TIBCO® Object Service Broker

Object Integration Gateway

Software Release 6.0
July 2012

two-second advantage™

Important Information

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

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

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

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

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

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

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

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

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

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

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

Contents

Preface

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

This manual describes how to install and use the Object Integration Gateway, a component of TIBCO Object Service Broker.

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 Getting Started with Object Integration Gateway (OIG)

This chapter gives a brief overview of Object Integration Gateway, describes its pre-requisites, and tells how to set up its administrative interface.

Topics

Overview

What Is OIG?

Object Integration Gateway (OIG) is a powerful application development tool for delivering web-enabled and business-to-business applications. It has the following key strengths:

- Legacy database integration
- Legacy application integration
- Highly scalable solutions, capable of running on z/OS, Windows, and Solaris
- A powerful business rules language and an integrated active repository that provide an application development tool set capable of very high levels of productivity

What Are the Key Components of OIG?

Enterprise JavaBeans Support

Object Integration Gateway provides a set of components that enable you to create EJB applications, including a base session bean component class, a home interface class, and a remote interface class.

Application Bean

Object Integration Gateway includes an application JavaBean that provides a standalone client-side implementation of all the functions provided by its EJB base class. You can use this application bean in a non-EJB environment where you need your standalone application or custom application server to access OIG.

JCA Adapter

Object Integration Gateway provides a Java Connector Architecture (JCA) adapter so that J2EE applications can interface with it.

.NET Support

Object Integration Gateway supports .NET applications via a .NET Class Library.

Prerequisites

J2EE Components on Open Systems

The OIG J2EE components for the different platforms require the Java 2 SDK, Standard Edition, Version 1.4 or higher, or Java 2 Runtime Environment, Standard Edition, Version 1.4 or higher (international edition).

You can download the Java SDK/JRE for free from the following web site:

http://java.sun.com/j2se/

COM Component

Before you can install the OIG COM component, you must have the following software already installed on the same computer:

- Java 2 SDK, Standard Edition, Version 1.4 or higher, or Java 2 Runtime Environment, Standard Edition, Version 1.4 or higher (international version)

You can download the Java SDK/JRE for free from the following web site:

http://java.sun.com/j2se/

OIG for .NET

To develop and run applications using OIG for .NET, you require the following software:

- Microsoft .NET Framework SDK Version 2.0
- Microsoft Visual J# .NET Version 2.0 Redistributable Package

Before installing OIG for .NET, ensure that the required software is already installed.

OIG Components on z/OS

If you plan to host your OIG web applications on z/OS, install the following software:

- z/OS UNIX System Services
- Java 2 for z/OS at the JDK 1.4 level or higher

OIG Components Installation on z/OS

1. Modify the Execution Environment JCL to add STANDBYNUM=*nn* to the Execution Environment's HRNIN DD cards, where *nn* is a number greater than the number of concurrent sessions on the Execution Environment to be used by OIG applications. This logs in *nn* user IDs at Execution Environment startup to accept connections from OIG, thus avoiding the overhead of multiple login/logout processes.

You can use the MAXSESSION parameter to limit the number of sessions for an OIG session pool. For more information, refer to Appendix A, Setting OIG Session Initialization Parameters, on page 113.

2. Recycle the Execution Environment and Data Object Broker to implement the changes.

Installing the OIG Administration Interface

The OIG administration interface is a tool used to manage OIG session pools. The web pages required to use the administration interface are placed in the %OS_ROOT%\admin directory.

Deploy the Administration Interface Web Pages

Before you can begin using the administration interface, deploy it as an OIG web application. To do this, copy the required web pages (JSPs, ASPs or ASPX, depending on your environment) from the %OS_ROOT%\admin directory to an application deployment directory.

The administration interface displays the status of OIG session pools within a Java Virtual Machine (JVM). Therefore, deploy the administration interface so that it runs in the same JVM as the applications you want it to monitor. Keep in mind that servlet/JSP applications can be deployed to run under one or more JVMs. When using the OIG COM component under IIS, the number of JVMs depends on the isolation level of your defined web applications.

The JSP version of the administration interface uses custom tags, so you must copy the tag library definition file, %OS_ROOT%\admin\jsp\WEB-INF\ectstaglib.tld, to the \WEB-INF directory under the application deployment directory defined to your servlet engine.

For information about how to use the administration interface, refer to Appendix D, Using the OIG Administration Interface, on page 161.

Chapter 2 Designing an OIG Application

This chapter describes how to design an OIG application.

Topics

Overview

Object Integration Gateway provides a powerful set of development tools and components for building enterprise-scale web applications. Before you can build your application, you must design it. In this chapter, we discuss the design options available to you, and the impact that different design approaches can have on your application.

Tools Available

TIBCO Object Service Broker UI

The TIBCO Object Service Broker UI is the companion tool to the TIBCO Object Service Broker text workbench for developing an OIG application. The TIBCO Object Service Broker UI is a development interface for creating and modifying TIBCO Object Service Broker and Object Integration Gateway objects. You use the TIBCO Object Service Broker UI to define the OIG objects that are the building blocks of your TIBCO Object Service Broker application: applications, transactions, XML documents, and XML field maps.

Components Available

You build an Object Integration Gateway application from the following components, referred to as Object Integration Gateway objects.

Rules

Rules are the native programming instructions for the rules engine. Rules execute the logic of your server-side application: they tell the rules engine what tasks to perform, such as what data to access from which data tables, and what to do with the data.

Tables

A data table is a defined unit of storage in the TIBCO Object Service Broker database system, used for holding pieces of related information. A data table contains fields (columns) and occurrences (rows). Each occurrence is uniquely identified by a primary key. TIBCO Object Service Broker has many table types to handle a wide range of permanent file storage types, from flat files to DB2 for IBM z/OS systems. Transient (temporary) tables are also used to hold intermediate results that are passed to the client or processed by another Object Integration Gateway object.

Transactions

An Object Integration Gateway transaction is a defined unit of processing that either succeeds or fails entirely. That is, it does not partially succeed or otherwise produce an ambiguous state after executing.

XML Documents

In Object Integration Gateway, an XML document is a definition that governs how OIG consumes (reads) or produces (writes) a particular XML document — that is, how the document is fed into or passed out of the TIBCO Object Service Broker database.

Designing a Transaction

Transaction

A transaction is a defined unit of processing that either succeeds or fails entirely. That is, it does not partially succeed or otherwise produce an ambiguous state after executing. An exception to this is a transaction that is executed as part of a larger compound transaction, specifically startTran and stopTran interfaces. This transaction does not affect the calling transaction if it succeeds and commits only data particular to that transaction. However, if the transaction fails, the compound transaction in which it is nested also fails.

Transactions are one of the basic building blocks of an Object Integration Gateway web application. They determine what data is accessed, and what actions are performed on the data.

Linking Other OIG Objects to a Transaction

When a transaction is built, it makes sense to have other Object Integration Gateway objects linked to the transaction. For example: you have a transaction that accesses all the sales data for a particular region and summarizes it into an intermediate table. You then need the results to appear on a web page containing both static content and dynamic content.

When the Object Integration Gateway objects are linked to a transaction, they are invoked every time the transaction is invoked. This is both a good thing and a bad thing. Perhaps, in some situations, the linked object should not be invoked, unless some condition is met. In this case, you can specify a post-build rule to be run at the end of the transaction. The post-build rule can test the required condition to determine whether the linked Gateway objects are invoked. For more information about the available functions for post-build rules, and their syntax, refer to Chapter 9, Using the OIG Rules Programming Interface, on page 101.

Designing an XML Document

XML Capabilities of OIG

Object Integration Gateway can both consume (read) and produce (write) XML documents.

In OIG, all XML documents are mapped to a set of database tables and their associated relational model. The OIG XML document definition is used to determine the structure of the XML document produced from the relational model by defining both the structure of the data and the formatting of the data. When the XML document is produced, only the initial data selection and the parent Gateway document definition are specified. The document definitions supply the data access specifications to navigate the data model. When an XML document is consumed, the OIG document definition is used to format the parsed data and enter the data into the associated databases.

Re

For a detailed example, see "Creating XML Documents" on page 127.

XML Field Map

An XML field map has two complementary purposes. First, when an XML document is being produced, the field map is used to define what data is to be included in the document, and to determine how this data is represented in the document. Second, when an XML document is being consumed, the field map defines the way that data parsed from the document is processed and where the data is entered into the database.

Root Names for XML Document Tables

Every table within an XML document has a root name. Unlike the root name for an XML document, the root name element for a table will appear once for every row of the table being mapped. Fields of a row of a table can be mapped to attributes of the table root name or some sub-element of the table root name, except as described in , Restrictions That Apply to Tables with No Root Name, page 13. The following sections contain examples for your use.

Example 1

In the example below, a single table is mapped to an XML document. The element "employees" is the root name of an XML document and "employee" is a table root name.

```
<?xml version="1.0"?>
<employees>
  <employee>
    <empno>80002</empno>
    <lname>SMYTHE</lname>
  </employee>
  <employee>
    <empno>80003</empno>
    <lname>CHANG</lname>
  </employee>
  <employee>
    <empno>80004</empno>
    <lname>GARZA</lname>
  </employee>
</employees>
```

Example 2

There are circumstances where an XML document required less structure. Such documents can be consumed or produced by mapping a table with a single row and not specify root name for the table. The following example shows this structure: PolicyOwner is the root name of the XML document, followed by only the field name Value.

```
<?xml version="1.0" ?>
<PolicyOwner>
  <Value>Fred Smith</Value>
</PolicyOwner>
```

Example 3

Within a single XML document you can specify one table with only one row as well as multi-row tables, as shown in the following example:

```
<?xml version="1.0" ?>
<PartRequest>
  <PartCount>2</PartCount>
  <Part>
    <Number>874jq</Number>
    <Description>Cylinder head</Description>
  </Part>
  <Part>
    <Number>860jq</Number>
    <Description>Valve</Description>
  </Part>
</PartRequest>
```

Restrictions That Apply to Tables with No Root Name

The following restrictions apply to specifying a table with no root name in your XML document:

- Only one table of an XML document can have no root name.
- The fields of a table with no root name cannot be mapped to attributes. This is true because the root element of an XML document cannot have attributes.
- Pre and post-processing rules are not permitted for tables without a root name.

Transferring OIG Objects Between Databases

Transferring Objects

To transfer Object Integration Gateway objects from one TIBCO Object Service Broker database to another, you can use the UNLOAD and LOAD tools, or use the TIBCO Object Service Broker promotion system.

OIG Object Types That Can Be Transferred?

Currently, you can transfer the following types of objects:

- TRAN—an OIG transaction
- XMLDOC—an XML document
- XMLFIELDMAP—an XML field map

Using UNLOAD and LOAD

You invoke the UNLOAD and LOAD tools from the TIBCO Object Service Broker text workbench. You use UNLOAD to export an Object Integration Gateway object from the TIBCO Object Service Broker database to an external UNLOAD file. You transfer this UNLOAD file to the system where the other TIBCO Object Service Broker database resides. Then, on the second system, you use LOAD to import the object from the UNLOAD file into the other TIBCO Object Service Broker database.

For more information about how to use the UNLOAD and LOAD tools, refer to *TIBCO Object Service Broker Shareable Tools* in the TIBCO Object Service Broker documentation set.

Promoting OIG Objects

Promoting Object Integration Gateway objects is no different than promoting other TIBCO Object Service Broker objects. The promotable object types appear, along with the other TIBCO Object Service Broker types, on the promotions screen.

For more information about TIBCO Object Service Broker promotions, refer to *TIBCO Object Service Broker Managing Deployment* in the TIBCO Object Service Broker documentation set.

Using Data Access Parameters

When accessing data through generic data access rules and XML generation, you can supply data access parameters. These take two forms: data parameter values and data key values. A data parameter value specifies the criteria that enables Object Integration Gateway to determine the parameter value required to access a parameterized data table. A data key value specifies the selection criteria that OIG uses to select data from the data table.

For information about the syntax required for data parameter values and data key values, refer to Appendix B, Understanding the Data Access Parameter Syntax, on page 123.

Data Access Parameters Can Be Dynamic

Data access parameters do not have to be static, but can be dynamic, either totally or partially. Specifying data access parameters as static values is a valid, but limiting, approach. The only requirement is that the resultant values make up a valid data access specification.

For example, the specification to access all rows of data from the table MYTABLE, where the field VALID is set to Y and the field VALUE is greater than a given integer, could be as follows:

```
Key Value = "VALID = 'Y' AND VALUE > 123"
```

In this example we could have any part of the predicate string as dynamic. A good example would be that we wanted to pass the value for the field VALUE from the web application as a session data value or as a name-value pair argument called SELECTEDVALUE. In this case, the key value for the object would be:

```
Key Value = "VALID = 'Y' AND VALUE > {SELECTEDVALUE}"
```

Passing Data to and Receiving Data from OIG

You can exchange data with OIG via:

- Name and value pairs
- Table recordsets
- XML

Passing Parameters Data

Collections of names and values can be passed as parameter data for an OIG transaction or XML document when the object is run. This data can be extracted by rules using the ECTSGETARGS utility. Such data is held in a session table called ECTSPARMS. Names for parameter data do not need to be unique. The same parameter name may be passed more than once.

When the .NET or the COM interfaces of OIG are used on a web page, query data for the page is automatically passed along with parameters for the OIG object.

Passing and Receiving Session Data

A collection of persistent data may also be passed when running an OIG object and is automatically received when the request has completed. Once a name and value pair has been added to the collection of persistent data, it will be passed for all calls afterwards unless it is removed from the collection. The Java, .NET, and COM OIG interfaces have functions to set and get these data. Unlike parameter data, names must be unique in the collection of persistent data.

A rule can extract persistent data using the ECTSGETARG or ECTSGETSESS utilities. ECTSGETSESS only gets persistent data whereas ECTSGETARG first checks for parameter data and then searches the collection of persistent data. Persistent data is placed in a session table named ECTSSESSION.

When called from a web page context, the .NET and COM interfaces of OIG automatically pass all web session variables in the collection of persistent data. Should a rule add a new name and value to the collection, then the value is automatically copied to the web page's session variable collection after the call to OIG.

The amount of data passed as persistent data should be carefully considered, as it can significantly affect the performance of your application.

Passing and Receiving Data as Tables or XML

More complex data may be passed and received as a table or XML document. XML is passed and received as strings but tables are exchanged using the appropriate object for the technology being used.

Language/Platform	Recordset Object
Java	webRowSet
.NET	ADO.NET recordset
COM	ADO recordset

Tables and XML are passed and received using get and set methods as name and value pairs. The name when setting a table or XML document corresponds to the name of a table or XML document defined in TIBCO Object Service Broker.

To pass a collection of tables and XML, the OIG application must call set interfaces to build the collection. On the subsequent call to run a transaction or XML document, these documents and tables are also passed along with parameter data for the object. Data from the collection is automatically available to rules for the OIG object. Table data is assigned to the named table and the appropriate XML document objects are invoked to process the XML before the OIG object is run. Finally, the collection of tables and XML is cleared when the call to run an OIG object is complete.

OIG XML documents and transactions can return XML and tables. Upon return from calling an OIG object an application can use appropriate get functions to extract the returned data.

Chapter 3 Using OIG for .NET

This chapter describes how to use OIG for .NET.

Topics

Overview

Object Integration Gateway Support for .NET

TIBCO Object Service Broker support for .NET contains the components necessary to define and execute transactions that access data residing in TIBCO Object Service Broker (or third-party databases accessible through TIBCO Object Service Broker) from a Microsoft Windows environment. A .NET class library provides native access to TIBCO Object Service Broker for ASP.NET applications, traditional forms-based applications, and COM+ applications. You can use an ASP.NET Web Control to paint web pages using the drag-and-drop painting facilities of Microsoft Visual Studio .NET.

The .NET class library provides access to all the features of OIG that are available from the OIG servlet and the OIG tag library, plus the ability to access data via an ADO.NET recordset. The .NET class library supports two public interfaces: the eCTSsession class, which provides all the methods needed to write an OIG application, and the eCTSexception class (derived from the .NET ApplicationException class), which is thrown should an error be detected.

The examples in this chapter use Visual Basic .NET syntax.

Configuring Pools for the .NET Class Library

The Pool Configuration Tool

The setup program automatically configures the .NET class library and the web control. However, you can change configuration settings at any time, using the .NET Pool Definer. This tool enables you to create aliases for a defined set of session parameters. You can then use the pool name in the code of your client application to specify the session parameters for your application.

Unlike the OIG COM component, the .NET class library does not require the installation of a Java Virtual Machine on the installation machine.

When you make changes to a pool definition, you can use the OIG administration interface to refresh the session pool, thereby putting the changes into effect immediately. Refer to Appendix D, Using the OIG Administration Interface, on page 161 for more information.

Instantiating the eCTSsession Object

Constructors

Before any functionality can be invoked, you must instantiate an eCTSsession object. The eCTSsession class has the following constructors:

```
eCTSsession()
eCTSsession(boolean useWebContext)
```

The second constructor enables you to instantiate an eCTSsession object that does not send variables out of the web session's request object to TIBCO Object Service Broker when an interaction is invoked. This is useful in cases where the component is instantiated in a web application, but the application programmer wants to explicitly control what web variables are sent to TIBCO Object Service Broker.

The first (default) constructor instantiates an eCTSsession object that sends all variables to TIBCO Object Service Broker on each interaction.

Example

The following example instantiates an eCTSsession object in Visual Basic .NET code:

```
Dim MySession As eCTSsession = New eCTSsession
'Invoke methods...
MySession.Close()   'Closes the session immediately
MySession = Nothing     'Frees the session object
```

Passing Data to the eCTSsession Object

The following sections describe the parameter syntax for the eCTSsession class methods.

Strings

A simple string value:

```
Transaction = "DEMOTX"
```

Hashtable and NameValueCollection Classes

Sets of parameters are passed to methods of the eCTSsession class as name-value pairs using .NET collection classes. Session parameters have unique key names, and are passed via the Hashtable class. Transaction parameters can have more than one value per key name, and so are passed via the NameValueCollection class. The following is an example that passes both session and transaction parameters:

```
Dim MySesssion As eCTSsession = new eCTSsession
Dim MySessParms As Hashtable = new Hashtable

MySessParms("U") = "USR40"
MySessParms("P") = "USR40"
MySessParms("L") = "USR40"
MySession.OpenApplication MySessParms

Dim MyArgs As new NameValueCollection
MyArgs.Add("DEPT", "Sales")
MyArgs.Add("EMPLOYEE", "Sally")
MyArgs.Add("EMPLOYEE", "Tom")
MySession.RunTrans("DEMOTX", MyArgs)
```

Recordsets

An ADO.NET DataTable object can be used both as input to and a result from eCTSsession methods. By definition, ADO.NET DataSets are disconnected from a data source. This means that they are not actively connected to the source database, so any updates made to a DataTable within a DataSet are not directly reflected in the source database. However, a DataTable returned by an eCTSsession object can be modified and passed back to TIBCO Object Service Broker by another eCTSsession method call.

eCTSsession Methods

Opening Object Integration Gateway Sessions

OpenApplication

Opens an Object Integration Gateway session. An application must open an OIG session to run an OIG transaction or XML document. The OpenApplication method enables you to either create a new session, or borrow one from a pool of existing sessions. Pooled sessions are shared between one or more applications.

Syntax:

```
MySession.OpenApplication()
MySession.OpenApplication(mode)
MySession.OpenApplication(poolName)
MySession.OpenApplication(poolName, mode)
MySession.OpenApplication(sessionParms)
```

```
MySession.OpenApplication(sessionParms, mode)
```

Parameter	Type	Value/Meaning
poolName	String	The case-sensitive name of a defined pool for a set of OIG session parameters. Pool names are held in the Windows registry. By default such sessions are pooled.
sessionParms	Hashtable	A collection of names and their value. By default the session created is not pooled. Refer to Appendix A, Setting OIG Session Initialization Parameters, on page 113.
mode	enumeration	If set to Mode.Pooled sessions are created once and pooled for reuse after being released. If set to Mode.Unpooled sessions are terminated after being released.

If you call the OpenApplication method without specifying either the *poolName* parameter or the *sessionParms* parameter, the default OIG session parameter settings are used. The default session parameters are determined in one of the following two ways:

- From the DEFAULT alias settings, if an alias of that name is defined using the Pool Definer.
- From the default session parameter settings, if a DEFAULT pool is not defined. For more information, refer to Appendix A, Setting OIG Session Initialization Parameters, on page 113.

Closing Object Integration Gateway Sessions

Close

Immediately closes an OIG session. If the underlying session was pooled, the session is returned to the pool of sessions for re-use by another application. If the session was not pooled, the session is terminated and the TIBCO Object Service Broker resources associated with the session are released. If there is a transaction in progress, this transaction is terminated and any updates pending are rolled back. Refer to “StartTran” on page 28.

Syntax:

```
MySession.Close()
```

The storage for objects under .NET's Common Language Runtime (CLR) Environment is managed by a garbage collector. Remember that objects are reclaimed, and the destructor for the object is invoked, when the CLR garbage collector chooses to do so.

A VB .NET programmer should not expect the underlying TIBCO Object Service Broker session of an opened eCTSsesssion object to be released when the program sets the variable's object reference to Nothing. For example:

MySession = Nothing

It is good practice to invoke the Close method to release the underlying resources for an eCTSsession when the object is no longer needed. Otherwise, your application could run out of TIBCO Object Service Broker sessions because the CLR garbage collector has not been invoked for a long period of time.

Dispose

Immediately closes an OIG session. This is an alias for the Close method.

Syntax:

```
MySession.Dispose()
```

Invoking Processing

The methods in this section can be used to invoke functionality defined to Object Integration Gateway. Each of the OIG objects (for example, transactions) can be explicitly invoked.

RunRule

Runs a rule within the current transaction. Before a rule can be called, a transaction must be started by calling the StartTran method.

Syntax:

```
MySession.RunRule(ruleName)
```

```
MySession.RunRule(ruleName, parms)
```

Parameter	Type	Value/Meaning
ruleName	String	The name of the rule to be executed.

Parameter	Type	Value/Meaning
parm	NameValueCollection	A collection of names and their values to be passed to the Gateway rule. The parameters specified for this argument override the parameters specified in the URL query string or passed by a POST from an HTML form.

RunTrans

Runs the named transaction.

Syntax:

```
MySession.RunTrans(transName)
MySession.RunTrans(transName, parms)
```

Parameter	Type	Value/Meaning
transName	String	The name of the transaction to be run.
parms	NameValueCollection	A collection of names and their values to be passed to the transaction. The parameters specified for this argument override the parameters specified in the URL query string or passed by a POST from an HTML form.

RunXmlDoc

Obtains or extracts the named XML Document.

Syntax:

```
MySession.RunXmlDoc(docName)
MySession.RunXmlDoc(docName, parms)
```

Parameter	Type	Value/Meaning
docName	String	The name of the XML document to be obtained or extracted.

Parameter	Type	Value/Meaning
parms	NameValueCollection	A collection of names and their values to be passed to the XML document. The parameters specified for this argument override the parameters specified in the URL query string or passed by a POST from an HTML form.

StartTran

Starts an OIG transaction. Subsequent RunXmlDoc or RunRule calls run within the transaction. The transaction is terminated by the next StopTran or Close call. A call to RunTrans while a transaction is active results in the nesting of a TIBCO Object Service Broker transaction to run the OIG transaction. Only one started transaction can be active for an OIG session at one time. Starting a second transaction results in an error.

Syntax:

```
MySession.StartTran(updateMode)
```

Parameter	Type	Value/Meaning
updateMode	bool	If true, updates are permitted while running the transaction. Otherwise, updates are not permitted and locks are not taken for tables.

StopTran

Stops an OIG transaction.

Syntax:

```
MySession.StopTran(commit)
```

Parameter	Type	Value/Meaning
commit	bool	If true, updates made to tables during the transaction are applied to the MetaStore when the transaction is terminated.

Handling Data

The methods in this section can be used to pass data to and from Object Integration Gateway either as a ADO.NET DataTable, or as an XML document.

GetDataSet

Returns all the OIG tables returned by the last RunTrans, RunXmlDoc, or RunRule call as a collection of ADO.NET DataTables in an ADO.NET DataSet object. The name of a DataTable in the collection is the name of the interface table.

Return type: ADO.NET DataSet object

Syntax:

```
dataset = MySession.GetDataSet()
```

GetDocument

Returns an XML document generated by the last execution of the RunTrans or RunXmlDoc methods.

Return type: String

Syntax:

```
xml = MySession.GetDocument()
xml = MySession.GetDocument(docName)
```

Parameter	Type	Value/Meaning
docName	String	The name of the XML document returned. If this parameter is not specified, GetDocument returns the first XML document in the result set.

GetDocumentFromTable

Returns the XML for an interface table returned by the last RunTrans or RunRule call. Normally interface tables are returned as ADO.NET DataTables using the GetTable method.

Return Type: String

Syntax:

```
dataset = MySession.GetDocumentFromTable(tableName)
```

Parameter	Type	Value/Meaning
tableName	String	The name of the table to be returned as a string.

GetTable

Converts an XML document from the last Run method call to an ADO.NET DataTable object, and returns the DataTable object. The XML document must be of type ADO.NET.

Return type: ADO.NET DataTable object

Syntax:

```
datatable = MySession.GetTable()
```

```
datatable = MySession.GetTable(documentName)
```

Parameter	Type	Value/Meaning
documentName	String	The name of the XML document to be extracted. The XML document must be of type ADO.NET. If this parameter is not specified, GetTable converts the first XML document in memory.

GetTableFromDocument

Converts an XML document from the last Run method call to an ADO.NET DataTable object, and returns the DataTable object. The XML document must be of type ADO.NET and is produced as a result of running an XML document rather than as a result of the Pass Data to Client option of an interface table.

Return type: ADO.NET DataTable

Syntax:

```
datatable = MySession.GetTableFromDocument(documentName)
```

Parameter	Type	Value/Meaning
documentName	String	The name of the XML document to be returned as an ADO.NET DataTable.

GetTableOfContent

Returns all data returned by the last RunTrans, RunXmlDoc, or RunRule call as an ADO.NET DataTable. This includes interface tables, XML documents, and persistent variables.

The table-of content information related to persistent data is present only if the OIG session has been opened with debug mode specified. Refer to the session parameter DEBUG on page 114 for more information.

The DataTable returned contains three columns, each row of the table describing a result returned.

Columns	Type	Description
Name	String	Contains the name of the OIG result.
Type	Integer	Identifies the result.
Value	String	Contains the value of the returned data.

Possible types of returned data are as follows:

Value	Description
3	XML for an interface table.
4	XML for a document.
10	Rules trace back in HTML format.
11	Rules trace back in text format.
30	Deleted persistent data.
31	Replaced persistent data.
32	Persistent data.
33	“Once only” persistent data.

Return type: ADO.NET DataTable object

Syntax:

```
datatable = MySession.GetTableOfContent()
```

GetXmlDocument

Returns an XML document generated by the last execution of the RunTrans, RunRule, or RunXmlDoc methods. The resultant XML data is held in a .NET XML DOM object.

Return type: System.Xml.XmlDocument object

Syntax:

```
xmlDoc = MySession.GetXmlDocument()
```

```
xmlDoc = MySession.GetXmlDocument(docName)
```

Parameter	Type	Value/Meaning
docName	String	The name of the XML document returned. If this parameter is not specified, GetXmlDocument returns the first XML document in the result set.

GetXmlTextReader

Returns an XML document generated by the last execution of the RunTrans or RunXmlDoc methods. The resultant XML data is held in a .NET XmlTextReader object. The XmlTextReader object provides a high-performance, forward-only and read-only way of accessing XML data. The model is similar to the SAX XML model, but implemented as a pull model rather than the SAX push model.

Return type: System.Xml.XmlTextReader object

Syntax:

```
xmlDoc = MySession.GetXmlTextReader()
```

```
xmlDoc = MySession.GetXmlTextReader(docName)
```

Parameter	Type	Value/Meaning
docName	String	The name of the XML document returned. If this parameter is not specified, GetXmlTextReader returns the first XML document in the result set.

SetDocument

Stages an XML document to be sent to OIG on the next Run method call. You can call the SetDocument method multiple times to stage multiple documents for the next Run method call.

Syntax:

```
MySession.SetDocument(docName, xmlDoc)
```

Parameter	Type	Value/Meaning
docName	String	The name of the OIG XML document definition used for processing the incoming XML document.
xmlDoc	String	The XML document string to be passed to an OIG application.

SetDocumentFromFile

Stages an XML document to be sent to OIG on the next Run method call. The source of the XML document is read from the file identified by the *fileName* parameter. The filename can be a complete path name or a filename that is relative to the current working directory.

Syntax:

```
MySession.SetDocumentFromFile(docName, fileName)
```

Parameter	Type	Value/Meaning
docName	String	The name of the document to be passed to TIBCO Object Service Broker.
fileName	String	The name of the file containing the XML document.

SetTable

Passes the contents of an ADO.NET DataTable object to an TIBCO Object Service Broker data table. The SetTable method converts the DataTable to an XML document of type MSSCHEMA, so that when the next OIG object is invoked, the XML document is sent along with any other parameters and loaded into the named TIBCO Object Service Broker data table.

You can use the SetTable method to pass data to TIBCO Object Service Broker that is required by an OIG object invoked by a subsequent Run method.

Syntax:

```
MySession.SetTable(tableName, datatable)
```

Parameter	Type	Value/Meaning
tableName	String	The name of the TIBCO Object Service Broker table the DataTable is loaded into when the next OIG object is invoked.
datatable	ADO.NET DataTable object	The ADO.NET DataTable to be passed to TIBCO Object Service Broker.

When the destination table is a TIBCO Object Service Broker screen table, and the subsequently invoked OIG transaction has an associated TIBCO Object Service Broker screen, the data is loaded into that screen table for the screen named in the OIG object definition.

SetXmlDocument

Stages an XML document to be sent to OIG on the next Run method call. The document is passed via a .NET DOM object (System.Xml.XmlDocument). You can call the SetXmlDocument method multiple times to stage multiple documents for the next Run method call.

Syntax:

```
MySession.SetXmlDocument(docName, xmlDoc)
```

Parameter	Type	Value/Meaning
docName	String	The name of the OIG XML document definition used for processing the incoming XML document.
xmlDoc	System.Xml. XmlDocument object	The XML document to be passed to an OIG application.

Handling HTML and Messages

GetEndMsg

Returns the end message from the last execution of RunTrans or RunXmlDoc. The GetEndMsg method is typically used after executing a Run method.

Return type: String

Syntax:

```
msg = MySession.GetEndMsg()
```

GetResult

Returns the HTML generated by the last execution of a Run method.

Return type: String

Syntax:

```
html = MySession.GetResult()
```

ASPX Example:

```
MySession.RunTrans("DEMOTX")
Response.Write(MySession.GetResult())
```

Handling Errors from eCTSsession

An eCTSsession object throws an eCTSexception object if an error occurs. The eCTSexception class is derived from the .NET ApplicationException class, and has additional properties you can use to get information about rule failures in your OIG application.

You can use the eCTSexception object's Message property to get the reason for the error. You can also use the following two eCTSexception properties to obtain more information about the error.

HtmlErrorLog

If a rule fails in the OIG application, the rules trace back is returned as string in HTML format. If the error is not due to a rules failure, an empty string is returned.

Syntax:

```
html = MyeCTSexception.HtmlErrorLog
```

TextErrorLog

If a rule fails in the OIG application, the rules trace back is returned as string. Each line of the trace back ends with a carriage return followed by a line feed. If the error is not due to a rules failure, an empty string is returned.

Syntax:

```
text = MyeCTSexception.TextErrorLog
```

Managing Persistent Data for Web Sessions

When using OIG with ASP.NET web applications, it is a common technique to store application state information within the IIS Session object. You can use the following methods for OIG to access relevant session variables by setting up a list of variables that are automatically passed to a Run method. The data that is passed is available via the {argname} syntax when you define OIG objects, or via the ECTSGETARG and ECTSGETSESS utility rules. You can also use the ECTSSETSESS utility rule to update the IIS Session object. The IIS Session object is updated at the end of the Run method.

AddSessionParm

Adds a name to the list of session variables passed to the rules engine at execution of RunTrans or RunXmlDoc.

Syntax:

```
MySession.AddSessionParm(varName)
```

Parameter	Type	Value/Meaning
varName	String	The name of the session variable to be added to the list.

RemoveSessionParm

Removes a name from the list of session variables passed to the rules engine at execution of RunTrans or RunXmlDoc.

Syntax:

```
MySession.RemoveSessionParm(varName)
```

Parameter	Type	Value/Meaning
varName	String	The name of the session variable to be removed from the list.

Managing Persistent Data for Non-Web Sessions

Applications that run in a web environment, such as those using ASP.NET, can use the AddSessionParm and RemoveSessionParm methods to manage a list of web session variables stored in the IIS Session object. However, applications running in non-web environments do not have access to the IIS Session object or its variables. You can use the following methods for a non-web application to create and access a list of session variables that is stored by the eCTSsession object. These non-web session variables are stored as a collection of name-value pairs that is passed to, and can be updated by, any of the Run methods.

The following methods can also be called by applications running in a web environment such as ASP.NET. In such cases, the session variables are stored as web session variables in the IIS Session object.

GetPersistentData

Returns the value of an OIG session variable.

Return type: String

Syntax:

```
varValue = MySession.GetPersistentData(varName)
```

Parameter	Type	Value/Meaning
varName	String	The name of the session variable whose value is to be returned.

SetPersistentData

Sets the value of an OIG session variable. If the named session variable is not set, the variable is added to the list of session variables passed to the rules engine on subsequent Run method calls. If the session variable is set to an empty string, the variable is removed from the list of variables passed.

Web applications can also use this method to add to, or remove from, the list of web session variables normally managed by the AddSessionParm and RemoveSessionParm methods.

Syntax:

```
MySession.SetPersistentData(varName, varValue)
```

Parameter	Type	Value/Meaning
varName	String	The name of the session variable whose value is to be set.
varValue	String	The value to be set for the named session variable.

Code Examples

Example 1

The following code fragments show the Page_Load method of a sample ASPX page. The page contains a single web control, a ListBox named listBooks. A non-pooled session is opened and the BOOKLIST XML document is called to return the data in the BOOKS TIBCO Object Service Broker table. The GetTable method is used to extract the data as an ADO.NET DataTable object. A "For Each" loop is used to extract the values of the TITLES attribute so that they can appear in the ASPX page's ListBox.

Visual Basic

```
Private Sub Page_Load( _
   ByVal sender As System.Object, _
   ByVal e As System.EventArgs) _
   Handles MyBase.Load

   Dim eSession As eCTSnet.eCTSsession = New eCTSnet.eCTSsession
   Dim sessParms As Hashtable = New Hashtable
   Dim row As DataRow

   sessParms("U") = "USR40"
   sessParms("P") = "USR40"
   sessParms("HOST") = "USR40"
   eSession.OpenApplication(sessParms)

   eSession.RunXmlDoc("BOOKLISTNET")
   Dim table As DataTable =
   eSession.GetTableFromDocument("BOOKLISTNET")
   For Each row In table.Rows
      listBooks.Items.Add(row("TITLE"))
   Next

   eSession.Dispose()
   eSession = Nothing
End Sub
```

C#

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

```
using eCTSnet;

namespace OIGDocExamp
{
   /// <summary>
   /// Summary description for WebForm2.
   /// </summary>
   public class WebForm2 : System.Web.UI.Page
   {
      protected System.Web.UI.WebControls.ListBox listBooks;

      private void Page_Load(object sender, System.EventArgs e)
      {
         // Put user code to initialize the page here
         eCTSsession eSession = new eCTSnet.eCTSsession();
         Hashtable sessParms = new Hashtable();
         sessParms.Add("U", "USR40");
         sessParms.Add("P", "USR40");
         sessParms.Add("HOST", "USR40");
         eSession.OpenApplication(sessParms,
            eCTSnet.eCTSsession.Mode.Pooled);
         eSession.RunXmlDoc("BOOKLISTNET");
         DataTable dt =
         eSession.GetTableFromDocument("BOOKLISTNET");
         foreach (DataRow row in dt.Rows)
            listBooks.Items.Add( (string)(row["TITLE"]));
         eSession.Dispose();
      }

      #region Web Form Designer generated code
      override protected void OnInit(EventArgs e)
      {
         // CODEGEN: This call is required by the ASP.NET Web Form
Designer.
         InitializeComponent();
         base.OnInit(e);
      }

      /// <summary>
      /// Required method for Designer support - do not modify
      /// the contents of this method with the code editor.
      /// </summary>
      private void InitializeComponent()
      {
         this.Load += new System.EventHandler(this.Page_Load);
      }
      #endregion
   }
}
```

Example 2

This example shows how you create an ASP.NET web service in Microsoft Visual Studio .NET, using the .NET class library for OIG.

First, open Visual Studio and create a new project. For the project type, select Visual Basic. For the template, select ASP.NET Web Service.

In the new project, add a reference to the .NET class library for OIG.

Next, in the generated .asmx file, add the following Visual Basic code:

```
<WebMethod()> Public Function GetBookDetails(ByVal BookID As
String) _
As DataTable
   Dim oSession As eCTSnet.eCTSsession = New eCTSnet.eCTSsession
   Dim userParms As NameValueCollection = New NameValueCollection
   oSession.OpenApplication()
   userParms("KEY") = BookID
   oSession.RunTrans("BOOKDETADO", userParms)
   GetBookDetails = oSession.GetTable()
   oSession.Dispose()
   oSession = Nothing
End Function
```

Compile and run the project. In debug mode, the project presents a web-based interface that you use to test the web service.

Chapter 4 Using the OIG COM Component

This chapter describes how to use the OIG COM component.

Topics

Overview

Object Integration Gateway COM Component

Object Integration Gateway comes with a COM component that provides a simple interface between the rules engine and any COM-enabled components, such as Visual Basic applications and Active Server Pages. The COM component provides access to all the features of Gateway that are available from the servlet and the tag library, plus the ability to access data via an ADO recordset.

The Object Integration Gateway COM component contains one class, eCTSsession. This class exposes all the methods needed to write an application.

The examples in this chapter use Visual Basic or ASP syntax.

Configuring the Object Integration Gateway COM Component

Object Integration Gateway COM Configurator

The Object Integration Gateway COM component is automatically configured when you install it. However, you can change the COM component's configuration settings at any time, using the COM Configurator. This tool enables you to specify which Java Virtual Machine and which binaries the COM component uses. It also enables you to create aliases for a defined set of session parameters. You can then use the alias in the code of your client application to easily set the session parameters for your application.

When you install the Object Integration Gateway COM component, the setup program selects the most current Java Virtual Machine on the installation machine. If the JVM's directory location subsequently changes (because you install a newer version of the Java Runtime Environment, for example), you need to specify the new location of the JVM in the COM Configurator.

If you install a newer version of the JRE, you can simply delete the line in the Object Integration Gateway COM Configurator that specifies the location of the JVM. When you restart the COM Configurator, it automatically searches for and selects the latest JVM.

When you make changes to an alias definition, you can use the administration interface to refresh the session pool, thereby putting the changes into effect immediately. Refer to Appendix D, Using the OIG Administration Interface, on page 161 for more information.

Instantiating the Object Integration Gateway COM Component

Examples

Before invoking any Object Integration Gateway functionality, you must instantiate the COM component, as shown in the following examples.

In Visual Basic

Use the following syntax to instantiate a COM object in Visual Basic code:

```
Dim MySession As eCTSsession
Set MySession = New eCTSsession
'Invoke methods...
MySession.Close  'Closes the session
Set MySession = Nothing
```

On Active Server Pages

Use the following syntax to instantiate a COM object in ASP code:

```
Dim MySession
Set MySession = Server.CreateObject("eCTScom.eCTSsession")
'Invoke methods...
MySession.Close  'Closes the session
Set MySession = Nothing
```

Passing Data to the Object Integration Gateway COM Component

Overview

In the following method descriptions, the required format for parameters is explained and the parameter syntax for the COM component methods is described.

Strings

A simple string value:

```
Transaction = "DEMOTX"
```

String Arrays

The COM object can accept string arrays in two formats. The first is an array of arrays, and the second is a two-dimensional array. The first format is an easy-to-use format that does not require any variable to be declared. The second format requires that a variable be defined, but the advantage of this is that the defined variable and its values can be reused within the application.

Format 1: Array of Arrays (Object Integration Gateway 1.x Compatible)

Example:

```
MySession.RunTrans "DEMOTX", Array(Array("NAME", "VALUE"))
```

Format 2: Two-Dimensional Arrays

```
Array(x, y) = "stringvalue"
```

where *x* is the vertical dimension of the array (indexed from zero) and *y* is the horizontal dimension of the array (indexed from zero).

- Visual Basic example passing arrays of strings:

  ```
  Dim MyArgs(0,1) As String

  MyArgs(0,0) = "NAME"
  MyArgs(0,1) = "VALUE"
  MySession.RunTrans "DEMOTX", MyArgs
  ```

- ASP and Visual Basic example passing variant arrays:

  ```
  Dim MyArgs(0,1)
  ```

```
MyArgs(0,0) = "NAME"
MyArgs(0,1) = "VALUE"
MySession.RunTrans "DEMOTX", MyArgs
```

Recordsets

An ADO 2.5 or later Recordset object can be used both as input to and a result from Object Integration Gateway methods. In the case of the result recordset, this is a disconnected recordset. This means that it is not actively connected to the source database, so any updates made to the recordset do not get reflected in the source database, only in the recordset. However, this result recordset can be amended and used to pass data to another method call.

eCTSsession Methods

Opening Object Integration Gateway Sessions

OpenApplication

Opens an Object Integration Gateway session. An application must open a session to run a transaction or XML document. The OpenApplication method enables you to either create a new session, or borrow one from a pool of existing sessions. Pooled sessions are shared between one or more applications.

Syntax:

```
MySession.OpenApplication alias [,, mode]
```

```
MySession.OpenApplication , sessParms [, mode]
```

Parameter	Type	Value/Meaning
alias (optional)	String	The case-sensitive name of a defined alias for a set of session parameters. Aliases are held in the Windows registry. By default, sessions are pooled.
sessParms (optional)	String	The list of session parameters. Refer to Appendix D, Using the OIG Administration Interface, on page 161. By default, sessions are unpooled.
mode (optional)	String	If set to "POOLED", sessions are created once and pooled for reuse after being released. If set to "UNPOOLED", sessions are terminated after being released.

Re parameters You can pass either the *alias* parameter or the *sessParms* parameter, but not both. If you pass the *alias* parameter, the session is pooled by default, unless UNPOOLED is specified for the optional *mode* parameter. If you pass the *sessParms* parameter, the session is unpooled by default, unless POOLED is specified for the option *mode* parameter.

Re default behavior If you call the OpenApplication method without specifying either the *alias* parameter or the *sessParms* parameter, the default Gateway session parameter settings are used. The default session parameters are determined in one of the following two ways:

- From the DEFAULT alias settings, if an alias of that name was defined using the COM Configurator
- From the DEFAULT session parameter settings, if no DEFAULT alias is defined. For more information, refer to Appendix D, Using the OIG Administration Interface, on page 161

Closing Object Integration Gateway Sessions

Close

Immediately closes an OIG session. If the underlying session was pooled, the session is returned to the pool of sessions for re-use by another application. If the session was not pooled, the session is terminated and the TIBCO Object Service Broker resources associated with the session are released. If there is a transaction in progress, this transaction is terminated and any updates pending are rolled back. Refer to StartTran on page 50.

Syntax: `MySession.Close`

Invoking Processing

You can use the methods in this section to invoke functionality that is defined to Object Integration Gateway. Each of the objects (for example, transactions) can be explicitly invoked.

RunRule

Runs a rule within the current transaction. A transaction must be started by calling the StartTran method before a rule can be called.

Syntax: `MySession.RunRule rulename[, parameters]`

Parameter	Type	Value/Meaning
rulename	String	The name of the rule to be executed.

Parameter	Type	Value/Meaning
parameters (optional)	String array	An array of name-value pairs to be passed to the rule. The parameters specified for this argument override any parameters specified in the URL query string or passed by a POST from an HTML form.

RunTrans

Runs the named transaction.

Syntax: `MySession.RunTrans transactionname[, parameters]`

Parameter	Type	Value/Meaning
transactionname	String	The name of the transaction to be run.
parameters (optional)	String array	An array of name-value pairs to be passed to the transaction. The parameters specified for this argument override any parameters specified in the URL query string or passed by a POST from an HTML form.

RunXMLDoc

Obtains or extracts the named XML document.

Syntax: `MySession.RunXMLDoc xmldocname[, parameters]`

Parameter	Type	Value/Meaning
xmldocname	String	The name of the XML document to be obtained or extracted.
parameters (optional)	String array	An array of name-value pairs to be passed to the XML document. The parameters specified for this argument override any parameters specified in the URL query string or passed by a POST from an HTML form.

StartTran

Starts an OIG transaction. Subsequent RuleXmlDoc or RunRule calls run within the transaction. The transaction is terminated by the next StopTran or Close call. A call to RunTrans while a transaction is active results in the nesting of a TIBCO Object Service Broker transaction to run the OIG transaction. Only one started transaction can be active for an OIG session at one time. Starting a second transaction results in an error.

Syntax: `MySession.StartTran updateMode`

Parameter	Type	Value/Meaning
updateMode	Boolean	If this parameter is set to true, updates are permitted while running the transaction. Otherwise, updates are not permitted and locks are not taken for tables.

StopTran

Stops an OIG transaction.

Syntax: `MySession.StopTran commit`

Parameter	Type	Value/Meaning
commit	Boolean	If this parameter is set to true, updates made to tables during the transaction are applied to the MetaStor when the transaction is terminated.

Handling Data

You can use the methods in this section to pass data to and from Object Integration Gateway, either as a recordset (ADO 2.5 or later) or as an XML document.

GetDocumentFromTable

Returns the XML for an interface table returned by the last RunTrans, RunXmlDoc, or RunRule call as a string. Normally interface tables are returned as ADO Recordset using the GetTable method (refer to GetTable on page 51.)

Return type: String

Syntax: `dataset = MySession.GetDocumentFromTable tablename`

Parameter	Type	Value/Meaning
tablename	String	The name of the table to be returned as a string.

GetTable

Converts an XML document from the last Run method call to an ADO Recordset object, and returns the recordset object. The XML document must be of type MSSCHEMA.

Return type: ADO 2.5 Recordset object

Syntax: `recordset = MySession.GetTable [xmldocname]`

Parameter	Type	Value/Meaning
xmldocname (optional)	String	The name of the XML document to be extracted. The XML document must be of type MSSCHEMA. If this parameter is not specified, GetTable converts the first XML document in memory.

GetTableFromDocument

Converts an XML document from the last Run method call to an ADO Recordset object, and returns the recordset object. The XML document must be of type MSSCHEMA and is produced as a result of running an XML document rather than as a result of the Pass Data to Client option of an interface table.

Return type: ADO 2.5 Recordset object

Syntax: `recordset = MySession.GetTableFromDocument docname`

Parameter	Type	Value/Meaning
docname	String	The name of the XML document returned as an ADO recordset.

GetTableOfContent

Returns all data returned by the last RunTrans, RunXmlDoc, or RunRule call as an ADO Recordset. This includes interface tables, XML documents, and persistent variables.

The table-of content information related to persistent data is present only if the OIG session has been opened with debug mode specified. Refer to the session parameter DEBUG on page 114 for more information.

The Recordset returned contains three columns, each row of the table describing a result returned.

Columns	Type	Description
Name	String	Contains the name of the OIG result.
Type	Integer	Identifies the result.
Value	String	Contains the value of the returned data.

Possible types of returned data are as follows:

Value	Description
3	XML for an interface table.
4	XML for a document.
10	Rules trace back in HTML format.
11	Rules trace back in text format.
30	Deleted persistent data.
31	Replaced persistent data.
32	Persistent data.
33	“Once only” persistent data.

Return type: ADO 2.5 Recordset object

Syntax:

```
recordset = MySession.GetTableOfContent
```

SetDocument

Stages one or more XML documents to be sent to OIG on the next Run method call.

Syntax: `MySession.SetDocument docname, document`

Parameter	Type	Value/Meaning
docname	String	The name of the OIG XML document definition used for processing the incoming XML document.
document	String	The XML document string to be passed to an OIG application.

SetDocumentFromFile

Stages an XML document to be sent to OIG on the next Run method call. The source of the XML document is read from the file identified by the *filename* parameter. The filename can be a complete path name or a filename that is relative to the current working directory.

Syntax: `MySession.SetDocumentFromFile docname, filename`

Parameter	Type	Value/Meaning
docname	String	The name of the document to be passed to TIBCO Object Service Broker.
filename	String	The name of file containing the XML document.

SetTable

Passes the contents of an ADO Recordset object to a TIBCO Object Service Broker data table. The SetTable method converts the recordset to an XML document of type MSSCHEMA, so that when the next OIG object is invoked, the XML document is sent along with any other parameters and loaded into the named TIBCO Object Service Broker data table.

You can use the SetTable method to pass data to TIBCO Object Service Broker that is required by an OIG object invoked by a subsequent Run method.

Syntax: `MySession.SetTable tablename, recordset`

Parameter	Type	Value/Meaning
tablename	String	The name of the TIBCO Object Service Broker table the recordset is loaded into when the next OIG object is invoked.

Parameter	Type	Value/Meaning
recordset	ADO 2.5 Recordset object	The recordset to be passed to TIBCO Object Service Broker.

When the destination table is a TIBCO Object Service Broker screen table, and the subsequently invoked OIG transaction has an associated TIBCO Object Service Broker screen, the data is loaded into that screen table for the screen named in the OIG object definition.

Handling Other Data

In some situations, the result from the Run method is not to be processed as a recordset. In these cases, the following methods manipulate the result string.

GetDocument

Returns an XML document generated by the last execution of the RunTrans or RunXMLDoc methods.

Return type: String

Syntax: `xml = MySession.GetDocument [xmldocname]`

Parameter	Type	Value/Meaning
xmldocname (optional)	String	The name of the XML document returned. If this parameter is not specified, GetDocument returns the first XML document in the result set.

GetEndMsg

Returns the end message from the last execution of RunTrans or RunXMLDoc. The GetEndMsg method does not take parameters.

The GetEndMsg method is typically used after executing a Run method.

Return type: String

Syntax: `msg = MySession.GetEndMsg`

GetResult

Returns the HTML generated by the last execution of a Run method. The GetResult method does not take parameters.

Return type: String

Syntax: `html = MySession.GetResult`

ASP Example:

```
MySession.RunTrans "DEMOTX"
Response.Write = MySession.GetResult
```

Managing Persistent Data for Web Sessions

When using OIG with web applications that use Active Server Pages or Visual Basic, it is a common technique to store application state information within the IIS Session object. You can use the following methods for OIG to access relevant session variables by setting up a list of variables that are automatically passed to a Run method. The data that is passed is available via the {*argname*} syntax when you define OIG objects, or via the ECTSGETARG and ECTSGETSESS utility rules. You can also use the ECTSSETSESS utility rule to update the IIS Session object. The IIS Session object is updated at the end of the Run method.

AddSessionParm

Adds a name to the list of session variables passed to the rules engine at execution of RunTrans or RunXMLDoc.

Syntax: `MySession.AddSessionParm varname`

Parameter	Type	Value/Meaning
varname	String	The name of the session variable to be added to the list.

RemoveSessionParm

Removes a name from the list of session variables passed to the rules engine at execution of RunTrans or RunXMLDoc.

Syntax: `MySession.RemoveSessionParm varname`

Parameter	Type	Value/Meaning
varname	String	The name of the session variable to be removed from the list.

Managing Persistent Data for Non-Web Sessions

OIG applications that run in a web environment, such as those using Active Server Pages, can use the AddSessionParm and RemoveSessionParm methods to manage a list of web session variables stored in the IIS Session object. However, OIG applications running in non-web environments do not have access to the IIS Session object or its variables. You can use the following methods for a non-web application to create and access a list of session variables that is stored by the COM component. These non-web session variables are stored as a collection of name-value pairs that is passed to, and can be updated by, any of the Run methods.

The following methods can also be called by applications running in a web environment such as Active Server Pages. In such cases, the session variables are stored as web session variables in the IIS Session object.

GetPersistentData

Returns the value of an OIG session variable.

Return type: String

Syntax: `value = MySession.GetPersistentData varname`

Parameter	Type	Value/Meaning
varname	String	The name of the session variable whose value is to be returned.

SetPersistentData

Sets the value of an OIG session variable. If the named session variable is not set, the variable is added to the list of session variables passed to the rules engine on subsequent Run method calls. If the session variable is set to an empty string, the variable is removed from the list of variables passed.

Web applications can also use this method to add to, or remove from, the list of web session variables normally managed by the AddSessionParm and RemoveSessionParm methods.

Syntax: `MySession.SetPersistentData varname, varvalue`

Parameter	Type	Value/Meaning
varname	String	The name of the session variable whose value is to be set.
varvalue	String	The value to be set for the named session variable.

Code Examples

Produces an XML document via the RunXMLDoc method, creates an ADO Recordset object from the RunXMLDoc method's result, and renders it as HTML.

```
<HTML>
<HEAD><TITLE>Object Integration Gateway COM Example
2</TITLE></HEAD>
<BODY >
<h2>Rows of table BOOKS</h2>
<p>Formating via Simple VB Script</p>
<TABLE BORDER="0">
<%@ LANGUAGE="VBSCRIPT" %>
<%
Dim eSession
Dim fld
Dim rs

' Create an ADO recordset based on the table BOOKS
Set eSession = Server.CreateObject("eCTScom.eCTSsession")
eSession.OpenApplication ,"U=USR40,P=USR40,HOST=USR40"
eSession.RunXMLDoc "BOOKLIST"
Set rs = eSession.GetTableFromDocument("BOOKLIST")

' Display the names of the fields of SCREEN as a title
Response.Write "<TR>"
For Each fld In rs.Fields
Response.Write "<TH bgcolor=PaleGreen>"
Response.Write fld.Name
Response.Write "</TH>"
Next
Response.Write "</TR>"

' Display the rows of BOOKS in a table
While Not rs.EOF
Response.Write "<TR>"
For Each fld In rs.Fields
Response.Write "<TD bgcolor=Wheat>"
Response.Write fld.Value & " "
Response.Write "</TD>"
Next
Response.Write "</TR>"
rs.MoveNext
Wend

' Cleanup the objects created
rs.Close
Set rs = Nothing
eSession.Close
Set eSession = Nothing
%>
</TABLE>
</BODY>
</HTML>
```

Chapter 5 Using OIG in Enterprise JavaBean (EJB) Environments

This chapter describes how to use OIG in Enterprise JavaBean (EJB) environments.

Topics

Overview

What Is an Enterprise JavaBean?

An Enterprise JavaBean (EJB) is a Java component that 'lives' within an Enterprise JavaBeans container running on a server. A client program creates an instance of the component by calling the create() method of its home interface. The create() method returns the EJB component's remote interface back to the client. The client can then access the services provided by the EJB component by calling methods on its remote interface.

How Does Object Integration Gateway Support Enterprise JavaBeans?

Object Integration Gateway provides the following components for building EJB applications:

- Base class for creating Gateway EJB session bean components
- EJB component home interface
- EJB component remote interface
- WebRowSet class for manipulating XML document data as recordsets
- Sample base class and interface extensions
- Sample deployment descriptor file

Object Integration Gateway EJB Components

ects2EJBbase Base Class

A ects2EJBbase base class is provided for Object Integration Gateway EJB session bean implementations. You extend this class to create your EJB session bean components.

This class contains methods for running Gateway objects. The source code for a sample class, ects2EJBSampleBean, is provided later in this chapter to illustrate how to extend the base class for your own EJB component class.

Methods

Method	Description
addSessionParm(String *persistKey*, String *value*) throws java.rmi.RemoteException	Adds a persistent data item to the session data collection.
addSessionParms(java.util.HashMap *persistData*) throws java.rmi.RemoteException	Adds a set of persistent data items to the session data collection.
deletePersistData() throws java.rmi.RemoteException	Deletes all persistent data items from the session data collection.
deletePersistData(String *persistKey*) throws java.rmi.RemoteException	Deletes a persistent data item from the session data collection.
ejbActivate() throws javax.ejb.EJBException, java.rmi.RemoteException	Implementation of the javax.ejb.SessionBean interface method.
ejbCreate() throws javax.ejb.EJBException, java.rmi.RemoteException	Implementation of the javax.ejb.SessionBean interface method. Create without a session pool.
ejbPassivate() throws javax.ejb.EJBException, java.rmi.RemoteException	Implementation of the javax.ejb.SessionBean interface method.
ejbRemove() throws javax.ejb.EJBException, java.rmi.RemoteException	Implementation of the javax.ejb.SessionBean interface method.

Method	Description
String getDocument(String *docname*) throws java.rmi.RemoteException	Returns the XML document from the last run method call. The XML document must be of type ROWSET or MSSCHEMA.
String getEndMsg() throws java.rmi.RemoteException	Returns the end message set by the last Gateway operation.
java.util.HashMap getPersistData() throws java.rmi.RemoteException	Returns the session data collection.
String getResult() throws java.rmi.RemoteException	Returns the HTML contents of the results area.
String getSessionParm(String *persistKey*) throws java.rmi.RemoteException	Returns a persistent data item from the session data collection.
java.util.HashMap getSessionParms() throws java.rmi.RemoteException	Returns the session data collection.
WebRowSet getTable(String *docName*) throws java.rmi.RemoteException	Converts an XML document from the last run method call to a WebRowSet object, and returns the WebRowSet object. The XML document must be of type ROWSET or MSSCHEMA.
WebRowSet getTableFromDoc(String *docName*)	Returns a WebRowSet by parsing an XML document returned as a result of running a defined XML document.
openApplication() throws ects2EJBException	Obtains an unpooled session with the parameters specified in the deployment descriptor for this EJB.
openApplication(boolean pooled) throws ects2EJBException	Obtains a pooled or unpooled session with the parameters specified in the deployment descriptor for this EJB.
openApplication(java.util.HashMap parms, boolean pooled) throws ects2EJBException	Obtains a pooled or unpooled session with the parameters supplied in the parms argument. Parameters not supplied in the parms argument are provided by the deployment descriptor for this EJB.

Method	Description
removeSessionParm(String *persistKey*) throws java.rmi.RemoteException	Deletes a persistent data item from the session data collection.
removeSessionParms() throws java.rmi.RemoteException	Deletes all persistent data items from the session data collection.
runRule(String *RuleName)* throws java.rmi.RemoteException	Runs the named rule within the context of the Object Integration Gateway rules framework. The named rule can access data processed via setTable and setDocument. The named rule can also produce and return XML documents, and so on.
runTrans(String *transname*) throws java.rmi.RemoteException	Runs the named Gateway transaction.
runXMLDoc(String *docname*) throws java.rmi.RemoteException	Runs the named Gateway XML document.
setDocument(String *docname*, String *document*) throws java.rmi.RemoteException	Stages one or more XML documents to be sent to the Gateway on the next run method call.
setDocument(String *docname*, WebRowSet *wrs*) throws java.rmi.RemoteException	Stages one or more XML documents to be sent to the Gateway on the next run method call.
setPersistData(java.util.HashMap *persistData*) throws java.rmi.RemoteException	Adds a set of persistent data items to the session data collection.
setTable(WebRowSet *wrs*) throws java.sql.SQLException, java.rmi.RemoteException	Passes the contents of a WebRowSet object to a TIBCO Object Service Broker data table. The method converts the WebRowSet to an XML document so that when the next Gateway object is invoked, the XML document is sent along with any other parameters and loaded into a TIBCO Object Service Broker data table. Since no table name argument is provided, the data table from which the WebRowSet was created is updated.

Method	Description
setTable(WebRowSet *wrs*, String *tablename*) throws java.sql.SQLException, java.rmi.RemoteException	Passes the contents of a WebRowSet object to a TIBCO Object Service Broker data table. The method converts the WebRowSet to an XML document so that when the next Gateway object is invoked, the XML document is sent along with any other parameters and loaded into a TIBCO Object Service Broker data table. Since a table name argument is provided, the specified data table is updated.
setUserData(String *userdata*) throws java.rmi.RemoteException	Sets user data (name-value pairs) passed with run method calls. Multiple name-value pairs must be separated using commas. A value that contains a space character must be enclosed in single quotation marks.
setUserData(String *name*, String *value*) throws java.rmi.RemoteException	Sets user data (name-value pairs) passed with run method calls. Each call to this method adds a name-value pair to the userdata collection passed on the next interaction. There are no restrictions on the naming of user data; the same name can be used multiple times.
startTran(boolean update) throws java.rmi.RemoteException	Starts a bean-managed transaction that can span multiple interactions. The update parameter controls the mode of the TIBCO Object Service Broker transaction started.
stopTran(boolean commit) throws java.rmi.RemoteException	Stops a bean-managed transaction with either rollback or commit processing.

Home Interface

The home interface exposes methods for creating instances of the EJB component class (your extensions of the Object Integration Gateway base class). When a client makes a call to a create() method of the home interface, the return value is the remote interface of your EJB component class.

Methods

Method	Description
ejbRemoteInterface create() throws javax.ejb.CreateException, java.rmi.RemoteException	Causes the EJB server to instantiate an EJB session bean that does not participate in session pooling. Returns an instance of this EJB's remote interface.

Remote Interface

The remote interface exposes to the client the methods of your EJB session bean component. These are the methods that actually do the work of your Object Integration Gateway application. The remote interface includes methods to run Gateway objects such as transactions.

Deployment Descriptor

The deployment descriptor is an XML document that contains information about the EJB components you want to deploy to your EJB container. It should be packaged at the root of your applicationname-ejb.jar file (where *applicationname* is the name of your EJB application).

In the deployment descriptor you specify the JNDI names of your EJB component, home interface, and remote interface classes. You can also specify Object Integration Gateway session initialization parameters, and any other values you want passed to your EJB component.

WebRowSet Class

You use WebRowSet objects to manipulate XML document data as recordsets. The WebRowSet class is an implementation of the javax.sql.RowSet interface. For information about the methods available in WebRowSet objects, refer to "Methods" on page 61.

Following are some general notes you should be aware of when using WebRowSet objects.

Column References

Because TIBCO Object Service Broker programmers are accustomed to referencing fields by name, the WebRowSet class is extended to let users get data by column name as well as by column number. If a column label is provided in the XML document definition in the TIBCO Object Service Broker UI (as specified in the Root Name field on the Properties tab), this value, and not the column name, is used for the column label property.

Supported Schemas

The WebRowSet class supports two XML schema types: MSSCHEMA and ROWSET. MSSCHEMA is the preferred schema type of Microsoft, whereas ROWSET is the preferred schema type of Sun Microsystems. MSSCHEMA is the schema type used by Microsoft's ADO components to encode recordsets. The ROWSET schema is part of a prerelease of the JDBC 3.0 specification. Neither requires a document type definition for validation.

There are some interesting differences between the two schema types:

- MSSCHEMA is an attribute-based schema.
- ROWSET is a character-based schema.
- ROWSET schemas tend to carry much more metadata than MSSCHEMA. You could find that you need this metadata.

For various reasons, one schema outperforms the other in certain circumstances. You should experiment to determine which schema type best suits your application needs.

Data Integrity

Because the WebRowSet object is a completely disconnected recordset, modifications to its data are done without a transactional context. No locks are held, and the data is not guaranteed to remain unchanged while the rowset travels between various servers. It is up to the application programmer to use these disconnected recordsets in such a manner that data integrity is maintained.

Data Sources

A recordset can be created using a wide variety of data sources. You can then use the recordset to populate a WebRowSet object, and send the WebRowSet to the Object Integration Gateway to update a table. Sending data to the Gateway from external data sources is not a supported feature but it should work correctly as long as a compatible table definition can be created. This technique has risks that you need to be aware of:

- TIBCO Object Service Broker is not guaranteed to support all the data types available in other data sources. Incompatible fields can cause either the population of the WebRowSet to fail, or the consumption of the XML document to fail.
- Most unsupported data types cause a java.sql.SQLException to be thrown from a WebRowSet object.

Constructors

Constructors	Description
WebRowSet() throws java.sql.SQLException	Default constructor. Instantiates the WebRowSet but does not initialize any data structures.

Methods

Methods	Description
String getSchemaType()	Returns the schema type that the WebRowSet is currently set to use.
readXml(java.io.Reader reader, int docType throws java.sql.SQLException	Reads an XML document from the reader and initializes the contents for the rowset accordingly. This docType provided to this method relieves the parser of the burden of dynamically detecting the document' schema.

Methods	Description
readXml(java.io.Reader reader) throws java.sql.SQLException	Reads an XML document from the reader and initializes the contents for the ROWSET accordingly.
setSchemaType(String schematype) throws java.sql.SQLException	Set the schema type that the ROWSET is to use.
writeUpdateOnly(java.io.Writer writer) throws java.sql.SQLException	Writes the contents of the WebRowSet to generate an XML document consisting of only the changed, deleted or inserted rows.
writeXml(java.io.Writer writer) throws java.sql.SQLException	Writes the contents of the WebRowSet to generate an XML document.
writeXml(java.sql.ResultSet resultset, java.io.Writer writer) throws java.sql.SQLException	Uses the contents of the named WebRowSet to generate an XML document.

EJB Code Examples

This section presents code for a sample Object Integration Gateway EJB component class, its home and remote interfaces, and its deployment descriptor file.

ects2EJBSampleBean

The following is sample code for an EJB component class that extends the Gateway EJB base class.

```
/* A reference implementation of the OIG EJB sessionbean. */

package com.ObjectStar.ects;

import javax.ejb.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import sun.jdbc.rowset.WebRowSet;
import com.ObjectStar.ects.*;

public class ects2EJBSampleBean extends ects2EJBbase implements
SessionBean {
```

```
// empty constructor
   public ects2EJBSampleBean() {
   }

   // EJB Container required methods. These are not available
   // to the client program, but are used only by the container.

   public void ejbActivate()
      throws javax.ejb.EJBException, java.rmi.RemoteException {
      System.out.println("ects2EJBSampleBean.activate");
      super.ejbActivate();
   }
   public void ejbPassivate()
      throws javax.ejb.EJBException, java.rmi.RemoteException {
      System.out.println("ects2EJBSampleBean.passivate");
      super.ejbPassivate();
   }
   public void ejbRemove()
      throws javax.ejb.EJBException, java.rmi.RemoteException {
      System.out.println("ects2EJBSampleBean.remove");
      super.ejbRemove();
   }

   // Create with no pool. Called when client
   // calls "create()" on the home interface
   public void ejbCreate()
      throws javax.ejb.EJBException, java.rmi.RemoteException {
      System.out.println("ects2EJBSessionBean.create()");
      super.ejbCreate(null);
   }

   // Create with pool. Called when client calls "create(poolname)"
   // on the home interface
   public void ejbCreate(String poolname)
      throws javax.ejb.EJBException, java.rmi.RemoteException {
      System.out.println("ects2EJBSessionBean.create()");
      super.ejbCreate(poolnm);
   }
}
```

ects2EJBSampleHome

The following is sample code for the home interface class. The create () methods of the home interface return an instance of the EJB component's remote interface to the client.

```
/* A reference implementation of the OIG EJB sessionHome class. */

package com.ObjectStar.ects;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface ects2EJBSampleHome extends EJBHome {
   ects2EJBSample create()
```

```
      throws RemoteException, CreateException;
   ects2EJBSample create(String poolName)
      throws RemoteException, CreateException;
}
```

ects2EJBSample

The following is sample code for the remote interface class. The methods exposed in the remote interface must match the methods of the EJB component class. A client call to a remote interface method is executed by the corresponding method of the EJB component.

```
/* A reference implementation of the OIG EJB session remote
interface. */

package com.ObjectStar.examples;

import com.ObjectStar.ects.*;
import javax.ejb.EJBObject;
import com.ObjectStar.jdbc.rowset.*;

public interface ects2EJBSample extends ects2EJBRemoteIntf   {
   StringBuffer runTag(String tagName)
      throws java.rmi.RemoteException;
   StringBuffer runXMLDoc(String docName)
      throws java.rmi.RemoteException;
   StringBuffer runTrans(String tranName)
      throws java.rmi.RemoteException;
}
```

Deployment Descriptor

This is a sample deployment descriptor file for the ects2EJBSample EJB component.

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://www.java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
   <description>OIG EJB support</description>
   <display-name>ectsEJB</display-name>
   <enterprise-beans>
      <session>
      <description>no description</description>
      <display-name>ectsSample</display-name>
      <ejb-name>ects2EJBSample</ejb-name>
      <home>com.ObjectStar.examples.ects2EJBSampleHome</home>
      <remote>com.ObjectStar.examples.ects2EJBSample</remote>

<ejb-class>com.ObjectStar.examples.ects2EJBSampleBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Bean</transaction-type>
      <env-entry>
         <env-entry-name>HOST</env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
```

```
   <env-entry-value>pipin</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>PORT</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>9068</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>USERID</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>HURON1</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>PASSWORD</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>HURON1</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>LIBRARY</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>ECTSSAMP</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>SEARCH</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>L</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>MAXSESSION</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>3</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>POOLTIMEOUT</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>4000</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>DATAIN</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>16384</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>DATAOUT</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>16384</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>XMLTRACEDIR</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value><null/></env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>DEBUG</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>false</env-entry-value>
</env-entry>
```

```
      <env-entry>
         <env-entry-name>TRACEMESSAGES</env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value><null/></env-entry-value>
      </env-entry>
      <env-entry>
         <env-entry-name>XMLPARSER</env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>org.apache.xerces.parsers.SAXParser</env-entry-va
lue>
      </env-entry>
      </session>
   </enterprise-beans>

<assembly-descriptor>
      <security-role>
         <description>Users</description>
         <role-name>users</role-name>
      </security-role>
   </assembly-descriptor>

</ejb-jar>
```

Chapter 6 Using the OIG JCA Adapter

This chapter describes how to use the OIG JCA adapter.

Topics

Overview

What Is J2EE Connector Architecture?

JCA adapters are just simple Java archives that have the file extension .rar instead of .jar. The structure of the archive looks like this:

```
oigadapter.rar
   |
   + - oigadapter.jar
   |
   + - ects.jar
   |
   + - xerces.jar
   |
   + - META-INF
        |
        + - ra.xml
```

The following is a brief description of each of the resource archive files that are listed above:

File	Description
oigadapter.rar	This file contains the implementations of the javax.resource.cci.* and javax.resource.spi.* interfaces. These interfaces embody the JCA specification.
ects.jar	This file contains Object Integration Gateway utility classes and the TIBCO Object Service Broker SDK (Java) classes.
xerces.jar	This file is the Xerces XML parser that is used for WebRowSet and generic XML support.
META-INF/ra.xml	This file is the deployment descriptor file for the adapter.

JCA Deployment Descriptor

What is the Deployment Descriptor?

The deployment descriptor is an XML document that contains information about the adapter, including deployment settings. For more information on JCA deployment descriptors, refer to the JCA 1.0 specification.

Deployment Descriptor Example

This section presents code for a sample Object Integration Gateway JCA deployment descriptor file.

Deployment Descriptor

The following is a code sample of the default deployment descriptor for the Object Integration Gateway JCA adapter:

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE connector PUBLIC '-//Sun Microsystems,
    Inc.//DTD Connector 1.0//EN'
    'http://java.sun.com/dtd/connector_1_0.dtd'>

 <connector>
  <display-name>OigAdapter</display-name>
  <vendor-name>Objectstar Software Inc.</vendor-name>
  <spec-version>1.0</spec-version>
  <eis-type>ObjectStar Integration Gateway</eis-type>
  <version>1.0</version>
  <resourceadapter>

   <managedconnectionfactory-class>
    com.ObjectStar.connector.cci.CciManagedConnectionFactory
   </managedconnectionfactory-class>
   <connectionfactory-interface>
    javax.resource.cci.ConnectionFactory
   </connectionfactory-interface>
   <connectionfactory-impl-class>
    com.ObjectStar.connector.cci.CciConnectionFactory
   </connectionfactory-impl-class>
   <connection-interface>
    javax.resource.cci.Connection
   </connection-interface>
   <connection-impl-class>
    com.ObjectStar.connector.cci.CciConnection
   </connection-impl-class>
   <transaction-support>
    LocalTransaction
   </transaction-support>
   <config-property>
```

```
 <config-property-name>HOST</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>pipin</config-property-value>
</config-property>
<config-property>
 <config-property-name>PORT</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>9068</config-property-value>
</config-property>
<config-property>
 <config-property-name>USERID</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>HURON1</config-property-value>
</config-property>
<config-property>
 <config-property-name>PASSWORD</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>HURON1</config-property-value>
</config-property>
<config-property>
 <config-property-name>LIBRARY</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>ECTSSAMP</config-property-value>
</config-property>
<config-property>
 <config-property-name>SEARCH</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>I</config-property-value>
</config-property>
<config-property>
 <config-property-name>DATAIN</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>8000</config-property-value>
</config-property>
<config-property>
 <config-property-name>DATAOUT</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>8000</config-property-value>
</config-property>
<config-property>
 <config-property-name>DEBUG</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>FALSE</config-property-value>
</config-property>
<config-property>
 <config-property-name>TRACEMESSAGES</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
 <config-property-value>
  C:\TEMP\TRACEMESSAGES.LOG
 </config-property-value>
</config-property>
<config-property>
 <config-property-name>XMLPARSER</config-property-name>
 <config-property-type>java.lang.String</config-property-type>
<config-property-value>
 org.apache.xerces.parsers.SAXParser
</config-property-value>
```

```
  </config-property>
  <config-property>
      </config-property>
  <authentication-mechanism>
   <authentication-mechanism-type>
    BasicPassword
   </authentication-mechanism-type>
   <credential-interface>
    javax.resource.security.PasswordCredential
   </credential-interface>
  </authentication-mechanism>
  <reauthentication-support>false</reauthentication-support>
 </resourceadapter>
</connector>
```

Working with the Deployment Descriptor

Changing Deployment Descriptor Settings

You only need to change the <config-property> and the <transaction-support> settings, under normal circumstances.

The following is a brief description of each of these settings:

Config-Property Settings

Setting	Abbreviation	Default Value	Description
DATAIN		32768	The size of the datain buffer to allocate for adapter sessions. This can be from 500 to 32 KB bytes.
DATAOUT		32768	The size of the dataout buffer to allocate for adapter sessions. This can be from 500 to 32 KB bytes.
DEBUG		FALSE	Set to TRUE or FALSE. Indicates whether debugging messages should be produced by the server. If the application server specifies an adapter log file, the debug messages are sent there; otherwise they go to the server's System.out stream. Because this writer is serialized, running in debug mode can impact server performance, particularly in an SMP environment.

Setting	Abbreviation	Default Value	Description
HOST	H	localhost	The host name of the Object Integration Gateway where the adapter connects.
LIBRARY	L		The library (if any) to use for the session.
PASSWORD	P	HURON1	The password that the adapter uses to log in.
PORT		9068	The port number of the osMon task on the HOST machine.
SEARCH	SEA	L	The search path for the adapter sessions.
TRACEMESSAGES		(empty string)	If the DEBUG parameter is set to TRUE and a filename is specified here, the contents of the datain and dataout buffers are written to this log file. There is a significant performance penalty for doing this.
USERID	U	HURON1	The user ID that the adapter uses to log in.
XMLPARSER		org.apache.xerces.parsers.SAXParser	The parser to use for operations involving XML. The Xerces SAX parser is the default.

Transaction-Support Settings

Setting	Description
LocalTransaction	Use to deploy the adapter to support the JCA 1.0 LocalTransaction contracts. The Gateway starts and stops LocalTransactions as per the deployment settings for the EJB's that access the JCA adapter.

Setting	Description
NoTransaction	Use to deploy the adapter without supporting transaction contracts. You are responsible for starting and stopping all transactions.

Many of these settings are particular to the application and the Gateway where the adapter is deployed. Some Gateways have facilities for modifying such settings as needed via an admin console, and others do not.

To modify these settings prior to deployment, you must extract the ra.xml file from the archive, modify it, and return it to the archive.

The following explains how you can modify these settings prior to deployment:

1. Open the Command Prompt.
2. Remove the deployment descriptor file with the command:
   ```
   jar xf oigadapter.rar META-INF/ra.xml.
   ```
3. Edit the META-INF/ra.xml file.
4. Return the edited file to the archive with the command:
   ```
   jar uf oigadapter.rar META-INF/ra.xml.
   ```

Some of these settings are not appropriate for use by all application sessions. You can override any or all of the parameter settings when you instantiate a connection in your application code. Applications have connections to multiple Object Integration Gateways, or to the same Gateway in a variety of modes.

When an instantiated connection has unique properties, it creates a new session. Creating multiple connections with unique properties that create new sessions defeats the session pooling mechanism and quickly exhausts the available session resources on the Gateway.

Installation and Deployment

The procedure for installing the adapter varies from server to server. Most servers can deploy the adapter as is, and some require a server-specific deployment descriptor to be included in the META-INF directory. Use the Gateway-specific deployment descriptors to specify values that are outside the scope of the JCA specification, such as pool size, idle timeout, and so on. The specification explicitly states a number of areas that server vendors implement in any way they choose, so long as the specification contracts are fulfilled. Refer to your application server documentation for information about these settings and for information about how to deploy JCA adapters.

As of J2EE 1.3, the definition of an application is expanded to include JCA adapters. Several examples are provided with Object Integration Gateway that demonstrate this. To make a JCA adapter part of an application, simply add a new <module> section to the application.xml file, and include the oigadapter.rar file in the application's .ear file.

This is an example of what such an application.xml would look like:

```
<application>
 <display-name>
  Object Integration Gateway 5.0 Sample Application
 </display-name>
 <module>
  <web>
   <web-uri>oigdemo.war</web-uri>
   <context-root>/oigdemo</context-root>
  </web>
 </module>
 <module>
  <ejb>oigdemo-ejb.jar</ejb>
 </module>
 <module>
  <connector>oigadapter.rar</connector>
 </module>
</application>
```

This is the easiest way to install an adapter and usually circumvents server-specific installation procedures.

Using the Adapter

One of the main advantages of using a JCA adapter in your application is that it provides a standard programming interface. Developers have a very short learning curve before they are able to use the adapter. In the case of the Object Integration Gateway adapter, the interface employed is the CCI interface described in the J2EE documentation for the javax.resource.cci package.

The following code sample shows one method of an EJB instantiating a JCA connection in its setSessionContext method:

```
public void setSessionContext(SessionContext sc) throws
EJBException

{
   this.sc = sc;
   String adapterJndiName = null;
   try
   {
      ic = new InitialContext();
      // ADAPTER is a deployment parameter for this ejb
      adapterJndiName = (String)ic.lookup("java:comp/env/ADAPTER");
      // lookup the connection factory for this adapter
      cf = (CciConnectionFactory)ic.lookup(adapterJndiName);
```

```
   }
   catch(NamingException ex)
   {
      debugPrint(ex.getMessage());
      ex.printStackTrace();
      throw new EJBException(ex.getMessage());
   }
}
```

The EJB now has a reference to the connection factory that it can to create a connection, like this:

```
private Connection getCCIConnection() throws ResourceException
{
   ConnectionSpec spec = new CciConnectionSpec(connectionParms);
   return cf.getConnection(spec);
}
```

The connection object represents a session that a method uses to perform some processing:

```
private void runInteraction(int interactionType,
      String interactionName)
      throws ResourceException
{
   try
   {
      con = this.getCCIConnection();
      Interaction ix = con.createInteraction();
      iSpec.setFunctionName(interactionName);
      iSpec.setFunctionType(interactionType);
      ((CciInteraction)ix).execute(iSpec);
      closeCciConnection();
   }
   catch(ResourceException ex)
   {
      printException(ex);
      if (con != null)
         closeCciConnection();
      iSpec.reset();
      throw ex;
   }
}
```

In this code sample, iSpec is a reference to the InteractionSpec object, which would normally be allocated in the EJB's create or init method, as follows:

```
iSpec = new CciInteractionSpec();
```

With CCI, all the EIS-specific information is contained in the InteractionSpec object. This enables a developer who is already familiar with CCI to use any JCA adapter simply by learning how to use the InteractionSpec object.

See Also The Javadoc file, which is installed as a downloadable zip file with your OIG installation, for information on OIG JCA Adapter APIs.

Chapter 7 Using the OIG Application Bean

This chapter describes how to use the OIG application bean.

Topics

OIG Application Bean

ects2AppBean Class

The Object Integration Gateway application JavaBean, ects2AppBean, is a standalone implementation of all the functions provided by the Gateway EJB base class. You use the Gateway application bean in a non-EJB environment where a standalone application or custom application server has to access the Gateway.

Constructors

Constructor	Description
ects2AppBean()	No-args constructor. Requires that you set session initialization parameters manually.
ects2AppBean(String *alias*, String *sessParms*, String *mode*)	*alias* - A session pool alias name. *sessParms* - Session parameters. *mode* - Set to "pooled" for session pooling.
ects2AppBean(String *alias*, java.util.Hashtable *sessParms*, String *mode*)	*alias* - A session pool alias name. *sessParms* - Session parameters. *mode* - Set to "pooled" for session pooling.

Methods

Method	Description
addSessionParm(String *persistKey*, String *value*)	Adds a persistent data item to the session data collection.
addSessionParms(java.util.HashMap *persistData*)	Adds a set of persistent data items to the session data collection.

Method	Description
close()	Close the current TIBCO Object Service Broker session. **Note** A running transaction is rolled back prior to the session being closed.
getDocument()	Returns the first document in the result area.
String getDocument(String *docname*) throws ects2AppBeanException	Returns the XML document from the last run method call. The XML document must be of type ROWSET or MSSCHEMA.
String getDocumentFromTable(String *tableName*) throws ects2AppBeanException	Returns the named table's XML document from the message area.
String getEndMsg()	Returns the end message set by the last Gateway operation.
String getResult()	Returns the HTML documents returned from the last Object Integration Gateway interaction.
String getSessionParm(String *persistKey*)	Returns a persistent data item from the session data collection.
java.util.HashMap getSessionParms()	Returns the session data collection.
WebRowSet getTable() throws ects2AppBeanException	Returns a com.ObjectStar.jdbc.rowset.WebRowSet object via the "pass data to client" option of the Object Integration Gateway Transaction definition.
WebRowSet getTable(String *docname*) throws ects2AppBeanException	Converts an XML document from the last run method call to a WebRowSet object, and returns the WebRowSet object. The XML document must be of type ROWSET or MSSCHEMA.
WebRowSet getTable(String *tableName*) throws ects2AppBeanException	Returns a com.ObjectStar.jdbc.rowset.WebRow

Method	Description
WebRowSet getTableFromDoc(String *docname*) throws ects2AppBeanException	Returns a com.ObjectStar.jdbc.rowset.WebRowSet object constructed by parsing an XML document from the current interaction.
java.util.LinkedList getToc()	Returns a java.util.LinkedList containing the table of contents from the last Object Integration Gateway interaction. The table of contents can be used in diagnostic or recovery code to find out what the interaction returned when unexpected results are encountered.
boolean isOpen()	Returns a boolean indication of whether there is a current session or not.
openApplication() throws ects2AppBeanException	Opens an unpooled application based on default parameters.
openApplication(boolean pooled) throws ects2AppBeanException	Opens a pooled or unpooled application based on default parameters.
openApplication(Stringsessparms, boolean pooled) throws ects2AppBeanException	Opens a pooled or unpooled application based on default parameters.
openApplication(java.util.Hashtable sessParms, boolean pooled) throws ects2AppBeanException	Opens a pooled or unpooled application based on default parameters.
removeSessionParm(String *persistKey*) throws ects2AppBeanException	Deletes a persistent data item from the session data collection.
removeSessionParms()	Deletes all persistent data items from the session data collection.

Method	Description
runRule(String ectsRuleName) throws ects2AppBeanException	Runs a TIBCO Object Service Broker rule. The rule runs within the context of the Object Integration Gateway framework and can programmatically interact with all normal Gateway objects. This method is provided as a way of accomplishing some discreet rules processing within the context of an explicitly started transaction.
runTrans(String *transname*) throws ects2AppBeanException	Runs the named Gateway transaction.
runXMLDoc(String *docname*) throws ects2AppBeanException	Runs the named Gateway XML document.
setDocument(String *docname*, String *document*) throws ects2AppBeanException	Stages one or more XML documents for sending to the Gateway on the next run method call.
setDocument(String *docname*, WebRowSet *wrs*) throws java.sql.SQLException	Stages one or more XML documents for sending to the Gateway on the next run method call.
setDocumentFromFile(String *docName*, java.lang.String fileName) throws ects2AppBeanException	Stores a file-based XML document so that the next interaction sends it to the Gateway for parsing.
setTable(WebRowSet *wrs*) throws java.sql.SQLException	Passes the contents of a WebRowSet object to a TIBCO Object Service Broker data table. The method converts the WebRowSet to an XML document so that when the next Gateway object is invoked, the XML document is sent along with any other parameters and loaded into a TIBCO Object Service Broker data table. Since no table name argument is provided, the data table from which the WebRowSet was created is updated.

Method	Description
setTable(WebRowSet *wrs*, String *tablename*) throws java.sql.SQLException	Passes the contents of a WebRowSet object to a TIBCO Object Service Broker data table. The method converts the WebRowSet to an XML document so that when the next Gateway object is invoked, the XML document is sent along with any other parameters and loaded into a TIBCO Object Service Broker data table. Since a table name argument is provided, the specified data table is updated.
setUserData(String *userdata*)	Sets user data (name-value pairs) passed with run method calls. Multiple name-value pairs must be separated using commas. A value that contains a space character must be enclosed in single quotation marks.
setUserData(String *name*, String *value*) throws java.rmi.RemoteException	Sets user data (name-value pairs) passed with run method calls. Each call to this method adds a name-value pair to the userdata collection passed on the next interaction. There are no restrictions on the naming of user data; the same name can be used multiple times.
setUserid(String *parmvalue*)	Sets the value of the USERID session initialization parameter.

Method	Description
startTran(boolean update) throws ects2AppBeanException	Explicitly starts an Object Integration Gateway transaction. When started, a transaction can span multiple interactions of types runRule and runXMLDoc. Data updated as a result of these interactions is committed when the stopTran(true) method is called. The runTrans interaction is always atomic and any updates related to it are committed as part of a nested transaction.
stopTran(boolean commit) throws ects2AppBeanException	Explicitly stops an Object Integration Gateway transaction.

ects2AppBean Code Example

This example illustrates a simple compound transaction. If rules TEST_A and TEST_B update tables, and if these updates are pending, they are committed at the stopTrans statement.

```
package examples;
import com.Amdahl.Cli.*;
import com.ObjectStar.ects.*;
public class Ex1 implements Runnable {
    ects2AppBean appBean = null;
    java.util.Hashtable map = new java.util.Hashtable();

    /** Creates a new instance of traceback */
    public Ex1() {
    }

     /**
      * @param args the command line arguments
      */
    public static void main(String[] args) {
        Ex1 me = new Ex1();
        new java.lang.Thread(me).start();
    }
        public void run() {
             map.put("HOST","localhost");
             map.put("PORT","9068");
             map.put("USERID","HURON1");
             map.put("PASSWORD","HURON1");
             appBean = new ects2AppBean();
             try {
                // Open a pooled session.
               appBean.openApplication(map, true);
               //Start an update mode transaction
               appBean.startTran(true);
```

```
            //Run TEST_A but do not commit any updates.
            appBean.runRule("TEST_A");

            //Run TEST_b but do not commit any updates.
            appBean.runRule("TEST_B");

           //Stop the transaction and commit the data.
            appBean.stopTran(true);
            appBean.close();
        }catch(ects2AppBeanException aex){
            aex.printStackTrace();
        }finally{
            appBean.close();
        }
        return ;
    }
}
```

ects2Result Class

The ects2Result class encapsulates the results of all OIG interactions. All documents and error logs returned as a result of an interaction are stored in the ects2Result. In debug mode persistent session data are also stored.

Inner Class

Class	Description
ects2Result.TocEntry	The ects2Result encapsulates the results of all Gateway interactions. All documents and error logs returned as a result of an interaction are stored in the ects2Result. In debug mode, persistent session data are also stored.

Methods

Methods	Description
getTocEntryFromTable() throws com.ObjectStar.ects.ects2ResultException	Returns the table of contents entry for the first table document from the result area.
getTocEntryFromTable(String tableName) throws com.ObjectStar.ects.ects2ResultException	Returns the table of contents entry for the named table from the result area.

Methods	Description
getTocEntryXMLDoc() throws com.ObjectStar.ects.ects2ResultException	Returns the table of contents entry for the first XML document in the result area.
getTocEntryXMLDoc(String docName) throws com.ObjectStar.ects.ects2ResultException	Returns the table of contents entry for the named XML document from the result area.

ects2AppBeanException Class

The ects2AppBeanException class is a specialized exception class for the ects2AppBean class. It provides a convenient way of capturing and retrieving exception messages. Many ects2AppBean methods throw an ects2AppBeanException object when an error is detected.

Constructors

Constructor	Description
ects2AppBeanException(String msg, Throwable th)	A constructor requiring error message text and a cause exception. If the cause exception is an ects2sessionbeanException the logs from that exception are accessible from the getlog and print stack trace methods here.
ects2AppBeanException(String *message*)	A constructor requiring only error message text.
ects2AppBeanException(com.ObjectStar.ects.ects2sessionbeanException sbex)	A constructor requiring an ects2sessionbeanException cause exception.

Methods

Method	Description
String getHtmlErrorLog()	Returns the text rules error log for this exception, if one exists, formatted as HTML.
String getLog()	Returns the error log information for this exception, if one exists, formatted as either plain text or HTML based on session or user data containing an "HTMLMSG" indication.
String getMessage()	Returns the exception message.
String getTxtErrorLog()	Returns the plain text rules error log for this exception if one exists.
printStackTrace()	Prints error information to the System.err stream. If there is TIBCO Object Service Broker rules error information available it is printed. If not, a Java stacktrace is printed.
printStackTrace(java.io.PrintStream stream)	Writes error information to the indicated stream. If there is TIBCO Object Service Broker rules error information available it is written. If not, a Java stacktrace is written.
printStackTrace(java.io.PrintWriter writer)	Writes error information to the indicated writer. If there is TIBCO Object Service Broker rules error information available it is written. If not, a Java stacktrace is written.

Chapter 8 Using XAL

This chapter describes how to use XAL.

Topics

Overview

What Is XAL?

XAL stands for XML Abstraction Layer. XAL is a toolkit for converting TIBCO Object Service Broker text-based applications to web-based or GUI-based applications, with little or no modification of the text-based application code.

The primary difficulty faced by customers when first adopting OIG is that TIBCO Object Service Broker applications frequently have a great deal of display logic embedded in the business portion of their code. To use the business logic from such an application, much of the code must be restructured to separate the business logic from the display logic.

What XAL provides is an ability to run a web site or a dedicated GUI application, using only the data and metadata that exists in the TIBCO Object Service Broker screens of a text-based application. This provides a way for users to benefit from OIG immediately, while beginning to modify their applications.

How Does XAL Work?

XAL works by starting the text-based application in an OIG session. The XAL framework is invoked just prior to the application relinquishing control at the DISPLAY statement, and again just prior to its regaining control. In the intervening period the screen data is encoded in XML and sent to the client along with any SCREEN or SCREENTABLE metadata required. During this period the XAL framework is available to service a wide variety of requests from its client, including, but not limited to, requests for metadata relating to the currently displayed screen.

The XAL client, which is constructed on top of the OIG client, is able to modify the screen's XML-based data in any way supported by a TIBCO Object Service Broker screen before returning it to the XAL framework for consumption. Things like field display attributes are all taken care of by XAL and are available, on request, in the client tier. The client supports data tables to which GUI controls can be bound in the normal way. See the source code for the XAL sample application for data binding examples.

Because a web browser is similar to a text terminal in its mode of operation, it is expected that most XAL applications are web applications. To that end, XAL is designed to work with web application servers and a sample web application is provided with XAL.

Creating an XAL Web Application

Before you can develop and run an XAL application, you must first install either the OIG J2EE components (for Java environments) or the OIG .NET components (for .NET environments). Refer to Chapter 1, Getting Started with Object Integration Gateway (OIG), on page 1 for more information.

The fastest and easiest way to develop an XAL web application is to deploy the sample XAL application, and to modify it to work with your existing text-based application.

A sample JavaServer Pages (JSP) application for XAL is provided in the WebFrame.war file, which is included when you install the J2EE components for OIG. A sample XAL application for .NET is also available by request.

Deploying the Sample JSP Application

Use the following steps to deploy the sample JSP application for XAL:

1. Deploy the WebFrame.war application file to your servlet/JSP container, using the procedures specific to your container.

   For example, if you are using Tomcat, copy the WebFrame.war file into the application deployment directory, which is named webapps by default.

2. Start your servlet/JSP container.

3. Open your browser, and go to http://localhost:8080/WebFrame/Xal/

   This assumes that your servlet/JSP container is also running as the web server, and that it is listening on port 8080 for HTTP requests.

   At this point you should be presented with a web page that has a single Start XAL button on it.

4. On the web page, click the Start XAL button.

   A login page should open in a separate browser window that has no navigation buttons. To better emulate 3270-style navigation using PF keys, the navigation buttons are absent.

5. On the login page, enter your TIBCO Object Service Broker connection parameters, and the name of a startup rule for an existing text-based application. The rule must take no arguments and the rule name must already be entered into the @XAL_APPAUTH table on the TIBCO Object Service Broker system where you are connected.

   Entering the name of the startup rule in the @XAL_APPAUTH table authorizes you to run the text-based application as an XAL application.

6. On the login page, click the Start button.

   You should see the first screen of the text-based application, displayed as a web page.

   The text-based application is now running as an XAL web application.

How the Sample Web Application Works

The sample web application is a very simple JSP application. Here is a more detailed description of how it works:

1. Entering the URL for the sample application in your web browser takes you to the default page for the site: index.html.
2. The index.html page has a start button. Clicking the start button opens the login.html page in a new browser window that has no forward/backward navigation buttons. The navigation buttons are removed because a text-based application requires that you use PF keys to display screens. Using browser navigation buttons to move from screen to screen would 'confuse' the text-based application and cause it to fail.
3. In the login.html page you enter the TIBCO Object Service Broker connection parameters and the name of a no-argument startup rule for a text-based application. When you click the submit button, the connection parameters and rule name are passed to the login.jsp page.
4. The login.jsp page instantiates an XalSession object and initializes it with the parameters in the web.xml file, as well as the connection parameters and startup rule name passed from the login.html page. To specify your own startup rule, simply enter the name of the rule in the RULE field on the login.html page. Remember that the rule must take no arguments, and must be entered in the @XAL_APPAUTH table.
5. The login.jsp page calls the XalSession object's start() method, which starts the text-based application and allows it to run until the first DISPLAY or DISPLAY & TRANSFERCALL statement is encountered. At that point the XAL framework extracts the displayed screen's data and metadata and returns them to the XalSession object. When all this is done the start() method returns.
6. The login.jsp page instantiates an HtmlProducer object and calls its paintScreen() method. The paintScreen() method writes HTML to the browser's output stream. The result is that the browser renders a web page generated by the HtmlProducer object.
7. When you click a PF key, the form is posted to the response.jsp page. The response.jsp page instantiates an HtmlConsumer object that, in turn,

processes the fields in the response object such that the XalSession object is updated with all changed data and cursor location information.

8. The response.jsp page calls XalSession.processFcnKey(). This method validates the changed data, and sends it to the XAL framework running in the TIBCO Object Service Broker Execution Environment. The XAL framework updates all necessary screen data, positions the cursor, and returns control to the application.
9. When another DISPLAY statement is encountered the process repeats.

Unless an error is encountered, the response.jsp page processes both the display and response actions on every screen in the application.

Handling Errors

The sequence described in the previous section represents an error-free interaction. This section describes what happens in the sample web application if there is a validation error on a modified field.

1. When data is modified, the ScrRowSet object validates the data against the metadata for the screen table, and throws an XalValidationException if constraints are violated.
2. When data is modified, the ScrRowSet object validates the data against any stored reference table information.
3. If a field has a reference table, but the data is not cached in the XalSession object, the data is sent to the Execution Environment where the validation occurs when the screen table is updated. If validation fails, an XalValidationException object is thrown.
4. If an XalValidationException object is thrown in the demo application, the ValidationError.jsp page is used to display the error message. You can correct the input data and resume at this point.

Modifying the Sample JSP Application

There are three basic ways to modify the sample application to achieve the "look and feel" you want for your XAL applications:

- Modifying the style sheet
- Modifying the screen metadata documents
- Extending the HtmlProducer and HtmlConsumer classes

This section describes each of these methods.

Modifying the Style Sheet

The sample application includes a cascading style sheet, xal.css, that is used to control the appearance of the application. You can freely modify this style sheet, or create a new style sheet, for your own applications.

The xal.css sample style sheet is listed here with comments describing the function of each section.

```
/* Attributes particular to the helpkey, which is not an input
control, but a button. */
button.helpkey {
   text-align: left;
   font-family: monospace;
   color: blue;
}
/* Screentable data is placed in scrollable divs. In this case they
have a thin border so you can see their boundaries. */
div.screentable {
   border-style: solid;
   border-width: thin;
}
input {
   text-align: left;
   font-family: monospace;
   color: blue
}
/* Display attributes for a text field that is protected. */
input.disabledinput {
   color: red;
   background-color: white;
   border-width: 0
}
/* Display attributes for a text field that is editable. */
input.enabledinput {
   color: black;
   background-color: white;
   border-width: 1pt;
}
/* Display attributes for a text field that is invisible. */
input.invisibleinput {
   color: white;
   background-color: white;
   border-width: 0
}
input.submitbutton {
}
/* Fields with reference tables attached could be rendered as
select controls.
   The default HtmlProducer does not do this because of horizontal
alignment problems. */
select.referenceselect {
   color: white;
   background-color: green;
   border-width: 1px
}
```

Modifying the Screen Metadata Documents

The screen metadata documents are XML documents. By default, they are retrieved by the XAL client at runtime and parsed into a DOM. This DOM is used by the HtmlProducer class to generate the HTML for the application web page.

A screen metadata document is composed of elements, each of which has several attributes and, potentially, child elements. A screen metadata document must conform to the XAL.xsd XML schema document, which is included with XAL. You can use this schema to validate any changes you make. Several of the attributes in a screen metadata document can by modified to affect the way HTML is produced:

- Each element has an ENABLED attribute, which when set to false causes the element and all its children to be skipped by the HtmlProducer object.
- Widths and heights can be adjusted as necessary.
- Text boxes can have their type property set to SUBMIT, BUTTON, CHECKBOX, and so on. See the schema document for values.
- A text box with a reference field attached can have its type set to SELECT, which causes a select control composed of the contents of the reference table to be rendered.
- Any of the other properties can be modified. You can even modify any part of the document in ways not supported by the default HtmlProducer class and create a new custom producer class that supports the modification. You could have to extend the XML schema as well.

To generate a metadata document for a screen, you run the XAL_GO_DSN rule in a TIBCO Object Service Broker session started with DSBIFTYPE=TEXT. This rule takes the following three arguments:

- SCREEN — The screen name. The resultant file is {SCREENNAME}.xml.
- SCHEMA_URL — The URL that is placed in the XML document for validation purposes. You should copy the XAL.xsd file onto a local web server and provide that URL. If null, no schema is included in the document and no validation occurs.
- OUTPUTDIR — The directory where to write the metadata documents. If null, the session's DSN directory is used.

After using the XAL_GO_DSN rule to write the screen metadata document to a file, you can modify it.

To produce an HTML rendition of the currently displayed screen, the HtmlProducer object must obtain the DOM representation of the screen's metadata. It does this by calling the XalSession.getScreenDomMetaData() method, which returns the DOM for the currently displayed screen. If the DOM

for the screen is not cached, and if the XALMETADIR session parameter is specified in the web.xml file, the XalSession.getScreenDomMetaData() method builds the DOM by parsing the screen's metadata document located in the directory specified in the XALMETADIR session parameter. If the XALMETADIR session parameter is not specified, a default DOM is retrieved from TIBCO Object Service Broker. New DOMs are added to the cache.

Extending the HtmlProducer and HtmlConsumer Classes

The HtmlProducer and HtmlConsumer classes are used in the sample application JSPs to produce and consume HTML, and therefore play a significant role in determining the 'look and feel' of the application. The source code for these two classes is provided with XAL so you can modify it for your own purposes. Alternatively, you can create extensions of these classes and use them in your JSPs.

When creating or modifying Java classes for use in XAL applications, refer to the API documentation for the XAL components. The API documentation is provided in Javadoc format, and is available in your installation folder under OIG for J2EE.

See Also Appendix A, Setting OIG Session Initialization Parameters, on page 113 for information about XAL-specific session initialization parameters.

Chapter 9 Using the OIG Rules Programming Interface

This chapter describes how to use the OIG rules programming interface.

Topics

Overview

What Is the Rules Programming Interface?

The Object Integration Gateway rules programming interface, or RPI, is the standard interface by which you access the Gateway rules engine.

Object Integration Gateway is designed to provide a relatively easy way to create sophisticated web applications, rapidly, without writing any code. The main development tool, the TIBCO Object Service Broker UI, is easy to use, yet powerful. With the TIBCO Object Service Broker UI, you can build complex functionality into your web application. And when you need to develop specialized functionality that goes beyond what can be achieved strictly within the TIBCO Object Service Broker UI, you must work directly with the Gateway rules programming interface.

Working with the RPI

Working with the rules programming interface means writing rules, or modifying existing rules, to achieve the functionality you need. Basically, rules are the native programming instructions for the Object Integration Gateway rules engine. Rules tell the Gateway rules engine what tasks to perform, such as what data to access from which database tables, and what to do with the data.

To write or modify rules, you use the same tool you use to define data tables: the TIBCO Object Service Broker UI.

What You Need to Know to Work with the RPI

Before you can work with the rules programming interface, you need to know how to perform tasks such as defining tables and editing rules in the TIBCO Object Service Broker environment.

What Reference Material Is Provided?

The following table describes the manuals to use when you are programming the Object Integration Gateway rules interface. You access these manuals from our web site at http://support.tibco.com/ .

Topic	Manual
Defining and managing your data.	*TIBCO Object Service Broker Managing External Data*
Rules language statements and language usage.	*TIBCO Object Service Broker Programming in Rules*
Re-usable, shareable code for data and definition manipulation.	*TIBCO Object Service Broker Shareable Tools*

See Also

System-specific documentation appropriate to the operating environment where Object Integration Gateway is used.

Types of Rules

Rule Types

Object Integration Gateway rules are divided into the following basic types:

- Build rules
- Format rules
- Process rules
- Utility rules

Each of these rule types is described in the following sections.

Build Rules

Build rules perform work for Object Integration Gateway objects. There are two types of build rules: pre-build rules, which perform initialization work, and post-build rules, which perform termination work.

The following Gateway objects have the ability to run build rules:

- Transactions (pre-build and post-build rules)
- XML documents (post-build rules only, for document consumption only)

Pre-Build Rules

A pre-build rule is the first action executed, when an Object Integration Gateway object, such as a transaction, is run. Its purpose is usually to perform some initialization work for the Gateway object. For example, a pre-build rule can be used to initialize a data table with data, set variables used by the object, or perform validation.

Post-Build Rules

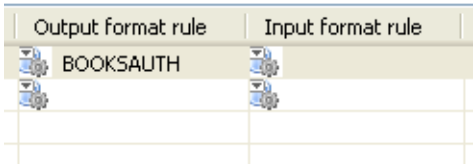
A post-build rule is the last action executed when an Object Integration Gateway object, such as a transaction, is run. You can use it for additional processing or clean-up work, after the object stops running.

Format Rules

Format rules transform, format, or otherwise process the data coming from a source data table, before displaying it on a web page. For example, if you want a date that is stored in the data table as 00/12/09 displayed as December 9, 2000, use a format rule.

Applying Format Rules

You can specify a separate format rule for each data field. You apply a format rule to a field using the TIBCO Object Service Broker UI, in the field map XML document. To apply a format rule to an XML document data field, enter the name of the rule in the **Output** field or **In** field of an XML field map, as shown in the following illustration.



In the **Output** field, you specify the name of a rule that formats the data contained in the field when producing the XML document. In the **In** field, you specify the name of a rule that formats the data parsed from the XML document being consumed.

Utility Rules

Accessing RPI Arguments

Object Integration Gateway provides the following utility rules to access argument data passed from the client web application.

value = ECTSGETARG(argname)

Returns the value of the parameter `argname` as passed from the client application. For example, the application could pass `argname` from a web browser via an HTTP POST or as part of the URL query string, or on a method call on the COM component. If `argname` is not found as a passed argument, the system looks for a session data value of the same name. Throws an ArgNotFound exception, if the parameter is not found.

value = ECTSRETURNARG(argname)

Same as ECTSGETARG, except that it returns an empty string if the parameter is not found.

Accessing Object Integration Gateway Session Parameters

Object Integration Gateway provides the following utility rules to access Gateway session parameters.

ECTSDELSESS(sessionparmname, sessionparmvalue)

Deletes the named value from both session context and the collection of variables provided to the Gateway for subsequent interactions.

value = ECTSGETSESS(sessionparmname)

Returns the value of the named Gateway session parameter.

ECTSREMSESS(sessionparmname, sessionparmvalue)

Deletes the named value from the collection of variables provided to the Gateway for subsequent interactions, but not session context.

ECTSSETONCE (sessionparmname, sessionparmvalue)

Sets the value of the named Gateway session parameter to the passed value such that it is placed in the web application's SESSION context, but not in the collection of variables provided to the Gateway for subsequent interactions.

ECTSSETSESS(sessionparmname, sessionparmvalue)

Sets the value of the named Gateway session parameter to the passed value.

Messaging

You use the following rules to control Gateway messaging.

ECTSMSG(message)

Sets message text that is sent to the client by a rule. By default, the message is sent via a persistent data parameter (session data on a web server) called ECTSMSG.

Execution/Generation

You use the following rules to execute or generate Object Integration Gateway objects.

See also the *TIBCO Object Service Broker Shareable Tools* manual.

XMLPARSE(docname, validate, docsource, docdata)

Begins parsing an XML document.

XMLSTART(xmldocname, predicate, parm)

Generates an XML document based on the passed data access arguments.

XMLSTARTDSN(outdsn, predicate, parm)

Generates an XML document based on the passed data access arguments, and places it in the specified file.

XMLSTARTSETDEST(tablespec, fieldspec)

Sets up the output table and field for XMLSTART.

XMLSTARTTAB(tablename, format, predicate, parm)

Returns a table to the OIG client.

Formatting and Linking

The following rules can be useful when writing format and link rules.

value = $TRANXMLSTRING(string)

For use with XML document format rules. Translates the given string into a format that is XML parser-safe, and returns it. For example, the string "x<y>z" could be interpreted as a valid XML tag. Normally to prevent this the data is automatically converted by Object Integration Gateway to an XML-safe format. However, in a format rule it could be desirable to return additional XML data as part of the string. In this case it is up to the developer to ensure that the result is valid and safe.

value = $URLENCODE(string)

For use with link rules. Some special characters are not permissible in a query string because they have meaning as part of the URL syntax. This rule translates the given string to a URL-safe format that can be used as an argument, and returns it.

Selecting an Application Profile

You can use the following rule to select an application profile. You use application profiles, which are defined in the TIBCO Object Service Broker UI, to specify default settings for an application.

ECTSSETAPP(appname)

Selects the named application profile for use with the current application.

Making an HTTP Request

Use the following tool to make an HTTP request.

See also the *TIBCO Object Service Broker Shareable Tools* manual.

value = $HTTPREQUEST(requesttype, url, header, data, result, message)

The rule returns the HTTP response code as an integer value. The return value has type C, syntax B, and length 4. A response code of 1000 indicates a non-HTTP error.

The following illustration shows an example of how $HTTPREQUEST can be called in a rule:

```
GETTEST();
-  LOCAL HTTPCODE, URL, HEADER, DATA, RESULT, MESSAGE;
_ ------------------------------------------------------------------------------
_ ------------------------------------------------------------------------------+-----
_ URL = 'HTTP://WWW.TIBCO.COM/POSTTEST.ASP?P1=1&P2=AAA';                        | 1
_ HTTPCODE = $HTTPREQUEST('GET', URL, HEADER, DATA, RESULT, MESSAGE);           | 2
_ CALL ENDMSG('RC=' || HTTPCODE || ', M=' || MESSAGE || ', R=' || RESULT);      | 3
_                                                                               |
  ------------------------------------------------------------------------------
```

Logging

Use the following rule to write messages to the Object Integration Gateway log file.

ECTSLOG

Writes a message to the Gateway log file.

RPI Variables

Object Integration Gateway Interface Variables

When a custom rule runs, it is executed in an environment that is not isolated within the Object Integration Gateway rules engine. This means that the rule has access to information about the environment where it is running and information that is specifically made available. This section describes local variables that are made available to custom rules and the information they can safely use. Be aware that other information is available to these variables, but there is no guarantee that this data is available across releases. Also, be aware that changing data outside the variables listed below is unsupported and can cause unexpected behavior.

Format Rule and Field Build Rule Variables

The following local variables are available when a format rule or field build rule is invoked:

NEW_FIELD_VALUE	The value field for which the format rule is invoked. Changing this value changes the value displayed.
OUTPUT	The current output buffer for the current field. Change with extreme caution.
RECORDCOUNT	The count of the number of rows processed so far.

Transaction Variables

The following additional local variables are available when an Object Integration Gateway object is invoked by a transaction:

TRANS	The name of the current transaction.
TXMODE	The mode of the current transaction (BROWSE or UPDATE).

Appendix A Setting OIG Session Initialization Parameters

This appendix describes the OIG session initialization parameters.

Topics

Session Parameters

You can set the following initialization parameters for an Object Integration Gateway session.

DATAIN

Specifies the datain size in bytes.

Valid Values	512 to 32768 bytes.
Default	32768 bytes

DATAOUT

Specifies the dataout size in bytes.

Valid Values	512 to 32768 bytes.
Default	32768 bytes

DEBUG

Specifies whether debugging messages are written to the System.out stream for significant internal events. It also causes session data to be included in the table of contents. The information provided in the system log can be requested by support if you have an issue under investigation.

Debug should be used carefully with high performance applications, because access to the System.out stream is serialized and can adversely affect performance.

Valid Values	TRUE, FALSE
Default	FALSE

HOST

The name of the host machine where the TIBCO Object Service Broker monitor process is running.

Abbreviated Name	H
Valid Values	0 to 64 characters.
Default	localhost.
Effects	HOST must be used in pair with the PORT parameter to fully identify the target TIBCO Object Service Broker monitor process (Open Systems) or Execution Environment (z/OS).

LIBRARY

The name of the local library for rule calls.

Abbreviated Name	L
Valid Values	0 to 8 characters.
Default	None.
Effects	The library specified here is searched when SEARCH=L is specified.

MAXSESSION

The maximum number of sessions that the Gateway can execute simultaneously.

Valid Values	Determined by the license key.
Default	1
Effects	Exceeding the limit specified by the MAXSESSION parameter setting results in the rejection of the offending HTTP request.

PASSWORD

The user's login password to be passed to TIBCO Object Service Broker. The value is case-sensitive on Open Systems.

Abbreviated Name	P
Valid Values	1 to 8 characters.
Default	HURON1

POOLTIMEOUT

The number of milliseconds the pool manager waits for a session to become available if all sessions in the pool are busy.

Valid Values	0 to 2147483647 milliseconds.
Default	0 milliseconds.
Effects	• If a session becomes available within the specified time, the pool manager returns the session to the requesting client. • If no session becomes available within the specified time, an exception is thrown indicating that MAXSESSION is exceeded.

PORT

The number of the TIBCO Object Service Broker monitor socket port.

Valid Values	0 to 65535.
Default	9068

PREFIX

The character value used as a prefix for generated user IDs.

Valid Values	1 to 7 characters.
Default	None.

Effects	• A generated user ID is composed of two parts: a character prefix that is constant, and a numeric suffix that is incremented. A generated user ID is always a total of eight characters long, so the number of digits used for the numeric suffix depends on the length of the prefix: if a prefix is, say, five characters long, three digits are used for the suffix (for example, HURON001). The digits start numbering at zero, and are incremented by one as each user ID is generated. • The number of user IDs that can be generated depends on the length of the prefix: if five characters are used for the prefix, three digits are used for the numeric suffix, accommodating a maximum of 1000 generated user IDs. • If no value is provided for the PREFIX parameter, user IDs are not generated. • You cannot specify an empty string for the PREFIX parameter.

SEARCH

The library search environment for the first rule to be executed.

Abbreviated Name	SEA
Valid Values	S, I, L
Default	L
Recommendation	Specify I or L only when one or more required rules reside in the site's installation or local library. Specification L causes overhead in finding rules; the local library must be searched whenever a new rule name is encountered for the first time in a transaction. Significant performance benefits can be realized when frequently used rules are promoted to the installation library.

Effects	If SEARCH=S is specified, TIBCO Object Service Broker searches for rules, external routines, or builtins in the following order: 1. The builtin library 2. The system library, as specified by the parameter SYSLIB 3. The external routines library that is contained in the ROUTINES table If SEARCH=I is specified, TIBCO Object Service Broker searches for rules, external routines, or builtins in the following order: 1. The installation library, as specified by the parameter INSTLIB 2. The builtin library 3. The system library, as specified by the parameter SYSLIB 4. The external routines library that is contained in the ROUTINES table If SEARCH=L is specified, TIBCO Object Service Broker searches for rules, external routines, or builtins in the following order: 1. The local library, as specified by the parameter LIBRARY 2. The installation library, as specified by the parameter INSTLIB 3. The builtin library 4. The system library, as specified by the parameter SYSLIB 5. The external routines library that is contained in the ROUTINES table

STANDBYWAIT

Specifies whether the start request of the Object Integration Gateway session is to wait for an available standby session if all the standby sessions are busy.

- A NO setting causes a reject of the request.

- A YES setting puts the request on a wait queue for a standby session. The STANDBYWAITLIMIT parameter controls the length of the wait time. See the description of that parameter in the *TIBCO Object Service Broker Parameters* manual. This value overrides STANDBYWAIT specified as a configuration or EE initialization parameter in TIBCO Service Object Broker.

This parameter is accepted by Open Systems TIBCO Service Object Broker on z/OS and Open Systems. However, it has no effect on Open Systems.

Valid User Values	Y, YES, N, NO
Default	Y
Effects	Effect on TIBCO Service Object Broker on z/OS: The exact behavior is defined by this parameter and the STANDBYWAITLIMIT and STANDBYWAITMSG settings of Object Service Broker. See *TIBCO Object Service Broker Parameters* for details. Effect on TIBCO Service Object Broker on Open Systems: None.

TRACEMESSAGES

Specifies the file to which debug tracing messages are written, if the DEBUG parameter is set to TRUE.

Default	Empty string.

USERID

The TIBCO Object Service Broker session user ID.

Abbreviated Name	U
Valid Values	1 to 8 characters.
Default	HURON1

XAL-Specific Session Initialization Parameters

The following session initialization parameters are specific to XAL sessions.

Parameter Name	Web or Standalone	Description
DATEFORMAT	Both	The default TIBCO Object Service Broker datemask for displaying date fields. Should be equivalent to the value in the session.prm file.
DEBUGDIR	Both	The name of a directory where debugging information for this session can be written. If a different directory is required for each session specify a unique value on the XAL constructor.
HTMLERRORLOG	Both	Require that error logs print in HTML format by default.
REFERENCECHECKING	Both	TRUE or FALSE depending on whether the user wants reference data to be collected and cached for screenfields with the autoprompt flag set. Because some applications use dynamic data for autoprompt fields and this data is cached at application context, it could be necessary to turn reference checking off in the web application. Reference checking is still be done in the execution environment if this feature is turned off.

Parameter Name	Web or Standalone	Description
SCREENRSMD_CACHE	Web only	This is the name of the collection that either exists or is to be created by the XAL session object for the purpose of storing ScrRowSet metadata documents. These documents are copies of the screen table definition and are cached at application context for performance reasons. This cache is an ideal candidate for persistence by the web application.
SCREENREFS_CACHE	Web only	This is the name of the collection that either exists or is to be created by the XAL component for the purpose of storing reference field information. Reference data is collected only if the REFERENCECHECKIG parameter is set to true (or not specified). This cache is an ideal candidate for persistence by the web application.
SCREENMD_CACHE	Web only	This is the name of the collection that either exists or is to be created by the XAL session object for the purpose of storing screen metadata documents. These documents reflect the definition of the TIBCO Object Service Broker screen and are cached at application context for performance reasons. This cache is an ideal candidate for persistence by the web application.
SCREENROWS	Both	The number of rows in the virtual screen used by the XAL session. Valid values are 0 to 43. This is a required parameter because the default is 0, which causes a display failure.

Parameter Name	Web or Standalone	Description
SCREENCOLS	Both	The number of columns in the virtual screen used by the XAL session. Valid values are 0 to 132. This is a required parameter because the default is 0, which causes a display failure.
XALMETADIR	Both	The name of the directory where the XAL session object should look for custom edited screen metadata documents. These documents are produced by the XAL_GO_DSN utility rule and can be hand edited to achieve screen-by-screen control of the rendering process.

Appendix B Understanding the Data Access Parameter Syntax

This appendix describes the syntax of the data access parameters.

Topics

Data Parameter Value Syntax

The data parameter value is provided in one or more comma-separated values that represent either the data value itself, or a reference to the data value.

The following example specifies two dynamic substitutions, `{ARG1}` (a numeric value) and `'{TABLEA}.{FIELDA}'` (a string value). If the value to be substituted is a string value, the expression must be enclosed in single quotation marks.

Example

```
'A Value', {ARG1}, '{TABLEA}.{FIELDA}'
```

Syntax in BNF Notation

```
<parm> ::=
   <parm value list>

<parm value list> ::=
   <parm_value{,parm_value}>

<parm_value> ::=
   <passed argument name>
   <table reference>
   <data value>

<passed argument name> ::=
   <{argument name}>

<table reference> ::=
   <{tablename}.{fieldname}>

<data value> ::=
   <Numeric Value>
   <Quoted String>

<Quoted String> ::=
   <'string'>
```

Data Key Value Syntax

The data key value provides table access predicates in a syntax very similar to the syntax used for rules, but without the table specification, the WHERE, or any ordering.

The following example specifies two dynamic substitutions, `{ARG1}` (a numeric value) and `'{TABLEA}.{FIELDA}'` (a string value). If the value to be substituted is a string value, the expression must be enclosed in single quotation marks.

Example

```
FIELD1 = 'A Value' AND FIELD2 > {ARG1} OR FIELD3 =
'{TABLEA}.{FIELDA}'
```

Syntax in BNF Notation

```
<key predicate> ::=
   <where not expression> {<logical operator>
   <where not expression>}

<where not expression> ::=
   [<not>] <where expression>

<where expression> ::=
   <where relation>
   (<where predicate>)

<where relation> ::=
   <field reference> <relational operator>
   <where expression>

<where expression> ::=
   [<unary operator>] <where expression term>
   {<add operator> <where expression term>}

<where expression term> ::=
   <where expression factor> {<multiplication operator>
   <where expression factor>}

<where expression factor> ::=
   <where expression primary> [<exponent operator>
   <where expression primary>]

<where expression primary> ::=
   <passed argument name>
   <table reference>
   <data value>

<passed argument name> ::=
   <{argument name}>

<table reference> ::=
```

```
   <{tablename}.{fieldname}>

<data value> ::=
   <Numeric Value>
   <Quoted String>

<Quoted String> ::=
   <'string'>
```

Appendix C Creating XML Documents

This appendix is a tutorial on how to create XML documents with TIBCO Object Integration Gateway and the TIBCO Object Service Broker UI. You must have installed and must be familiar with that UI. For details on its use, see the TIBCO Object Service Broker UI online help.

Topics

Performing the First Steps

The tree view in the TIBCO Object Service Broker UI OSB Projects view (or the Windows > Show View menu) shows two XML-related items: XML Documents and XML Field Maps. Choosing XML Documents with no filter applied displays a list of XML documents.

The documents you create are usually of the XML type. The `MSSCHEMA`, `ADO.NET`, and `ROWSET` document types are mostly for use within TIBCO Object Integration Gateway and you seldom need to create your own documents of those types.

TIBCO Object Integration Gateway assumes that the XML document being processed is parsed and placed into a relational table structure by TIBCO Object Service Broker for further processing and that the document is created by reading the processing results from a relational table. The overall process looks like this:

data entry → document → table → processes → table → document → exit

To map this process into the TIBCO Object Service Broker UI, create an XML document definition to define the XML document's format and content and the relationship to the underlying relational table or tables.

An XML document definition contains two parts:

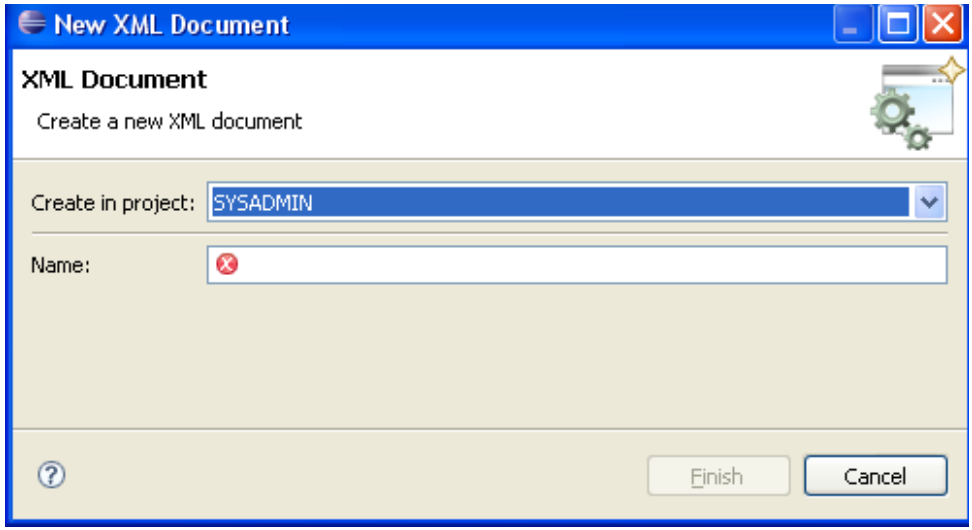
- The document definition itself.
- A table field map with the fields to be included in the document and their attributes.

The document definition can include other XML documents, called child documents, with which you process hierarchical data. For example, an XML document might list all the departments in your organization and a child document might list the employees within a department. A later section in this tutorial describes child documents in more detail.

Defining XML Documents

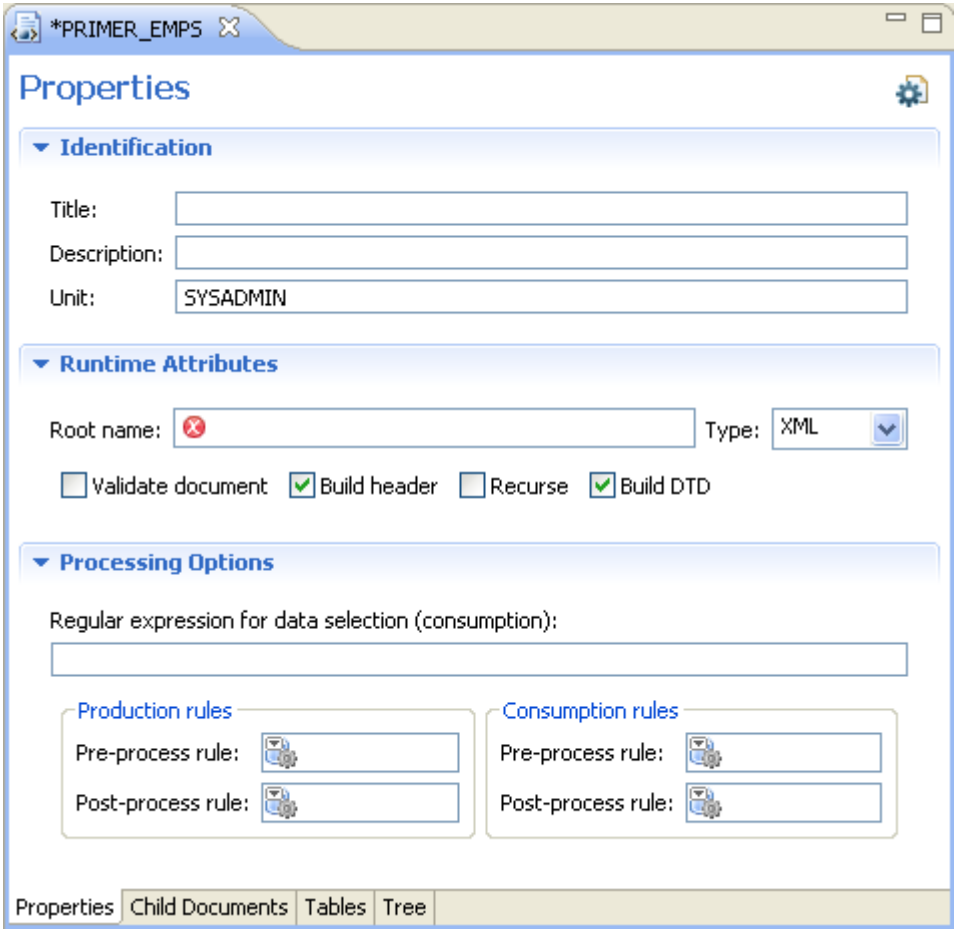
To define an XML document:

1. Choose File > New > XML Document to open the New XML Document wizard.



2. Type the name of the XML document in the Name field and click Finish.

A template is displayed.



3. Fill in the name, title, description, and unit.

   Even though the document name must be unique among all the XML documents defined on your instance of TIBCO Object Service Broker, that name can be the same as, for example, that of a table or transaction.

In the tutorial that follows, you create an XML document called `PRIMER_EMPS` to display all the employee data from the table `ECTS_EMPLOYEES`. You must fill in all the information in the Identification section.

Note that the Runtime Attributes section specifies the document type, which defaults to `XML`. Leave it unchanged for your document.

You must also specify the name of the opening and closing tags, which enclose all the occurrences, as a document root name in the “Root name” field under Runtime Attributes.

Every XML document must be well-formed, that is, you must enclose all its XML tags within an opening tag and a closing tag. Feel free to name those tags however you desire.

For this tutorial, the root name is `EMPLOYEES`. Once you have set the root name and saved it, your XML document definition is complete for now and you can add the field map.

Adding Tables

To add a table to the example XML document:

1. Make the source table, `ECTS_EMPLOYEES`, known to the example XML document: Click the Tables tab, locate the table in the Tables view, and then drag the table name to the area called Table.
2. Type `ects_employees` in the "Root name" text field.

   The table root name is the name assigned as the opening and closing tags for all the occurrences of the table to be included in the XML document, that is, the name of the tag that encloses one table occurrence. The table and document root names must be different.
3. Type `ECTS_EMPLOYEES` in the "Map name" text field.

   You will specify this map later.
4. **Optional.** To include all the fields even if they have null values, select Generate empty.

   Otherwise, if a field has a null value in an XML document, that field is omitted.
5. **Optional.** Select the **"**Null after update**"** option to clear the table buffer after each occurrence is loaded into the output table during the reading of an XML document.

   Otherwise, the buffer remains uncleared at the end of each occurrence. This selection is important only if some occurrences in the XML document contain missing fields. If the buffer is not cleared, a missing field from one occurrence acquires a value anyway from a prior occurrence.

   If all the fields are present for every occurrence, you can ignore this option.
6. Specify the parameters. For this tutorial, specify a selection criterion of `deptno=10` in the Key predicate field.

   If you parameterize the source table, the criteria for selecting occurrences and ordering field values apply only when you create an XML document. You can specify variable values identified by curly brackets, for example, `{REGION}`, constants, or both as the format of the entries here.
7. **Optional.** Specify the preprocess and postprocess production and consumption rules in their respective fields (Production Rules and Consumption Rules).

Object Service Broker executes the production rules when producing the document and the consumption rules when reading the document. for the production of the document.

In the rest of the fields, specify how you want the output table updated when the XML document is read, that is, when to commit the updates and whether the updates replace existing data. Do the following:

1. **Optional.** Select **"**Replace existing data**"** to cause the data from the XML document to overwrite the existing table data.

   Deselecting that checkbox means that you expect all the data occurrences in the XML document to be new, that is, the primary key values do not already exist in the table.

2. **Optional.** Select **"**Update at end of element**"** when specifying the options for updating the occurrences of child XML documents in relation to updating the occurrences of the parent document. This selection ensures simultaneous processing for all the updates to each occurrence.

   Deselecting that option means that processing of the updates from the child documents occurs as partial-occurrence updates, after which the updates from the parent document are processed separately. You usually select this option for a child document definition to prevent a partially complete occurrence from being added to the database.

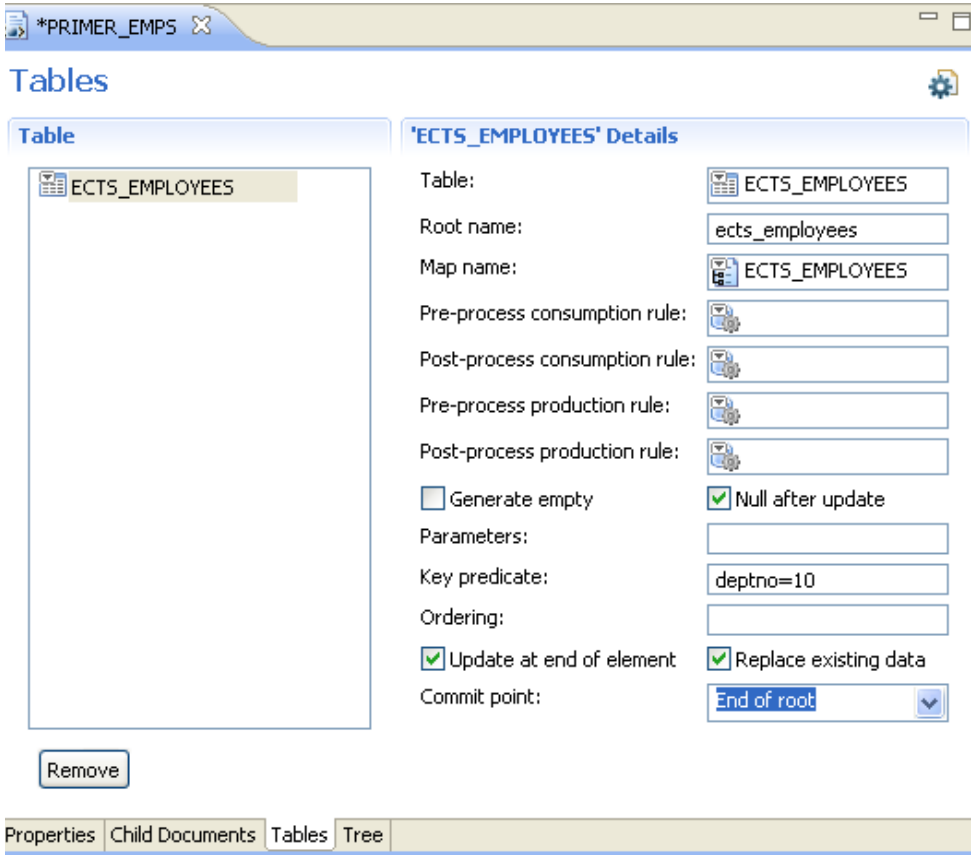
3. Select a commit point.

   The default is to commit updates at the end of all data processing or at every `COMMITLIMIT` exception, whichever occurs first. Selecting None means no explicit commit and that you opt for TIBCO Object Service Broker's default commit at the end of the transaction.

If you specify the attributes of a parent XML document, specify the handling of the child documents by performing either of these two steps:

- Select "End of root" to set a commit point at the end of each child document root.
- Select "After N roots" to set a commit point after every N roots from the child document. An additional option is then displayed, in which you specify a value for N.

A dialog box similar to this one is then displayed:



Save the XML document.

When producing XML documents, you can specify any table type for the definition. Remember, however, that this table then applies to both the produce and consume messages when it's part of an XML document.

TDS tables work well for static data. For dynamic data, we suggest TEM tables to avoid unnecessary updates and commits to TDS. Similarly, opt for TEM tables when consuming XML documents.

Adding Field Maps

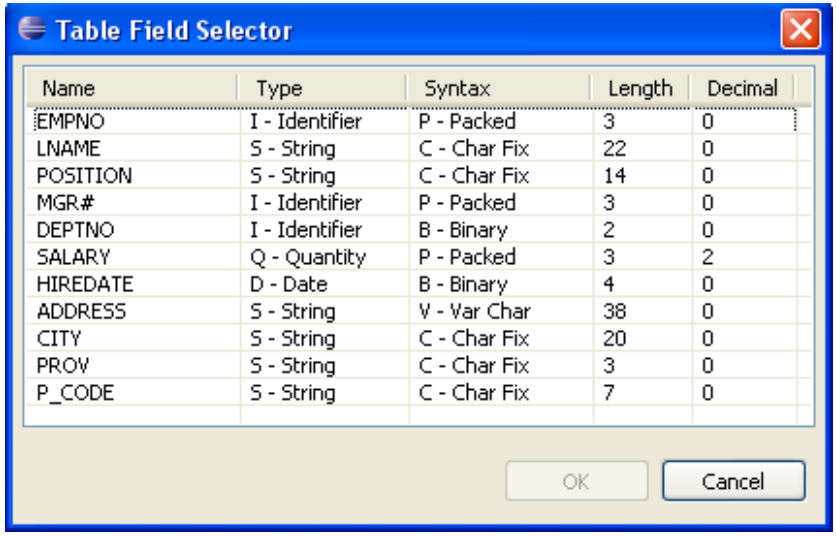
Now add a field map to your table for specifying fields in the XML document.

If the map already exists, click the icon beside the map in the Tables tab for the XML document name and choose Open from the menu. Otherwise, create a new XML Field Map in the XML Field Map wizard, as follows:

1. In the open XML Field Map editor, type the identification information on the Properties tab, if desired, and click the Field Map tab.
2. Click "Add from table" for a list of the fields associated with the field map.

   Note: To associate the XML field map name with a table, specify that name in the XML document's Tables tab.

   Your Table Field Selector dialog box should look like this:

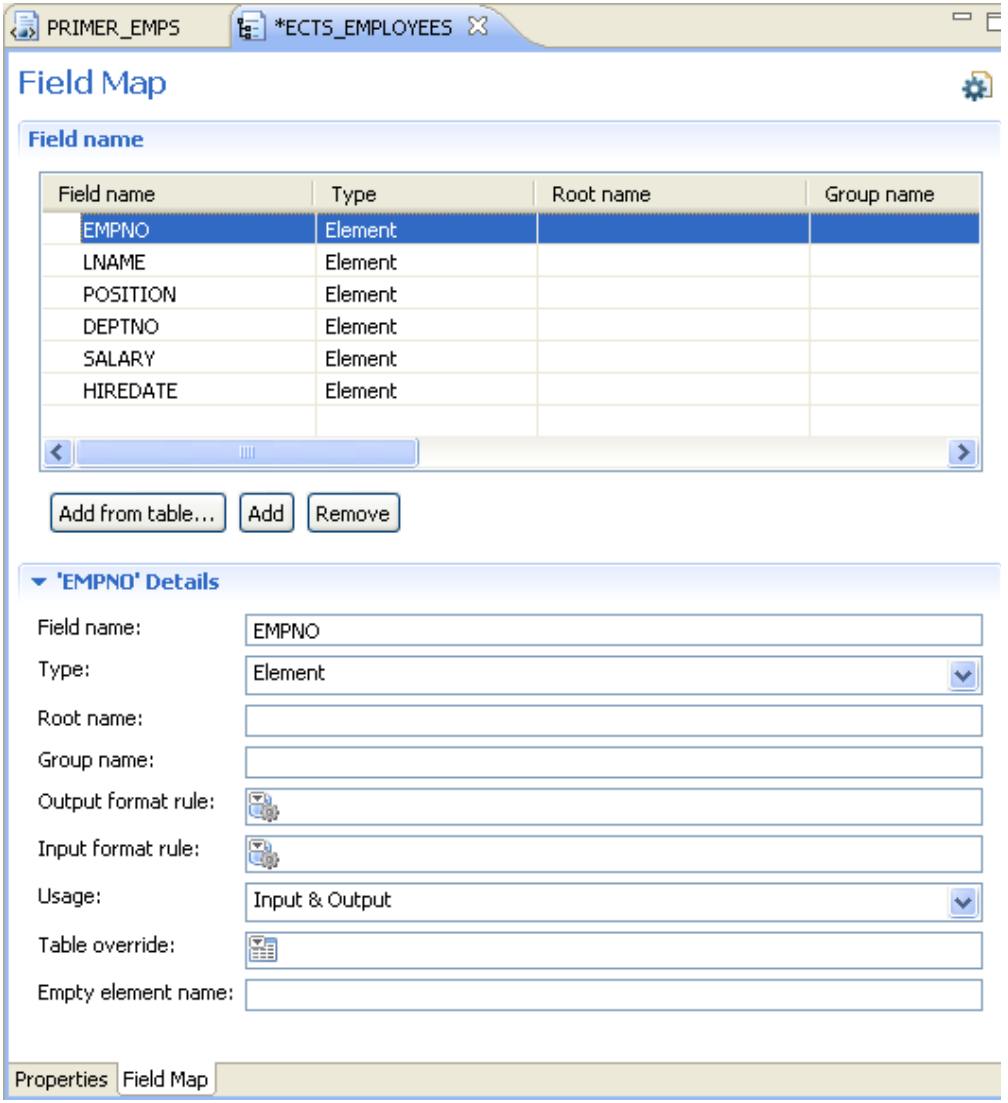


3. Select the fields you would like to include in your field map. For this tutorial, select all the fields up to and including `HIREDATE` except `MGR#`.

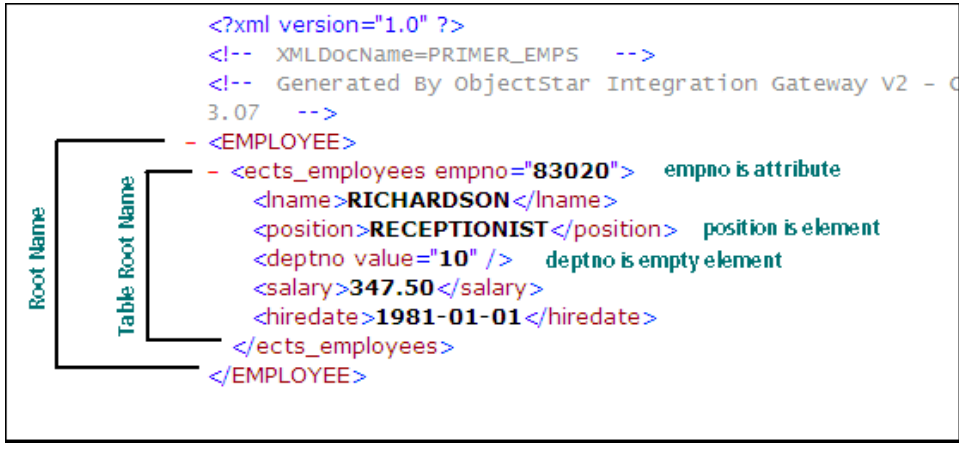
   You must exclude `MGR#` because it contains `#`, a special XML control character, which, if it is part of a field name, cannot be correctly handled by Object Service Broker. If your tables contain field names with the `#` character, rename those fields for the XML document in the Root Name column.

4. Close the Table Field Selector.

The selected fields are then displayed in the document field map, as shown in the following figure.

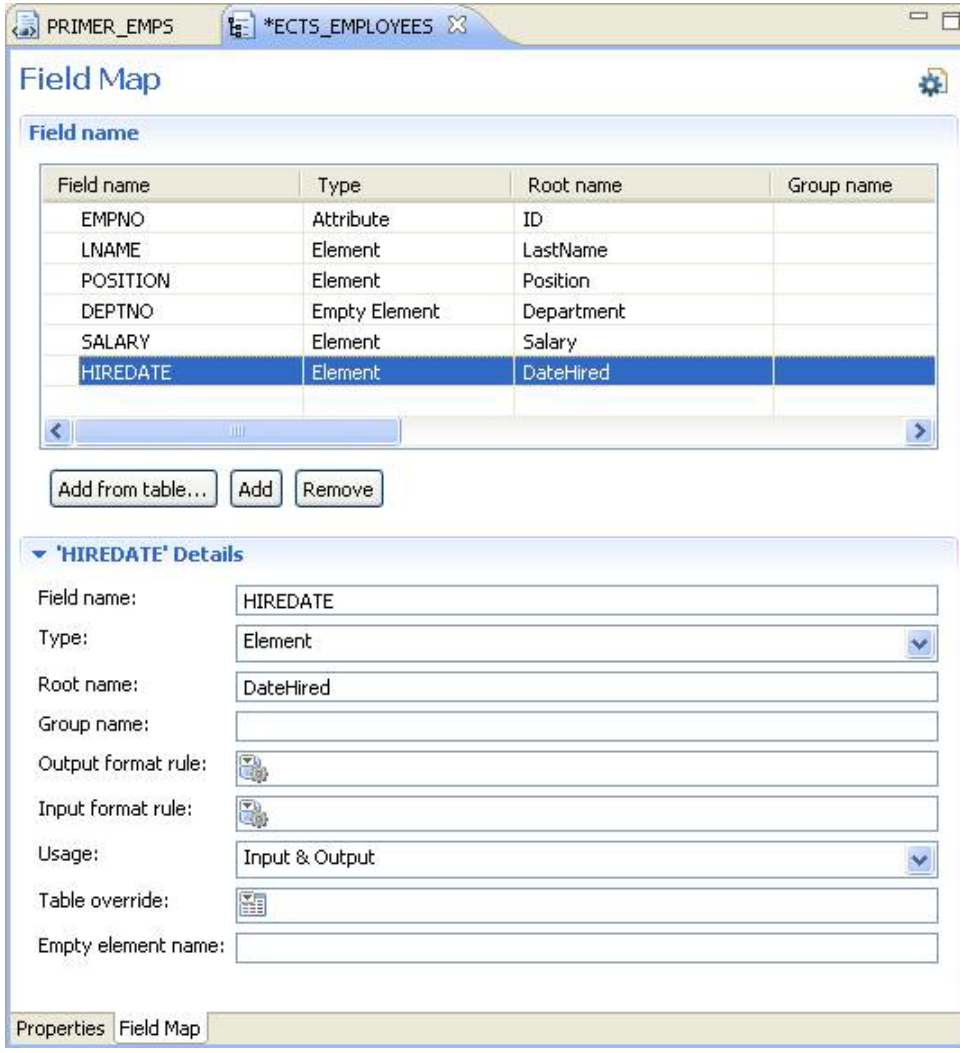


Next, assign an XML type to each field. After you've added the fields, each of them in the Field Map is, by default, assigned an XML type of `Element`. Your type choices are `Attribute`, `Empty Element`, and `Element`. What's their difference? The following figure shows the XML code generated for one occurrence from the table `ECTS_EMPLOYEES` with annotations to show how the three attribute types affect the generated XML.

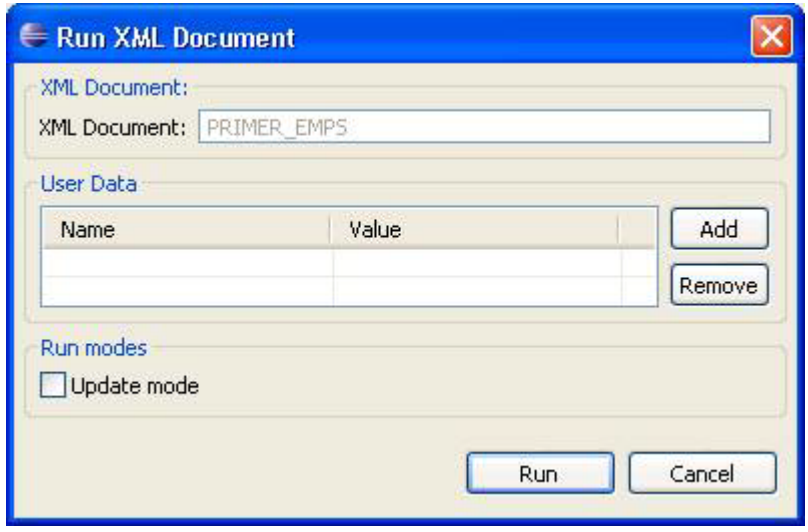


The field names that appear in the XML document are the actual table field names. You can rename any of them by specifying a new, case-sensitive name in the Field Map's "Root name" column. The name must **not** contain the # character or any spaces.

Here is an example of the Field Map that shows several different names in the same occurrence.



To execute an XML document, select the document and choose Run As from the short-cut menu in the XML Documents or OSB Projects view. The following figure shows the Run XML Document dialog box.



Here is a segment of the resulting XML document:

```
<?xml version="1.0" ?>
<!--  XMLDocName=PRIMER_EMPS  -->
<!--  Generated By ObjectStar Integration Gateway V
  -->
- <EMPLOYEE>
  - <ects_employees ID="83020">
      <LastName>RICHARDSON</LastName>
      <Position>RECEPTIONIST</Position>
      <Department value="10" />
      <Salary>347.50</Salary>
      <DateHired>1981-01-01</DateHired>
    </ects_employees>
  </EMPLOYEE>
```

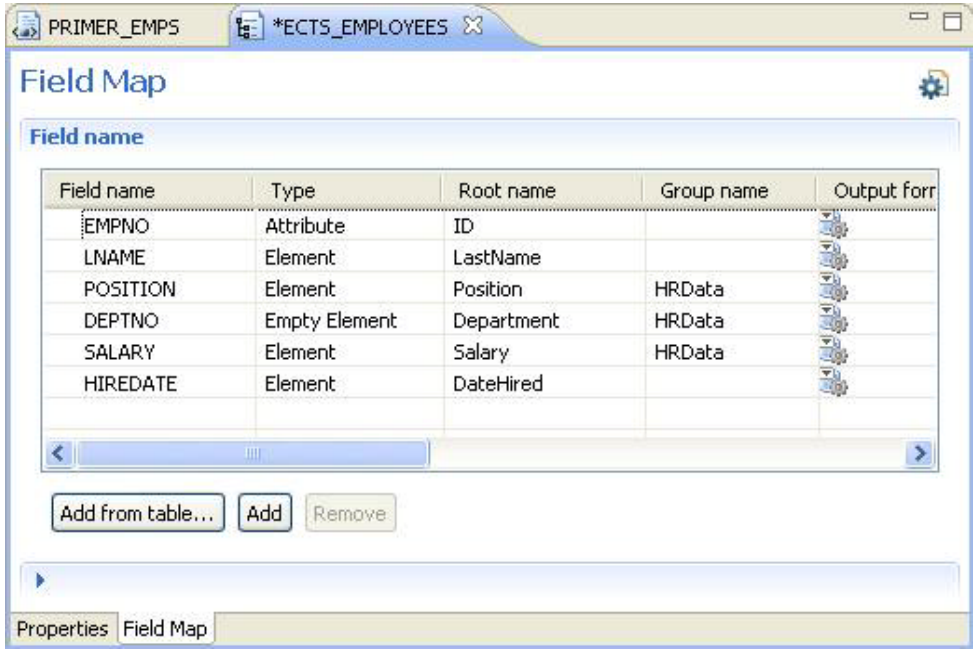
When an XML document is produced, the order of the elements changes if you add a group to the field map associated with the document. This is the intended behavior. The XML standard states that elements are not positional within a document. The receiving application must be able to determine how to handle each element according to the element name.

Understanding Other Field Attributes

You'll have seen several other columns in the field map. Learn about them, from left to right, in this section.

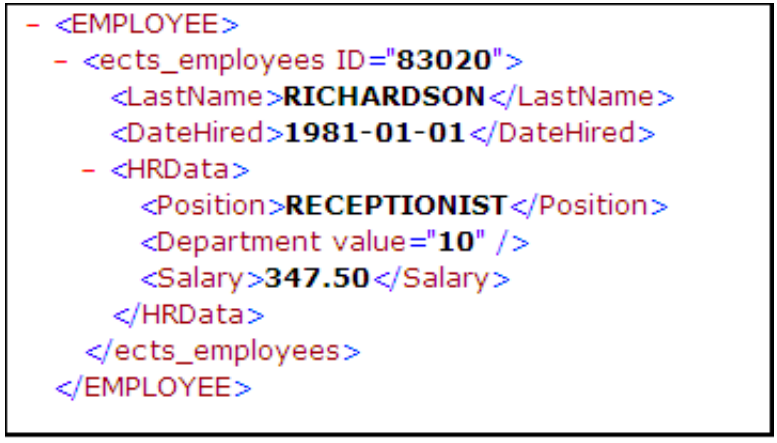
Group Name

Occasionally, you might have to have a starting tag and an ending tag that enclose a group of related fields. For example, you might want to collect the fields Position, Department, and Salary in a group called `HRData` to denote that they are the responsibility of the Human Resources Department, as illustrated here:



In this example, the three fields to be grouped happen to be in contiguous locations in the field map, but that is not a requirement. Fields are grouped based only on the case-sensitive value in the Group Name column. You can also set up multiple group names.

The following figure shows an example of the XML code.



Format Rules

In the two columns for format rules, you specify the name of a rule that reformats the field value. One column applies to the production or writing of the XML document and the other to its consumption or reading.

On entry to the rule, the local variable `NEW_FIELD_VALUE` contains the current field value. Your rule can change that value to the one you want displayed (on output) or passed to the database table (on input).

Usage

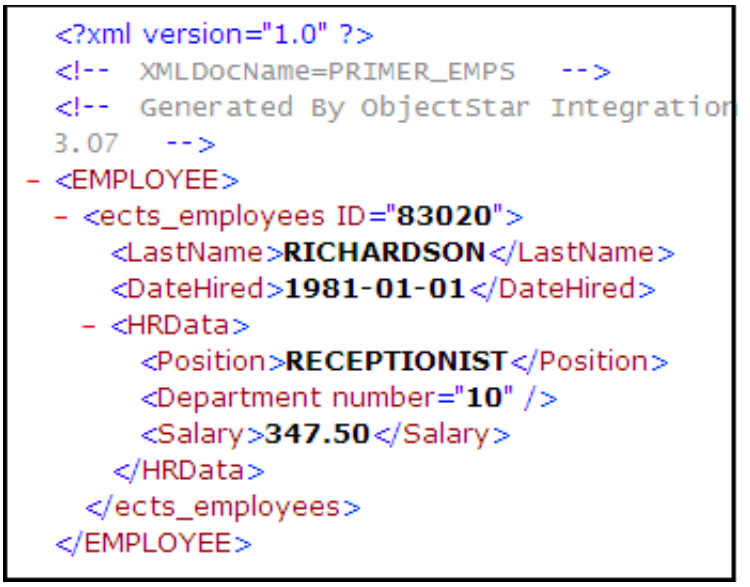
The Usage column defines the context for the field. For example, an XML document can contain fields that are present only on input and not on production of the document. For most of your applications, the fields have the default of Input & Output.

Table Override

When an XML document is being consumed or read, the data normally updates the table associated with the field map. To direct the data to a different table, specify the table name in this column. This step could be useful, for example, for saving incoming data away from production data until you have verified that the former fully conforms to your requirements.

Empty Element Name

In the display for Employee 83020, notice that the empty element DEPTNO appears as <Department value="20" />. You can specify a name in the Empty Element Name column to replace the word value, for example, replace it with number. See the following figure.



Understanding Child Documents

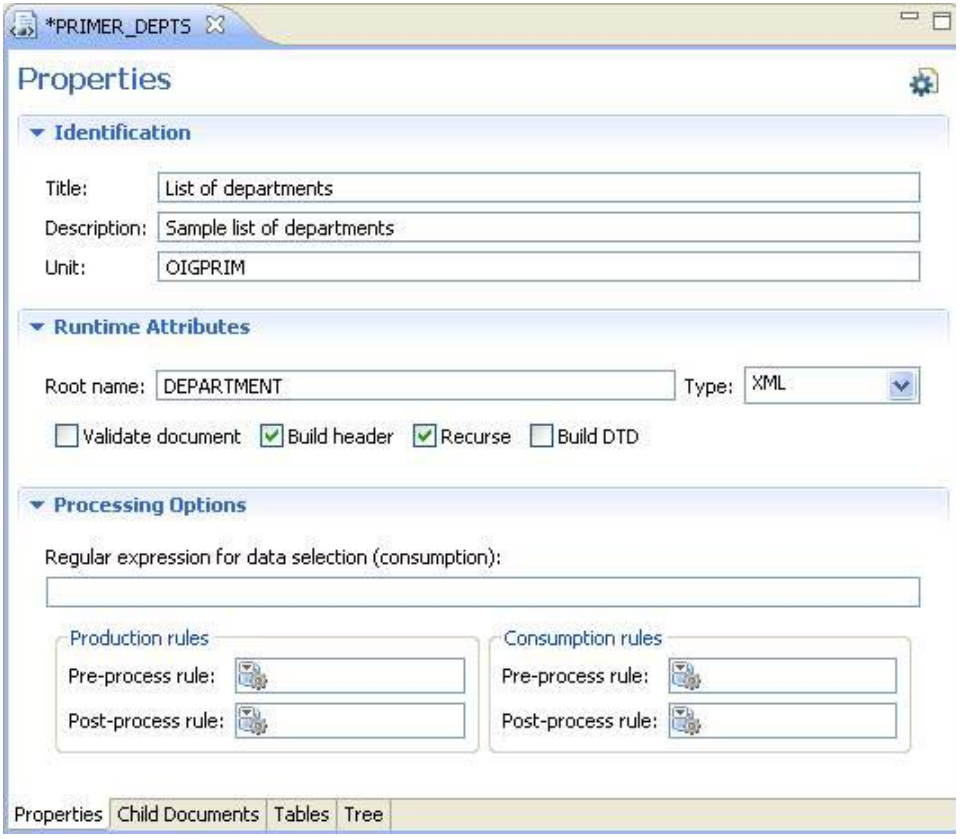
A child XML document is one that is attached to a parent XML document and that is processed at the end of each occurrence in the parent document.

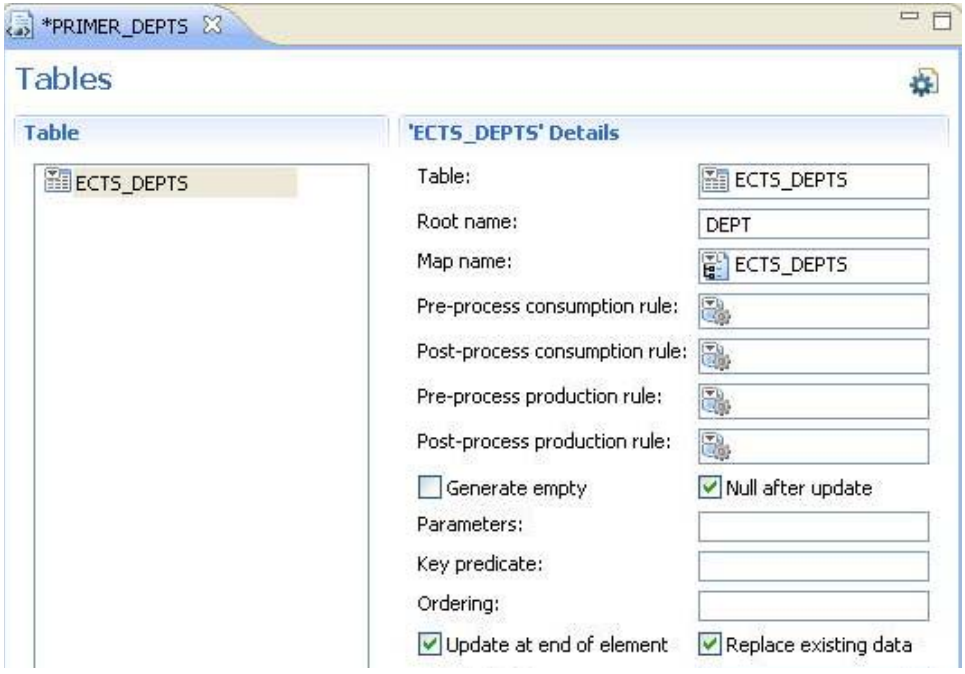
Take the sample tables, ECTS_DEPTS and ECTS_EMPLOYEES. Suppose you want to create an XML document that lists all the departments from ECTS_DEPTS and, for each department, all its employees. You can do that with nested FORALL statements in the TIBCO Object Service Broker rules language, like this:

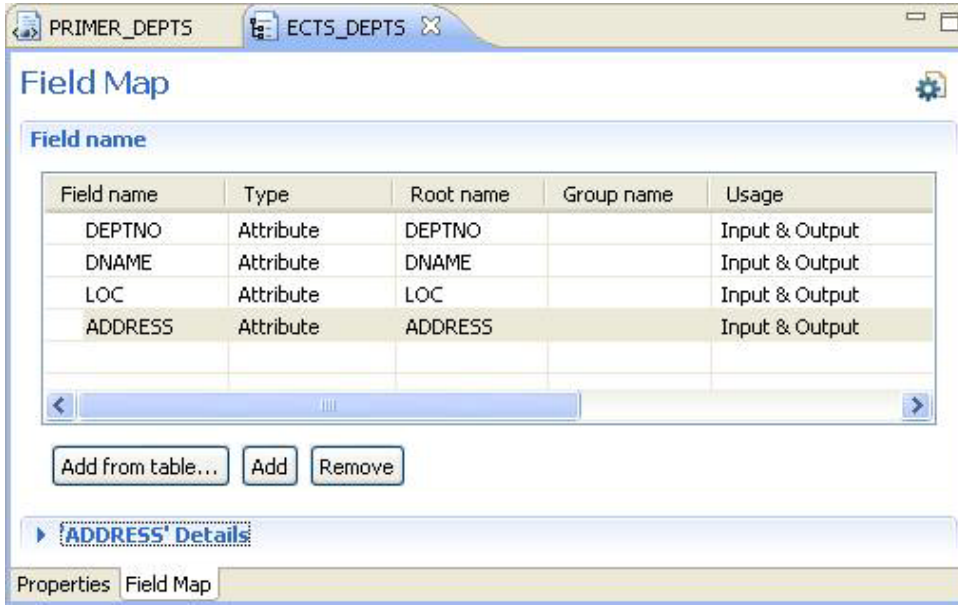
```
FORALL ECTS_DEPTS:
     FORALL ECTS_EMPLOYEES WHERE DEPTNO = ECTS_DEPTS.DEPTNO:
     (process one employee)
     END;
END;
```

In an XML document, you can perform that task with a child document. For the preceding example, define an XML document for the ECTS_DEPTS table, which then becomes your parent document, and name it PRIMER_DEPTS.

Follow the steps as illustrated by the three figures below to define PRIMER_DEPTS.



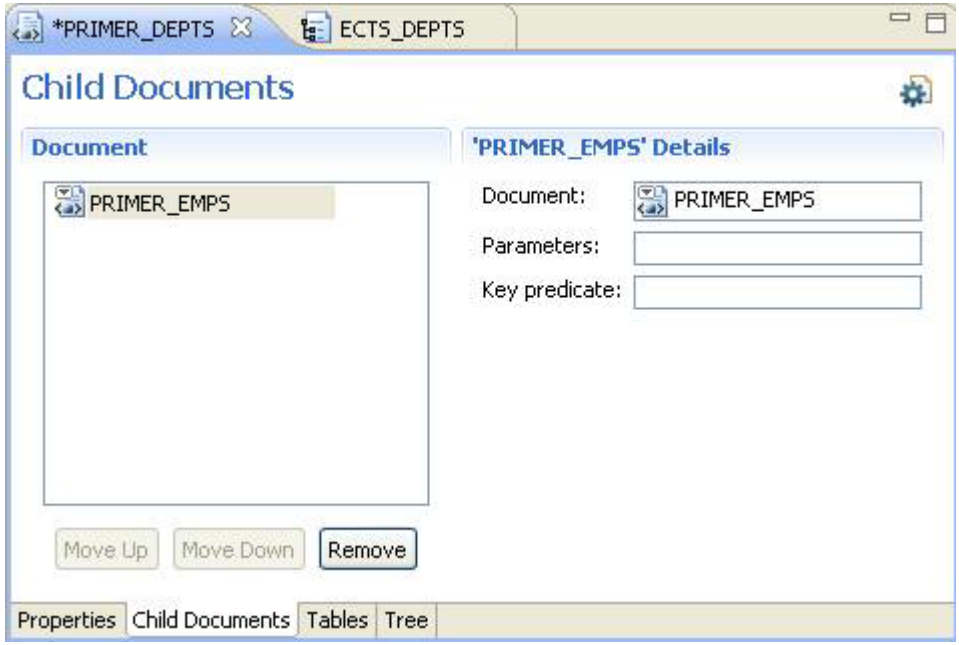

*PRIMER_DEPTS
Tables
Table
ECTS_DEPTS
'ECTS_DEPTS' Details
Table:
ECTS_DEPTS
Root name:
DEPT
Map name:
ECTS_DEPTS
Pre-process consumption rule:
Post-process consumption rule:
Pre-process production rule:
Post-process production rule:
Generate empty
Null after update
Parameters:
Key predicate:
Ordering:
Update at end of element
Replace existing data


PRIMER_DEPTS
ECTS_DEPTS
Field Map
Field name
Field name
Type
Root name
Group name
Usage
DEPTNO
Attribute
DEPTNO
Input & Output
DNAME
Attribute
DNAME
Input & Output
LOC
Attribute
LOC
Input & Output
ADDRESS
Attribute
ADDRESS
Input & Output
Add from table...
Add
Remove
'ADDRESS' Details
Properties
Field Map

For simplicity, on the preceding image, the columns “Output format rule” and “Input format rule” have been collapsed.

Earlier in this chapter, you created an XML document named PRIMER_EMPS to display a list of employees. Perform the following steps to reuse that document for selecting employees in one department by applying data- selection criteria:

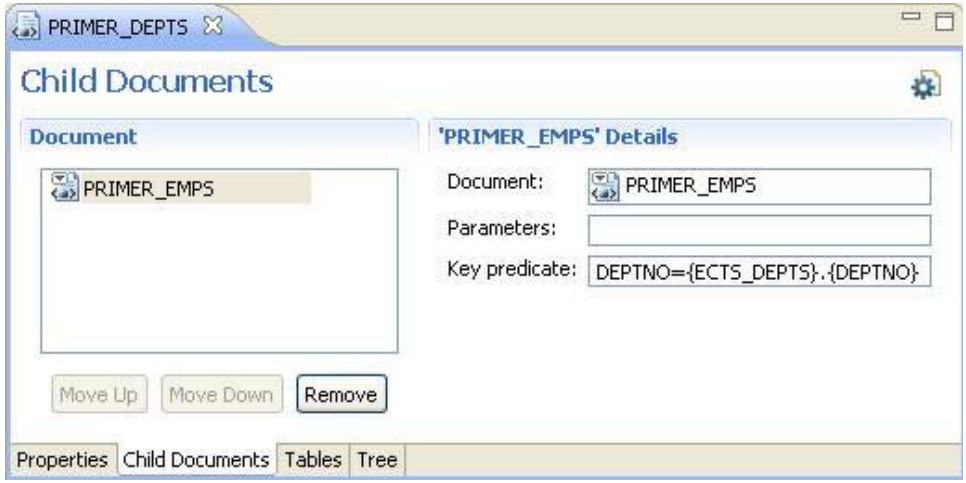
1. Add the PRIMER_EMPS document to the PRIMER_DEPTS document as a child: Open the XML Documents view and the PRIMER_DEPTS document definition.
2. Click the Child Documents tab.
3. Drag PRIMER_EMPS from the XML Documents view into the Select Child Document box on the Child Documents tab. The result should look like this:



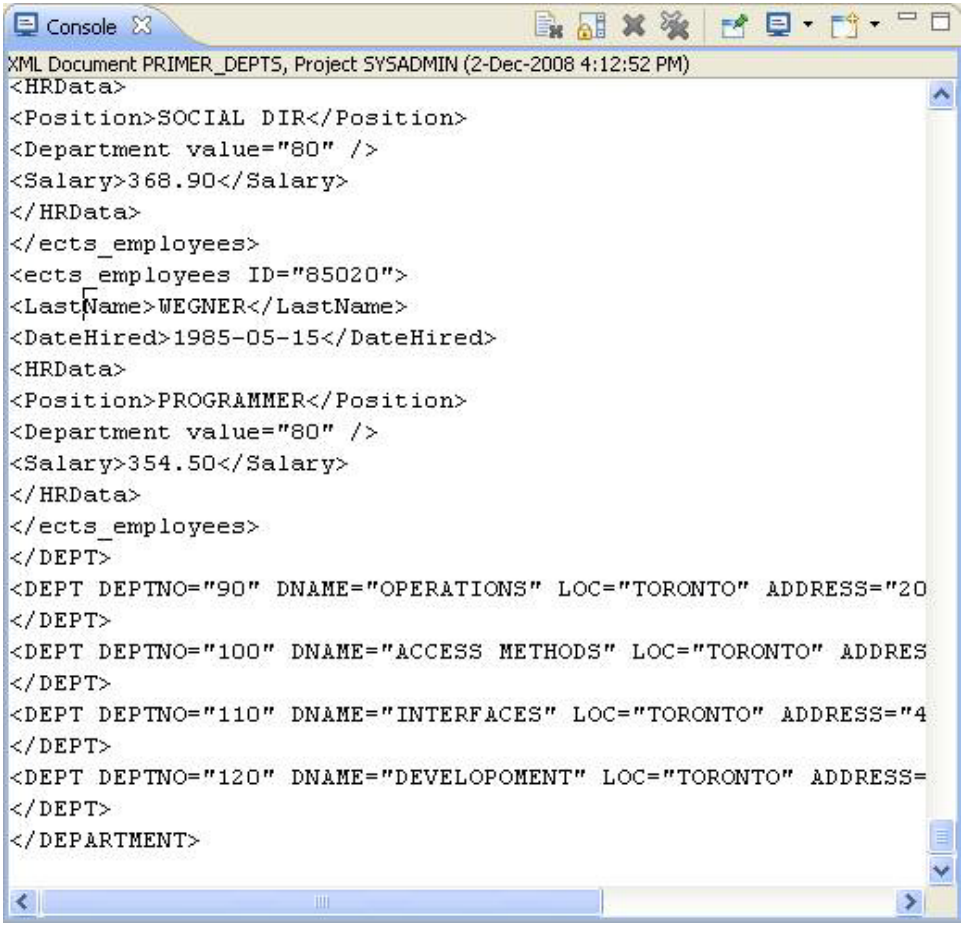
During processing of child documents, their document type is always inherited from the parent document regardless of the type defined in the child document.

Now add a few selection criteria to our child document, keeping in mind that you want to list only the employees in the department currently being processed. In the PRIMER_DEPTS document definition under the Child Documents tab is an area in which to specify parameter values for the table, as appropriate, and any necessary data selection criteria. Those properties override the values that already exist in the child document. That is, you need not revise any of the properties of the child document that would change its function as a parent.

In this case, you do not need any parameter values because the ECTS_EMPLOYEES table is not parameterized. Your selection criterion is to list all the employees in the department currently being processed. Recall that the department number in both tables is in the field DEPTNO. Simply apply the curly-bracket ({}) notation to access the field in the ECTS_DEPTS table to phrase the selection, as shown here:



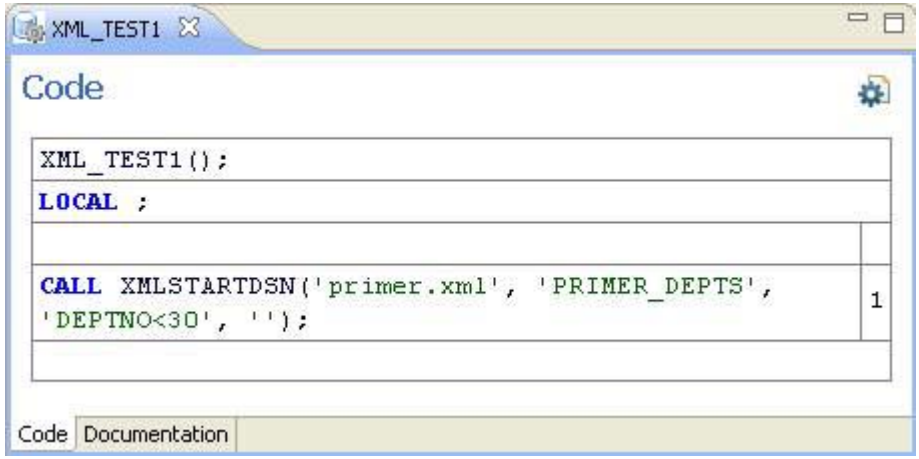
Finally, to test the result, run PRIMER_DEPTS by choosing Run As from the short-cut menu for the selected PRIMER_DEPTS item in the XML Documents view. Here is the output:



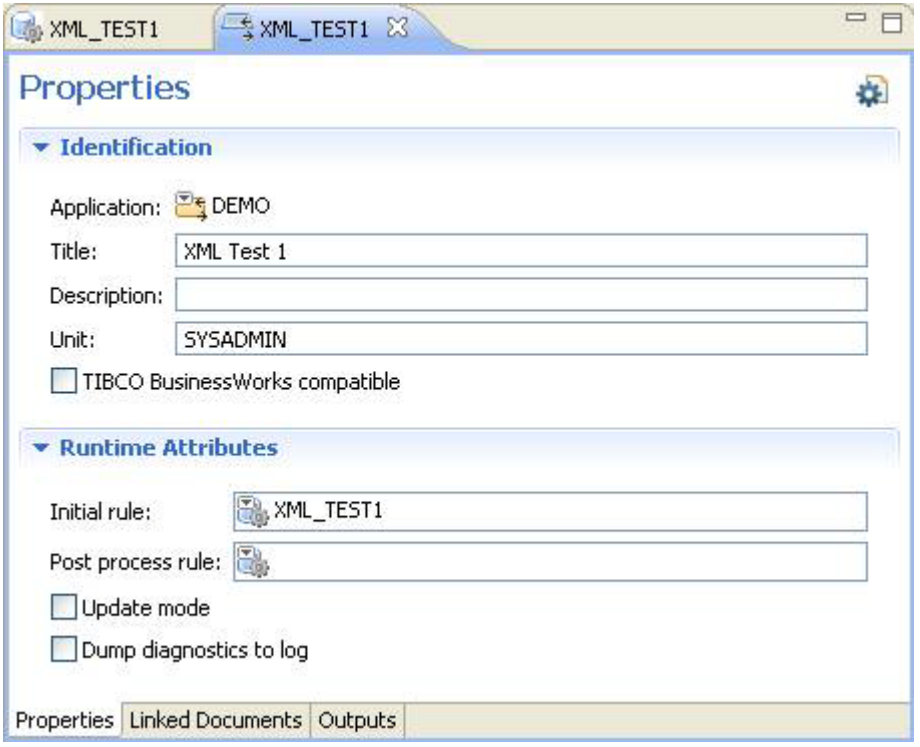
Using XML Documents

You can use XML documents in many ways. A primary way is through the Object Integration Gateway programmatic interfaces, such as the `runXMLDoc` method of the `ects2AppBean` class. You can also create a rule to output an XML document into a file or a data set.

The following figure is an example of a rule that retrieves an XML document and saves it to a file.



This other example shows a transaction that uses the `XML_TEST1` rule:



Here is a segment of the `primer.xml` file produced by the transaction:

```
<?xml version="1.0" ?>
<!-- XMLDocName=PRIMER_DEPTS -->
<!-- Generated By ObjectStar Integration Gateway V2.5 - Code Level 1.00 -->
<DEPARTMENT>
<DEPT DEPTNO="10" DNAME="ACCOUNTING" LOC="TORONTO" ADDRESS="200 UNIVERSITY AVE, M8C
3V1">
<ects_employees ID="80003">
<LastName>CHANG</LastName>
<DateHired>1987-02-21</DateHired>
<HRData>
<Position>ASSOC.ANALYST</Position>
<Department value="10" />
<Salary>589.91</Salary>
</HRData>
</ects_employees>
<ects_employees ID="80006">
<LastName>MILMAN</LastName>
<DateHired>1985-05-15</DateHired>
<HRData>
<Position>ANALYST</Position>
<Department value="10" />
```

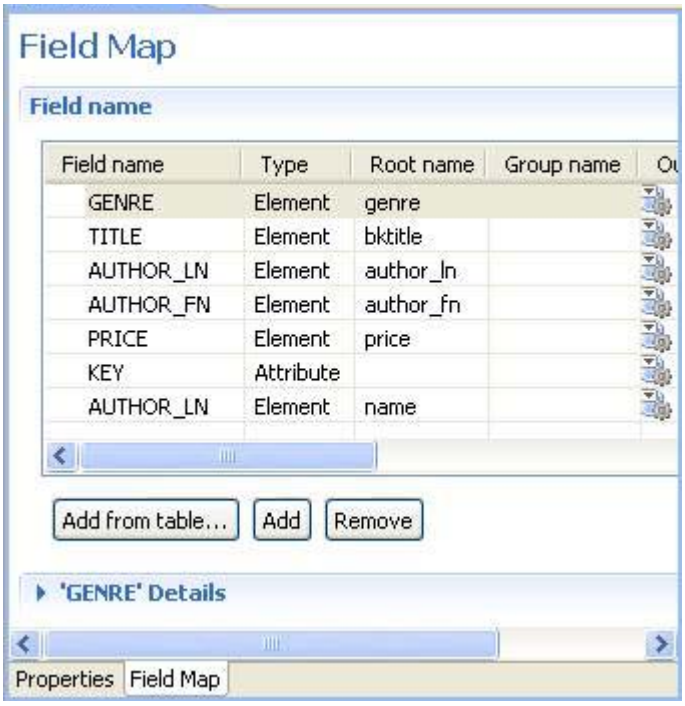
```
<Salary>699.49</Salary>
</HRData>
</ects_employees>
<ects_employees ID="82009">
<LastName>BOIVIN</LastName>
<DateHired>1983-10-30</DateHired>
<HRData>
<Position>ASSISTANT MGR</Position>
<Department value="10" />
<Salary>543.50</Salary>
</HRData>
</ects_employees>
...
```

You need not execute the rule you created through a transaction. Feel free to run the rule from any other interface, such as the Text Workbench.

Defining Attribute Relationships

This section shows you how to define attribute relationships with the XML Field Map `Attribute Of` type.

Consider this XML field map from the Books example:

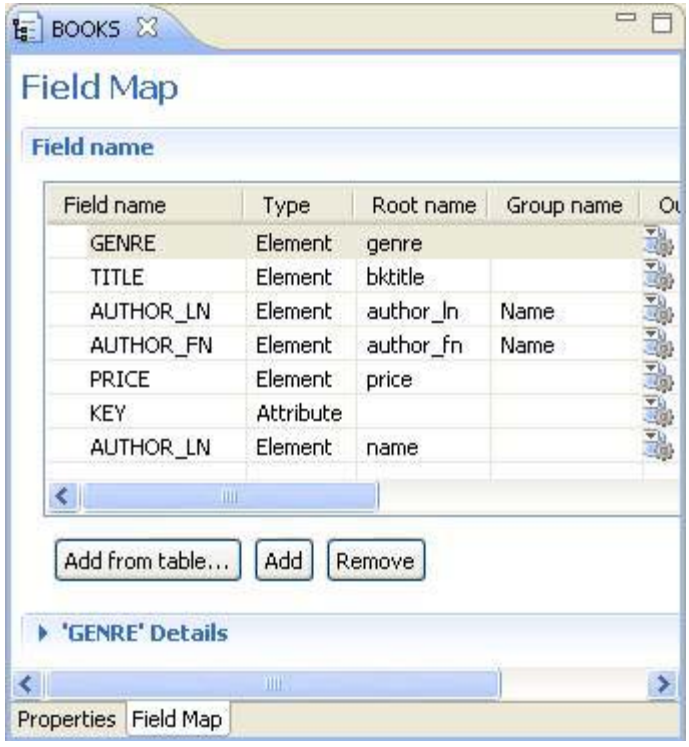


Executing the XML document `BOOKS` normally produces an XML document, a segment of which looks like this:

```
<book KEY="2">
    <genre>Literature and Fiction</genre>
    <bktitle>Pride and Prejudice</bktitle>
    <author_ln>Austin</author_ln>
    <author_fn>Jane</author_fn>
    <price>4.80</price>
</book>
```

Follow these steps to learn how to use the `Attribute Of` type:

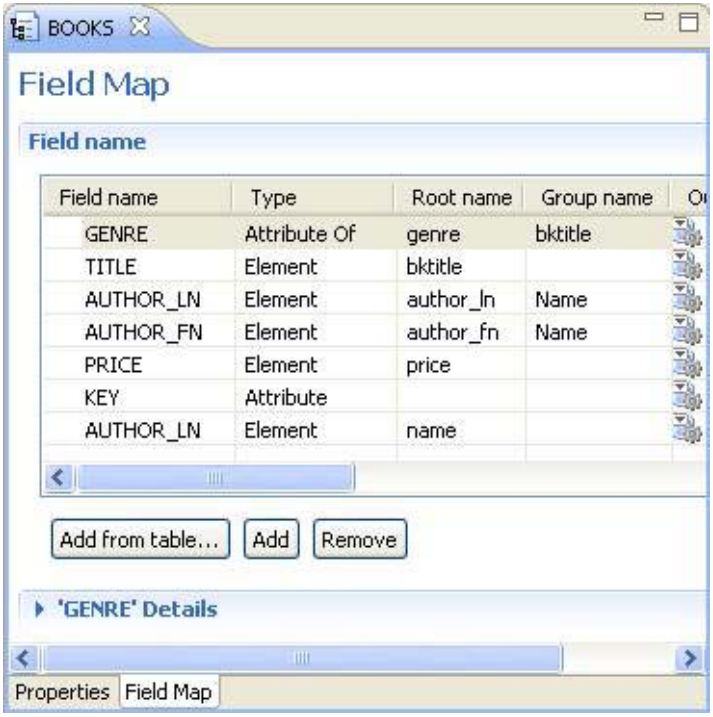
1. Group the author's first and last names in the XML Field Map group `Name`, as illustrated here:



The resulting XML document entry looks like this:

```
<book KEY="2">
    <genre>Literature and Fiction</genre>
    <bktitle>Pride and Prejudice</bktitle>
    <price>4.80</price>
    <Name>
        <author_ln>Austin</author_ln>
        <author_fn>Jane</author_fn>
    </Name>
</book>
```

2. Make GENRE an attribute of the bktitle element, as illustrated here:

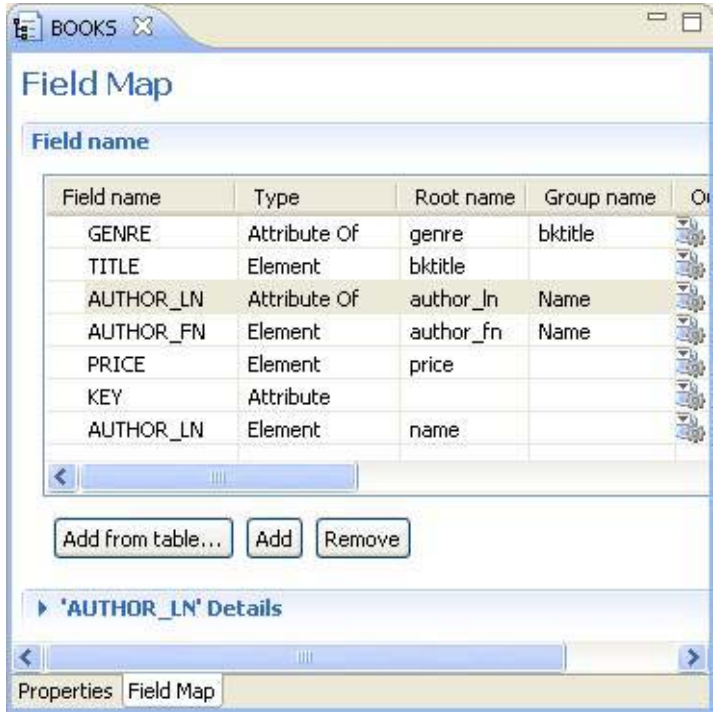


The resulting XML document entry looks like this:

```
<book KEY="2">
    <bktitle genre="Literature and Fiction">Pride and Prejudice</bktitle>
    <price>4.80</price>
    <Name>
        <author_ln>Austin</author_ln>
        <author_fn>Jane</author_fn>
    </Name>
</book>
```

Next, create attributes within the groups:

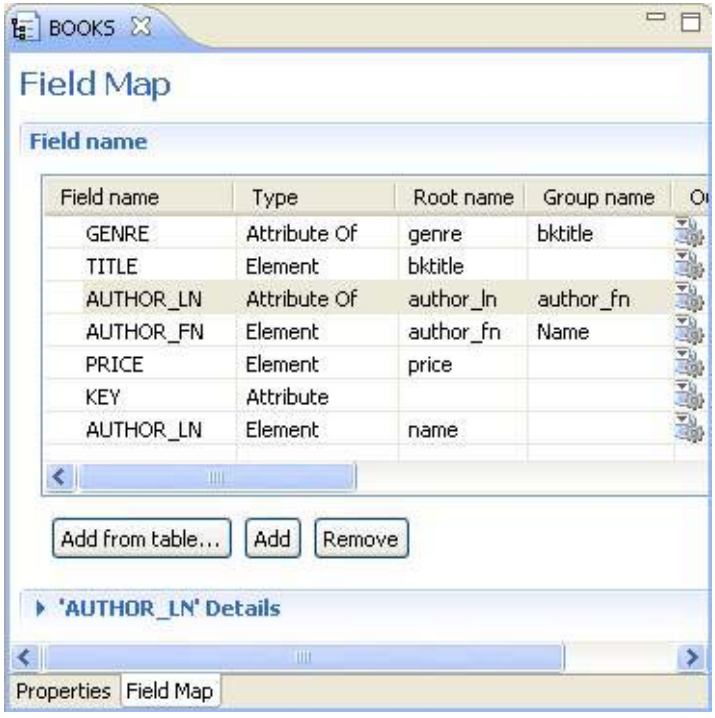
1. Make the last name an attribute of the group element `Name`, as illustrated by the following figure.



The resulting XML document entry looks like this:

```
<book KEY="2">
    <bktitle genre="Literature and Fiction">Pride and Prejudice</bktitle>
    <price>4.80</price>
    <Name author_ln="Austin">
      <author_fn>Jane</author_fn>
    </Name>
</book>
```

2. Make the last name an attribute of the first name instead of the group element `Name`, as illustrated by the following figure.

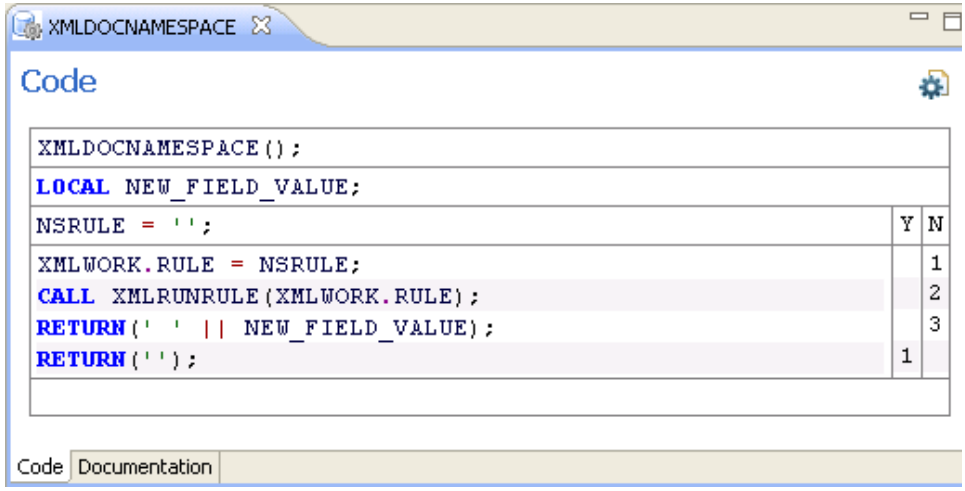


The resulting XML document entry looks like this:

```
<book KEY="2">
    <bktitle genre="Literature and Fiction">Pride and Prejudice</bktitle>
    <price>4.80</price>
    <Name>
        <author_fn author_ln="Austin">Jane</author_fn>
    </Name>
</book>
```

Customizing XML Declarations

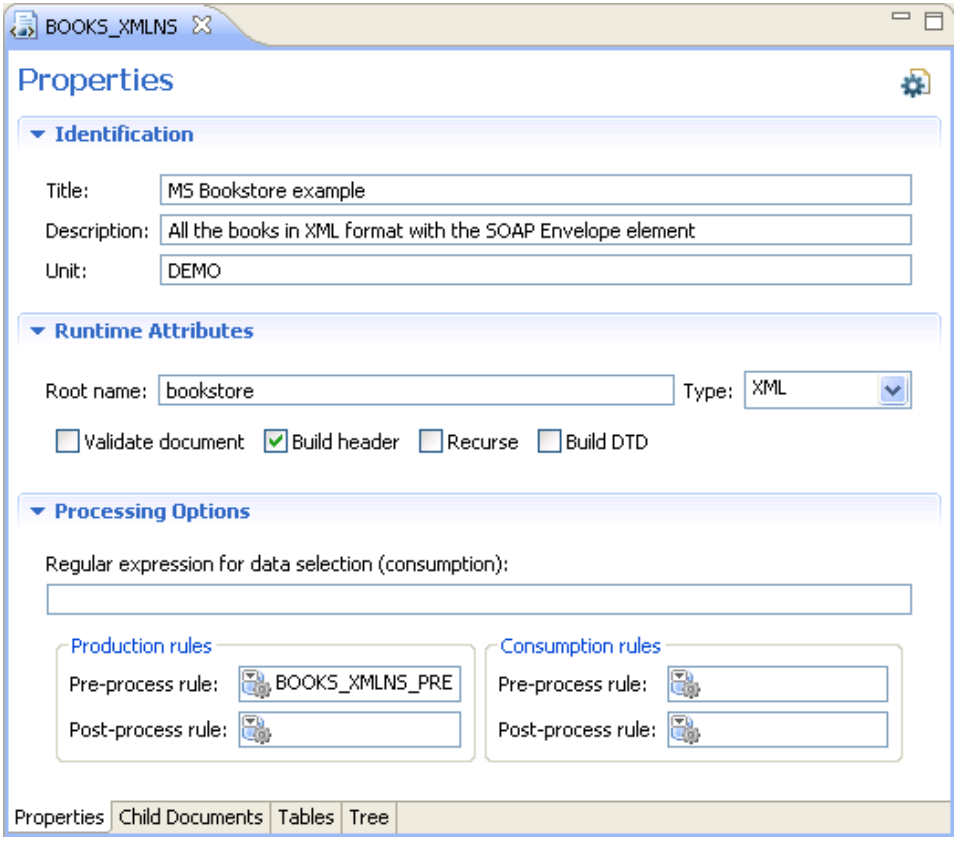
This section shows you how to customize the XML namespaces that are based on the function XMLDOCNAMESPACE called from the rule XMLSTART4 to produce the soap:Envelope portion of the XML request message. See the following figure.



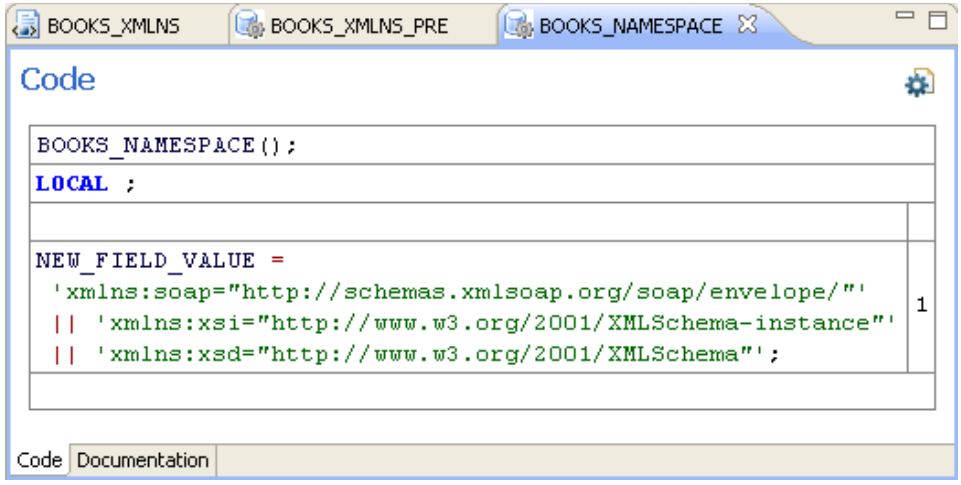
Here is the procedure:

1. Create a copy of the BOOKS XML document and name it BOOKS_XMLNS.
2. Delete all the consumption rules and define a new preprocess production rule BOOKS_XMLNS_PRE. Leave all the other settings unchanged.

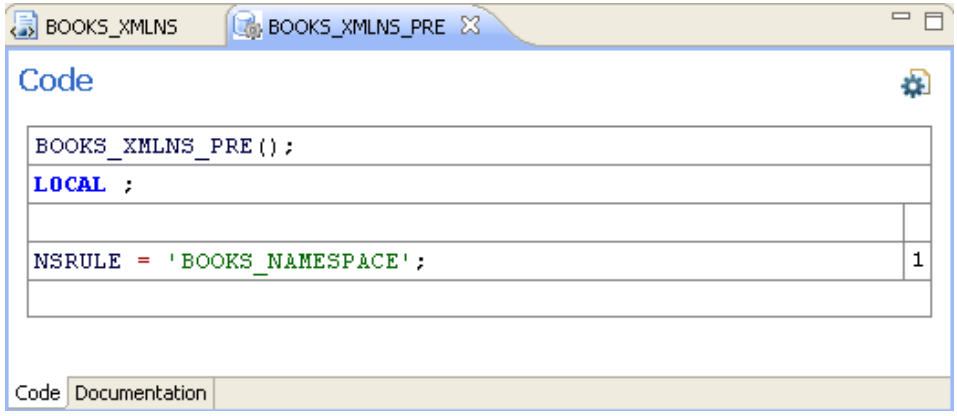
The resulting XML document looks like this:



3. Create a rule BOOKS_NAMESPACE that contains the required SOAP envelope element in NEW_FIELD_VALUE, which will be used in the XMLDOCNAMESPACE rule, as illustrated here:



4. Create a rule BOOKS_XMLNS_PRE, which sets a local variable NSRULE to enable BOOKS_NAMESPACE to run, as illustrated here:



5. Run the XML document BOOKS_XMLNS_PRE. The result looks like this:

```
End message:
OK

<?xml version="1.0" ?>
<!-- XMLDocName=BOOKS_XMLNS -->
<!-- Generated By ObjectStar Integration Gateway V2.5 - Code Level 1.00 -->
```

```
<bookstore
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<book KEY="1">
<bktitle genre="Classics">The Odysey</bktitle>
<price>2.98</price>
<Name>
</Name>
</book>
<book KEY="2">
<bktitle genre="Literature and Fiction">Pride and Prejudice</bktitle>
<price>4.80</price>
<Name>
<author_fn author_ln="Austin">Jane</author_fn>
</Name>
</book>
<book KEY="3">
<bktitle genre="Science Fiction">Stranger In A Strange Land</bktitle>
<price>7.99</price>
<Name>
<author_fn author_ln="Heinlein">Robert A</author_fn>
</Name>
</book>
<book KEY="4">
<bktitle genre="Science Fiction">I, Robot</bktitle>
<price>5.99</price>
<Name>
<author_fn author_ln="Asimov">Isaac</author_fn>
</Name>
</book>
...
```

Appendix D Using the OIG Administration Interface

This appendix describes how to use the OIG administration interface.

Topics

Overview

What Is the Administration Interface?

The OIG administration interface is an OIG web application you can use to manage OIG session pools. Using the administration interface, you can view the list of active pools and the current state of sessions in each pool. You can also use the administration interface to recycle the sessions.

For information about how to install the administration interface, refer to Installing the OIG Administration Interface on page 5.

Using the Administration Interface

Logging In

To use the administration interface, you first navigate to the login page. The login page is either eCTSadmin.jsp for a JSP web application, or eCTSadmin.asp for an ASP web application, or eCTSadmin.aspx for a .NET web application. (The URL you enter depends on where you installed the administration interface web application.) You must log in with a user ID and password for a level-7 user, and enter the host name and port number for your TIBCO Object Service Broker Execution Environment. After you successfully log in, you should see the Pool List page.

See Also *TIBCO Object Service Broker Managing Security* for information about TIBCO Object Service Broker security levels.

Using the Pool List Page

The Pool List page displays a list of the active session pools. The session pool list appears in a table that contains the following fields:

Field Name	Description
Pool Name	The name of the pool. For JSP or servlet applications, this is the name of the eCTSSession bean or the name of the servlet. For ASP, ASPX or COM applications, this is the alias name for the pool, or a generated name if the OpenApplication method of the COM component is called with the mode parameter set to POOL. Click the pool name to see details for the pool.
Sessions	The number of sessions currently allocated to the pool.
Use Count	The number of times a session was granted from the pool. For servlet applications, it is the number of times a servlet was invoked for the pool. For ASP, ASPX or COM applications, it is the number of times an OpenApplication method call was made for the pool.
Last Used	The last time the pool was accessed.

Field Name	Description
Action	Click the Restart link to shut down all sessions in the pool. If a session is active, the session is terminated after it completes the request it is processing. After a restart, new requests for the pool result in the creation of new sessions.

Options

- To see the information for a pool on the Pool Details page, click the pool name.
- To recycle the sessions in a pool, click the pool's Restart link.

Using the Pool Details Page

The Pool Details page displays detailed information about the session pool you selected in the Pool List page. The information appears in two tables.

The first table, with the heading Pool Parameters, displays the initialization parameter settings for sessions in that pool. For more information about session initialization parameters, refer to Appendix A, Setting OIG Session Initialization Parameters, on page 113.

The second table, with the heading Sessions in Pool, displays a list of active sessions. It contains the following fields:

Field Name	Description
Id	The unique identifier for the session.
In Use	The current processing state of the session. Displays True if the session is processing a request; False if the session is idle.
Use Count	The number of times the session was used.
Last Used	The last time the session was used.
Last Function	The name of the last operation and the object operated on.

Field Name	Description
To Server	The number of message buffers sent to the Object Integration Gateway for the session. Some requests require multiple messages to be sent to the Gateway because of the amount of persistent data and the size of the XML documents being sent. You can manage this count by changing the size of outgoing message buffers. Refer to the DATAOUT parameter in Appendix A, Setting OIG Session Initialization Parameters, on page 113 for more information.
From Server	The number of message buffers received from the Gateway. Some requests require multiple buffers to hold returning persistent data and large XML documents. You can manage this count by changing the size of incoming message buffers. Refer to the DATAIN parameter in Appendix A, Setting OIG Session Initialization Parameters, on page 113 for more information.

Exiting the Administration Interface

To exit the administration interface, just close your browser to end the session. You do not have to explicitly log out.

Appendix E Creating a Silent Installer for the OIG COM Component

This appendix describes how to create a silent installer for the OIG COM component.

Topics

Overview

What Is a Silent Install?

A silent install of software is one that is performed without user intervention, that is, without a user executing installation commands on the target machine. Such an install is described as silent because the user is not even aware that the installation is taking place.

Silent installs are frequently executed on multiple target machines across a network.

Creating a Silent Installer

What the Installer Has to Do

A silent installer for the COM Component must do the following three tasks:

1. Check for other required software, page 169
2. Install the program files, page 169
3. Make the registry entries, page 170

Task A Check for other required software

The COM Component requires you install the following software on each target machine:

- Microsoft Data Access Components (MDAC) 2.5 or higher
- Java 2 Runtime Environment (JRE) 1.4 or higher

Refer to Prerequisites on page 3 for more information about the software versions required by the COM Component.

The JRE can be installed silently on the target machine using the following steps:

1. Copy the JRE directory tree, intact, to any directory on the target machine.
2. Ensure that the Gateway JAVABINPATH registry entry on the target machine points to the location of the jvm.dll file in the JRE directory.

For more information on redistributing the JRE files, refer to the README.txt file in the JRE directory.

Task B Install the program files

The installer must copy the following COM Component files to a directory on the target machine:

- eCTScom.dll
- ects.jar

After copying the files to the target machine, the installer must register the eCTScom.dll file on the target machine, using the Windows registration utility regsvr32.exe.

Task C Make the registry entries

The installer must make the following entries in the Windows registry on the target machine. These registry entries specify the location of the JRE and the ects.jar file, and the alias definitions used by your OIG applications.

Refer to Chapter 4, Using the OIG COM Component, on page 41 for more information about defining and using aliases.

Registry Key	Value Name	Value Data
\HKEY_LOCAL_MACHINE\SOFTWARE\ObjectStar\eCTS	ECTSBINPATH	An REG_SZ value that specifies the full pathname of the directory where the ects.jar file is located.
\HKEY_LOCAL_MACHINE\SOFTWARE\ObjectStar\eCTS	JAVABINPATH	An REG_SZ value that specifies the full pathname of the Java Virtual Machine file, jvm.dll. Java infers the root directory of the JRE from the path of the jvm.dll file.
\HKEY_LOCAL_MACHINE\SOFTWARE\ObjectStar \eCTS\Alias	DEFAULT	An REG_SZ value that specifies the default Gateway session parameters for the Object Integration Gateway COM Component, used if no other aliases are defined or specified.

If other alias definitions are required, add them under the \HKEY_LOCAL_MACHINE\SOFTWARE\ObjectStar\eCTS\Alias registry key. Refer to Appendix A, Setting OIG Session Initialization Parameters, on page 113 for more information about OIG session parameters.

After Installing

After the silent install, applications can use the COM Component. You do not have to restart Windows, if the prerequisite software is already installed.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Z