To view only trace information associated with your rules, you require a dedicated server. This can be accomplished using the SERVERID parameter. To change the server ID of a DAT table definition to an alternate dedicated server ID, execute the CHANGE_SERVERID tool as follows:

```
CHANGE_SERVERID(table_name,old_serverid,new_serverid)
```

You can view the information included in the PXX Trace using the standard CA Datacom PXX reporting utility.

See Also

- Refer to TIBCO Object Service Broker *Parameters* for information about parameters.
- TIBCO Object Service Broker *Application Administration* and TIBCO Object Service Broker *for z/OS Installing and Operating* for more information on distributed access, and on the RESOURCE MANAGEMENT option of the Administration menu.
- TIBCO Object Service Broker *Programming in Rules* for more information on the Rule Debugger.

Debugging Server Problems

To show CA Datacom errors as they are returned from CA Datacom or problems processing the requests in the server, add one or more of the following values in the field **DEBUGPARMS** in the @SERVERDEBUG(DAT) table. SERVERID is the primary key value for this table.

A	Prints resource usage statistics at the end of the transaction (for example, CPU time or table access calls).
C	Logs every CA Datacom return code in the trace. Use this value only for support activities.
D	Dumps CA Datacom specific work areas such as area, work area, request qualification area, element list, and user information when an unexpected CA Datacom return code is encountered. You should always specify this value. Contact your CA Datacom database administrator for interpretation.
E	Logs non-CA Datacom errors when the server cannot convert data. You should always specify this value.
F	Causes the server to close all its tables at each synchronization point, while the server continues to run. Use of this option can cause significant server wait time as CA Datacom opens and closes the files. If the files are still in use by other users, the wait time is minimal.
I	Logs the messages received by the server. Use only for support activities.
O	Logs the messages sent by the server. Use only for support activities.

P	Records trace information to SYSOUT. If not specified, the trace information is recorded in the @DATTRACE TIBCO Object Service Broker table.
T	Logs the requests received by the server.
X	Turns on CBS trace in CA Datacom. Reports can then be produced analyzing CA Datacom access path usage. The first three characters of the User Information Block are set to $$$. This causes additional recording of CBS runtime information in the CA Datacom PXX.

Reporting Problems

Refer to How to Contact TIBCO Support on page xx for information about reporting problems with the server to TIBCO Support.

Have the following information available when reporting server related problems to TIBCO Support:

- DAT table definition(s) and sample data.
- Listings of all the TIBCO Object Service Broker Datacom control tables.
- Server job log and control cards.
- Output from the server trace (if applicable).

Chapter 8 Defining the Accesses to CA Datacom

This chapter describes how to define the TIBCO Object Service Broker DAT tables for accessing the CA Datacom data.

Topics

Overview

To access CA Datacom data from TIBCO Object Service Broker, you must define a TIBCO Object Service Broker table of type DAT. A DAT table can have one or more CA Datacom fields as fields, up to 16 CA Datacom fields as the composite primary key, and an optional location parameter.

To define a DAT table, complete the following tasks

1. Task A: Define a DAT Table on page 135
2. Task B: Define Fields for the DAT Table on page 140
3. Task C: Add Control Information on page 147
4. Task D: Document the DAT Table on page 152

Task A: Define a DAT Table

There are three methods of defining a DAT table:

- Use the Workbench Table Definer.
- Use the TIBCO Object Service Broker UI.
- Use the DATACOM tool to generate DAT table definitions.

An instance of the server does not have to be running to define a DAT table, as the information that you extract in Extraction of Table Information on page 111 is stored in a TIBCO Object Service Broker TDS table.

Invoking the Table Definer

Invoke the Table Definer from the workbench using the DT define table option or the primary command field. You can modify an existing definition or define a new DAT table. Use the DATACOM tool to specify the initial values for your definition.

Accessing Existing Tables

To display the definition of an existing DAT table from the workbench, do one of the following:

- Type the name of an existing table beside the DT define table option and press Enter to display its definition.
- Type the name of an existing table in the primary command field, for example:

  DT PERSONNEL<Enter>.
- Move the cursor to the DT define table option and press Enter to display the Object Manager screen, which contains a list of existing tables.

  Scroll through this list to see which table you require. To select a table, type **S** beside the name and press Enter.

See Also TIBCO Object Service Broker *Getting Started* for more information on the Object Manager.

Defining a New Table

To define a new table, complete the following steps:

1. Type the name of a new DAT table beside the DT define table option or in the primary command field and press Enter.

   A TDS definition template appears.

2. Change the Type field at the top of the screen from TDS to DAT and press Enter.

   A Table Definition screen appears similar to the following:

```
 COMMAND==>                          TABLE DEFINITION

       Table: PERSONNELD       Type: DAT   Unit: USR40


 ServerId:
 URT Name:

 Dictionary Name:
  Cobol Copybook:
   DataCleansing:


Datacom Table Attributes
------------------------
           DBID:
  Internal Name:
  Duplicate Key:

   Element List:


 PFKEYS: 1=HELP 4=CORE 5=EXTENSIONS 13=PRINT 3=SAVE 22=DELETE 12=CANCEL
```

Fill in the fields, as follows:

Table	If the definition is generated using the DATACOM tool, this field displays the table name supplied by default; otherwise, the field is blank. The default is the name of the CA Datacom table converted to a valid TIBCO Object Service Broker name. Modify the default name as required. A valid entry is a character string up to 16 characters, beginning with a letter (A-Z) or a special character ($ or #), and continuing with more letters, special characters, digits (0-9), or underscore characters (_). A table name starting with an @ symbol denotes a table supplied by TIBCO Object Service Broker.
Type	The table type DAT that you changed in Defining a New Table on page 135.
Unit	The user unit associated with the table. To modify this field, use the Core screen. Refer to Task C: Add Control Information on page 147 for more information.
ServerId	The ID for the server or group of servers associated with the table. This ID must match the SERVERID parameter specified in the server JCL. Refer to Specifying the Startup Parameters on page 119 for more information.
URT Name	**Optional.** The URT name. Ensure that a load module with this name and default entry point DBNTRY is available to load for the server at runtime.
Dictionary Name	**Optional.** The Data Dictionary name of the CA Datacom table to be accessed. A valid entry is any valid CA Datacom table name. Press PF1 to access a list of the CA Datacom tables loaded into TIBCO Object Service Broker.
COBOL Copybook	The fully qualified name of the file that contains the respective COBOL copybook if you created this definition with the Eclipse UI-based definer. You cannot modify this field.
DataCleansing	**Optional.** The data-cleansing actions, if any, the server is to carry out. This field is optional.
DBID	**Optional.** The CA Datacom identification number for the CA Datacom database that is being used.
Internal Name	**Required.** The CA Datacom name that uniquely identifies the table within the CA Datacom database being used.

Duplicate Key	A value of Y if duplicate master keys are allowed for the CA Datacom table; a value of N if only a single master key is allowed for the CA Datacom table.
Element List	The CA Datacom element list used by the server at runtime when this table is accessed. You cannot modify this field.

The fields DBID, Internal Name, and Duplicate Key are filled in with default information after you have entered the value of the Dictionary Name field. Refer to Using the DATACOM Tool on page 138 for more information.

You can, however, manually enter or modify the DBID, Internal Name, and Duplicate Key fields. You cannot enter or modify the value of the Element List field.

Using the DATACOM Tool

Execute the DATACOM tool from the EX execute rule option on the workbench to display the CA Datacom table management facility screen illustrated below. Use this facility to generate a definition for your DAT table. See the screen in Extraction of Table Information on page 111.

Generating a Definition

To display a list of CA Datacom tables available for mapping as TIBCO Object Service Broker DAT-table definitions (that is, those CA Datacom tables whose metadata has previously been extracted) and the existing DAT definitions, follow these steps:

1. Position your cursor next to the “Manage TIBCO OSB DAT-type table definitions” option and press Enter.

   A list of tables appears.

2. Find the table you need, type **G** in the line command field, and press Enter.
3. In the SERVERID field (prompted), type the name of the server that you are using.
4. Scroll through the list to the name of the new table, type **S** in the line command field, and press Enter.

Using the Table Definer PF Keys

Use the following PF keys and their corresponding primary commands (or their abbreviations) from the Table Definition screen.

PF Key	Primary Command	Description
1	**HELP**	Displays Help for the Table Definer. Also displays a list of available CA Datacom table names if you position your cursor on the **DATACOM Name** field. You can select one table from this list.
2	**DOCUMENT**	Displays the documentation screen, where you can document the table definition. Refer to , Task D: Document the DAT Table, on page 152 for more information.
3	**SAVE**	Validates the definition information specified in the Table Definition, Core, and Extensions screens. If valid, it returns you to the workbench.
4	**CORE**	Displays the Core screen, where you define control information for your table.
5	**EXTENSIONS**	Displays the Extensions screen, where you select the Datacom fields that you want defined as fields in the DAT table.
12	**CANCEL**	Cancels the changes to the definition and returns you to the workbench.
13	**PRINT**	Prints the definition of the table. You remain in the Table Definer.
22	**DELETE**	Deletes the definition of the DAT table. You are prompted to confirm the deletion.

Task B: Define Fields for the DAT Table

A field definition consists of both the TIBCO Object Service Broker and CA Datacom definitions. The TIBCO Object Service Broker field definition is extended by the CA Datacom field definition. Use the Extensions screen to define the TIBCO Object Service Broker portion of the field definition.

Defining Fields

To define fields, press PF5 on the Table Definition screen. The following illustration shows the leftmost portion of the screen. Press PF11 to view the CA Datacom field information displayed to the right.

```
FIELDS INFORMATION:
 #   Name             Elmt Key  Syn Len Dec Xsyn Xlen Xdec Xoff Datacom/Copybook Name
--- ---------------- ----- -Typ-- ----- -- ---- ----- -- ----- ----------------------
1    NUMBER           EMDTA P     P     3  0    M     5  0     0 NUMBER
2    NAME                         V    24  0    V    24  0     5 NAME
3    STREET_ADDRESS               V    24  0    V    24  0    29 STREET-ADDRESS
4    CITY_ADDRESS                 V    15  0    V    15  0    53 CITY-ADDRESS
5    STATE_ADDRESS                V     2  0    V     2  0    68 STATE-ADDRESS
6    ZIP_CODE_LOC                 V     5  0    V     5  0    70 ZIP-CODE-LOC
7    SOCIAL_SECURITY              P     5  0    P     5  0    75 SOCIAL-SECURITY
```

- If you enter the Table Definer with the DATACOM tool, TIBCO Object Service Broker and CA Datacom field values are provided in this Extensions screen. Modify the TIBCO Object Service Broker field values as required; modifications cannot be made to the CA Datacom values.
- If you entered the Table Definer directly, values are not provided unless you are modifying an existing DAT table.

Editing Extensions Screen Fields

Fields in CA Datacom are not necessarily in the same sequence as in TIBCO Object Service Broker.

Either type in or modify values directly or press PF2 to display the Fields screen for the selection of fields. If you select the fields using the Fields screen, default values are provided for the definition. You can modify the following fields:

#	Specifies the position of the field in the table. You can re-order the fields by typing new values in this field or you can delete the field by blanking out the number.

Name	Specifies the field name. This name must be unique within the table. There is no limit as to the number of fields you can name, either using the Table Definer or generating the definition. The number of fields you can select depends on the length of the sum of all fields, primary keys, and parameters. This sum plus control information must be less than or equal to 31,744 bytes. A valid entry is a character string of up to 16 characters beginning with a letter (A - Z) or a special character ($ or #), and continuing with more letters, special characters, digits (0 - 9), or underscore characters (_), for example, DEPTNO.
Elmnt	Elements are field collections that CA Datacom rows consist of. Enter the five-character element name on this screen, once per element, in the line that corresponds to the field with the lowest row offset (Xoff) within that element. When you return to the table screen, the element list there will contain all the element names you have entered.
Key	The value that uniquely identifies each occurrence in the table. At least one field must be defined as a primary key field. The primary key can be a single field or a composite of up to 16 fields, with a maximum length of 127 bytes. For more information, see Selecting Fields on page 143. The valid entries are as follows: • P — Primary key. • Blank — Non-key field.
Type	Specifies the TIBCO Object Service Broker semantic data type of the CA Datacom field. The valid entries are as follows: • C — Count. • D — Date. • I — Identifier. • L — Logical. • Q — Quantity. • S — String. Refer to Translating Data Types on page 144 for information on the default mapping of Datacom data types to TIBCO Object Service Broker semantic types and syntax.
Syn	Specifies the TIBCO Object Service Broker syntax of the CA Datacom field. The valid entries are B, C, P, V, F, W, UN, and RD. Refer to Translating Data Types on page 144 for information on the default mapping of Datacom data types to TIBCO Object Service Broker semantic types and syntax.

Len	Specifies the TIBCO Object Service Broker length of the Datacom field, in bytes. Refer to Translating Data Types on page 144 for the default mapping of Datacom data types to TIBCO Object Service Broker semantic types and syntax.
Xsyn, Xlen, Xdec, and Xoff	When reading or writing data from or to CA Datacom, the server converts the data from or to the format described by these attributes. Exercise caution if you change them. **Xsyn** — TIBCO Object Service Broker's external syntax mapping the CA Datacom data type of the field in the CA Datacom table row. **Xlen** — The length in bytes of the field in the CA Datacom table row. **Xdec** — The number of digits to the right of the decimal point in the field in the CA Datacom table row. **Xoff** — The offset of the field in the CA Datacom table row. You cannot change this value, which has been computed either by the metadata extractor or by the UI Table Editor, depending on which of these two tools you used to generate the definition.
Datacom/ Copybook Name	The name that originates from either the CA Data Dictionary or, if you created your DAT definition through the TIBCO Object Service Broker Eclipse UI table definer, from the COBOL copybook you used.
Dec	Specifies the number of digits to the right of the decimal point. This field is relevant only for syntax P. The valid entries are as follows: • Syntax P — Value must be smaller than twice the length of the entire field. • Syntax B, C, W, V — 0. • Semantic type C — 0.
Req	Specifies if the field is required. Any field can be a required field. By definition, a primary key field is a required field. The valid entries are as follows: • Y — **Required.** Every occurrence in the table must have a value for this field. Inserting or editing an occurrence without valid values in required fields causes an exception to be raised. • N — Not required. • Blank — Not required.

Reference	If a table is to be referenced when a user is inserting or replacing a value in the field, enter the name of a table that is to be referenced. The added or modified value must exist as a primary key value in the referenced table or the action fails. Reference checking is not done if a null value is given for a field that is not required and does not have a default value. A valid entry is the name of any table except a table of type SCR, RPT, or EXP; the primary key field on the table must hold the values for the referenced field. The table cannot be parameterized.
Default	If you specify a default value for a field and no data is provided, the default value is used instead of a null value. A valid entry is any valid value for the field. If arithmetic operations are to be performed on numeric fields of type Q or C, you must enter a numeric default value such as `0.00`; arithmetic operations cannot be performed on data containing null values.

Using Extensions Screen PF Keys

Use the following PF keys from the Extensions screen:

PF2	Displays the Fields screen used to select the Datacom fields.
PF3	Validates changes and, if valid, returns you to the Table Definition screen.
PF12	Cancels changes and returns you to the Table Definition screen.

Selecting Fields

From the sample Extensions screen shown in Defining Fields on page 140, use PF2 to display a listing of available fields for selection.

Type an **S** in the line command field of each field that you require. Press PF3 to save the selection and return to the Extensions screen. Press PF12 to cancel and return to the Extensions screen.

```
t of Fields in DATACOM Table  PERSONNEL

S DATACOM Field Name                 MKS   Off   Type Sign Len Semantic
- ---------------------------------  ---  ----    -   -   --- --------
_ NUMBER                             1     0     N   N     5
_ NAME                               0     5     C   N    24
_ STREET-ADDRESS                     0    29     C   N    24
_ CITY-ADDRESS                       0    53     C   N    15
_ STATE-ADDRESS                      0    68     C   N     2
_ ZIP-CODE-LOC                       0    70     C   N     5
_ SOCIAL-SECURITY                    0    75     D   Y     5


      < Place “S” beside the item(s) you wish to have Selected on PF3 >

 PFKEYS: 3=SAVE 12=CANCEL
```

Note the following when selecting CA Datacom fields for a DAT table definition:

- Group fields and simple fields are available for selection. You must not have overlapping fields in your DAT table definition. Use the fields **Off** and **Length** to determine if fields are overlapping. Pressing PF3 from the Extensions screen also validates for overlapping.
- The primary key of the DAT table must include all the master key fields of the CA Datacom table, with no overlapping of fields. You can include both group fields and simple fields in the primary key.

  TIBCO Object Service Broker holds a maximum of 16 fields in a composite primary key; however, you can use the Table Editor and the Table Browser only on tables that have a maximum of eight fields.
- Component fields of a master key must have the same order in the primary key. If the field is part of a master key, a value greater than 0 in the **MKS** field indicates the sequence of the fields in the master key.
- Newly selected fields are placed last in the order of existing fields, and are automatically numbered when you return to the Extensions screen.

Translating Data Types

The following table illustrates CA Datacom data types that can be converted to TIBCO Object Service Broker syntax. Default translations are shown in the following table.

```
CA Datacom    TIBCO Object Service Broker

TYPE SIGN  LENGTH    Xsyn Syn   Len  Meaning
---- -   -------     --   --  ----- -------------------------------------
B    N       2        K    B      3 halfword unsigned
B    N       3        H    V      6 time, HHMMSS
B    N       4        K    B      5 word, unsigned
B    N       8        K    B      9 doubleword, unsigned
B    N      10        H    V     20 timestamp, CCYYMMDDHHMMSSNNNNNN
B    Y       2        B    B      2 halfword, signed
B    Y       4        B    B      4 word, signed
B    Y       8        B    B      8 doubleword, signed
C    N   32719[1]     V    V      L char
D    N      16[1]     U    P      L packed, unsigned
D    Y      16[1]     P    P      L packed, signed
DATE N       4        H    V      8 date, CCYYMMDD
E    Y      16        F    F     16 expanded float
G    N   32718[1]     UN   UN     L graphic
H    N   32718[1]     V    V      L hexadecimal, two-byte display
K    N   32718[1]     UN   UN     L Kanji, same as G
L    Y       8        F    F      8 long float
N    N      31[1]     M    P  L/2+1 zoned decimal, unsigned
N    Y      31[1]     N    P  L/2+1 zoned decimal, signed
RAW  N   32719[1]     H    RD   L+4 all "other" data types
S    Y       4        F    F      4 small float
Y    N   32718[1]     UN   UN     L DBCS, same as G
Z    N   32719[1]     W    W      L mixed (DBCS & SBCS)
2    N       2        K    B      3 halfword, unsigned
2    Y       2        B    B      2 halfword, signed
4    N       4        K    B      5 word, unsigned
4    Y       4        B    B      4 word, signed
8    N       8        K    B      9 doubleword, unsigned
8    Y       8        B    B      8 doubleword, signed
```

[1]The reported/assumed length of the CA Datacom field is less or equal to this number. The notation L in the Len column refers to this length.

Translating Nulls

The CA Datacom interface used by TIBCO Object Service Broker has no concept of nulls. As a result, the following translations occur:

- Numeric nulls are translated to zeros.
- String nulls are translated to spaces.

For more information on null processing within TIBCO Object Service Broker, refer to the TIBCO Object Service Broker *Programming in Rules* manual.

Changing the Defaults

You can modify any attribute in the TIBCO Object Service Broker field definition section of the Extensions screen. You can:

- Change the default order in which the fields appear, by typing new numbers in the **#** field. When you are defining the order, specify the primary key(s) as the first field(s).
- The other selected fields follow the primary key(s) in numeric order from top to bottom of the list. Fields without numbers are deleted.
- Change the entry in the **Name** field to a new name to uniquely identify the field within the DAT table.
- You can name a field the same as a field in another table; if you are moving data between tables, giving the fields the same name simplifies the process.
- Change the TIBCO Object Service Broker semantic type (**Type** field) and syntax (**Syn** field) of the field.

  You can use any valid TIBCO Object Service Broker semantic type (except date) and syntax (except floating point), provided the combination is valid. Refer to the TIBCO Object Service Broker *Programming in Rules* manual for a list of valid combinations.

Changing the TIBCO Object Service Broker field syntax can cause a conversion error, since the server must convert each affected field of each row to the new syntax as defined in the DAT Table Definition.

Task C: Add Control Information

Use the Core screen to add or delete additional control information in your definition. To access this screen, press PF4 from the Table Definition screen.

```
CONTROL TABLE INFORMATION for DAT Table PERSONNELD
     UNIT    IDGEN  Fix      Source       Segment#
   --------    -     -  ----------------    ---
   USR40       N     N                      0

PARAMETERS INFORMATION:
  #       Name           Class Type Synt Lgth  Reqd  Decimal Src   Source Name
  - ----------------      -    -     -   ---    -     ---    - ----------------
  1 LOCATION               L    I     C    16           0
EVENTRULE INFORMATION:              |   ORDERING INFORMATION:
 #  Type Access       Rule          |      #      Field Name    Sequence
---  -     -    ----------------    |     ---  ----------------    -




PFKEYS: 4=ADD 16=DELETE 3=SAVE 12=CANCEL
```

Core Screen Field Entries

The top portion of the screen contains the following fields:

UNIT	The user unit with which the table is associated is entered here by default. The Unit marks the table as belonging to a particular application or to a logical unit. This value can be changed. The default unit for your user ID is specified in your TIBCO Object Service Broker user profile. A valid entry is a character string up to eight characters long. Valid units can be provided by your system administrator.
IDGEN	Not used for DAT tables.
Fix	Whether the table definition is bound (Y) or not bound (N). For more information, refer to Binding DAT Table Definitions on page 114 and TIBCO Object Service Broker *Application Administration*.

Source	Not used for DAT tables.
Segment#	Not used for DAT tables.

You can also enter the following types of information into this screen:

- Modifications to the default location parameter information
- Event rules
- Ordering criteria

Parameters

Use this section of the Core screen to define a location parameter for the DAT table. A location parameter is required only if you want to access Datacom data through a peer server associated with a different Data Object Broker (remote node). If you do not require a location parameter, position your cursor in this section and press PF16 to delete the parameter.

If you always access the CA Datacom table remotely, the node from which you request the access can have either a minimal or a full definition.

Minimal Definition

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 through the use of the **`Default`** field, **`Src`** field, or **`Src`** and **`Sourcename`** field. Data parameters are defined on the full definition, not a minimal definition.

A minimal definition with a location parameter means you always access data at a remote node. 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 Table Definition

A full definition with a location parameter indicates you can access data at either the local node or a remote node.

The table type specified in a full definition must match the data on the local node. For example, a full definition of type TDS used to access TDS data on the local node can also be used to access a DAT table with the same name on a remote node.

See Also TIBCO Object Service Broker *Managing Data* for more information on location parameters, event rules and minimal table definitions.

Event Rules

Using the event rule feature, you can provide additional controls over access to a table. To provide these controls you define event rules based on defined accesses. These accesses can be one of the following, for each rule:

W	Any write access.
I	Insert only.
R	Replace only.
D	Delete only.
G	Any retrieval access.

Types

The event rules are always called when the table is accessed in the access type specified. All the rules that apply to a specific access are executed in the order in which they are entered in the event rule section. They cannot access tables on a remote node. You can define two types of event rules, as follows:

Validation	Verify the value of an occurrence when the table is being modified, such as checking the validity of a field value
Trigger	Cause additional processing to take place when a table is accessed. For example, a trigger rule can be used to create an audit trail or update other tables.

Fields

The event rule information is entered in the scrollable event rule section. To define event rules, position your cursor in this section and press PF4. Complete the following fields:

Line#	Type in a line number, starting at 1 for the first line, with one event rule per line. The line numbers must be numbered consecutively.
Type	The type of event rule. The valid entries are as follows: • **V** — Validation rule. No database updates are allowed during the validation process. The rule must be a function that returns Y (yes), the validation was successful, N (no), the validation was not successful, or a message explaining why it was not successful. • **T** — Trigger rule. No restrictions on coding other than that it must not be a function, it cannot change the contents of the triggering row, and it cannot use the TRANSFERCALL statement. Nested triggers are possible.
Access	The type of access (or manipulation) to be performed on the data, causing the event to be executed. See the table below for the valid entries.

Validation Rules	Trigger Rules
W — Any Write (Insert, Replace, Delete)	W- Any Write (Insert, Replace, Delete)
I — Only Insert	I - Only Insert
R — Only Replace	R - Only Replace
D — Only Delete	D - Only Delete
	G - Any Retrieval

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

Ordering Information

Although ordering of fields is possible, it is more efficient to use the ORDERED clause on a retrieval statement. However, to add ordering to your definition, use the scrollable ordering information section.

Restrictions: You will not be able to issue a GET on a table if you specify ordering in the table definition or on a GET statement.

Position your cursor in this section, press PF4, and enter the following in the fields:

Line#	Type in a line number, starting at 1 for the first line, with one field per line. Lines must be numbered consecutively.

Sequence	Specify if the field is to be ordered in ascending or descending order. Rows are normally returned in primary key order. Valid entries:

A	Ascending.
D	Descending.
blank	Ascending.

Field Name	Type the name of the field that is to be used for ordering.

Core Screen PF Keys

Use the following PF keys in the Core screen:

PF3	Validates changes and, if valid, returns you to the Table Definition screen.
PF4	Adds an initial entry into the section of the screen where your cursor is positioned.
PF12	Cancels changes and returns you to the Table Definition screen.
PF16	Deletes an entry. Position your cursor on the appropriate entry and use the key.

Task D: Document the DAT Table

Each table definition in TIBCO Object Service Broker has a Documentation screen associated with it. You use this screen to create or modify documentation for the table. To display the Documentation screen for a DAT table, press PF2 from the Table Definer.

The sample screen below shows the Documentation screen associated with the PERSONNELD table:

```
DESCRIPTION OF TABLE            PERSONNELD                  UNIT: USR40

MODIFIED ON 20 JAN 2000 BY ACC            CREATED ON 02 JAN 2000 BY USR40

KEYWORDS: PERSONNEL
SUMMARY : DATACOM TABLE CONTAINING EMPLOYEE DATA

                              DESCRIPTION
_ -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
_ This table contains all information on current employees.




PFKEYS: 3=END 5=VIEW DOCUMENT 13=PRINT 12=EXIT
```

Field Values

The Table Definer updates some of the fields on this screen, but you must maintain the **KEYWORDS**, **SUMMARY**, and **DESCRIPTION** fields. Complete these fields as follows:

KEYWORDS	Type individual words that briefly describe the table. These words are used by the Keyword Search facility in TIBCO Object Service Broker. This field is one line long and can contain multiple entries, separated by commas or blanks.

SUMMARY	Type a one line summary of the **DESCRIPTION** field.
DESCRIPTION	Type information about the table (for example, what its role is, what it does, and how it works) using TIBCO Object Service Broker **SCRIPT** commands. There is no limit to the amount of information you can type in this field.

PF Keys

The Documentation screen supports the following program function keys:

1	Displays corresponding help for the field or screen where your cursor appears.
3	Saves changes and returns you to the Table Definer.
5	Toggles between browse and edit modes.
12	Cancels changes and returns you to the Table Definer.
13	Prints the version of the documentation that you are viewing.

See Also TIBCO Object Service Broker *Shareable Tools* for more information on the SCRIPT tool.

Chapter 9 Using TIBCO Object Service Broker to Process CA Datacom Data

This chapter provides information on how to access and process data in TIBCO Object Service Broker DAT tables and to take advantage of CA Datacom features in TIBCO Object Service Broker

Topics

Processing the Data

When you access CA Datacom data from TIBCO Object Service Broker, the following occurs:

1. TIBCO Object Service Broker requests are translated into native CA Datacom commands.
2. CA Datacom field data types are translated to the TIBCO Object Service Broker field types defined in the DAT table.

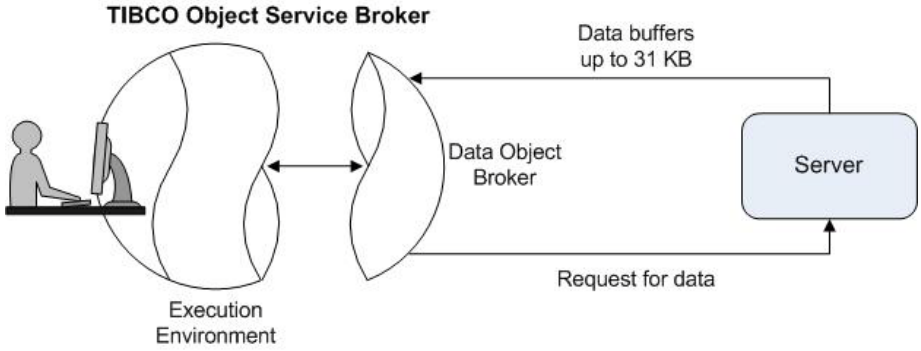
You can access the data using either of the following methods:

- The Table Browser and Table Editor
- The rules language

The following sections describe the mechanisms available to process the data and the procedures for taking advantage of the CA Datacom functionality and for handling handle errors.

The initializer program for the server is passed a number of parameters. One of these parameters instructs the program to attach a number of instances of the server, allowing multiple servers in a single server address space. Each instance of the server uses a CA Datacom thread to connect to CA Datacom. The thread is disconnected only when the server is disconnected.

When CA Datacom data is requested, the server starts a FORALL. Data is sent to the Data Object Broker in variable length buffers up to a maximum of 31 KB, as shown in the following illustration. If a single request requires more than 31 KB of data, multiple 31 KB buffers are sent until the request is complete.



Using the Table Browser

You can use the Table Browser to browse a defined DAT table by typing the table name next to the BR browse table option and using Enter.

In the sample screen below, PERSONNELD is the name of the DAT table. The CA Datacom data is presented to you in TIBCO Object Service Broker table format.

```
BROWSING TABLE   :  PERSONNELD
COMMAND ==>
                                                                  SCROLL: P
  NUMBER          NAME              POSITION      MGR#  DEPTNO  SALARY
_ ------ ---------------------- -------------- ------ ------ ---------
_  22001 DRABEK                 CUST SUPPORT    56112     30    900.00
_  22007 ROEDER                 CUST SUPPORT    56112     30    900.00
_  30058 HOEGSON                PRE-SALES       37219     20    675.00
_  34111 TERAMURA               PRE-SALES       37219     20    710.00
_  34121 LEES                   CUST SUPPORT    56112     30    700.00
_  36162 MORANG                 JR OPERATOR     44798     80    575.00
_  41001 CROFTON                TECH WRITER     80002     70    675.00
_  41007 STEVENSON              EDUCATOR        80002     60    700.00
_  41009 SMITH                  TESTER          79912     50    600.00
_  44385 SOUZA                  SALES           37219     10    719.00
_  44622 SAUNDERS               ACCOUNTANT      98895     40    800.00
_  51111 HRODEK                 ANALYST         79912     50    710.00
_  51121 CANNON                 ANALYST         79912     50    700.00
_  51162 KIMURA                 JR PROGRAMMER   79912     50    575.00
_  61219 WONG                   SENIOR ANALYST  79912     50    820.00
_  61385 DHILLON                EDUCATOR        80002     60    685.00
_  61622 SCHULTZ                SENIOR ANALYST  79912     50    800.00
PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND
```

Exceptions

You can browse and edit a DAT table in the same way you would browse or edit another table, except:

If…	Then…
The **Duplicate Master Keys** field is set to Y.	You cannot update the table using the Table Editor. You can, however, use the Table Browser and the Single Occurrence Editor.

If…	Then…
Your table definition contains fields that are longer than 260 bytes.	• You must use the Single Occurrence Editor from the Table Editor to edit them. • You must use **SELECT LIKE** instead of **SELECT** to access fields of this length.

Insertion of Duplicate Primary Keys

Depending on its indexes, you can use CA Datacom to insert duplicate primary keys. A duplicate primary key results in the following:

- If you delete an occurrence using the Single Occurrence Editor, the occurrence shown is deleted.
- If you delete or update an occurrence using a rule, the last occurrence retrieved is the one deleted or updated.

See Also TIBCO Object Service Broker *Managing Data* for more information about browsing and editing tables in TIBCO Object Service Broker.

Using Rules

Accessing CA Datacom data using the TIBCO Object Service Broker rules language is similar to accessing TIBCO Object Service Broker data. The main difference is in the way CA Datacom interprets the request.

The following sections outline the differences encountered while using rules and point out normal TIBCO Object Service Broker rules behavior to consider when building applications.

Transaction Streams

If you issue a TIBCO Object Service Broker EXECUTE statement within a main (parent) transaction, it creates another (child) transaction stream, to a maximum of ten streams. The number of streams allowed in a TIBCO Object Service Broker transaction depends on the TRANMAXNUM Execution Environment parameter, which has a default of nine streams. Each transaction stream in TIBCO Object Service Broker that accesses CA Datacom data requires its own server thread.

Ensure that your system administrator is aware of the number of server threads required to accommodate all transaction streams accessing CA Datacom data in a single transaction.

Using TRANSFERCALL or DISPLAY & TRANSFERCALL statements in your rules minimizes server threads and reduces the possibility of CA Datacom locking contention.

Transaction Limitations

The number of DAT tables you can access per transaction depends on the POOLSIZE server parameter. Refer to Specifying the Startup Parameters on page 119 for more information.

If you use the default parameter values, you can access at least 16 DAT tables in one transaction; more, depending on the size of the DAT table definitions. Refer to Estimating the CTABLESIZE Parameter on page 121 for more information.

Retrieval Processing

The **SELxx** CA Datacom commands are used to retrieve data from CA Datacom for each retrieval statement (GET or FORALL) in your rule.

When TIBCO Object Service Broker runs in browse mode, no locks are taken in CA Datacom. When TIBCO Object Service Broker runs in update mode, the server retains an image of all the occurrences read (CA Datacom imposes a primary exclusive locking limit of 9999 occurrences). When the update to the occurrence completes, the server re-reads the occurrence that is being updated, and if the image matches the image originally retained, the occurrence is replaced or deleted. If the images do not match, the update is not done and LOCKFAIL is signalled.

GET Statement

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

FORALL Statement

A FORALL statement returns rows to TIBCO Object Service Broker in the order in which CA Datacom passes them. If you require a different order, you must include an ORDERED clause in your FORALL statement.

When running in update mode, the equivalent of an exclusive lock is applied to the row being updated. To accommodate the largest number of rows to be updated within a single transaction, increase the size of the EXCTLNO parameter in the CA Datacom master list.

Replace (Update) or Delete Processing

If the **`Duplicate Master Keys`** field is set to N, the row is re-read, checked against its previous image, and if the images match it is replaced or deleted. If the **`Duplicate Master Keys`** field is set to Y, only one row at a time is returned to the Execution Environment and the last row retrieved is replaced or deleted.

The DELETE *where primary key*= statement is not supported for DAT tables.

Insert Processing

To insert rows to CA Datacom tables, ensure your DAT table definition includes the entire row.

See Also

- TIBCO Object Service Broker *Parameters* for more information about Execution Environment parameter.
- TIBCO Object Service Broker *Programming in Rules* for information about the rules language and rules processing.

Handling of Errors in the Server

This section describes how the server handles TIBCO Object Service Broker requests with respect to:

- Synchronization and recovery
- Error handling

Synchronization and Recovery

Locking of CA Datacom data is determined by the server and the CA Datacom Multi-User Facility, not by TIBCO Object Service Broker. A CA Datacom transaction spans the same length of time as a TIBCO Object Service Broker transaction. A CA Datacom COMMIT causes all locks to be released. Therefore, COMMITs are sent from the server to CA Datacom at the end of a transaction (even if no updates were made, so any locks taken are released). The server insists on transaction end after a TIBCO Object Service Broker rule issues a COMMIT.

- A COMMIT request sent to the server or a normal end of TIBCO Object Service Broker transaction results in a COMMIT being sent to CA Datacom.
- A ROLLBACK request sent to the server or a transaction failure sends a ROLLBACK to CA Datacom, and the TDS intent list is discarded.
- Intermediate ROLLBACKs are allowed and further updates can be made in the same transaction.
- TRANSFERCALL and DISPLAY & TRANSFERCALL statements are allowed after a ROLLBACK.

`COMMITLIMIT Exception`

The COMMITLIMIT exception does not apply to CA Datacom tables. Requests to update CA Datacom data are processed as they are encountered, and are not buffered in the Intent List. However, you can be limited by CA Datacom's limit of rows locked in primary exclusive control (EXCTLNO Master List parameter). Refer to Defining the Master List on page 97 for more information.

Updates of TIBCO Object Service Broker and CA Datacom Data

TIBCO Object Service Broker provides a method of ensuring data integrity when both CA Datacom and TIBCO Object Service Broker data are updated in the same transaction. This method is referred to as Fail Safe processing level 1.

If you do not request Fail Safe processing, transactions that update both CA Datacom and TIBCO Object Service Broker data can result in discrepancies if the server or the Data Object Broker abnormally terminates during the transaction end processing. Refer to Implementing Fail Safe Processing on page 126 for more information.

System Exceptions

The TIBCO Object Service Broker runtime environment signals system exceptions to enable 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 CA Datacom data through the server are trapped under one of the following exceptions:

ERROR	An error is detected and no lower level exception exists in the application.

ACCESSFAIL	A table access error is detected.

GETFAIL	No occurrence satisfies the selection criteria. The CA Datacom error code 14 raises this exception.
DELETEFAIL	The primary key specified for a DELETE statement does not exist.
INSERTFAIL	The primary key provided for an INSERT statement already exists. The CA Datacom error code 10 raises this exception.
REPLACEFAIL	The primary key provided for a REPLACE statement does not exist.

INTEGRITYFAIL	An attempt to violate data integrity is detected.
LOCKFAIL	An attempt was made to read more rows with the CA Datacom primary exclusive lock. Restructure your application to take fewer locks in a transaction, run in browse mode so locks are not taken, or increase the EXCTLNO parameter in the CA Datacom Master List.

SECURITYFAIL	Permission for the requested action on the TIBCO Object Service Broker object is denied. This also occurs if the CA Datacom authorization ID does not have permission to perform the requested action on the specified object.
SERVERBUSY	A new transaction requested an instance of the server and no server is available to process the request. Control is passed back to the rule, giving the rule the opportunity to try the transaction again. If this exception is raised too often, consider starting more servers or reviewing the amount of work being done in your transactions.
SERVERERROR	The server made a request to CA Datacom, and CA Datacom returned an error code that does not map to one of the specific TIBCO Object Service Broker exceptions. The ON SERVERERROR handler can call @SERVERERROR to parse the error message (contained in ENDMSG). Refer to Error Handling on page 164 for more information.
SERVERFAIL	A transaction was in progress when the connection to a server was broken or the server failed. Control is passed back to the rule, giving the rule responsibility for transaction cleanup.

Error Handling

You must pass @SERVERERROR the contents of RETURN_MESSAGE, which has the following format:

*ppp*DM*nnnx serverid serveruserid source: message*

The following list describes the variables necessary to pass the RETURN_MESSAGE contents to @SERVERERROR:

ppp	Represents the user-specified 3 character product ID.
nnn	The CA Datacom external message number.
x	The message severity (E for error, W for warning, and I for information).
serverid	The server ID of the server.
serveruserid	The server user ID (IDPREFIX + ###) of the server.

source	The code portion of the server that trapped the error and returned the message (for example, CSECT, rule, or function).
message	The actual error message text.

If a specific message from a specific server has some information that is required to process the error, the table-driven approach to the execution of @SERVERERROR causes a rule (specified for that error by the developer using @SERVERERROR) to execute. The error message is interpreted in the @SERVERERROR processing and put into a temporary table until required.

Customization

To customize error handling, you must update data in the @DATERRORCODES control table. The definition of the @SERVERERROR and @DATERRORCODES tables is owned by TIBCO Object Service Broker and must not be modified. The data you update in them is owned by you.

Table Processing with SERVERERROR exception

Here is how the table is processed when the SERVERERROR exception is raised and the @SERVERERROR rule is called by your application:

- @SERVERERROR reads the @SERVERMSGCNTL table and looks up the specific message identifier handlers.
- The appropriate message handler looks up the external error codes in the correct server control tables.
- If any codes are found, they call the associated user-written handler.
- The user-written handler can use other functions and data stored in specific tables to handle any specific external error/status code.

@SERVERERROR can be called at any time, although it is useful only for parsing server messages generated due to external CA Datacom errors. The original message can always be retrieved using @SE_MSG after @SERVERERROR has been called. The information parsed by @SERVERERROR has transaction scope.

You can add your own instances in the @SERVERMSGCNTL table, provided that the OWNER specified begins with letters A to Z and the key values in the instance are message identifiers in the form DM*nnnx* mentioned above.

See Also

- TIBCO Object Service Broker *Programming in Rules* about system exceptions.
- TIBCO Object Service Broker *Shareable Tools* for more information on the @SERVERERROR and RETURN_MESSAGE tools.

Chapter 10 Accessing and Processing VSAM LDS Data

This chapter shows you how to access and process VSAM Linear Data Set (LDS) data from within TIBCO Object Service Broker with the SDK.

Topics

Setup of Accesses to VSAM LDS Data

Access to VSAM LDS data is only supported via the SDK. Once the SDK has been installed you need to define the appropriate tables to facilitate the manipulation of the VSAM LDS data and then write the rules to process the data. Your applications used to access the VSAM LDS data are comprised of these table definitions and rules.

Installation of the TIBCO Object Service Broker Base Component

You must install TIBCO Object Service Broker base component in order to use the SDK to access VSAM LDS data. The base component can reside on z/OS, Windows or Solaris. Installation instructions for all platforms are located in *TIBCO Object Service Broker Installation and Operations.*

Installation of the SDK

Installation instructions for the Service Gateway for Files, which includes the SDK, are located in *TIBCO Object Service Broker Managing External Data*.

Required Tables

VSAM LDS data is accessed via a combination of TIBCO Object Service Broker tables: VSAM LDS and VSAM MAP. An access to VSAM LDS is performed via a VSAM LDS table. Data manipulation is performed via VSAM MAP table support.

See Also *TIBCO Object Service Broker Managing External Data* for information about the Service Gateway for Files and about the VSAM accesses discussed in this chapter.

TIBCO Object Service Broker Managing Data for information about MAP tables.

TIBCO Object Service Broker Shareable Tools for information about the tools used for MAP tables.

Definition of VSAM LDS Data

To access VSAM LDS data, first define a table of type of VSM from TIBCO Object Service Broker. This table is used to map the VSAM data for use within TIBCO Object Service Broker. For details on defining VSAM tables within TIBCO Object Service Broker, refer to *TIBCO Object Service Broker Managing External Data*.

The VSAM LDS definition is very similar to that for other VSAM data sets, with the following key differences:

- The Server ID field must be set as access is only permitted via the Service Gateway for Files SDK
- The data set type must be LDS or L
- The load parameter is ignored

The definition of the table must consist of 3 binary fields of the length 4, the first of which must be the primary key. While the Field names may be different their usage will remain constant.

The fields of the table definition are as follows:

Field	Description
RBA	This field is the Relative Byte Address of the LDS record to be processed. It must be on a control interval boundary and for an INSERT must be higher than the current High Used Relative Byte Address for the data set.
ADDRESS	This is the storage address of the buffer that contains the LDS record being processed.
CISIZE	This is the control interval size of the VSAM LDS being processed.

Following is the definition for a sample VSAM LDS table:

```
  COMMAND==>                          TABLE DEFINITION

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

     File  : S6B.XX.LINEAR
     DDname:              Read Only: N     Load: N     Data Set Type: LDS
     Ignore:
     Server ID: FGSERVER
    Parameter Name  Typ Syn Len Dec Class             '      Event Rule   Typ Acc
   ----------------  -   -- --- --    -              '   ---------------- -   -
_  LOCATION           I   C   16  0    L              ' _
_                                                    ' _
                    -------- VSAM --------|---------- Metadata Definition ------
      Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len  Dec Ord Rqd Default
   ---------------- ---- -----  --   -----  -  -   -- ----- ---  -   -  --------

_  RBA              B        4   0       0  P       B      4   0
_  ADDRESS          B        4   0       4          B      4   0
_  CISIZE           B        4   0       8          B      4   0
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

Definition of Associated MAP Table

In addition, you must also define an associated MAP table. For details on defining MAP tables within TIBCO Object Service Broker, refer to *TIBCO Object Service Broker Managing Data*.

The definition is very similar to a normal MAP table. For VSAM MAP tables the following definition rules apply:

- The Data Set Type is set to MAP or M.
- The value of IDgen must be Y.
- The definition must contain an ADDRESS parameter as shown above.
- The key field must be of defined as shown above.
- The first non key field should start at offset 0.

The table definition may also contain a count parameter.

Following is the definition for a sample VSAM MAP table:

```
  COMMAND==>                          TABLE DEFINITION

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

     File  : S6B.XX.LINEAR
     DDname:             Read Only: N     Load: N     Data Set Type: MAP
     Ignore:
     Server ID: FGSERVER
    Parameter Name  Typ Syn Len Dec Class            '      Event Rule   Typ Acc
   ----------------  -   -- --- --   -              '   ---------------- -   -
_ ADDRESS            I   B    4  0    A              ' _
_ LOCATION           I   C   16  0    L                _
                    -------- VSAM --------|---------- Metadata Definition ------
      Field Name    Xsyn Xlen  Xdec Offset Key Typ Syn Len  Dec Ord Rqd Default
   ---------------- ---- -----  --   -----  -  -   -- ----- ---  -   -  --------

_ KEY               B        4   0       0  P  I   B      4   0
_ DATA              B        4   0       0         B      4   0
_
_
_
_
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRT 14=FIELDS 6=OFFSET 21=DATA 2=DOC
```

Operations for Processing Data

You process the data using the TIBCO Object Service Broker rules language. Access to the VSAM LDS returns the address of the data buffer that can be manipulated using an associated VSAM MAP table.

Supported Operations

Only the GET, INSERT and REPLACE operations with fully qualified primary keys (the relative byte address of the record) are supported. Samples are shown in VSAM LDS Samples on page 172.

DELETE processing is not supported.

See Also *TIBCO Object Service Broker Programming in Rules* for detailed information about processing data using the TIBCO Object Service Broker rules language.

Insert Processing

To commence INSERT processing if no previous GET accesses have been performed the address of the buffer in the interface needs to be obtained. This is achieved by accessing the VSAM LDS with a special key of -1:

```
GET vsamlds_table WHERE RBA=-1;
```

This access returns the address of the processing buffer in the VSAM LDS interface. You can then use VSAM MAP table processing to manipulate this buffer.

VSAM LDS Samples

The following JCL defines the Linear Data Set (LDS) in the subsequent sample rules.

```
//IDCAMS JOB (ACCT),'CREATE LINEAR VSAM',
//    MSGCLASS=A,CLASS=A,TIME=30,
//    NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM LINES=999999
//    EXEC PGM=IDCAMS
//SYSIN DD *
  DELETE S6B.XX.LINEAR SET LASTCC=0
  DEFINE CLUSTER (NAME(S6B.XX.LINEAR) -
         TRACKS(5) -
         CISZ(8192) -
         SHAREOPTIONS(1,3) -
         LINEAR)
//SYSPRINT DD SYSOUT=*
```

Creation of Initial Data Set

You can use the following three rules to create control intervals in the sample LDS. Each full word in the LDS contains its relative byte offset from the start of the data set.

Rule LINEAR is key to this process. It obtains the CISIZE of the sample VSAM LDS and the address of the buffer storage in the VSAM LDS interface. It then populates this buffer with the desired data before inserting the contents of the buffer into the physical LDS.

```
LINCREATE(FIRST, LAST);
_ LOCAL A, CISIZE;
_ ------------------------------------------------------------------------
_ ------------------------------------------------------------------+-------
_ A = FIRST;                                                        ¦ 1
_ UNTIL EQ :                                                        ¦ 2
_    CALL LINEAR(A);                                                ¦
_    A = A + CISIZE;                                                ¦
_    CALL @EQ(A, LAST);                                             ¦
_    END;                                                           ¦
_ ------------------------------------------------------------------------
```

```
LINEAR(RBA);
_ LOCAL OFFSET, ADDRESS;
_ ------------------------------------------------------------------------
_ ------------------------------------------------------------------+----
_ GET LINEAR WHERE RBA = - 1;                                       ¦ 1
_ CISIZE = LINEAR.CISIZE;                                           ¦ 2
_ LINEAR.RBA = RBA;                                                 ¦ 3
_ OFFSET = 0;                                                       ¦ 4
_ ADDRESS = LINEAR.ADDRESS;                                         ¦ 5
_ UNTIL EQ :                                                        ¦ 6
_    CALL @EQ(OFFSET, CISIZE);                                      ¦
_    LINEARMAP.DATA = RBA + OFFSET;                                 ¦
_    LINEARMAP.KEY = 1;                                             ¦
_    CALL LINEARREP;                                                ¦
_    OFFSET = OFFSET + 4;                                           ¦
_    END;                                                           ¦
_ INSERT LINEAR;                                                    ¦ 7
_ ------------------------------------------------------------------------
```

```
LINEARREP;
_
_ ------------------------------------------------------------------
_ ------------------------------------------------------------+------
_ REPLACE LINEARMAP(LINEAR.ADDRESS + OFFSET);                 ¦ 1
_ ------------------------------------------------------------------
_ ON COMMITLIMIT :
_    COMMIT;
_    REPLACE LINEARMAP(LINEAR.ADDRESS + OFFSET);
```

The result of executing LINCREATE(0,8192) is that the first control interval in the VSAM LDS is created as follows:

```
PRINT INDATASET(S6B.XX.LINEAR)
IDCAMS SYSTEM SERVICES TIME: 04:44:44
LISTING OF DATA SET -S6B.XX.LINEAR
RBA OF RECORD - 0
00000 00000000 00000004 00000008 0000000C 00000010 00000014 00000018 0000001C
00020 00000020 00000024 00000028 0000002C 00000030 00000034 00000038 0000003C
00040 00000040 00000044 00000048 0000004C 00000050 00000054 00000058 0000005C
00060 00000060 00000064 00000068 0000006C 00000070 00000074 00000078 0000007C
00080 00000080 00000084 00000088 0000008C 00000090 00000094 00000098 0000009C
000A0 000000A0 000000A4 000000A8 000000AC 000000B0 000000B4 000000B8 000000BC
000C0 000000C0 000000C4 000000C8 000000CC 000000D0 000000D4 000000D8 000000DC
000E0 000000E0 000000E4 000000E8 000000EC 000000F0 000000F4 000000F8 000000FC
etc., etc. until
01F80 00001F80 00001F84 00001F88 00001F8C 00001F90 00001F94 00001F98 00001F9C
01FA0 00001FA0 00001FA4 00001FA8 00001FAC 00001FB0 00001FB4 00001FB8 00001FBC
01FC0 00001FC0 00001FC4 00001FC8 00001FCC 00001FD0 00001FD4 00001FD8 00001FDC
01FE0 00001FE0 00001FE4 00001FE8 00001FEC 00001FF0 00001FF4 00001FF8 00001FFC
```

Once this initial control interval has been created further control intervals may be created starting from RBA 8192.

Reading of VSAM LDS data

You can use the following rule to read a word of data at a given RBA in the sample VSAM LDS and display the results as a decimal number. It calculates the control interval to be read, reads the LDS VSAM table to read this control interval into the VSAM LDS interface buffer. It then uses the VSAM MAP table to read the actual data from the correct word in this buffer.

```
LINGET(RBA);
_ LOCAL RECORD, OFFSET;
_ ------------------------------------------------------------------------------
_ REMAINDER(RBA, 4) = 0;                                                      ¦ Y N
_ ------------------------------------------------------------------------+--------
_ GET LINEAR WHERE RBA = - 1;                                               ¦ 1
_ RECORD = RBA / LINEAR.CISIZE * LINEAR.CISIZE;                             ¦ 2
_ OFFSET = RBA - RECORD;                                                    ¦ 3
_ GET LINEAR WHERE RBA = RECORD;                                            ¦ 4
_ GET LINEARMAP(LINEAR.ADDRESS + OFFSET);                                   ¦ 5
_ RETURN(LINEARMAP.DATA);                                                   ¦ 6
_ RETURN('RBA MUST BE A MULTIPLE OF 4');                                    ¦   1
_ ------------------------------------------------------------------------------
_ ON GETFAIL :
_    RETURN('GETFAIL RBA ' || RBA);
```

For example, EX LINGET(2868) returns 2868.

Replacement of VSAM LDS Data

You can use the following rule to read a word of data at a given RBA, increment it by 1, and replace the data in the sample VSAM LDS. It calculates the control interval to be read and reads the LDS VSAM table to read this control interval into the VSAM LDS interface buffer. It then uses the VSAM MAP table to read the actual data from the correct word in this buffer. It increments it by 1 and replaces the data in the VSAM LDS interface buffer using the VSAM MAP table and then replaces the contents of the VSAM LDS control interval using the VSAM LDS table.

```
LINREP(RBA)
_ LOCAL RECORD, OFFSET;
_ ------------------------------------------------------------
_ REMAINDER(RBA, 4) = 0;                                    ¦ Y N
_ ----------------------------------------------------------+------
_ GET LINEAR WHERE RBA = - 1;                               ¦ 1
_ RECORD = RBA / LINEAR.CISIZE * LINEAR.CISIZE;             ¦ 2
_ OFFSET = RBA - RECORD;                                    ¦ 3
_ GET LINEAR WHERE RBA = RECORD;                            ¦ 4
_ GET LINEARMAP(LINEAR.ADDRESS + OFFSET);                   ¦ 5
_ LINEARMAP.DATA = LINEARMAP.DATA + 1;                      ¦ 6
_ REPLACE LINEARMAP(LINEAR.ADDRESS + OFFSET);               ¦ 7
_ REPLACE LINEAR;                                           ¦ 8
_ RETURN('RBA ' || RBA || ' HAS BEEN INCREMENTED BY 1');    ¦ 9
_ RETURN('RBA MUST BE A MULTIPLE OF 4');                    ¦    1
_ ------------------------------------------------------------
_ ON GETFAIL :
_    RETURN('GETFAIL RBA ' || RBA);
```

For example, EX LINREP(2868) returns the following:

```
RBA 2868 HAS BEEN INCREMENTED BY 1
```

EX LINGET(2868) now returns the following:

```
2869
```

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X