TIBCO® Object Service Broker for Open Systems

Managing Backup and Recovery

Software Release 6.0
July 2012

two-second advantage™

TIBCO®
The Power of Now®

Important Information

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

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

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

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

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

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

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

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

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

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

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

Contents

Preface

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

This manual explains the backup and recovery features of TIBCO Object Service Broker for Open Systems. It describes system key components and describes how you can back up your data and recover from errors. You can use this information, with assistance from your TIBCO Support representative, to develop the best customized solution for your unique backup and recovery requirements.

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 Introducing TIBCO Object Service Broker Backup and Recovery Components

This chapter describes the TIBCO Object Service Broker backup and recovery components for open systems.

Topics

Overview

Operational Components

There are two major operational components of a TIBCO Object Service Broker system that are key elements of a backup and recovery strategy. They are:

- Data Object Broker
- Execution Environment

These components are discussed in the following sections.

File Components

The Data Object Broker uses a number of key operational files to provide complete data integrity and maximum recoverability:

File	Refer to page…
Pagestore	7
Redolog	8
Contingency log	9
Journals	9
dbdef file	10

Data Object Broker

Definition

The Data Object Broker is the server for TIBCO Object Service Broker data. It is responsible for most operations against the Pagestore, which is the repository for all TIBCO Object Service Broker data. For more information about the Pagestore, refer to Pagestore on page 7.

Data Object Broker Functionality

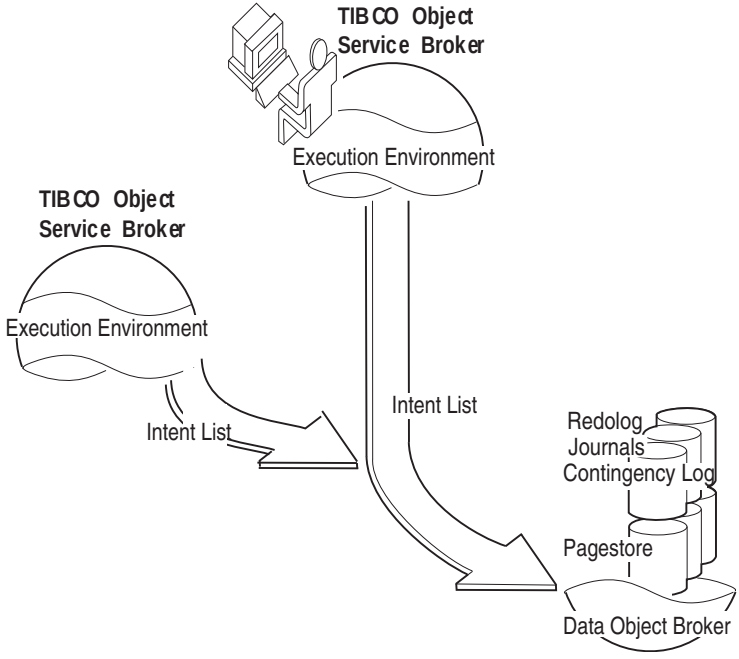
When the Data Object Broker receives a commit request from the Execution Environment, it performs the following:

1. Obtains and locks the required data pages from the Pagestore
2. Places the pages in memory
3. Records the update requests in the redolog
4. Updates the pages, placing them on a queue for checkpoint processing
5. When a checkpoint occurs, writes the pages to the journal and propagates the changes to the Pagestore

Committing Updates

Most accesses and changes to the Pagestore go through the Data Object Broker. If a commit request involves another system (an external database or a peer TIBCO Object Service Broker system), the Data Object Broker logs the request in the contingency log until the update request is confirmed by the other system. For more detail, refer to Chapter 3, Understanding Fail Safe Processing, on page 17.

The Role of the Data Object Broker and Related Data Sets



Data Object Broker Communication

The method of communication between the Data Object Broker and Execution Environments depends upon their location. If they are on the same machine, they use Inter-Process Communication (IPC), while communication between regions on different machines uses TCP/IP.

Execution Environment

Definition

The Execution Environment runs, or executes, the TIBCO Object Service Broker applications for the end user. It manages such things as:

- Screen I/O
- Rules interpretation
- Logical data control

Execution Environment Functionality

The Execution Environment deals with the Data Object Broker on behalf of the user or application and passes on requests for data and commands to update the database.

The Execution Environment makes requests to the Data Object Broker to obtain information, processes the information, and presents it in the form of screen displays or reports. An Execution Environment can connect to only one Data Object Broker, whereas a Data Object Broker can support many Execution Environments. In short, the Execution Environment provides the user and application interface to the physical data.

Types of Execution Environments

You can have several types of Execution Environments. These include:

- Multi-user Execution Environment accessed through the Object Integration Gateway (OIG), ostty, TIBCO Object Service Broker SDK (C/C++), or TIBCO Object Service Broker SDK (Java) clients, or via a 3270 emulator using Telnet 3270.
- Batch Execution Environment (osBatch), which is single-user.

Parameters

Execution Environments require a number of parameters. System-wide defaults for each type of Execution Environment are defined during installation and each Execution Environment can specify its own defaults at startup.

See Also

- *TIBCO Object Service Broker Parameters* for more information about Execution Environment parameters.
- *TIBCO Object Service Broker Object Integration Gateway* for more information about Object Integration Gateway.
- *TIBCO Object Service Broker for Open Systems External Environments* for more information about the SDK (C/C++) and the SDK (Java) clients.
- *TIBCO Object Service Broker for Open Systems Utilities* for more information about osBatch.
- *TIBCO Object Service Broker Getting Started* for more information about the Telnet 3270 access.

TIBCO Object Service Broker Files

The following sections describe the files used by TIBCO Object Service Broker. For better performance, each TIBCO Object Service Broker file should be located on a different device. This is very important for high-use files such as the MetaStor, the redolog, and the contingency log.

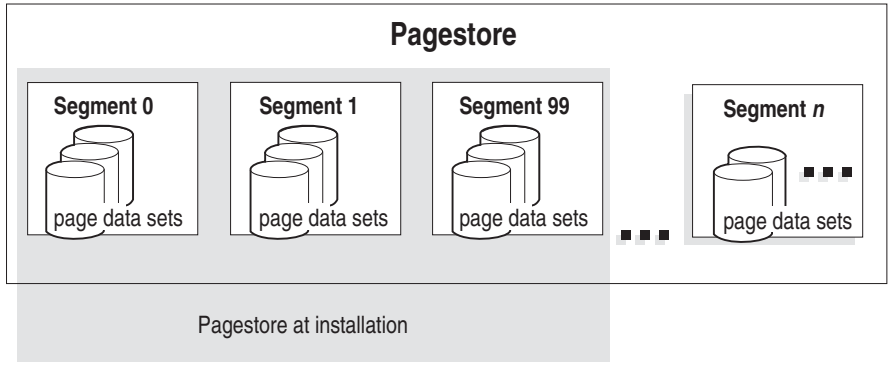
See Also *TIBCO Object Service Broker for Open Systems Installing and Operating* for more information on the definition, placement, and sizing of these files.

Pagestore

The Pagestore is a collection of files used by TIBCO Object Service Broker for storing data in tables. In simple terms, this is the database. The Pagestore stores data using the TDS (Table Data Store) method. TDS tables are stored in a B+ tree structure.

Pagestore Segments

The Pagestore is divided into partitions known as segments. Each segment consists of a number of files containing data, as shown in the following illustration.



Segment 0: The MetaStor

The base segment (segment 0) contains the MetaStor, the central repository for all metadata: definitions, characteristics, access paths, and storage locations of all data and programs in TIBCO Object Service Broker. Segment 0 is the most heavily used segment. It contains three files by default (PAGE1, PAGE2, and PAGE3). These files can be expanded as required.

Pagestore Capacity

The Pagestore can consist of up to 256 segments. Each segment can consist of up to 128 files. A file can hold 500,000 x 4 KB pages. Given these parameters, the potential capacity of the Pagestore is approximately 256 GB per segment. When you initially set up your Pagestore, leave room for expansion in each page file. For example, if you require a 6 GB segment 1, define six page files with 250,000 4 KB pages rather than three page files with 500,000 x 4 KB pages.

The capacity of each segment of the Pagestore is effectively limited by the maximum size of the backup files.

Page File Usage

Each segment of the Pagestore contains a number of page files. If possible, each file should be stored in a different directory within a different SCSI controller than the one containing the redolog and journals. This allows TIBCO Object Service Broker to minimize I/O contention in the segment. When a new 4-KB page is allocated to a segment, TIBCO Object Service Broker uses the files contained in the segment in sequence. It places each new page on the next file in turn so that the pages are evenly distributed across the files.

Initial Pagestore Configuration

When you first install TIBCO Object Service Broker, your Pagestore configuration consists of six components:

- Base segment or MetaStor (segment 0), which initially contains three page files
- Segment 1, which contains one page file
- Segment 99, which contains the security audit log (ACCESSLOG table)
- Redolog
- Contingency log
- Journals

See Also *TIBCO Object Service Broker Application Administion* for information about expanding the Pagestore.

Redolog

The redolog contains records of all update operations recently performed, or about to be performed, against the database. TIBCO Object Service Broker uses the redolog to reconstruct the committed updates made between checkpoints and maintain database integrity.

The redolog file should be defined at the head of a SCSI controller that is a different controller than the one containing the Pagestore page files.

Contingency Log

The contingency log contains a record of every in-doubt or contingent transaction.

Contingent Transactions

Contingent transactions are the local TIBCO Object Service Broker portion of a transaction that can perform either of the following updates:

- Updates both local and remote TIBCO Object Service Broker data in a single transaction
- Updates local TIBCO Object Service Broker data and external database data in a single transaction

In-doubt Transactions

In-doubt transactions are those that involve an external database server or a peer TIBCO Object Service Broker and that could not be completed. They are kept in the contingency log until the update is confirmed by the other systems, then written to the redolog.

In the event of a power failure or external system failure, TIBCO Object Service Broker uses the contingency log to ensure consistency of updates across external database, peer, and local TIBCO Object Service Broker systems.

For more information about the role of the contingency log in distributed data environments, refer to Chapter 3, Understanding Fail Safe Processing, on page 17.

Journals

TIBCO Object Service Broker contains two journals that perform the following tasks:

- Create write-ahead logs for checkpoint recovery

When the system performs a checkpoint, all modified data pages in the resident page manager buffers are copied to the cache portion of the journal, therefore assuring their recoverability in the event of a failure. The resident page manager is the internal Data Object Broker component that manages local memory containing images of all Pagestore pages currently in use.

- Provide an audit trail of all changed physical pages

The system switches between journals when one reaches capacity and the contents of the full journal are saved. The procedure to save the full journal is called a *spin*. Refer to Chapter 5, Understanding Journal Processing, on page 37 for more information. The journals can be placed on the same SCSI controller as the redolog, but they must be in a different directory.

Database Definition File

The database definition file, dbdef, defines the Pagestore. The *dbdef* file contains:

- Definitions of the segments comprising the Pagestore
- Definitions of the redolog, journals, and contingency log

TIBCO Object Service Broker uses the dbdef file during initialization. The dbdef file centralizes most control information. For more information on the TIBCO Object Service Broker Pagestore, refer to Pagestore on page 7.

Chapter 2 Understanding Transaction Processing

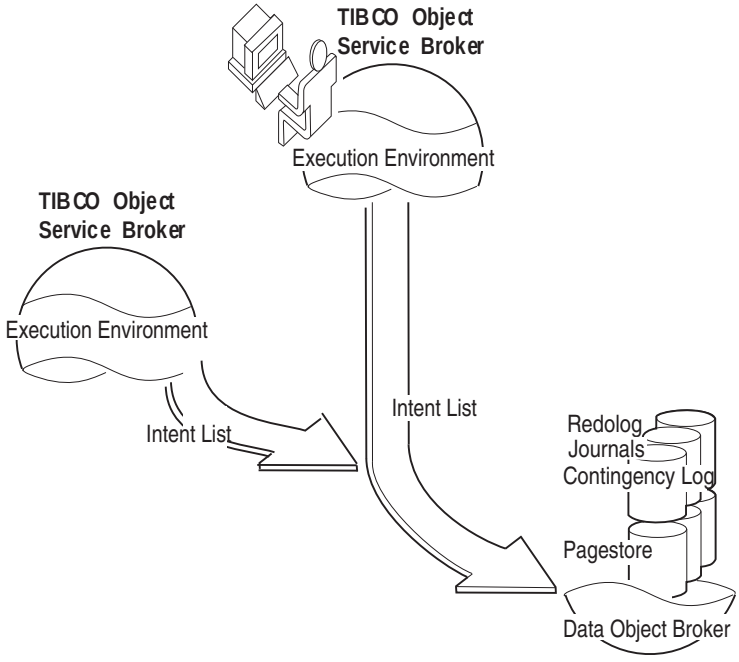
This chapter describes transaction processing in TIBCO Object Service Broker.

Topics

Overview

This chapter takes you through a typical TIBCO Object Service Broker transaction and explains what goes on behind the scenes from an operations perspective. The transaction used in the example is assumed to be running in update mode and updates a local TDS table.

Key Components Involved in Transaction Processing



Sample Transaction

Task A The transaction requests data from a table

This request can be a GET or a FORALL on a table. The following occurs:

1. The Execution Environment sends the request to the Data Object Broker requesting a logical shared-lock on the rows being accessed.
2. The Data Object Broker checks whether the required data pages are in the resident page manager buffers.

   If the required data pages are already in the buffers, they are available for use. If the pages are not in the buffers, the Data Object Broker retrieves them from the Pagestore and loads them into the buffers.
3. The logical row in the table has a shared-lock placed on it because the transaction is running in update mode.
4. The Data Object Broker passes the data to the Execution Environment.

Task B The transaction performs an update

Update requests include:

- INSERT
- DELETE
- REPLACE

During the update request, the following occurs:

1. If no other transaction has a shared-lock on the rows being updated, the Execution Environment stores the updates in its intent list.
2. An exclusive lock is requested for the updated rows. This means that no other transaction can obtain a shared-lock on the rows until the transaction terminates.

The data is not yet physically changed, and is not until a commit is issued. The Execution Environment keeps track of the changes, however, to ensure integrity and consistency are maintained.

Task C A commit command is issued

The following occurs after the commit command is issued:

1. The Execution Environment passes the database update request (intent list) to the Data Object Broker.
2. The Data Object Broker writes the intent list to the redolog. This operation guarantees that all committed updates are written to the database.
3. The Data Object Broker updates the data pages in the resident page manager buffers. This ensures that these new database updates are reflected in future accesses to the same data by other transactions.
4. If an updated segment reaches its predefined capacity threshold, that is, it is nearing full capacity, warning messages are sent to the operator console.
5. The Execution Environment is notified to proceed with the application. It clears the intent list, but not the database locks that are held by the Data Object Broker, and continues execution of the application.
6. If a checkpoint is required, the Data Object Broker invokes its asynchronous checkpoint processor, which writes all the updated pages to the journals and the Pagestore.
7. If the current journal fills up during a checkpoint, the Data Object Broker starts the spin process and switches to the other journal.

Task D The transaction ends

The following occurs:

1. The Execution Environment passes the updates left in the intent list to the Data Object Broker. These are updates made after the most recent commit.
2. The Data Object Broker processes the database updates as before, writing them to the redolog, updating the resident page manager buffers, and so on.
3. All locks held on behalf of the transaction are released.

Implications for Backup and Recovery

Note the following potential implications of transaction processing on your backup and recovery system:

Frequency of Checkpoints

You must ensure that you monitor the frequency of checkpoints incurred when new batch processes are added to your TIBCO Object Service Broker system. If the process causes a large number of pages to be updated, the resident page manager buffers fill quickly causing a significant increase in checkpoint requests. Both the redolog and journals must be large enough to accommodate the increased activity. If a new batch process causes a sharp increase in checkpoints, examine the application to see if updates can be clustered together to reduce the number of data pages that require updating.

Merge Process Speed

You must ensure that the merge process completes before the redolog or resident page manager buffers fill up. Otherwise, the system quiesces or abends.

Chapter 3 Understanding Fail Safe Processing

This chapter describes fail safe processing in TIBCO Object Service Broker.

Topics

Fail Safe Processing

Definition

Fail Safe is the TIBCO Object Service Broker term for the commit strategy.

Fail Safe processing is controlled and co-ordinated by the Data Object Broker where the commit for the unit of work is issued. The co-ordinating Data Object Broker synchronizes all participating service providers to ensure that all updates, local and remote, either commit or rollback as one unit of work. This ensures data integrity.

A service provider is one of the following:

- A local Data Object Broker
- A local service for TDS tables
- A peer Data Object Broker
- An external database server

Fail Safe Strategies

Determining a Fail Safe Strategy

If you have a multiple-systems environment, you must decide on an appropriate Fail Safe strategy. To do this, you must determine the Fail Safe level that meets your needs.

Fail Safe Levels

Use the following table to determine your required Fail Safe level:

Transaction Requirements	Fail Safe Level
Commits can be handled serially. External service providers commit first followed by local updates to TDS tables.	0
Transaction involves a local service for TDS tables and an external service provider. Local updates are contingent upon the completion of external updates.	1
Transaction involves multiple Data Object Brokers.	2

The following sections describe each Fail Safe level in detail.

During commit processing, the commit coordinator determines, within defined limits, which commit strategy to use:

Type of Update [a]	Type of Commit Given to Peer Server
One resource only (local resource or remote resource).	Fail Safe level 0
Local and one remote resource.	Fail Safe level 1
Local and/or at least two remote resources.	Fail Safe level 2

a. For all three update types, the local resource is TDS on the local node and the remote resource is a peer.

Fail Safe Level 0 (Serial)

Fail Safe level 0 provides for the issuance of commit requests to each service provider in a commit group, one at a time. If there is only one updated resource, Fail Safe level 0 is adequate. When there are two or more service providers, there is potential for data to get out of sync if an error occurs.

Consider, for example, a commit group that has three service providers:

- Server 1
- Server 2
- Local TDS tables

This is how the Fail Safe process proceeds:

1. Server 1 is issued a commit. When it responds successfully, the commit continues. Otherwise, the commit aborts.
2. Server 2 is issued a commit. When it responds successfully, the commit continues. Otherwise, if DBPROFILE=1, the commit group is terminated, abandoning all further updates; if DBPROFILE=0, the commit continues.
3. The local TDS tables are issued a commit.

Data synchronization errors between the service providers can occur if failures or communication loss happens after Server 1 responds successfully to the commit. If Server 1 does not respond, it is unknown whether it committed or not.

Fail Safe Level 1 (Contingent)

Fail Safe level 1 is restricted to environments where there are local updates and *one* service provider with remote (external) updates. An external service provider is capable of Fail Safe level-1 processing if a transaction database is defined and available. Refer to Transaction Database on page 22 for more information.

Commit processing under Fail Safe level 1 functions as follows:

1. The (local) Data Object Broker saves the intent list (that is, the TIBCO Object Service Broker component of the transaction) in the contingency log.
2. The (remote) external service provider generates a transaction control record and adds it to the pending database updates, then commits. The (local) Data Object Broker commits the intent list after receiving confirmation that the external update was successful.
3. If connection to the external service provider is lost before commit success or failure notification is received by the originating Data Object Broker, the associated transaction on the contingency log becomes an in-doubt transaction.

4. When communication is re-established, the external service provider can be queried for the presence or absence of the transaction control record that it added to the pending database updates. This verifies whether it was able to commit the unit of work.
5. If the control data is *not* found, the external system did not successfully commit and therefore any associated updates must be abandoned. If it is found, the updates are committed.

Fail Safe Level 2 (Two-Phase Commit)

Fail Safe level 2 coordinates the commit of a database unit of work that is distributed across multiple service providers. To operate at Fail Safe level 2, the participating service providers—a combination of Data Object Brokers and external database servers—must be capable of working with the TIBCO Object Service Broker commit coordination protocol.

In phase 1, when the co-ordinating service provider signals all other participants to prepare to commit, each participating service provider logs the pending unit of work in its own contingency log.

Having received acknowledgments of a successful phase 1, the co-ordinator starts phase 2, giving the commit signal, and each service provider actually performs the commit.

Contingent Two-Phase Commit

In this instance, a unit of work is spread across multiple Fail Safe level-2 capable resources, and one Fail Safe level-1-capable resource.

Commit processing under this condition functions as follows:

1. Fail Safe level 2 begins and the commit coordinator signals a prepare to commit to all participants.
2. The Fail Safe level-1-capable resource is signalled to commit and all its commits are completed.
3. Fail Safe level-2 commits are completed.

Components Supporting Fail Safe Processing

There are two key components that support Fail Safe processing:

- Transaction database
- Contingency log

These components are described in the following sections.

Transaction Database

The transaction database, which resides on the external database system, is a part of Fail Safe processing for all external service providers. It consists of a table (or equivalent in the external database's terminology) containing a number of fields identified by TIBCO Object Service Broker. In the case of a peer TIBCO Object Service Broker Data Object Broker, the peer's local contingency log is used as the transaction database. The fields are:

Contents of field	Syntax	Length
Data Object Broker name.	C	8
Server identifier.	C	8
Transaction ID.	B [a]	4
Date/Time.	B	8
Server registration data.	C	25

a. Some databases require defining binary fields as character syntax.

Each field is assigned a name. The field name varies depending on the requirements of the external database. In most cases, this table (or equivalent) has a default name (such as HRNTRXDB). This name can be customized provided that the customized name is reflected in the server parameter TRXDB.

See Also The appropriate *TIBCO Service Gateway* manual for more information on the server parameters.

Contingency Log

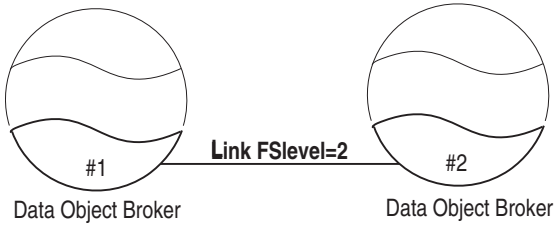
The second file is the contingency log, which is used by the Data Object Broker to hold transaction updates that are contingent upon service provider acknowledgments. This file is used when updates to multiple databases (whether local TDS or external service providers) have occurred in the same TIBCO Object Service Broker transaction.

Sample Distributed Processing Scenarios

The following scenarios illustrate Fail Safe processing in environments where there are multiple TIBCO Object Service Broker systems with or without external database servers.

Two Data Object Brokers

Consider a transaction in which a TDS table is updated in Data Object Broker #1 and another TDS table is updated in Data Object Broker #2. The Data Object Brokers are assumed to be connected by a link where the Fail Safe level is 2 as illustrated in the following diagram:



In this case, Data Object Broker #2 is capable of Fail Safe level 2; however, because there is only one service provider and local TDS updates, Fail Safe level 1 provides adequate data integrity.

Commit Cycle for Two Data Object Brokers

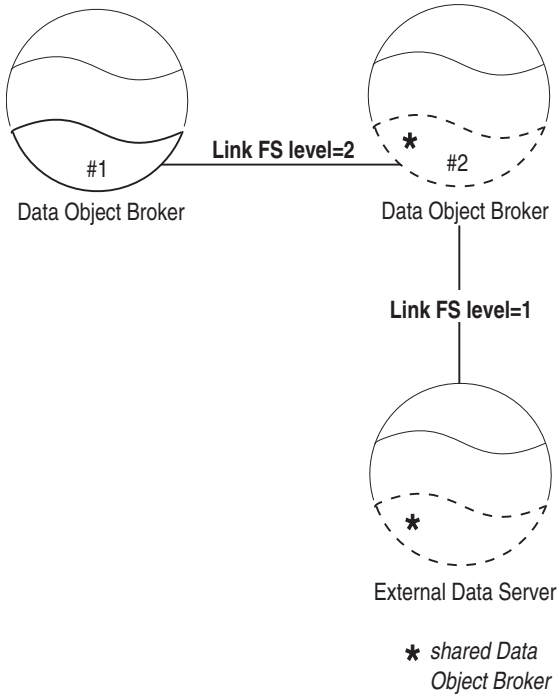
The sequence of events for this commit cycle is as follows:

1. Data Object Broker #1 stores its pending updates in the contingency log.
2. Data Object Broker #1 sends a Fail Safe level-1 commit to Data Object Broker #2.
3. Data Object Broker #2 stores its pending updates in its contingency log.
4. Data Object Broker #2 commits its updates.
5. When Data Object Broker #1 receives the success message from Data Object Broker #2, it commits its updates (if a failure message is received, it abandons the updates).
6. Data Object Broker #1 sends a contingency log release directive to Data Object Broker #2 so that the remote contingency log entry can be released.

The same processing occurs if Data Object Broker #2 is an external database server, except for steps #3 and #6.

Two Data Object Brokers with an External Database Server

If Data Object Broker #2 has an external database server attached to it involved in the same transaction, it processes the request as for Fail Safe level 1. This process is illustrated as follows:



Commit Cycle for Two Data Object Brokers with an External Database Server

The sequence of events for this commit cycle is as follows:

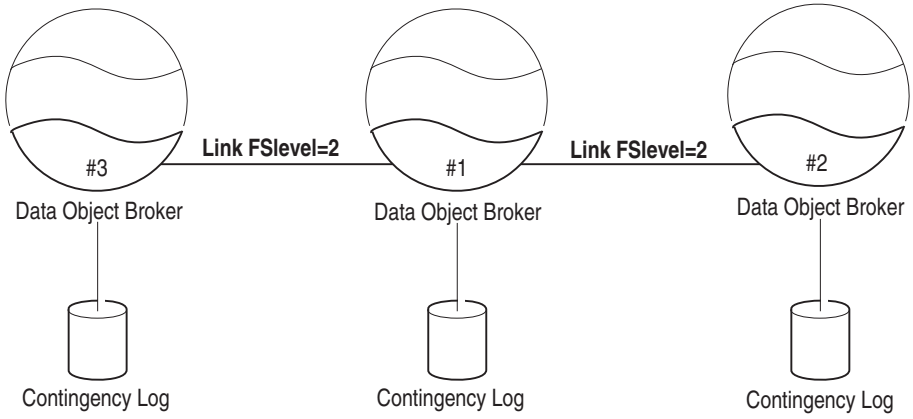
1. Data Object Broker #1 logs its intent list to the contingency log.
2. Data Object Broker #1 issues a Fail Safe level-1 commit to Data Object Broker #2.
3. Data Object Broker #2 writes its intent list to the contingency log.
4. Data Object Broker #2 sends a Fail Safe level-1 commit to the external database.
5. Data Object Broker #2 commits its TIBCO Object Service Broker transaction, previously saved in the contingency log, and signals Data Object Broker #1 (the commit coordinator) that the commit was successful.

6. Data Object Broker #1 completes its commit processing.

Data Object Broker #1 is not aware that the commit protocol between Data Object Broker #2 and the External Data Server is at Fail Safe level 1.

Three Data Object Brokers

Another possible scenario is a TIBCO Object Service Broker transaction with TDS table updates on three Data Object Brokers, illustrated as follows:



Commit Cycle for Three Data Object Brokers

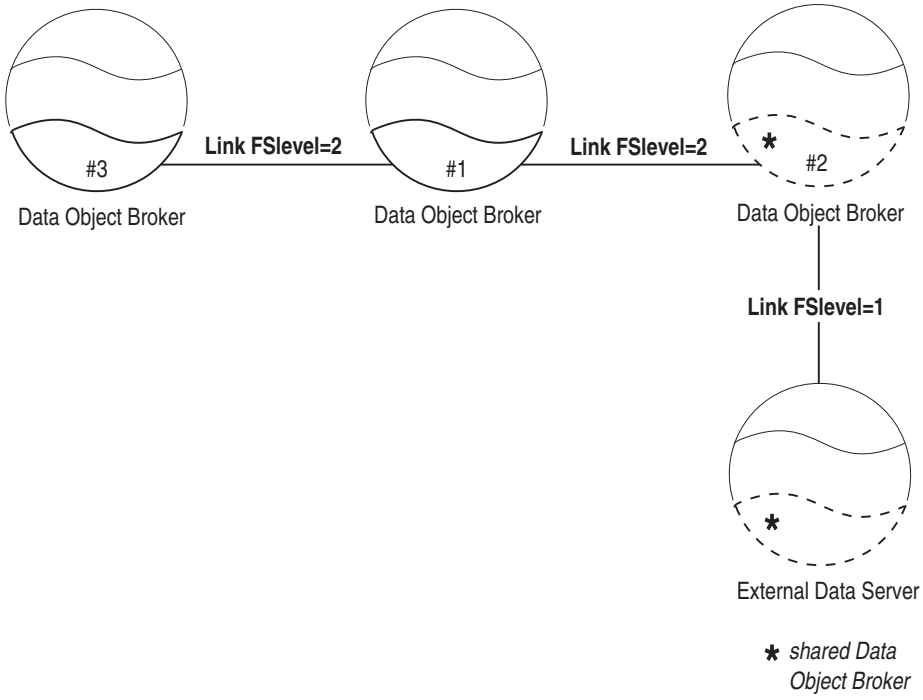
Since more than one updated service provider is involved in the transaction and each is capable of supporting Fail Safe level-2 processing, the sequence of events becomes:

1. Data Object Broker #1 issues a prepare to commit to Data Object Brokers #2 and #3.
2. Data Object Broker #1 saves its intent list in its contingency log.
3. Data Object Brokers #2 and #3 each save their intent lists in their contingency logs and then signal Data Object Broker #1 that the prepare to commit was successful.
4. Data Object Broker #1 issues a commit to Data Object Brokers #2 and #3 and does its own local commit.
5. After Data Object Brokers #2 and #3 confirm their commits and release their contingency logs, the user is notified.

Fail Safe level 2 does not require a contingency log release after the commit request.

Multiple Data Object Brokers with an External Database Server

The following scenario involves three Data Object Brokers and an external database server:



Commit Cycle for Three Data Object Brokers with an External Database Server

At transaction end, the sequence of events is as follows:

1. Data Object Broker #1 issues a prepare to commit request to Data Object Brokers #2 and #3 and it writes its intent list to the contingency log.
2. Data Object Broker #3 responds that the prepare to commit was successful.
3. Data Object Broker #2 sends back a message indicating it can handle only Fail Safe level-1 requests.
4. Data Object Broker #1 sends another message to Data Object Broker #2 telling it to perform a Fail Safe level-1 commit.
5. When this is complete, Data Object Broker #2 sends a success message back to Data Object Broker #1, which then completes the commits in Data Object Brokers #1 and #3.

Commit Behavior Across Multiple Data Object Brokers

The following table shows the commit-related actions for the more common combinations of TDS and Fail Safe level settings across two service providers:

TDS	Transaction updated: Service Provider #1	Transaction updated: Service Provider #2	Action
Y	Y, FSlevel = 0	N	1. Commit service provider #1. 2. Commit TDS.
Y	Y, FSlevel = 1	N	1. Write TDS updates to contingency log. 2. Commit service provider #1 including updates to the transaction database or equivalent. [a] 3. Commit TDS.
N	Y, FSlevel = 1	N	Commit service provider #1.
Y	Y, FSlevel = 1	Y, FSlevel = 1	Commit is rejected because a single transaction is not allowed to update more than one service provider with FSlevel=1.
Y	Y, FSlevel = 1	Y, FSlevel = 0	Commit is rejected because you cannot mix FSlevel=0 with FSlevel>0 in the same transaction.
N	Y, FSlevel = 1	Y, FSlevel = 0	Commit is rejected.

TDS	Transaction updated: Service Provider #1	Service Provider #2	Action
Y	Y, FSlevel = 1	Y, FSlevel = 2	1. Write intent to contingency log. 2. Issue prepare to commit to service provider #2. 3. Wait for response. 4. Issue Fail Safe level 1 commit to service provider #1. 5. Wait for response. 6. Commit local intent and issue commit to service provider #2.
Y	Y, FSlevel = 2	Y, FSlevel = 2	1. Prepare to commit issued for all three. 2. All three are committed.

a. When the service provider is another Data Object Broker, the contingency log provides an equivalent function to the transaction database for Fail Safe level-1 commits. This database is identified by the TRXDB server parameter. Refer to Transaction Database on page 22 for information on the TRXDB parameter.

Chapter 4 Understanding Checkpoint Processing

This chapter describes checkpoint processing in TIBCO Object Service Broker.

Topics

Checkpoint Processing

What is a TIBCO Object Service Broker Checkpoint?

A TIBCO Object Service Broker checkpoint is the process that synchronizes the committed transactions held in the redolog and the affected pages held in the resident page manager buffers with journals and the physical Pagestore.

When is a Checkpoint Created?

Based on...	A checkpoint starts when...
System administrator action	The administrator forces a checkpoint.
	The administrator forces a journal spin.
	The administrator varies a segment offline.
Time	The number of minutes elapsed since the last checkpoint reaches the limit set in the CHPTINTERVAL Data Object Broker parameter.
File usage	The redolog is 50% full.
	The resident page manager buffers are 15% full.
Activity on the database	The number of changed pages reaches the limit set in the CHPAGELIMIT Data Object Broker parameter.
	The number of commits issued from Execution Environments connected to this Data Object Broker reaches the limit set in the CHTRANLIMIT Data Object Broker parameter.

Steps Performed by the Checkpoint Process

There are three major steps performed by the checkpoint process.

1. Compress the data pages.

   The checkpoint process compresses all affected data pages in the resident page manager by removing the part of each page that does not contain data.

2. Calculate checkpoint size.

The process calculates how much space is required to write the checkpoint to the active journal.

3. Check active journal space availability.

   The process checks to see if the active journal has enough space to hold the checkpoint, and finds one of the following conditions:

   — Active journal has sufficient space for checkpoint.

   — Active journal has insufficient space for checkpoint.

Refer to the following sections for more information.

Active Journal has Sufficient Space for Checkpoint

When the active journal has sufficient space to hold the checkpoint, the following occurs:

1. The changed data pages are written to the active journal.
2. The affected area of the redolog is released back to the system.
3. The changed data pages are physically written to the Pagestore.
4. A control page in the journal is updated to indicate that the checkpoint was successful.

   The journal pages written out in the checkpoint are now available for use by the continuous backup process. They are no longer considered to be part of the cached portion of the journal.

Active Journal has Insufficient Space for Checkpoint

When the active journal does not have sufficient space to hold the checkpoint, the following occurs:

1. The journal is off-loaded to a spin file. The checkpoint process starts the spin01 or spin02 batch file/script, depending on which journal is active.
2. The alternate journal is made active.
3. The number of spin files is checked against the *SPINLIM* environment variable to determine if a merge of the spin files is required.
4. If a merge is required, the checkpoint process starts the spin03 batch file/script to merge all existing spin files into a master journal accumulation file (spinout.000).

5. The spun journal is released back to the system.

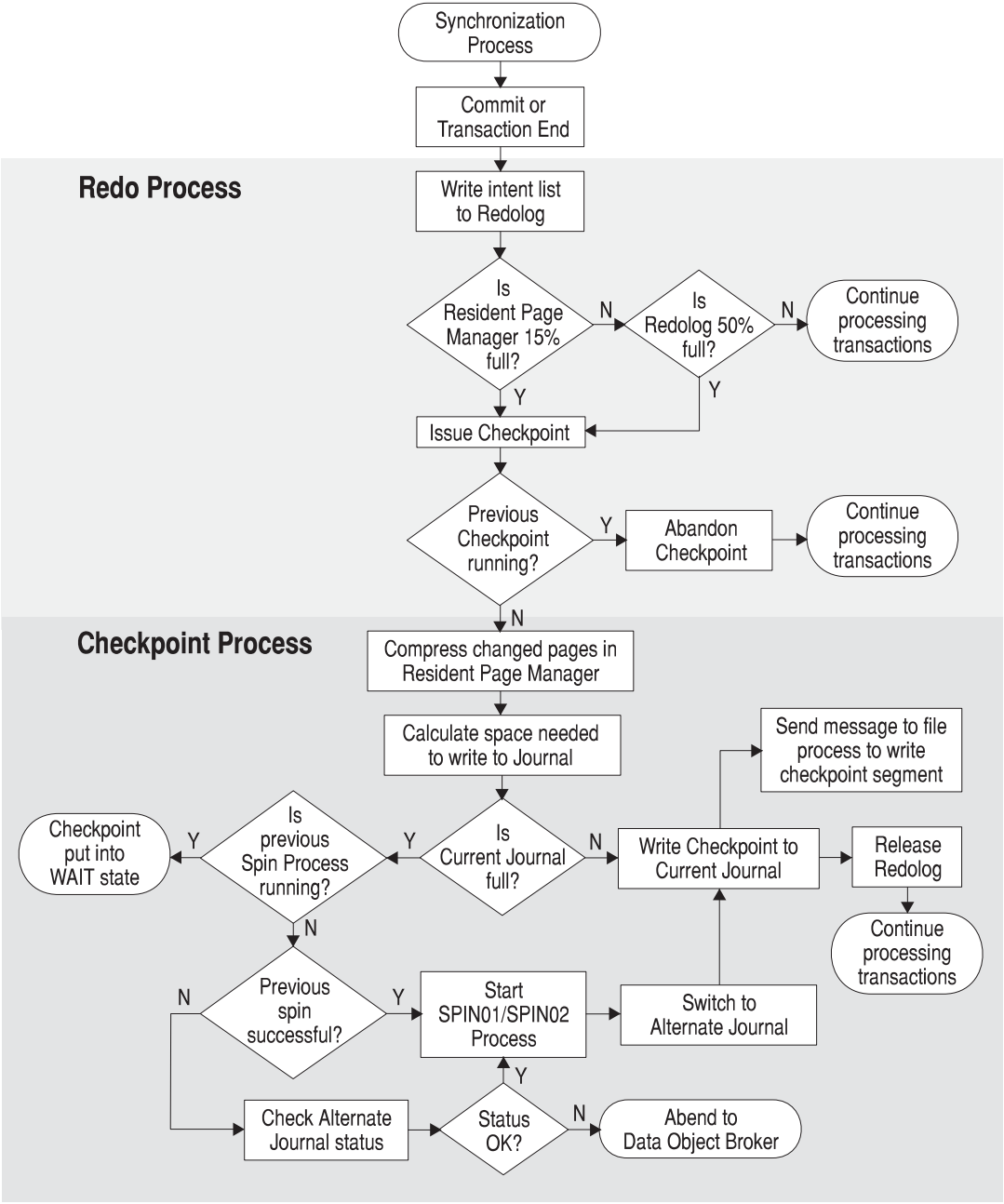
Only one checkpoint can be in progress at a time. If a segment is being varied offline while a checkpoint is in progress, its state is internally adjusted to indicate that it is not available for transactions, but it is actually put offline only after the next checkpoint (which propagates all related changes) is finished.

If a journal spin is forced during a checkpoint, it is also delayed until the checkpoint in progress is finished. The request for the checkpoint is ignored if the checkpoint is already in progress.

Completing Pending Transactions

At system startup, the active journal is checked to see if the last checkpoint completed. If it did not complete, the checkpoint is restarted using the cached portion of the journal. This portion contains the changed pages that were not physically written to the Pagestore. In addition, the redolog is checked and used to complete any pending transactions.

Checkpoint Process Illustration



Chapter 5 Understanding Journal Processing

This chapter describes journal processing in TIBCO Object Service Broker.

Topics

Journal Processing

Definition

TIBCO Object Service Broker journals are data sets that provide a record of all modified page images. Journals are essential to the TIBCO Object Service Broker continuous backup and recovery procedures.

By default, continous backup is not enabled. It must be explicitly enabled for a system. Refer to , Performing Continuous Backup, page 51 for more details.

Overview

This chapter explains how TIBCO Object Service Broker:

- Spins (off-loads) a journal
- Switches between journals

Journal backups can be merged together and used to refresh your current system backup. For guidance in developing a continuous backup procedure that meets your unique backup and recovery requirements, read this chapter and then refer to Chapter 6, Backing Up Your System, on page 45.

When are Journals Spun?

When one journal file reaches capacity, the Data Object Broker automatically switches to the other. The Data Object Broker still continues processing transactions during this switch. The journal that is almost full is then *spun* (emptied of its contents) and made available for use again.

Journal spinning is normally performed under the control of the batch files (for Windows) or scripts (Solaris): spin01, spin02, and spin03. These files are shipped with a default behavior, which can be tailored at any time by the TIBCO Object Service Broker system administrator.

Tailoring the Spin Batch Files/Scripts

The sample batch files or scripts for journal spinning can be found in the following directory:

- Windows: %OS_ROOT%\bin
- Solaris: ${OS_ROOT}/bin

In Windows, the spin01.bat and spin02.bat files reset the journal for reuse. In Solaris, the spin01 and spin02 scripts reset the journal for reuse. To turn on journal spinning for continuous backup, do one of the following:

- Windows: Copy the spin01.nt, spin02.nt, spin03.nt, spin03A.nt, spin04.nt, spin05.nt and spin05A.nt files to the corresponding name with a .bat extension.
- Solaris: Copy the spin01.alt and spin02.alt scripts to the corresponding name without an extension.

Edit each batch file/script to ensure the following environment variables are set correctly:

SPINLIM	The number of spin files to be merged into a master accumulation file. Set this variable only in spin01.bat and spin02.bat in Windows or spin01 and spin02 in Solaris.

SPINDIR	Set this variable as follows:
	Windows:
	• In spin01.bat and spin02.bat:
	Type the full path name of where the spin files are to be located, for example:
	`c:\Ostar\database\JOURNAL`
	• In spin03.bat and spin05.bat:
	Type the path name of the spin files without specifying the drive, for example:
	`\Ostar\database\JOURNAL`
	Solaris:
	In spin01, spin02, spin03, and spin05, type the full path name where the spin files are to be located, for example:
	`${OS_ROOT}/database/JOURNAL`

SPINVOL (**`Windows only`**)	The drive where the spin files are to be located, for example: `C`. Set this variable in spin03.bat and spin05.bat only.
BACKNAME	The name of your latest full backup. Set this variable only in spin05.bat in Windows or spin05 in Solaris. The default is backup.000.

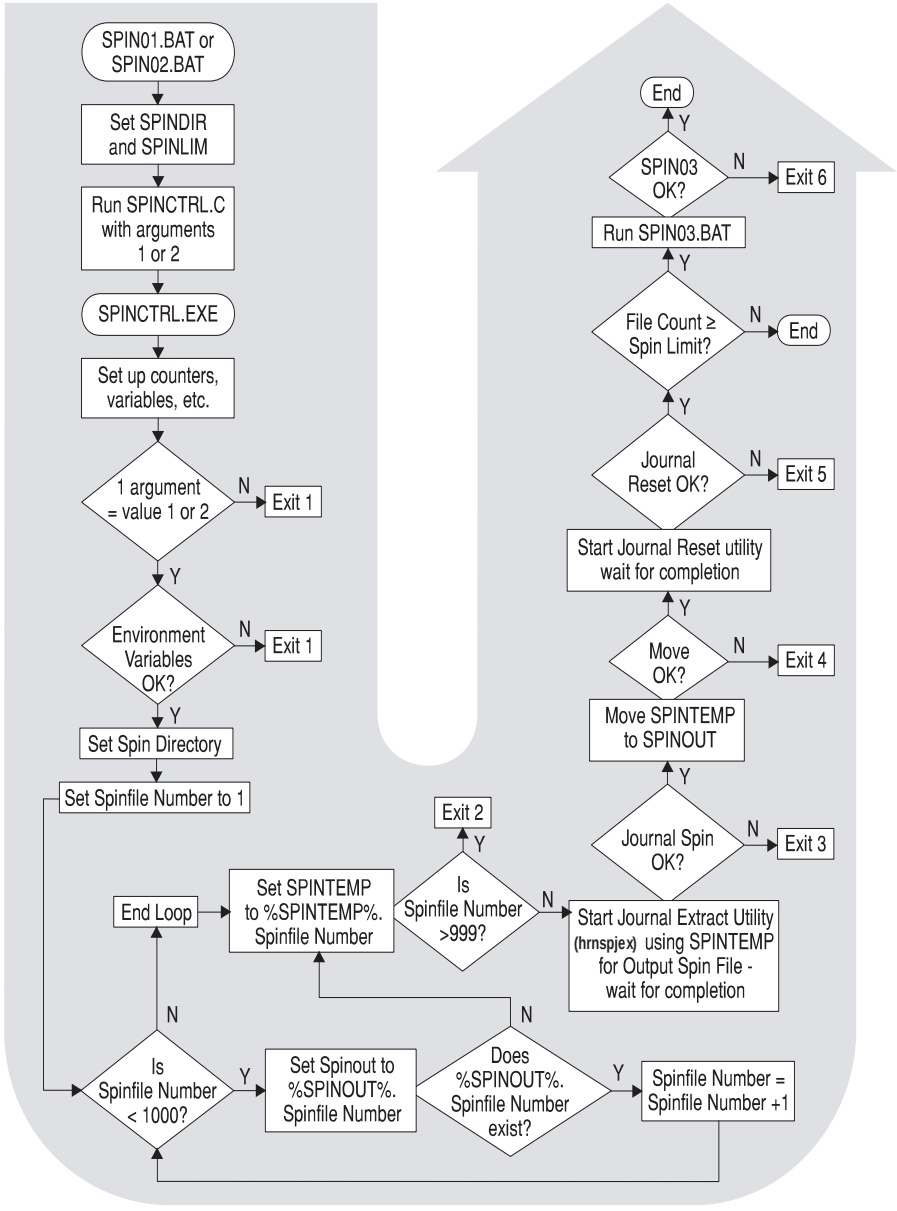
Journal Spin Process

The journal spin process uses the hrnspjex (Journal Extraction) utility to extract relevant records from JOURNAL1 or JOURNAL2 and produces an audit trail. At the end of the journal off-load, the hrnspset (Reset Journal) utility resets the journal and changes its state to EMPTY so that it can be reused.

If the second journal fills up before hrnspset can be run for the first journal, the system abends.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information about the hrnspjex and hrnspset utilities.

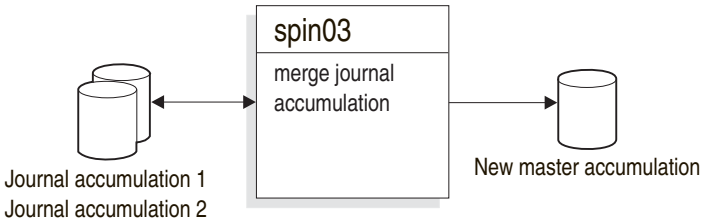
Journal Spin Process Illustration



Journal Operations

Merging the Journals

When the hrnspset utility is done, the spin process determines how many journal accumulations exist. If the limit is exceeded (default is 3), spin03 uses the hrntlmrg (Journal Merge) utility to merge the journal accumulations, as shown in the following diagram:



Adjusting the Accumulation Limit

Adjust your accumulation limit according to the following:

- Disk space availability

  For example, to reduce the amount of disk space taken up by journal accumulation files, reduce the limit.

- Your requirements for immediate recovery of critical data

  For example, if your data is critical, merge your journal accumulation files after every generation, then merge your master accumulation with your latest backup.

Setting Journal Size

Journals must be large enough so they do not fill too often during the day, but they must also be small enough that the contents of a full journal can be saved before the other journal fills up. Experiment to find a good journal size for your system. As a guideline, try sizing your journals to hold about four hours of work (that is, a journal fills up in about four hours under average conditions).

Minimum Journal Size

There is a minimum size for journals: the journal should have at least as many pages as there are resident pages defined in the crparm file. Both formatting and opening fail if the journal has fewer pages than the minimum size, as determined by the current value of the RESIDENTPAGES Data Object Broker parameter.

Full Journals

If a journal fills before the other journal is available, the Data Object Broker abends and displays a message indicating that the journal is unavailable.

Upon the restart of the Data Object Broker under this condition, the Data Object Broker reruns the spin job on the journal that is currently active. If the original problem that caused the filling up of the journal is not corrected before restart, the Data Object Broker shuts down again. It is therefore necessary to correct whatever condition caused the other journal to be unavailable before restarting the Data Object Broker. To retain the journal data, spin the journals manually using the hrnspjex utility before restarting the Data Object Broker.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on using the hrnspjex and hrntlmrg utilities.

Spin Batch File/Script Failures

If a spin job fails, a spinlog.nnn file is created, where *nnn* matches the TIBCO Object Service Broker Data Object Broker log file (hrncr.nnn) number in use when the error occurred. The log contains the date and time of the error along with any messages from the hrnspjex and/or hrnspset utilities. Messages concerning any errors encountered in the following are available in the Data Object Broker log:

- Windows: the spin batch files or the spinctrl C program
- Solaris: the spin scripts

In the illustration in Journal Spin Process Illustration on page 41, several exit codes are included. They correspond to the following conditions:

Exit Code	Description
1	Invalid spin # — must be 1 or 2. (or) Environment variables incorrect.
2	More than 999 spinout files.

Exit Code	Description
3	hrnspjex error — check spinlog.nnn for information.
4	Could not open SPINTEMP to SPINOUT.
5	hrnspset failed — check spinlog.nnn for information.
6	spin03 failed.

Chapter 6 Backing Up Your System

This chapter describes different option for backing up your system.

Topics

Overview

Introduction

You can use TIBCO Object Service Broker to meet your backup and recovery requirements, whether you need regularly scheduled full system backups or a continuous backup approach that you can use to stay fully operational at all times.

You can also back up and recover your Data Object Broker database using the TIBCO Object Service Broker Database Administrator tool. Refer to *TIBCO Object Service Broker for Open Systems Installing and Operating*.

Backup Approaches

There are two different approaches to developing a backup strategy for your TIBCO Object Service Broker system:

- Full system backup
- Continuous backup

These approaches are outlined in the following sections. Your TIBCO Support representative can help you build a backup and recovery plan that integrates one or both of these approaches to suit your requirements.

Full System Backup

We recommend that you perform a full backup immediately after you install TIBCO Object Service Broker. You can also use this method as part of your regularly scheduled backup procedure. There are two ways you can perform a traditional full system backup:

Full System Backup…	Refer to page…
Using TIBCO Object Service Broker backup utilities.	48
Using non-TIBCO Object Service Broker backup utilities.	50

Continuous Backup

The TIBCO Object Service Broker continuous backup process enables you to maintain 24-hour operations and ensure full system recoverability at all times. Using continuous backup, you never have to shut down TIBCO Object Service Broker for a routine backup. Continuous backup combines the most recent backups with updates in the journals to produce a complete and current backup copy.

TIBCO Object Service Broker Backup Utilities

Advantages to this Approach

There are several advantages to using the TIBCO Object Service Broker backup utilities method over other backup tools:

- You can back up the entire system or individual segments. You can back up any segment other than segment 0 by varying the segment offline, backing it up, and varying it back online. To back up segment 0, you must shut down TIBCO Object Service Broker.
- The backup copies only those data pages actually being used and compresses the data before writing it to disk.
- The backup can be merged with journals created at a later time to create a more up-to-date backup.
- The hrnbrptr (Batch Pointer Check) utility can be used to verify backup integrity.

If you use FTP to transfer segment backups between Windows or Solaris and z/OS, you must reformat the unloaded data using the S6BBRFRU (Reformat TIBCO Object Service Broker Files Transferred with FTP) z/OS utility before it can be used by TIBCO Object Service Broker for z/OS. Refer to *TIBCO Object Service Broker for z/OS Utilities* for more information on the S6BBRFRU utility.

Using TIBCO Object Service Broker Backup Utilities

Before starting this procedure, you must shut down the Data Object Broker. Complete the following steps from a Windows or Solaris command prompt:

1. Change to the directory that contains your backup files.

   It is recommended that you place backup files in a different directory than TIBCO Object Service Broker, for example:

   ```
   c:\OSTAR_BKUPS (Windows)
   /usr1/OSTAR/OSTAR_BKUPS (Solaris)
   ```

2. Use the hrntlbps (Backup Pagestore) utility to create a backup of the MetaStor and segment 1.

   The following two commands create a backup in the file backup.000:

   ```
   hrntlbps -s 00 -w backup.000
   hrntlbps -s 01 -a backup.000
   ```

where:

-w	Specifies overwrite backup file if it exists.
-a	Specifies append to end of an existing backup file.

3. Restart the Data Object Broker.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on using the hrnbrptr and hrntlbps utilities.

Using Non-TIBCO Object Service Broker Backup Utilities

Limitations to this Approach

While non-TIBCO Object Service Broker backup methods have the advantage of being relatively simple and fast, they are not space efficient. They save the entire Pagestore instead of just the used pages. when you have backed up your system using one of these methods, your only method of restoring the system is to completely overlay the existing system with the backup copy. In addition, you cannot use the hrnbrptr (Batch Pointer Check) utility to verify backups taken using non-TIBCO Object Service Broker backup utilities.

Creating a Backup Using Non-TIBCO Object Service Broker Methods

To create a backup of the system using a non-TIBCO Object Service Broker backup utility, complete the following steps:

1. Shut down TIBCO Object Service Broker.
2. Use the product normally.

Performing Continuous Backup

Using the Continuous Backup Process

Instead of taking full or partial system backups at regular intervals, use the following continuous backup process and journals to update the latest backup:

1. Customize the spin01, spin02 and spin03 batch files/scripts.

   The checkpoint process automatically submits spin01 and spin02 as required. When spin01 and spin02 are customized, they invoke spin03 when the spin limit is reached. The remaining steps are not started automatically and must be scheduled to run as required.

   For the full procedure to customize the spin files, refer to Chapter 5, Understanding Journal Processing, on page 37.

2. Merge your journal accumulation files into a master accumulation file (spin03.bat in Windows, spin03 in Solaris).

   Spin03.bat in Windows and spin03 in Solaris are provided to perform this step. The output from this step is a master accumulation file called spinout.000. The next time the merge process runs, the spinout.000 file is merged with the existing journal accumulation files.

For Windows, the merge is automatically performed by spinctrl when the number of journal accumulation files reaches the value of the *SPINLIM* variable set in spin01 and spin02.

3. Merge your master accumulation file with your latest complete backup (sample process in spin05.bat in Windows, spin05 in Solaris).

   You can merge your master accumulation file with your latest complete backup as frequently as required. In most cases, we recommend a daily refresh of your complete backup. The spin05 batch file/script is provided as a sample to perform this merge. Alternatively, you can use the method described in Alternate Method to Refresh the Latest Backup.

4. Run the hrnbrptr (Batch Pointer Check) utility against any complete backup you produce to validate the integrity of all pages.

Alternate Method to Refresh the Latest Backup

Use the hrntlmrg (Journal Merge) utility to read in your journal accumulation file (or files) and your latest complete backup. The output is a new and complete backup that includes all page updates recorded in the journal accumulation files. For example, the following command creates a new backup file (backup.new) from the old backup file (backup.old) and the journal accumulation files (SPINOUT.001 and SPINOUT.002):

```
hrntlmrg backup.old SPINOUT.001 SPINOUT.002 backup.new
```

Since no segment is specified, all pages from all segments in the journal accumulation files are included in the new backup file. You must manually delete the old files.

Considerations

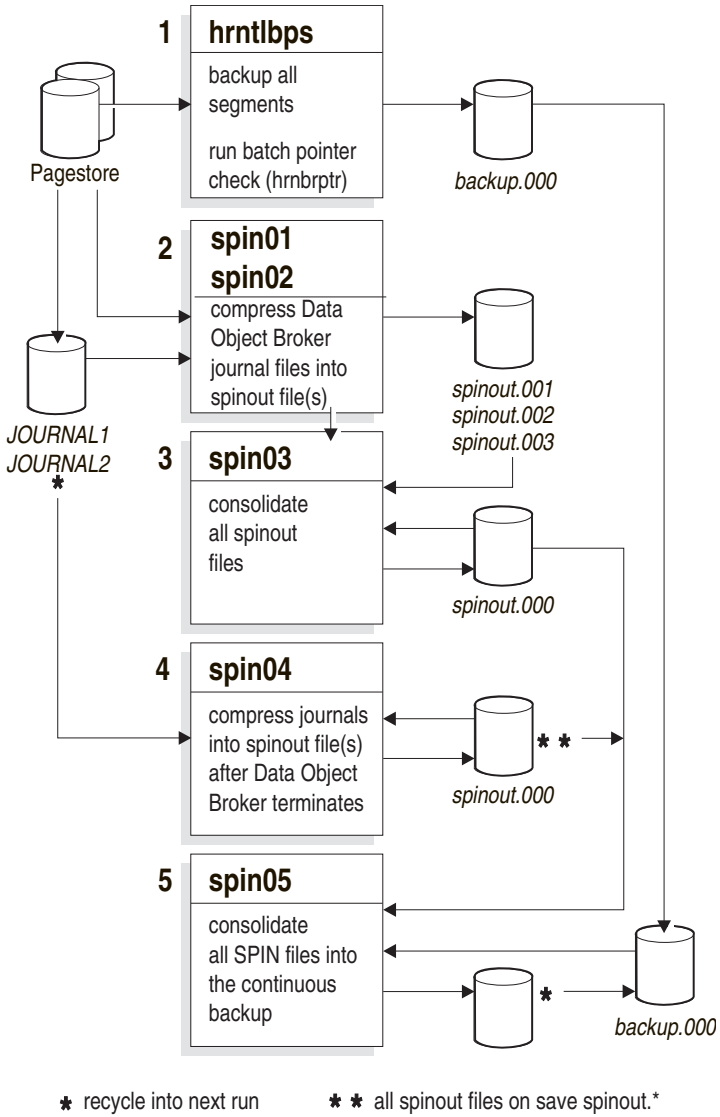
- Make sure your master accumulation file (spinout.000) is not in use by the automatic merge process spin03.bat when you run this step. This process should be scheduled to run at a time when the system is not busy running frequent spins.
- There is a limit to the number of bytes that can be passed as a parameter on the command line: approximately 200 bytes or 16 filenames that are each 12 bytes in length. If your *SPINLIM* variable is set higher than 14, ensure your procedure merges only 14 journal accumulation files at a time.
- Continuous backup and recovery works on completed (committed) transactions. Transactions in the contingency log are not complete and are not part of the backup process. For more information on the contingency log, refer to Chapter 1, Introducing TIBCO Object Service Broker Backup and Recovery Components, on page 1.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on using the hrnbrptr and hrntlmrg utilities.

Some utilities can modify the contents of the database but do not write journal records. After running such utilities, you must recommence your continuous backup.

Sample Continuous Backup Implementation

The following diagram illustrates a sample continuous backup strategy.



* recycle into next run * * all spinout files on save spinout.*

Continuous Backup Steps

To develop a customized plan, consult TIBCO Support. The numbers in this diagram correspond to the following steps:

1. Prime the continuous backup process with an initial set of page images using hrntlbps (backup.000).

The MetaStor (segment 0) must be included in your backup. The MetaStor identifies in which segment each table resides. Always run the hrnbrptr utility to ensure valid database pointers.

2. All page update images are saved in JOURNAL1 and JOURNAL2. When one fills, the Data Object Broker automatically starts spin01 or spin02 to save the journal data.
   - For Windows, spin01 and spin02 start a C program called spinctrl. Spinctrl copies journal update images into journal accumulation files called spinout.001, spinout.002, spinout.003, and so on
   - For Solaris, *spin01* and *spin02* copy journal update images into journal accumulation files called spinout.*001*, spinout.*002*, spinout.*003*, and so on
3. The defined limit of the *SPINLIM* variable is checked each time spinctrl (for Windows) or *spin01*/*spin02* (for Solaris) is started. When the number of spinout files reaches the limit, spin03 consolidates them into one file, spinout.000. All other spinout files are deleted. The hrntlmrg utility keeps only the most current page images.

When spin03 is run again, it merges spinout.000 with any new spinout files to create a new spinout.000.

4. The information in JOURNAL1 and JOURNAL2 is merged with existing spinout files by executing the spin04 batch file/script. The Data Object Broker must be shut down for spin04 to be successful.
5. Run spin05 on a regular basis to consolidate all updated journal page images into the continuous backup file backup.000. After spin04 and spin05 have run, backup.000 is identical to a complete TIBCO Object Service Broker backup of all segments using hrntlbps.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on using the hrntlbps and hrntlmrg utilities.

Chapter 7 Recovering From Errors

This chapter describes different options for recovering your system from errors.

Topics

Overview

Types of Recovery

There are two types of recovery to consider:

- Restoring your entire system from a previous full backup
- Restoring portions of your system through the continuous backup procedures

This chapter discusses both of these approaches.

Running the Batch Pointer Check Utility

We recommend that you run the hrnbrptr (Batch Pointer Check) utility against each new backup to ensure its integrity. If you did not run this utility against the backup you plan to use for recovery, you should do so before attempting your recovery.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on the hrnbrptr utility.

Deciding How Much To Restore

The first step in the recovery process is to determine the nature and scope of the damage from which you need to recover.

Task	Type of Recovery	Refer to page…
Restore entire system to its level just before problem occurred.	Full recovery	57
Restore only affected segments to their level before the problem occurred.	Partial recovery	60

Full Recovery

When Should the Entire System be Restored?

A full system recovery replaces the entire current system with a backup copy. You normally perform a full recovery only when the system is corrupted (data is altered or destroyed) or after a media failure.

Deciding on a Restoration Point

Before beginning, decide on an appropriate restoration point. You can:

- Restore your most recent full backup copy.

  If the backup was taken three days ago, at the end of the restore process, TIBCO Object Service Broker is exactly as it was three days ago. None of the changes made to the system since the backup was taken are reflected.

- Use the continuous method to update the backup before you restore.

  Run the spin04 and spin05 batch files/scripts, if necessary, to ensure that the backup is as up-to-date as possible.

Using TIBCO Object Service Broker Backup Utilities to Perform a Full Recovery

1. Determine the number of 4 KB pages assigned to each TIBCO Object Service Broker file to be restored.

   This can include:

   — The page files in segment 0 and segment 1

   — The redolog

   — The journals

   — The contingency log

2. Shut down TIBCO Object Service Broker.
3. Create a file containing the latest page images.

   These images are used to restore the damaged segment. For recovery up to your last checkpoint, off-load the journals and merge them with the journal accumulations and your system backup using the spin04 and spin05 batch files/scripts.

4. Rebuild each segment as follows:

   a. Reformat the page data files using the hrntlfps (Format Pagestore) utility. For example:

Windows	`hrntlfps -w %OS_ROOT%\database\seg01\PAGE1`
	`hrntlfps -w %OS_ROOT%\database\seg01\PAGE2`
	`hrntlfps -w %OS_ROOT%\database\seg01\PAGE3`

Solaris	`hrntlfps -w ${OS_ROOT}/database/seg01/PAGE1`
	`hrntlfps -w ${OS_ROOT}/database/seg01/PAGE2`
	`hrntlfps -w ${OS_ROOT}/database/seg01/PAGE3`

   b. Restore the files. Use the hrntlrps (Restore Pagestore) utility for each affected segment to restore its contents from the backup file. For example:

```
hrntlrps -s 0 backup.000
hrntlrps -s 1 backup.000
```

The MetaStor and each segment must be restored using backups taken in the same time frame. The MetaStor defines all tables, their fields, and the segment where each table resides.

5. Rebuild the operations files:

Ensure you spin any pages held in the journals (spin04.bat in Windows, spin04 in Solaris) prior to executing this step.

   a. Use the hrntlfjr (Format Journal) utility to reformat JOURNAL1 and JOURNAL2. For example:

Windows	`hrntlfjr -w %OS_ROOT%\database\JOURNAL\JOURNAL1` `hrntlfjr -w %OS_ROOT%\database\JOURNAL\JOURNAL2`
Solaris	`hrntlfjr -w ${OS_ROOT}/database/JOURNAL/JOURNAL1` `hrntlfjr -w ${OS_ROOT}/database/JOURNAL/JOURNAL2`

   b. Use the hrntlfrl (Format Redolog) utility to initialize the redolog. For example:

Windows	`hrntlfrl -w %OS_ROOT%\database\REDO\REDOLOG`
Solaris	`hrntlfrl -w ${OS_ROOT}/database/REDO/REDOLOG`

6. Manually abort any in-doubt transactions in the contingency log affecting the restored segments and timestamped later than the time to which the segments were restored.

   To do this, invoke the hrntlfcl (Format Contingency Log) utility as follows:

Windows	`hrntlfcl -w %OS_ROOT%\database\CLOG\CLOG`
Solaris	`hrntlfcl -w ${OS_ROOT}/database/CLOG/CLOG`

7. Restart TIBCO Object Service Broker.

Restoring from a Non-TIBCO Object Service Broker Backup

If the system is backed up using a third-party backup program, it must be restored in the same way as any other disk backed up using that utility. After restoration, your system is back to the state at which the backup was taken; all subsequent changes are lost.

See Also *TIBCO Object Service Broker for Open Systems Installing and Operating* or *TIBCO Object Service Broker for Open Systems Utilities* for detailed instructions for using the utilities in the examples provided.

Partial Recovery

If your analysis indicates that the damage is isolated to a specific Pagestore segment, the following steps indicates how to recover just that segment. If you are unsure if other segments are affected, the safest approach is to restore all segments.

Do not restore a segment to a point prior to the last physical definition change for any table held in that segment. Access errors can result when users attempt to use the table, due to a mismatch between the physical data and the recovered table definition. Make sure your MetaStor (segment 0) backup corresponds with the segment being restored.

Implementing a Partial Recovery

1. Make sure that there are no pending updates.
   a. To restore segment 0 (the MetaStor), shut down TIBCO Object Service Broker.
   b. To restore any other segment, vary the affected segment offline by using the following command:

      ```
      hrncr dboffline=segmentname or segmentnumber
      ```

   For example:

   ```
   hrncr dboffline=sales
   hrncr dboffline=3
   ```

   You can also vary a segment offline using the Administration menu. Refer to *TIBCO Object Service Broker for Open Systems Installing and Operating* for more information about the menu's options and operator commands.
2. Use the hrntlrps (Restore Pagestore) utility to rebuild the segment to be recovered from the backup.

   Use a recent backup of the affected segment or your most recent full system backup. For example, to rebuild segment 1 only, type the following:

   ```
   hrntlrps -s 1 backup.000
   ```
3. Bring the affected segment online or restart TIBCO Object Service Broker.

   The command to vary a segment online is:

   ```
   hrncr dbonline=segmentname or segmentnumber
   ```

   You can also bring the segment online from the Administration menu.

Recovering From Non-Page File Failures

While journal and backup information can be used to recover page data files, you can encounter other failures from which you must be prepared to recover. Other files that can fail include: JOURNAL1, JOURNAL2, the redolog, or the contingency log. The following describes these situations.

Redolog Failure

If the redolog becomes damaged, delete and re-initialize it.

- Windows: *%OS_ROOT%*\database\REDO\REDOLOG
- Solaris: *${OS_ROOT}*/database/REDO/REDOLOG

All committed changes made since the last checkpoint are lost in the event of the loss of the redolog.

Run hrntlfrl to re-initialize the redolog. When the redolog is re-initialized, restart the Data Object Broker.

Contingency Log Failure

If the contingency log becomes damaged, delete and re-initialize it using the hrntlfcl (Format Contingency Log) utility.

- Windows: *%OS_ROOT%*\database\CLOG\CLOG
- Solaris: *${OS_ROOT}*/database/CLOG/CLOG

Loss of contingency log data can result in data inconsistency in a distributed data environment. If your contingency log fails, contact TIBCO Support for assistance.

Failure of the contingency log impacts the distributed network of which the node is a part. This can require manual intervention, using the Administration menu (hrntladm), on some or all of the other networked nodes.

Journal Failure

The journals provide an audit trail of all changed physical pages. If one of your journals fails, you are unable to recover for subsequent page file failures. For this reason, you should immediately reset the continuous backup process with a new master backup.

After the Data Object Broker is started, use subsequent journal images, along with your backup, to recover any page file failures up to the time of failure. Although the page images contained within the failing journal are lost, your repository and other control files are still intact.

Recovering a Journal

1. Back up your segments immediately.

   This primes your continuous or discrete backup process.

2. Check the backups using the hrnbrptr (Batch Pointer Check) utility.
3. Reformat and re-initialize your journal files using the hrntlfjr (Format Journal) utility.
4. Restart the Data Object Broker.

The journal has a two-fold function: as part of continuous backup, and as part of checkpoint processing. When acting as a part of continuous backup, the loss of the journal means that a new full system backup must be created. For checkpoint processing, the journal is performing a caching role; if it is damaged before the last checkpoint is fully propagated to the Pagestore, it can impact Pagestore integrity.

See Also *TIBCO Object Service Broker for Open Systems Utilities* for information on the utilities.

Index

Numerics

A

B

C

D

E

F

H

I

J

L

M

N

O

P

R

S

T

U

V