Objectivity/DB

Objectivity/DB High Availability

Release 12.

Objectivity/DB High Availability

Part Number: 11.2.6-HA-0

Release 11.2.6, May 6, 2015

Contents

About This Book **7**
Audience 7
Organization 7
Conventions and Abbreviations 8
Getting Help 9

Part 1 INTRODUCTION

Chapter 1 Autonomous Partition Basics **13**
Autonomous Partitions 14
Partitioned Federated Databases 14
Databases Controlled by a Partition 15
Control and Containment 15
Objectivity/DB System Resources 16
Resources for an Unpartitioned Federated Database 16
Resources for a Partitioned Federated Database 17
Boot Autonomous Partition 19
Location of Files and Processes 19
Access to Databases 20
Improved Fault Tolerance and Performance 21
Improved Fault Tolerance 21
Improved Performance 22
Example 22
Usage Guidelines 25

Access to Global Federated-Database Information 26
Quorum of Partitions 26
Support for Restricted Access 27

Chapter 2 Data Replication Basics 29

Data Replication 30
Database Images 31
Images and Autonomous Partitions 31
Location of Images 32
Logical and Physical Model 32
Access to Replicated Databases 33
Access to an Individual Image 33
Access to a Quorum of Images 34
A Closer Look at Quorums 35
Image Weight 35
Quorum Calculation 35
List of Quorum Images 36
Automatic Quorum Calculation 36
If a Quorum is No Longer Possible 37
How a Quorum Ensures Data Integrity 37
Strategies for Assigning Weights to Images 38
Example of Evenly Weighted Images 38
Example of Unevenly Weighted Images 40
Hot-Failover Configurations 42
Tie-Breaker Partitions 43
Two-Machine Handler Functions 44
Restoring Images Into Service 45
Quorum of Partitions Revisited 46

Part 2 USAGE

Chapter 3 **Autonomous Partition Tasks** 49

About Autonomous Partitions 50

Required Access to Autonomous Partitions 51

General Administrative Tasks 52

- Backing Up a Partitioned Federated Database 52
- Restoring a Partitioned Federated Database 53
- Recovering Incomplete Transactions 53
- Installing a Partitioned Federated Database 54
- Exporting From a Partitioned Federated Database 55
- Importing Into a Partitioned Federated Database 55

Enabling Objectivity/Assist to Show Partitions 56

Getting Autonomous-Partition Information 57

- Listing All Partitions and Their Properties 57
- Listing the Properties of a Partition 57
- Listing Control Relationships for a Partition 57

Creating an Autonomous Partition 58

- Planning for the New Partition 58
- Procedure for Creating the New Partition 58

Changing Autonomous-Partition Properties 59

- Moving a Partition's System Resources 59
- Changing the Offline Status 60
- Changing the Weight 60

Managing Controlled Databases 62

- Creating a Database in the Controlling Partition 62
- Transferring Control of a Database to Another Partition 62

Deleting an Autonomous Partition 63

Purging Autonomous Partitions 63

Chapter 4 **Data Replication Tasks** **65**

About Database Images 65

Required Access to Autonomous Partitions 66

Getting Database-Image Information 67

- Listing Images of All Databases 67
- Listing Images of a Particular Database 68
- Listing the Properties of a Database Image 68

Creating a Database Image 68

Changing Database-Image Properties 69

- Changing a Database Image's Weight 70
- Moving a Database Image's File 70
- Transferring Control of an Image to Another Partition 71
- Making Database Images Read-Only or Read/Write 71

Managing Tie-Breaker Partitions 72

- Listing the Tie-Breaker Partition 72
- Setting the Tie-Breaker Partition 72
- Removing the Tie-Breaker Partition 73

Deleting a Database Image 73

Checking the Accessibility of Database Images 74

Restoring Database Images into Service 74

Part 3 REFERENCE

Chapter 5 **Tools** **77**

Index **91**

About This Book

This book describes the concepts and tasks you must understand to deploy and administer workgroup and enterprise applications using Objectivity/DB High Availability (Objectivity/HA). It describes how to use the Objectivity/HA tools to administer autonomous partitions and replicated databases in development and end-user environments.

You can perform similar tasks from within an application; see the documentation for your Objectivity/DB programming interface.

NOTE Earlier releases of Objectivity/HA were divided into two products—namely, Objectivity/DB Fault Tolerant Option (Objectivity/FTO) and Objectivity/DB Data Replication Option (Objectivity/DRO). You may still see these names, or abbreviations such as (*FTO*) and (*DRO*), in some Objectivity books.

Audience

This book is intended for database administrators who set up and administer Objectivity/HA. This book assumes you are familiar with *Objectivity/DB Administration*.

Organization

- Part 1 introduces Objectivity/HA and the basic concepts that are fundamental to understanding autonomous partitions and replicated data.
- Part 2 describes the tasks for setting up and administering Objectivity/HA.
- Part 3 provides reference descriptions of Objectivity/HA tools.

Conventions and Abbreviations

Navigation

In the online version of this book, table of contents entries, index entries, cross-references, and underlined text are hypertext links.

Typographical Conventions

`cd`	Command, literal parameter, code sample, filename, pathname, output on your screen, or Objectivity-defined identifier
`installDir`	Variable element (such as a filename or a parameter) for which you must substitute a value
Browse FD	Graphical user-interface label for a menu item or button
lock server	New term, book title, or emphasized word

Abbreviations

(administration)	Feature intended for database administration tasks
(HA)	Feature of the Objectivity/DB High Availability product
(*ODMG*)	Feature conforming to the Object Database Management Group interface

Command Syntax Symbols

[...]	Optional item. You may either enter or omit the enclosed item.
{…}	Item that can be repeated.
...|...	Alternative items. You should enter only one of the items separated by this symbol.
(…)	Logical group of items. The parentheses themselves are not part of the command syntax; do not type them.

Command and Code Conventions

In code examples or commands, the continuation of a long line is indented. Omitted code is indicated with the ellipsis (…) symbol. “Enter” refers to the standard key (labeled either Enter or Return) for terminating a line of input.

Getting Help

The Objectivity Developer Network provides technical information and resources, such as tutorials, documentation, FAQs, code examples, and sample applications. You can also access information about supported platforms and compilers. The Developer Network is found at:

http://support.objectivity.com

You can log onto your existing customer account from this location. Your customer account gives you access to product downloads and information about known bugs and bug fixes. Contact Customer Support if you need a login.

How to Reach Objectivity Customer Support

You can contact Customer Support by:

- **Telephone:** Call 1.408.992.7100 *or* 1.800.SOS.OBJY (1.800.767.6259) Monday through Friday between 6:00 A.M. and 6:00 P.M. Pacific Time, and ask for Customer Support.

  The toll-free 800 number can be dialed *only* within the 48 contiguous states of the United States and Canada.
- **Fax:** Send a fax to Objectivity at 1.408.992.7171.
- **Email:** Send electronic mail to *help@objectivity.com*.

Before You Call

Please be ready to submit the following information:

- Your name, company name, address, telephone number, fax number, and email address
- Detailed description of the problem
- Information about your workstation, including the type of workstation, its operating system version, and compiler or interpreter
- Information about your Objectivity products, including the version of the Objectivity/DB libraries

You can use the Objectivity/DB `oosupportinfo` tool to obtain information about your workstation and your Objectivity products.

Part 1 INTRODUCTION

1

Autonomous Partition Basics

NOTE This chapter applies only to *non-placement-managed* federated databases.

Objectivity/DB High Availability (Objectivity/HA) is a separately licensed product that improves the availability of distributed data in an Objectivity/DB federated database. With Objectivity/HA, you can divide a federated database into independent units, called *autonomous partitions*, which improve fault tolerance by limiting the impact of host, network, or Objectivity-server failures. When a failure occurs in one partition, users can continue to access data managed by the others.

Objectivity/HA also serves as the basis for replicating databases; see Chapter 2, "Data Replication Basics."

This chapter describes:

- General information about autonomous partitions and the organization of a a partitioned federated database.
- The Objectivity/DB system resources of a partitioned federated database.
- Access to individual databases in a partitioned federated database.
- How autonomous partitions can improve fault tolerance and application performance.
- Access to global information in a partitioned federated database.
- Support for restricted data in a partitioned federated database.

NOTE You must install AMS (Advanced Multithreaded Server), available with Objectivity/DB, on every host machine where autonomous partitions are to be located. For information on installing AMS, see the installation and configuration documentation on the Objectivity Developer Network. For information on using AMS, see Chapter 9, "Advanced Multithreaded Server," of *Objectivity/DB Administration*.

Autonomous Partitions

Autonomous partitions are a mechanism for limiting the effect of host, network, or server failures on an Objectivity/DB federated database. Collectively, partitions impose a level of administrative organization on distributed data.

Every autonomous partition:

- *Controls* a group of databases in the federated database.
- Has its own set of *Objectivity/DB system resources* for managing access to the controlled databases.

With autonomous partitions, you can organize databases into independently managed groups, so that when a failure occurs in one partition, users can continue to access the databases that are controlled by the remaining partitions. Autonomous partitions also improve application performance, allow you to restrict access to data that is for private use, and serve as the foundation for data replication.

Because autonomous partitions superimpose a separate administrative structure on a federated database, they do not affect the logical relationships among objects. Consequently, you can set up and manage partitions solely with Objectivity/DB and Objectivity/HA tools, without having to modify your database applications. If you choose, you can give your applications greater control over their use of partitions; see the documentation for your Objectivity programming interface. You can also use the programming interface to create your own tools for managing partitions.

Partitioned Federated Databases

A *partitioned federated database* is a federated database that has two or more autonomous partitions. At creation, a federated database is *unpartitioned*—that is, the federated database itself serves as a single implicit autonomous partition that controls all of its databases. The first time you create an autonomous partition explicitly, Objectivity/DB adds a *second* partition to the federated database. Although the initial (original) partition is automatically given the system name of the federated database, this partition has no special status among the subsequently created partitions. A federated database can have as many or as few partitions as are useful to you.

Databases Controlled by a Partition

An autonomous partition can control any arbitrary group of databases in a federated database, regardless of their physical location or usage. However, a partition is most effective when it controls a group of databases that are:

- Shared by the same users
- Meaningfully accessed independently (for example, because objects in these databases have comparatively few references to objects in databases controlled by other partitions)
- Located near each other and near the applications that access them (for example, on the same host or distributed among several hosts in the same portion of a local-area network)

Autonomous partitions divide the set of databases into nonoverlapping subsets, so that every database in the federation is controlled by exactly one partition. When a new partition is created, it has no databases under its control; you assign databases to a new or existing partition by:

- Creating new databases directly in the partition.
- Transferring control of existing databases to the partition.

Similarly, you can remove a database from a partition by deleting the database from the federation or by transferring the database to another partition. Partitioning is flexible: you can create empty partitions first and then populate them with databases, or you can create databases first and then organize them into partitions.

NOTE When a partition controls a database, it implicitly controls all of the containers in the database, including any external containers stored in their own container files. A container and its database cannot be controlled by different partitions. A controlled database cannot be replicated if it has any external containers.

Control and Containment

Strictly speaking, the relationship between a partition and a database is *control*, although, for convenience, partitions are sometimes said to “contain” the databases “in” them. Containment in this context is different from containment in the *logical storage hierarchy,* which describes the logical relationships between a federated database and its databases, a database and its containers, and a container and its basic objects. Whereas logical containment is the basis for object identifiers, containment in partitions refers to administrative control.

Objectivity/DB System Resources

Applications interact with various *Objectivity/DB system resources* that manage access to databases in a federated database. These resources enable applications to open a connection to the federated database, find individual database files, obtain locks on objects, read representations of the objects into memory, and, if necessary, recover any incomplete update transactions by rolling back changes. Partitioning a federated database adds a set of Objectivity/DB system resources for each partition.

Resources for an Unpartitioned Federated Database

An unpartitioned federated database has a single set of Objectivity/DB system resources for managing access to its databases—namely:

- A boot file, which identifies the Objectivity/DB system resources to be used by an application while opening a connection to the federated database.
- A system-database file containing the *system database*, which stores:
  - The *global catalog* of all databases in the federation and their locations.
  - The *schema* describing the types of the objects that can be read from or written to the federated database.
  - Any indexes and scope names defined for the entire federated database.
- A lock server, which manages concurrent access from multiple transactions.
- A journal directory, in which applications can write journal files to record recovery information for rolling back incomplete transactions.

Figure 1-1 shows the Objectivity/DB system resources for the unpartitioned federated database `MyFD`. After an application uses the boot file to open a connection to `MyFD`, it consults the global catalog in the system database to locate databases `DB2` through `DB6`. The application uses `MyFD`'s lock server to obtain locks on objects in these databases, and uses `MyFD`'s journal directory to record recovery information for these objects.

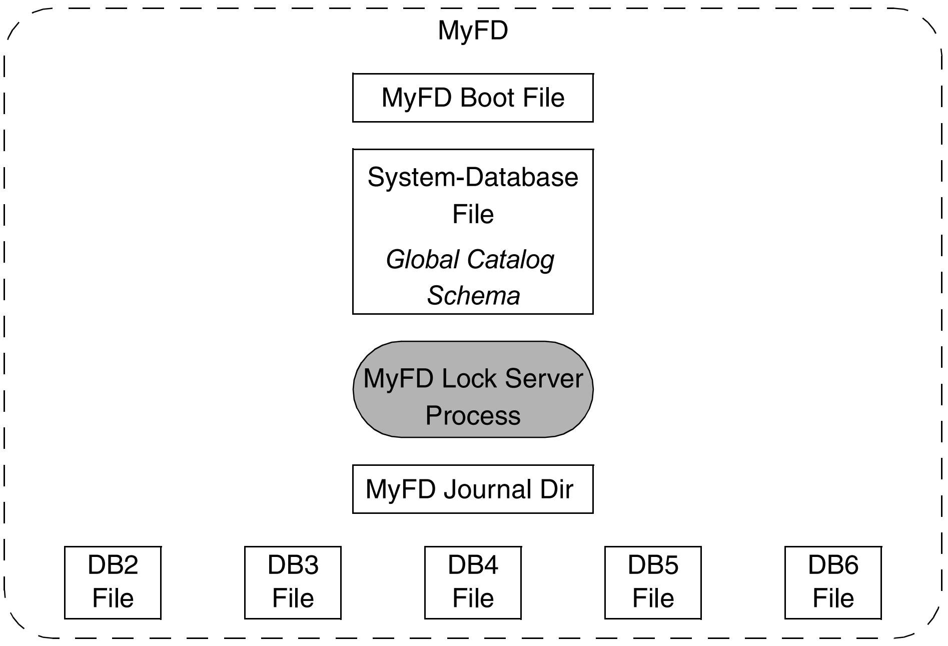

Figure 1-1 Files and Processes of an Unpartitioned Federated Database MyFD

Resources for a Partitioned Federated Database

Partitioning a federated database augments the original set of Objectivity/DB system resources by adding a separate set of system resources for every added autonomous partition. The Objectivity/DB system resources for each partition include:

- A boot file that identifies the Objectivity/DB system resources associated with the partition.
- A copy of the system-database file (specifically, a replicated *image* of the system database; see "Database Images" on page 31). In a partitioned federated database, the global catalog identifies the controlling partition of every database in the federation.
- A lock server that manages concurrent access to the data controlled by the partition. Different partitions are normally serviced by different lock servers.
- A journal directory for writing recovery information that pertains to the data controlled by the partition.
- AMS running on the data-server host on which the system-database file resides. (AMS is required for accessing images of replicated databases.)

Figure 1-2 shows the Objectivity/DB system resources for the federated database `MyFD`, in which two autonomous partitions (`AP2` and `AP3`) have been created explicitly. Control of databases DB3 and DB4 has been transferred from MyFD to AP2; control of databases DB5 and DB6 has been transferred from MyFD to AP3. A surrounding dashed line indicates the files and processes for each partition.

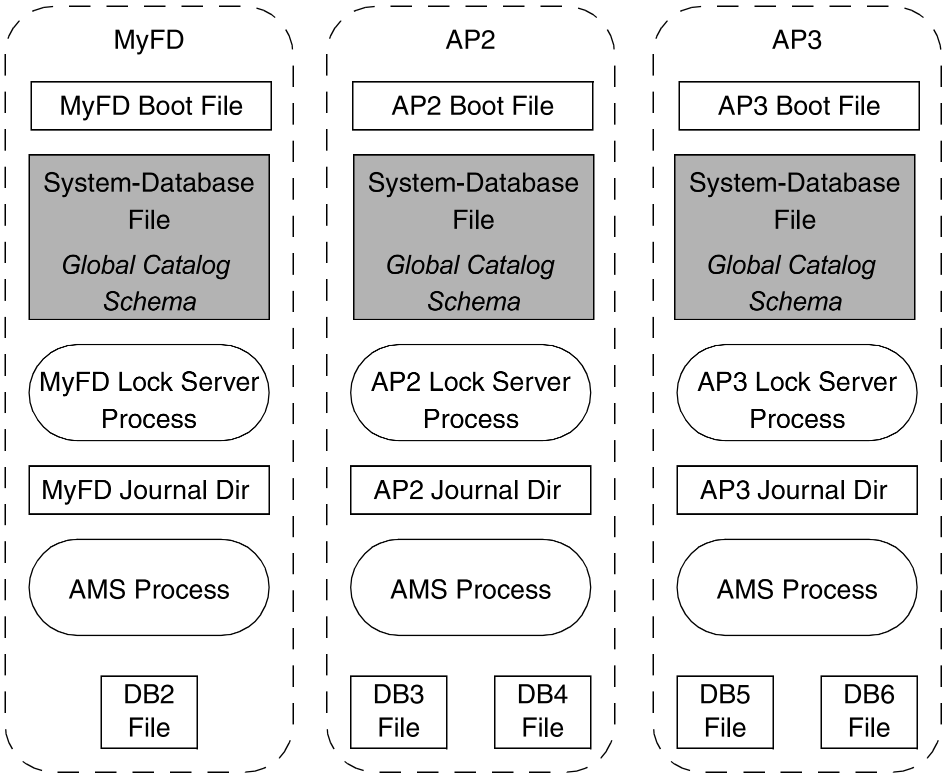

Figure 1-2 Files and Processes in a Federated Database With Three Partitions

After using any boot file to open a connection to MyFD, an application can find any database and its controlling partition by reading the global catalog in an image of the system database; see "Boot Autonomous Partition" on page 19. When accessing objects in DB2, the application requests locks from MyFD's lock server and records recovery information in MyFD's journal directory. Similarly, when accessing objects in DB3, the application contacts AP2's lock server and writes to AP2's journal directory; when accessing objects in DB5, the application contacts AP3's lock server and writes to AP3's journal directory.

As Figure 1-2 illustrates, the initial partition has the same system name (MyFD) as the federated database and has the original set of Objectivity/DB system resources. However, the initial partition itself is not the entire federated database. Rather, a partitioned federated database is a logical entity that is manifested as a set of autonomous partitions sharing the same replicated system database.

Boot Autonomous Partition

A partitioned federated database has multiple boot files associated with it (one boot file per autonomous partition). An application or tool can use any of these boot files to open a connection to a partitioned federated database. For best performance, an application or tool normally uses the boot file of the partition whose system resources are closest to the application or tool on the network. The partition whose boot file is used by an application or tool is called the *boot autonomous partition* (or *boot partition*). Most administration tools require that you identify the boot partition by providing a `bootFilePath` argument (the path to its boot file).

When the application starts, it uses the specified boot file to find the boot partition's system-database file and lock server, so that the application can read schema and catalog information during initialization. The choice of boot partition does not restrict data access in any way, unless the boot partition has been marked offline; see "Support for Restricted Access" on page 27. When schema or catalog information is required during subsequent operations, the application normally obtains this information from the boot partition's system-database file, although if that file becomes inaccessible for some reason, Objectivity/DB automatically substitutes the system-database file from another partition.

An application can use a different boot partition each time it starts. Furthermore, multiple applications that have each booted from a different partition can access the federated database concurrently.

Location of Files and Processes

Figure 1-1 and Figure 1-2 show the various files and processes that comprise a federated database, but do not indicate where these files and processes might be located. In general, Objectivity/DB allows you to *distribute* a federated database's files and processes among multiple hosts on the same network, as described in *Objectivity/DB Administration*. Appropriate distribution can improve performance, usually by shortening the network path between an application and the files or processes it uses, or by reducing the number of requests to be handled by any single network path, processor, or storage device.

The unpartitioned federated database `MyFD` in Figure 1-1 could be configured in various ways—for example:

- The lock server, journal directory, and all files could be on the same host.
- The lock server, journal directory, system-database file, boot file, and two of the database files could all be on one host, with the remaining three database files on each of three other hosts.
- The lock server, journal directory, and each file could each be on a separate host.

A partitioned federated database can be distributed in similar configurations, although normally the files and processes for each partition are most usefully hosted separately from the files and processes of the other partitions. Thus, in Figure 1-2, each dashed line indicating a partition could correspond to a separate host or group of hosts. The only requirement is that AMS must run on the data-server host for each system-database file.

Access to Databases

An application performs the following operations whenever it accesses a database in a partitioned federated database:

- Reads the global catalog to find the relevant database file and identify the autonomous partition that controls it.
- Requests the necessary locks from the controlling partition's lock server.
- Reads or updates the database file as appropriate.
- Records any recovery information in the controlling partition's journal directory.

To perform these operations, an application must be able to access not only the database file itself, but also various Objectivity/DB system resources, the host(s) on which they reside, and the network link(s) that connect them to the application.

Specifically, a database is *accessible* to an application only if the application can:

- Read or write to the desired database file itself. The file's data-server host must be operational and connected to the client host by an operational network link. (If the database is replicated, then AMS must be running on the data-server host, as well.)
- Connect to the controlling partition's lock server. The lock server must be running on an operational lock-server host, and this host must be connected to the client host by an operational network link.
- Write to the controlling partition's journal directory. The journal-directory host must be operational and connected to the client host by an operational network link.
- Read the system-database file in at least one partition. The file's data-server host must be operational and connected to the application's client host by an operational network link. AMS must be running on the data-server host.

  The system-database file may, but need not, belong to the partition that controls the database to be accessed. To read the system-database file in a partition, the application must be able to access that partition's lock server and journal directory, and the relevant hosts and network links must be operational.

The account running AMS must have read and write access permissions to the system-database file and to any database file or journal file accessed through AMS. The account running the application must have read and write permissions to any local database file or journal file accessed directly by the application.

Improved Fault Tolerance and Performance

Autonomous partitions improve an application's ability to access data in a federated database by *decentralizing* the Objectivity/DB system resources that make databases accessible. A partitioned federated database decentralizes its system resources by:

- Splitting concurrency management among multiple lock servers.
- Splitting transaction journaling among multiple journal directories.
- Providing multiple redundant images of the system-database file.

In contrast, the system resources of an unpartitioned federated database are centralized (even if they are physically distributed on different hosts). Without partitions, a *single* lock server manages concurrent access to *all* databases; a *single* journal directory contains recovery information for *all* databases; and a *single* system-database file is consulted by *all* applications.

The decentralization of Objectivity/DB system resources improves a partitioned federated database's fault tolerance and potentially improves the performance of client applications.

Improved Fault Tolerance

Partitions improve the fault tolerance of a federated database by maximizing the number of databases that remain accessible to applications after a host, network, or Objectivity-server failure occurs. The impact of such a failure is either eliminated or limited to the partition in which the failure occurs. Without partitions, a similar failure can prevent access to the entire federated database.

For example, assume that an application boots successfully from a particular partition, and that the application uses the boot partition's system-database file in subsequent operations. If a network failure then isolates the host of this system-database file, the application can continue without interruption, transparently using the system-database file from another partition. All user databases in the federation remain accessible (except any on the isolated host), and new applications can be started from different boot partitions. In contrast, no databases in an unpartitioned federated database can be opened after a network failure isolates the only system-database file.

Similarly, assume that every partition in a partitioned federated database is serviced by a different lock server. The failure of a particular partition's lock-server host prevents applications from accessing the databases controlled by that partition. Applications can continue to access the databases in all of the other partitions whose lock servers remain operational. In contrast, no databases in an unpartitioned federated database can be updated or opened after a host failure stops the only lock server.

NOTE A failure that prevents access to an individual database file has the same impact whether or not the federated database is partitioned—objects in that database cannot be read or updated until the database file can be accessed again. You can improve the availability of a database by replicating it; see Chapter 2, "Data Replication Basics."

Improved Performance

Autonomous partitions can improve throughput and runtime speed when multiple applications access the same federated database, especially if these applications run on geographically distant hosts on the same wide-area network. That is, by decentralizing Objectivity/DB system resources, partitions enable you to:

- Reduce resource contention. For example, without partitions, all lock requests are handled by the same lock server, possibly creating a bottleneck. Appropriating partitioning allows you to eliminate such bottlenecks by designating different lock servers to handle the lock requests for objects in different databases.
- Reduce network traffic. For example, without partitions, many applications may be located at a distance from the federation's only lock server and system-database file. Appropriate partitioning allows you to provide lock servers and system-database files that are local to widely distributed applications, reducing the network path between each such application and the resources it uses most frequently.

Example

Consider an unpartitioned federated database that contains employee information for a chain of warehouse stores with three regional offices—in San Francisco, Chicago, and New York. As shown in Figure 1-3, the employee information for each region is distributed among several database files on one or more data-server hosts located in that regional office.

Users at each regional office run applications that frequently access their local database files (across a local-area network) and occasionally access the database files of the other regions (across a wide-area network). Thus, an application running on a client host in the San Francisco regional office uses a local network to access the databases `SFEmp1` and `SFEmp2`, but must occasionally go over a wide-area network to access the databases `NYEmp1` and `NYEmp2`.

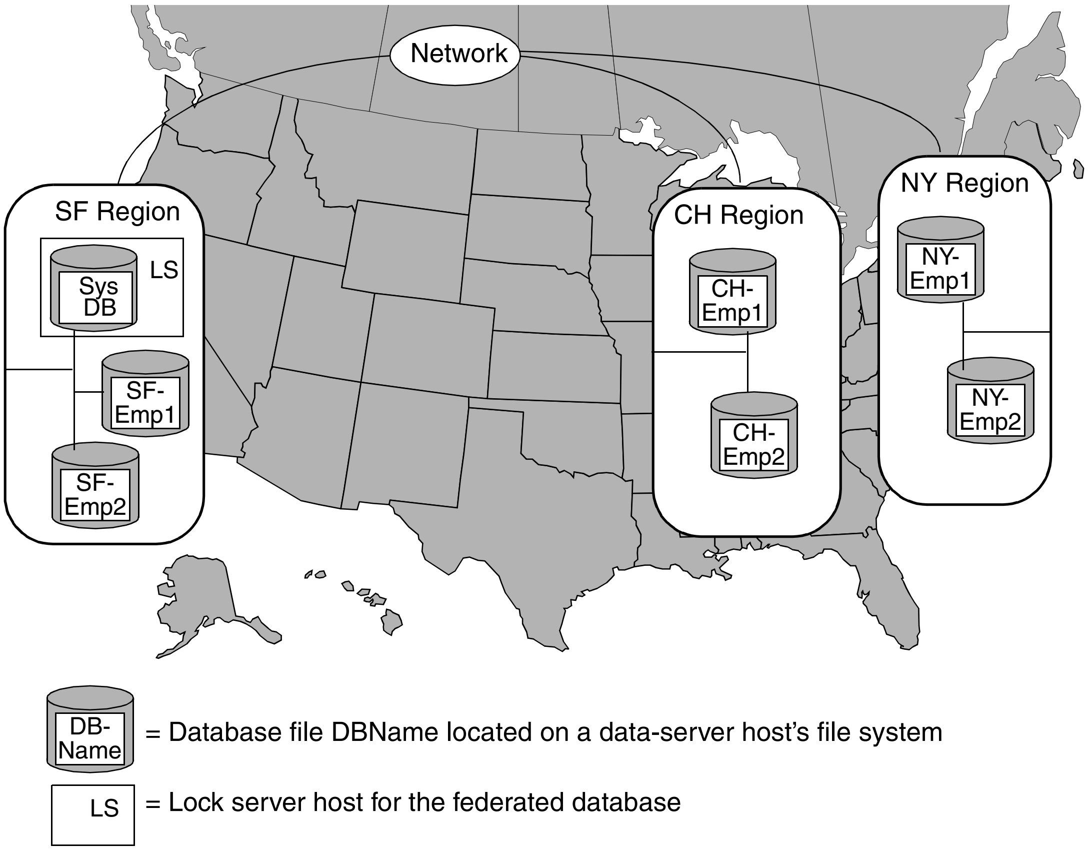
Network
SF Region
LS
Sys DB
SF-Emp1
SF-Emp2
CH Region
CH-Emp1
CH-Emp2
NY Region
NY-Emp1
NY-Emp2
DB-Name
= Database file DBName located on a data-server host's file system
LS
= Lock server host for the federated database

Figure 1-3 Widely Distributed Databases in an Unpartitioned Federated Database

Distributing the database files allows each database to be placed near the application that accesses it most frequently, as well as providing a degree of fault tolerance—for example, if the host containing `SFEmp1` fails, users in the Chicago office can still access `CHEmp1` and `CHEmp2`.

However, because this federated database is unpartitioned, its system resources are centralized—that is, it has a single lock server, system-database file, and journal directory located on a host in the San Francisco office. Centralization reduces the performance of the remote applications, which must communicate over the wide-area network to obtain locks or look up databases. Furthermore, a host or network failure in the San Franciso office could isolate the lock server, preventing users in Chicago and New York from committing transactions in their local databases.

To improve fault tolerance and performance, the database system administrator decides to partition the federated database, assigning the databases for each regional office to a different autonomous partition, as shown in Figure 1-4 on page 25.

Each regional office has its own a local lock server, system-database file, and journal directory (not shown). The lock server and system-database file for a partition may, but need not be, placed on the same host.

The applications in each regional office can now:

- Use the local autonomous partition as the boot partition, normally consulting the local system-database file to find databases during subsequent operations. For example, an application in the Chicago regional office boots from the Chicago region’s partition and uses that partition’s system-database file when locating databases. If, however, the host for this local system-database file were to fail, the application would automatically use the system-database file in San Francisco or New York.
- Connect to a local lock server when accessing local databases and to a remote lock server when data from a different region is needed. For example, a Chicago application requests locks from the Chicago lock server when accessing `CHEmp1` or `CHEmp2`, and requests locks from the San Francisco lock server when accessing `SFEmp1` or `SFEmp2`.

Work can continue in each regional office despite host or network failures occur in the other offices. For example, users in Chicago and New York can keep accessing local data even if a network failure isolates the San Francisco office.

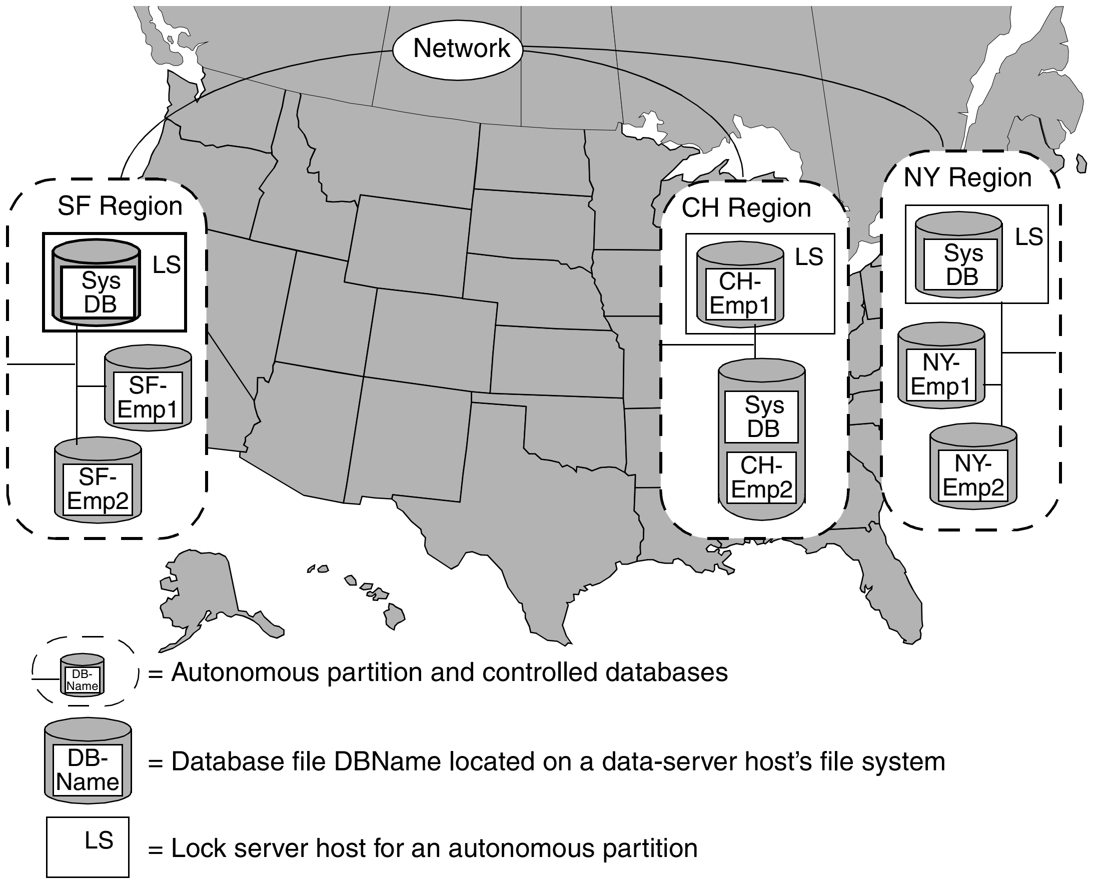
Network
SF Region
LS
Sys DB
SF-Emp1
SF-Emp2
CH Region
LS
CH-Emp1
Sys DB
CH-Emp2
NY Region
LS
Sys DB
NY-Emp1
NY-Emp2
DB-Name
= Autonomous partition and controlled databases
DB-Name
= Database file DBName located on a data-server host's file system
LS
= Lock server host for an autonomous partition

Figure 1-4 Widely Distributed Databases in a Partitioned Federated Database

Usage Guidelines

The design you choose for a partitioned federated database depends greatly on your network configuration, the object model, and the distribution of your users. Here are some general recommendations for using autonomous partitions to maximize access to data in a federated database:

- Use a different lock server for each partition.
- Use a different data-server host for the system-database file of each partition.
- Assign databases to partitions as suggested in "Databases Controlled by a Partition" on page 15 to support separate groups of users working independently.
- Place the processes and files of a boot partition close to the applications that boot from it; assign any databases to an application's boot partition if those databases contain objects that are frequently accessed by the application.

Access to Global Federated-Database Information

Every autonomous partition has a replicated copy of the system database, which stores general federated-database information such as the global catalog and schema. All tools and applications perform operations that read the system database—for example, to find a database file and its controlling partition. To read the system database, a tool or application must be able to access the system-database file of (at least) one partition, along with that partition's lock server and journal directory, and AMS on the file's data-server host. Put another way, a tool or application that needs to read federated-database information must be able to access the Objectivity/DB system resources for a single partition. (This partition is normally the boot partition, although if the boot partition becomes inaccessible after the tool or application has started, a different partition is automatically substituted.)

Some tools and applications perform operations that modify the system database. Such operations include creating or deleting an autonomous partition, creating or deleting a database, or performing schema evolution. Because the information in the system database is global, changes to this information must be made in the system-database files of all partitions. For a single operation to update all system-database files, however, a tool or application would need simultaneous access to all Objectivity/DB system resources for all partitions.

To improve fault tolerance, Objectivity/HA allows the system database to be updated even when some Objectivity/DB system resources cannot be accessed. More specifically, a tool or application that needs to update the system database must be able to access the Objectivity/DB system resources for a *quorum* of partitions. At a later time, Objectivity/HA automatically propagates the updates to the partitions that were not in the quorum.

Quorum of Partitions

In the simplest case, a quorum of autonomous partitions is a majority of the partitions in the federated database. You can assign a positive integer *weight* to one or more individual partitions to increase their importance for purposes of modifying the system database. A quorum of partitions thus consists of any set of partitions whose combined weight is more than half of the sum of all partition weights.

By default, the weight of each partition is 1, so all partitions are equal. You typically assign weights when a federated database has an even number of partitions, so a quorum can still be achieved even when the system resources of only half the partitions can be accessed.

Objectivity/DB automatically calculates whether a quorum of partitions is accessible whenever a tool or application attempts to read or write to the system database. To ensure consistency, a write operation is permitted only if a quorum can be accessed. A read operation is permitted even if a quorum of partitions cannot be accessed, although the federated-database information could be out-of-date. A quorum calculation may cause a previously inaccessible partition to be added back into the quorum; when this happens, the partition's system-database file is automatically brought up-to-date.

The boot partition may, but need not, be a member of the quorum. Because a tool or application must be able to access its boot partition at startup, the boot partition is initially a member of the quorum. However, if the boot partition later becomes inaccessible, a subsequent quorum calculation may find that the remaining accessible partitions constitute a quorum.

The quorum mechanism is a consequence of the replication of the system database. For details about quorum management, see "A Closer Look at Quorums" on page 35.

Support for Restricted Access

You can prevent tools and applications from accessing an autonomous partition, even if it is physically accessible. For example, you may want to prevent access to partitions that are used to archive containers or databases. You also may want to create a partition for private use by a particular workgroup or individual.

A partition's *offline status* controls whether it is available for access. To make an autonomous partition unavailable, you can configure it as *offline*. You can later configure it as *online* if you need to access it.

By default, applications enforce the offline status of autonomous partitions. This means that an application can access data in an offline autonomous partition only if that partition is the boot autonomous partition; otherwise, an error occurs. Furthermore, when an application's boot autonomous partition is offline, the application cannot access resources outside that partition.

Within an application, you can specify that a transaction can ignore the offline status of all autonomous partitions. Attempts to access resources in an offline autonomous partition will then succeed regardless of the boot file used by the application, provided the autonomous partition is physically accessible. Similarly, transactions can access resources outside the offline autonomous partition even if that partition is the boot autonomous partition.

2

Data Replication Basics

NOTE This chapter applies only to *non-placement-managed* federated databases.

Objectivity/DB High Availability (Objectivity/HA) improves the availability of particular databases by allowing you to *replicate* them—that is, to create and distribute managed copies (called *images*) among multiple autonomous partitions. When a failure or disconnect prevents access to one or more images of a database, users can still continue to access the data, provided that enough other images are accessible.

Data replication may be performed only in a partitioned federated database. You should read Chapter 1, "Autonomous Partition Basics" carefully before you read this chapter.

This chapter describes:

- General information about data replication.
- Images of a replicated database.
- Access to data in replicated databases.
- The quorum mechanism that coordinates data access across multiple weighted images.
- Examples of strategies for configuring weighted images.
- Techniques for establishing a quorum in a hot-failover configuration.
- How images are synchronized (brought up-to-date) after being restored to service.
- Further information about a quorum of partitions.

NOTE You must install AMS (Advanced Multithreaded Server), available with Objectivity/DB, on every host machine where database images are to be located. For information on installing AMS, see the installation and configuration documentation on the Objectivity Developer Network. For information on using

AMS, see Chapter 9, "Advanced Multithreaded Server," of *Objectivity/DB Administration*.

Data Replication

Data replication is a mechanism for improving the availability of individual databases in a partitioned federated database. With data replication, you can create and distribute managed copies of databases (called *images*), where each image of a given database is controlled by a different autonomous partition. The data in a replicated database can be read or updated by any application that can access a representative subset (or *quorum*) of the database's images.

Data replication is a natural extension of partitioning. Just as multiple autonomous partitions decentralize the Objectivity/DB system resources of a federated database, multiple images decentralize the data of individual databases. Data replication can thus:

- Improve fault tolerance. When network, host, or lock-server failures isolate some images of a database, an application can use the other images to continue accessing the data (provided the remaining images constitute a quorum).
- Improve performance. Appropriate partitioning and replication allow you to place images of the same data near widely distributed applications, thereby reducing the network path between each such application and the data it reads most frequently.
- Support planned disconnects. You can place images of the same data on different laptop computers, which are disconnected from and reconnected to the network on a regular basis. While a particular laptop is disconnected, a network application can continue accessing quorum images on the connected machines; when the laptop is reconnected, its images can be restored into service.

NOTE A federated database must be partitioned before you can replicate data in it.

Data replication does not affect the logical relationships among objects, so you can set up and manage replicated databases entirely through Objectivity/DB and Objectivity/HA tools (without having to modify your database applications). If you choose, you can give your applications greater control over their access to replicated data; see the documentation for your Objectivity programming interface. You can also use the Objectivity/HA programming interface to create your own tools for managing database images.

Database Images

When you replicate a database, it is physically represented as two or more *images*, where each image is a separate file containing a complete copy of all the objects in a database. The images of a replicated database are *managed* copies, in that Objectivity/DB automatically coordinates all updates to the database to keep the images *synchronized* with each other.

When you first create a database, it is *unreplicated*—that is, the database is physically represented as a single file, which is its only image. If you decide to replicate the database, you can do so at any time simply by explicitly creating a second image; see "Creating a Database Image" on page 68. The initial image has no special status among the subsequently created images.

NOTE A database with external containers cannot be replicated.

Images and Autonomous Partitions

Every image of a particular database is controlled by exactly one autonomous partition; furthermore, each such image must be controlled by a different autonomous partition. Figure 2-1 shows the replicated database `DB2` with an image in each of three autonomous partitions. (The files and processes of the partition's Objectivity/DB system resources are not shown in this figure.)

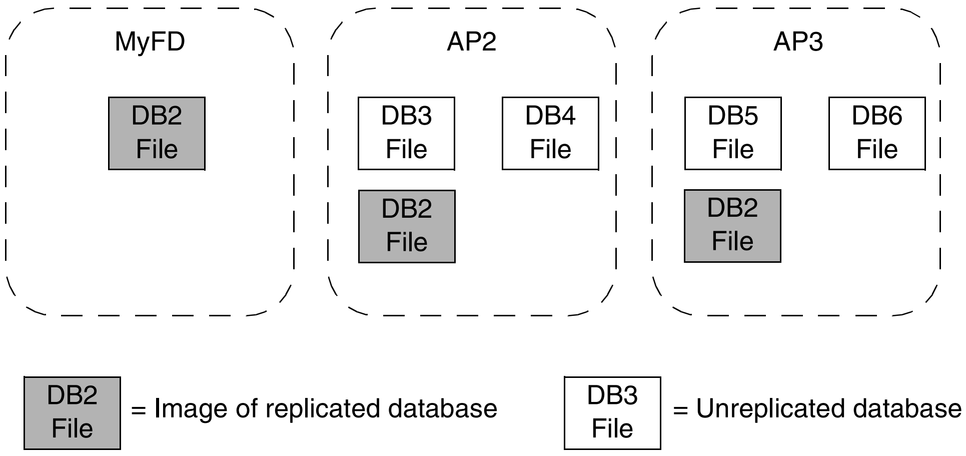

Figure 2-1 Database Images in Multiple Autonomous Partitions

A database can have as many or as few images as are useful to you. The maximum number of images per database is limited by the number of partitions in the federated database. If all users need fast, frequent access to critical data in a particular database, that database should have an image in every partition. But if the users of a particular partition require only occasional or noncritical access to a database, that partition need not control its own image of the database. For example, in Figure 2-1, you could replicate database `DB3` by creating an image of it in partition `MyFD` (but not in `AP3`).

As indicated in Figure 2-1, an autonomous partition controls a group of images, where some are images of replicated databases and others are the sole images of unreplicated databases. You can create a new image in a partition or move an existing image from one partition to another, provided the destination partition does not already control an image of the same database. You can also delete an image from any partition. Deleting an entire partition automatically deletes the images it controls, unless one of these images is the last (or only) image of a database.

Location of Images

Like unreplicated database files, the individual images of a replicated database can be located anywhere in a network, and you can move an image to a new location at any time. The images of a database can be placed on the same host or distributed each to a different host, as long as every image is controlled by a different autonomous partition.

Typically, the images controlled by a particular partition are placed:

- Close to each other, to the partition's Objectivity/DB system resources, and to the relevant client applications.
- Separately from the files and processes of the other partitions.

Thus, in Figure 2-1, each dashed line indicating a partition typically corresponds to a separate host or group of hosts. The only requirement is that AMS must run on every data-server host that has an image file.

Logical and Physical Model

Data replication affects the physical representation of databases, but not the logical containment hierarchy of a federated database or the logical object model of an application. Even when replicated, a database is a single logical entity with a unique database identifier and system name. A database's identifier and system name belong to the logical database, and not to any particular image, so the object identifiers (OIDs) for objects in a database are the same across all of the database's images. Consequently, a database application does not reference any individual image to find data in a database.

Of course, tools and applications that perform administrative operations such as moving or deleting an image must be able to identify the individual affected image. In such cases, the desired image is designated by identifying both the relevant database and the controlling autonomous partition.

Access to Replicated Databases

Accessing the data in a replicated database involves the same general operations as accessing the data in an unreplicated database—the accessing application must be able to read the global catalog, request the relevant locks, find the data on disk, read or update the data, and record any recovery information.

For an unreplicated database, such operations depend on the accessibility of a single image (the database itself); see "Access to Databases" on page 20. For a replicated database, however, such operations depend on the accessibility of multiple images, where each image must meet the conditions listed in "Access to an Individual Image" below. A *quorum* mechanism determines whether an application can access enough individual images of a database to be allowed to read or update the database.

Access to an Individual Image

An individual image of a replicated database is considered *accessible* to an application if the application can:

- Open the image file itself. This file's data-server host must be operational and connected to the client host by an operational network link. AMS must be running on the data-server host.
- Connect to the controlling partition's lock server. The lock server must be running on an operational lock-server host, and this host must be connected to the client host by an operational network link.
- Write to the controlling partition's journal directory. The journal-directory host must be operational and connected to the client host by an operational network link.
- Read the system-database file in at least one partition. This file's data-server host must be operational and connected to the application's client host by an operational network link. AMS must be running on the data-server host.

  The system-database file may, but need not, belong to the partition that controls the database image to be accessed. To read the system-database file in a partition, the application must be able to access that partition's lock server and journal directory, and the relevant hosts and network links must be operational.

The account running AMS must have read and write access permissions to the image file, to the system-database file, and to any journal file accessed through AMS. The account running the application must have read and write permissions to any local journal file accessed directly by the application.

Access to a Quorum of Images

An application can access the data in a replicated database only if a representative subset, or *quorum*, of the database's images are accessible. Under ideal conditions (where all network links, hosts, and Objectivity servers are operational), the quorum for a database includes the entire set of its images. When a network disconnect or a host or Objectivity-server failure prevents access to one or more images, an application automatically determines whether the remaining subset of images constitutes a quorum (see "Automatic Quorum Calculation" on page 36). By default, a quorum is a simple majority of images.

When an application can access a quorum of images for a database, the database is *available* for read and update operations. During such operations, the application accesses the database as follows:

- *Read operation*. The application obtains read locks in each image in the quorum, but actually reads storage pages from only one image, called the *read image*.

  By default, Objectivity/DB chooses the read image from among the quorum images, giving preference to the image that is closest on the network to the application. If an application has knowledge of which images it can access most efficiently, it can set the read image explicitly; see the documentation for your Objectivity programming interface.
- *Update operation*. The application obtains update locks in each image in the quorum, reads storage pages from just the read image, and writes any changes to each image file in parallel. Recovery information is recorded in the journal files of the controlling partitions. At a later time, updates are propagated to any images that were not in the quorum; see "Restoring Images Into Service" on page 45.

NOTE Because the individual images of a database are controlled by different autonomous partitions, locks on the database's objects are granted in parallel by multiple lock servers working in conjunction with each other.

A quorum of images is required for every update operation, but is required only by default for a read operation. If an application can tolerate potentially stale data, it may choose to allow *nonquorum reading* while a quorum of images is not accessible. During a nonquorum read, the application obtains locks in (and reads data from) a single, accessible image. For information about nonquorum reading, see the documentation for your Objectivity programming interface.

A Closer Look at Quorums

Objectivity/HA uses its quorum mechanism to guarantee the integrity of replicated data. The quorum mechanism ensures that updates to a replicated database are made consistently across its images, so the current state of the data exists in a well-defined set of images. An application determines whether it can access a quorum of images for a database by performing a *quorum calculation*, which is based on the *weight* of each image.

Image Weight

Every image has a *weight* that determines its relative importance for purposes of constituting a quorum. The greater an image's weight (relative to the weights of the other images of the same database), the more necessary it is for that image to be accessible to applications seeking to update the database. An image's weight is an integer with a positive value (1 or greater).

You can assign a weight to an image when you create it; you can change the weight of an existing image at any time. If you do not explicitly assign a weight to an image, the default weight is 1. See "Creating a Database Image" on page 68 and "Changing Database-Image Properties" on page 69.

Quorum Calculation

An application performs an operation called *quorum calculation* to determine whether it can access a quorum of images for a database. During the quorum calculation for a particular database, the application:

1. Tests whether each image of the database is accessible by trying to connect to the controlling partition's lock server and to AMS on the image's data-server host.
2. Adds the weights of the accessible images.
3. Compares the combined weight of the accessible images to the total weight of all of the database's images.

The accessible images constitute a quorum if their combined weight is greater than half of the total image weight for the database. For example, if all images of a database have the default weight (1), a simple majority of these images would need to be accessible to constitute a quorum for the database. For more examples, see "Strategies for Assigning Weights to Images" on page 38.

Special cases of quorum calculation exist for determining a quorum in particular configurations; see "Hot-Failover Configurations" on page 42.

List of Quorum Images

When an application performs a quorum calculation for a given database and finds that the accessible images constitute a quorum, the list of particular images in this quorum is recorded by every lock server that services a partition of the federated database. Each image in the recorded list is one of the database's *quorum images*, and any images not in the recorded list are *nonquorum images*. Once a quorum is recorded, subsequent operations that open the database or commit updates to it continue to use the same set of quorum images, until the next quorum calculation occurs; see "Automatic Quorum Calculation" below.

If the database is accessed by multiple applications, all of the applications use the same set of quorum images, regardless of which application actually calculated the quorum. A database's quorum images thus represent the database's current state, because each quorum image contains the latest updates made by any accessing application.

Data in nonquorum images may become stale because no updates can be made in them. Each such image is automatically brought up-to-date, or *synchronized*, with the quorum images whenever it is added back into the quorum; see "Restoring Images Into Service" on page 45.

Automatic Quorum Calculation

Automatic quorum calculation occurs whenever necessary to maintain a valid quorum. A recorded quorum is considered valid as long as each application can successfully:

- Lock every quorum image when the database is opened for read or update.
- Write to every quorum image when update transactions commit.

Whenever an application detects that it cannot access (lock or write to) one of the quorum images, the application automatically performs a quorum calculation to update the quorum.

During the quorum calculation, *all* of the database's images are tested for accessibility. If the application finds that it can access enough images to constitute a quorum, the recorded quorum is updated as follows, and the application proceeds with the desired operation using the updated quorum:

- Images that are still accessible are kept in the quorum.
- Images that are no longer accessible are dropped from the quorum.

- Accessible images that were inaccessible during the last quorum calculation are restored into service (added back into the quorum and synchronized).
- A new read image is selected from among the current quorum images, if the previous read image is no longer accessible.

If, however, the application finds that it cannot access enough images to constitute a quorum, the desired operation fails. The recorded quorum is left unchanged, in case it is valid the next time the application tries to open the database. Furthermore, the recorded quorum may still be valid for some other application running elsewhere in the network.

A number of operations trigger automatic quorum calculation besides an application's failure to access a quorum image. For example, quorum calculation for a database is performed whenever an application or tool makes a catalog change involving an image of that database—that is, whenever the weight of an image is changed, a new image is created, or an existing image is deleted. Applications also check intermittently for previously inaccessible images that have become accessible again, and perform automatic quorum calculation to add these images back into the quorum; see "Restoring Images Into Service" on page 45.

NOTE If you need to control the timing of quorum calculation for a database, you can request a quorum update explicitly. Explicit quorum calculation is typically used for restoring images into service (see page 45).

If a Quorum is No Longer Possible

Sometimes network links or host machines fail permanently, making it impossible for any application to access enough images to constitute a quorum for a database. If this happens, you can purge the obsolete autonomous partitions from the federated database, so that the images in the remaining partitions can constitute a quorum. See "Purging Autonomous Partitions" on page 63.

How a Quorum Ensures Data Integrity

The quorum mechanism safeguards data integrity by ensuring that all updates are made to a well-defined set of images, particularly when a network disconnect or failure occurs. For example, suppose a network failure separates the images of a database into two groups, with client applications still running on either side of the network break. The quorum mechanism ensures that at most one group of images can be updated—the group that constitutes a quorum for the database. Applications with access to this quorum can continue to read and update the database; the other applications will have to wait until the network failure is fixed. The quorum mechanism prevents updates from being made independently to both groups, which would lead to loss of data integrity.

NOTE If a data-server host experiences a machine failure, then *no* application can access the images that reside on the failed host. In this situation, there is no danger of an inaccessible image being updated independently of other images of the same database, whether or not the accessible images constitute a quorum.

Strategies for Assigning Weights to Images

The availability of data in a database depends not only on the number and location of database images, but also on the weight you assign to each image. Following are general guidelines for assigning weights to images of a particular database:

- If the database has an odd number of images, you can assign the same weight (usually 1) to each image and expect to achieve a quorum of images in most cases of network disconnect or machine failure.
- If the database has an even number of images, you can assign a greater weight to one image so that if a network failure isolates half of the images from the other half, the set of images with the greater weight constitutes a quorum. (Alternatively, you could choose to use a tie-breaker partition; see "Tie-Breaker Partitions" on page 43.)
- If you want to guarantee the ability to update an image in a specific partition, you can assign to the image a weight that is higher than the sum of the other images of the database. This ensures that no other set of images will be able to constitute a quorum during a network failure—but if the heavily weighted image becomes inaccessible, the remaining images cannot constitute a quorum and all client applications lose update access to the database.

The following examples illustrate several strategies for assigning image weights.

Example of Evenly Weighted Images

Figure 2-2 shows a simple example of autonomous partitions and databases in three buildings on a corporate campus. Each of the three buildings (A, B, and C) has its own local-area network and several databases within its own autonomous partition. Each building has a database (`A1`, `B1`, and `C1`) for maintaining the schedule of that building's conference rooms. Any employee can schedule a conference room; due to a shortage of conference rooms on the campus, employees often schedule conference rooms in other buildings.

Because the conference room schedules are frequently read (to establish availability) but only infrequently written (to reserve a room), databases `A1`, `B1`, and `C1` are replicated in every building. When an employee in building A wants to read the schedule for any conference room, Objectivity/DB reads the schedule from a server local to that building; reading from all three databases is equally fast, because databases `B1` and `C1` are replicated in building A's partition.

When an employee in building A reserves a conference room in building B, Objectivity/DB must update all three images of the `B1` database. The system's designers decided to tolerate slightly slower writes in exchange for much faster reads, given that users do not update as often as they read.

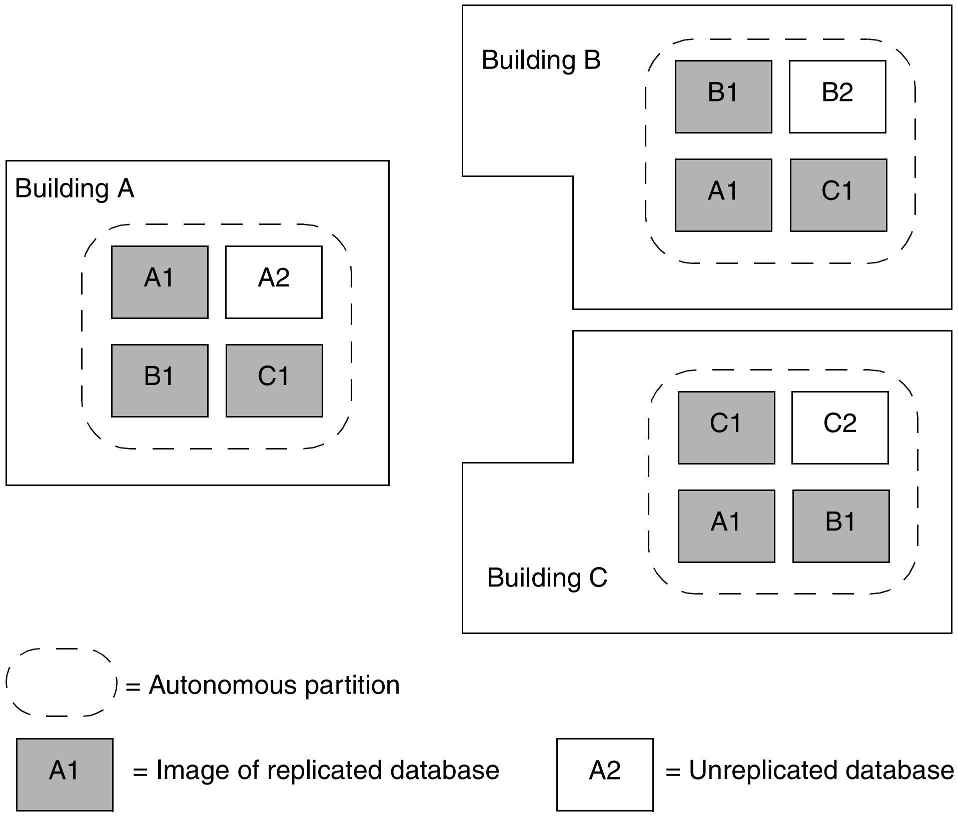

Figure 2-2 Evenly Weighted Database Images

Because each scheduling database has an odd number of images, the system designers assigned a weight of 1 to every image of these databases. Assigning the same weight to each image allows a quorum to result if any single image becomes inaccessible.

Consider what happens if building B is isolated due to a lost network connection. If a user in building A can access A’s image of `B1` and C’s image of `B1`, that user can schedule a meeting in building B. Another user in building B, able to access only B’s image, cannot schedule a meeting. By default, the user in building B cannot even read from B’s potentially out-of-date image unless the scheduling application can request the ability to read without a quorum (and therefore read potentially stale data).

After the network link from building B is restored, an administrative procedure synchronizes B’s image of the `B1` database with the quorum images in buildings A and C; see “Restoring Images Into Service” on page 45.

Example of Unevenly Weighted Images

Figure 2-3 shows a simplified example of unevenly weighted database images. This example is based on a chain of warehouse stores that requires membership of its customers. The chain of stores is divided into three regions—San Francisco, Chicago, and New York. Each region has its own local-area network connecting its regional stores, and all regions are connected by a wide-area network.

Each region has a membership database for storing customer information about the members in that region. Every store in a particular region must be able to:

- Add new members to the local region’s membership database.
- Verify customer information in the appropriate membership database whenever a purchase is made. A store normally reads the local region’s membership database because most people shop where they live, but occasionally a store must read a remote region’s membership database to verify the purchase of a member visiting from another region.

Furthermore, in the event of a network failure, every store in a region must be:

- Able to read and update the local region’s membership database (to add new members and verify unlimited purchases by local members).
- Allowed to read possibly stale information from each remote region’s membership database (to approve limited purchases by visiting members).

To meet these requirements:

- Each regional main office has its own autonomous partition.
- The membership database for each region is replicated, with an image in each partition.

- Within its "home" partition, the image of a membership database is assigned a weight of 3, while the images in the other regional partitions each have a weight of 1. This way, if a network failure isolates a regional office, the region's local image of its own database always constitutes a quorum, so the stores in the region can continue to update the local image.
- The application for verifying customer information allows nonquorum reads when purchases are made by visiting members during a network failure.

Consider what happens if a network failure isolates the San Francisco regional office from the other two regional offices. Because the image of `SFDB` has a weight of 3 in the San Francisco partition, that image constitutes a quorum, so stores in that region can continue to add members and verify local purchases. Because the images `CHDB` and `NYDB` are now nonquorum images in the San Francisco partition (and because the database application allows stale reads from nonquorum images), the San Francisco stores can still approve limited purchases by members visiting from Chicago or New York.

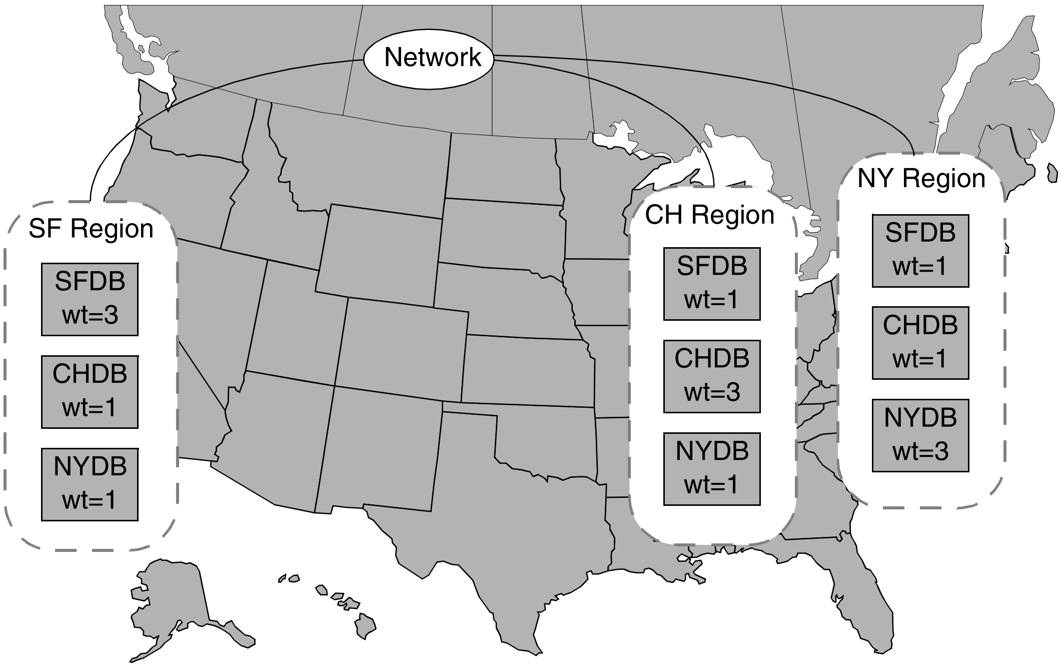

Figure 2-3 Unevenly Weighted Database Images

Hot-Failover Configurations

A *hot-failover* configuration is a pair of machines set up so that if one machine fails, the other machine automatically becomes a hot backup for it. To ensure continuous availability of data, you can use a replicated database in a hot-failover configuration by putting one image of the database on each of the two machines. You must also use one of the following special techniques for establishing a quorum in a hot-failover configuration:

- Adding a tie-breaker partition (page 43)
- Defining two-machine handler functions (page 44)

Figure 2-4 shows a hot-failover configuration for database `DB7`. Two autonomous partitions each control a database image of `DB7`. Each partition (and the image it controls) resides on a separate machine within a separate local-area network; the local-area networks are connected by a wide-area network.

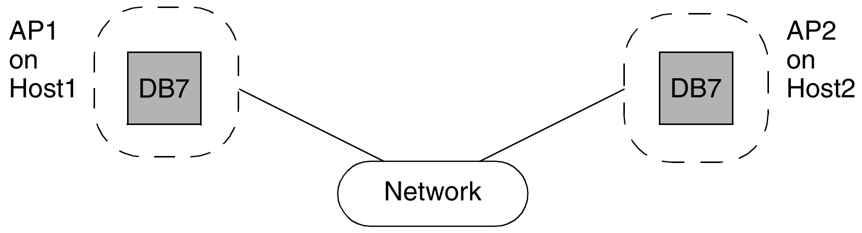

Figure 2-4 Hot-failover configuration

In a hot-failover configuration, quorum calculation should produce the following results:

- If one of the machines fails, the single database image on the remaining machine should constitute a quorum. (Thus, if `Host1` fails, the image on `Host2` should be sufficient for a quorum, and vice versa.)
- If a network failure isolates the two machines, one of the images should constitute a quorum and the other should not.

Manipulating image weights cannot achieve the desired results. Suppose you give the image on `Host1` a greater weight than the image on `Host2`. Then, if `Host1` fails, the image on `Host2` cannot constitute a quorum, so it cannot supply the necessary backup. You must use either a tie-breaker partition or a two-machine handler function, instead.

Tie-Breaker Partitions

The most common technique for establishing a quorum for a database in a hot-failover configuration is to set up a *tie-breaker partition* for the database. A tie-breaker partition is a specially designated autonomous partition that functions as a *pseudo image* of the database during quorum calculation. Whenever a quorum is calculated for the database, the tie-breaker partition acts as if it controls a database image with weight 1, although no such image actually exists in that partition.

Any partition that does not already control an image of the database can serve as a tie-breaker for that database. Often a new partition is created for the sole purpose of acting as a tie-breaker. The same partition can serve as a tie-breaker for more than one database; however, a given database can have only one tie-breaker partition at a time. A tie-breaker partition can be set for a database that has any number of images, although tie-breakers are normally most useful for databases with exactly two images (the hot-failover configuration).

A tie-breaker partition for a database can be located on any machine in the network. However, to be effective, the tie-breaker partition should not reside on the same host as the database's real images.

Figure 2-5 shows the hot-failover configuration from Figure 2-4 after a partition `AP3` has been created on `Host3` and set as a tie-breaker partition for `DB7`. This configuration supports hot-failover as follows:

- If either `Host1` or `Host2` fails, applications on the remaining hosts can continue updating `DB7`, because such applications can access both the remaining image and the tie-breaker partition, which constitute a quorum.
- If either `Host1` or `Host2` is isolated by a network failure, applications on the host connected to `Host3` can continue updating `DB7`, because they can access both the image on the local host and the tie-breaker partition, which constitute a quorum. Because applications on the isolated host can no longer access the tie-breaker partition, they cannot access a quorum.

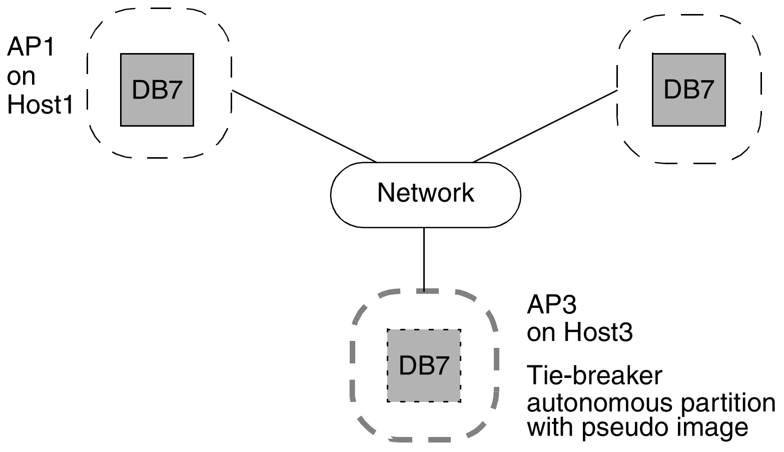

Figure 2-5 Tie-breaker scenario

For purposes of quorum calculation, setting a tie-breaker partition for a database is equivalent to adding another image to the database. However, tie-breaker partitions have several advantages:

- A tie-breaker partition adds only minimal I/O during quorum calculation, so it has no significant impact on application performance. In contrast, each added image increases the I/O performed during operations that read or update the database, because the new image must be locked or updated in parallel with the database's other images.
- A tie-breaker partition does not control a real image of the database, so it can be hosted on a less powerful machine with a smaller disk than a partition controlling a real image.

Two-Machine Handler Functions

A second approach to establishing a quorum in a hot-failover configuration is available to Objectivity/C++ applications if special hardware and software on the two machines enable applications on each machine to check on the status of the other machine. In such a configuration, each application that accesses the database can register a *two-machine handler function*.

NOTE Two-machine handler functions are used *only* in federated databases that contain exactly two partitions.

An application's two-machine handler function is called when the application can access only one image of a database, typically when the application can access the local database image but not the image on the other machine. The function tests whether the local database image can be considered to constitute a quorum by itself.

The important characteristic of a two-machine handler function is the ability to distinguish between a machine failure and a network failure. The function should check whether the other machine is still running. If not, a machine failure has occurred and applications on the remaining machine should be able to access the available image. If the other machine is running, however, a network failure has occurred; applications on one, but not both, of the two machines should be able to access the local image.

For additional information, see the database images chapter in *Objectivity/C++ Programmer's Guide*.

Restoring Images Into Service

When an image of a database becomes inaccessible for any reason (such as a network, host, or Objectivity-server failure, or a planned disconnect), Objectivity/DB automatically removes the inaccessible image from service—that is, the image is removed from the database's quorum, and is therefore unavailable for further updates. If applications continue to update the quorum images, the inaccessible image becomes out-of-date. As such updates occur, Objectivity/DB keeps track of the modified pages.

After you correct the failure (or otherwise restore the relevant connections), the out-of-date image becomes accessible again, but it is not actually restored into service until it is added back into the quorum. Adding a previously inaccessible image back into the quorum causes the image to be *synchronized* with the quorum images (updated with the recorded modifications), and makes the image available for subsequent read and update operations.

Objectivity/DB applications promote the restoration of images by automatically checking for nonquorum images that are currently accessible. Whenever an application detects such images, it performs an automatic quorum calculation to restore them into service. (Such images are also restored into service when quorum calculation is performed for any other reason; see "Automatic Quorum Calculation" on page 36.)

For performance reasons, automatic restoration of images is performed only infrequently, and not on a predictable schedule. Although a newly accessible image will be added back into the quorum eventually, an application may have to wait several hours for this to happen automatically. For more control over image restoration, you can supplement the automatic mechanisms by providing your deployment sites with standard or custom administrative tools that restore images into service on demand. Such tools can manage a replicated database in either of the following ways:

- By requesting a full quorum calculation for the database. This ensures that the quorum is as inclusive as possible, but takes time (because the accessibility of every image must be tested).
- By adding a particular image to the database's quorum without performing a full quorum calculation. This quickly synchronizes the desired image with the quorum images, but may overlook other images of the database that have recently become accessible.

For example, a custom administrative application might recalculate the quorum for each database that has an image in a specified partition. The custom application could be run as part of the recovery process when the partition's host is restarted.

For information about using the standard tool (`ooreplstat`) for restoring database images into service, see "Restoring Database Images into Service" on page 74. For information about developing custom applications for managing replicated databases, see the chapter on database images in the guide for your Objectivity programming interface.

Quorum of Partitions Revisited

The quorum mechanism that manages updates to replicated user data also manages updates to global schema and catalog information (see "Access to Global Federated-Database Information" on page 26). Such global information is stored in the system database, which is replicated in every partition. A tool or application is said to access a quorum of autonomous partitions when it can access a quorum of the system database's images. Thus:

- The weight of a partition is actually the weight of the image of the system database in that partition.
- Each *quorum partition* is a partition that controls a quorum image of the system-database file.

Because the system database contains critical schema and catalog information, Objectivity/DB performs automatic quorum calculations much more frequently for the system database than for ordinary databases. (Whenever a tool or application attempts to update the system database, Objectivity/DB checks whether any partitions are newly accessible; if so, the quorum is recalculated.) Furthermore, Objectivity/DB ensures maximum access to the system database by always allowing nonquorum reading.

Part 2 USAGE

3

Autonomous Partition Tasks

NOTE This chapter applies only to *non-placement-managed* federated databases.

An autonomous partition is a unit of administrative control for a subset of the databases in a federated database. Each autonomous partition is associated with a complete set of Objectivity/DB system resources that applications use when accessing the partition's databases. Administering an autonomous partition typically involves establishing the set of databases it controls and relocating its system resources.

This chapter describes:

- Structure and properties of autonomous partitions
- Access required to perform autonomous-partition tasks
- General administrative tasks such as backup and recovery
- Configuring Objectivity/Assist to display autonomous partitions
- Getting information about an autonomous partition, its properties, and the databases it controls
- Creating an autonomous partition
- Changing the properties of an autonomous partition, such as the locations of its system resources, its offline status, or its weight.
- Managing databases controlled by autonomous partitions
- Deleting autonomous partitions
- Purging autonomous partitions that are permanently inaccessible

Unless otherwise indicated, the tasks described in this chapter also apply to partitions that contain replicated data.

About Autonomous Partitions

Physically, an autonomous partition consists of two files:

- A system-database file containing a copy of the federated database's system database, which includes the schema and the global catalog.
- A boot file identifying the Objectivity/DB system resources (system-database file, lock server, and journal directory) for the partition.

NOTE Before you can work with autonomous partitions, you must install and run AMS on every data-server host where a system-database file is to reside. For more information, see Chapter 9, "Advanced Multithreaded Server," of *Objectivity/DB Administration*.

An autonomous partition has a number of *properties*. Two properties specify the partition's identity:

- Integer identifier—the unique positive integer that is assigned by Objectivity/DB when the partition is created. The identifier is a single, nonnegative number. The first partition, which is created implicitly when you create the federated database, is given the reserved identifier 1; partitions created subsequently have sequential identifiers starting with 2.
- System name—logical system name that is specified when the partition is created. The system name is unique among the system names of all autonomous partitions and databases in the federated database.

The following properties specify the locations of the Objectivity/DB system resources for the partition:

- System-database file host and file path—the host system and full directory path (including the filename) of the partition's system-database file.
- Journal-directory host and path—the host system and directory path for the partition's journal files, which store information for Objectivity/DB to use when rolling back incomplete transactions against the databases controlled by the partition.
- Lock-server host—the host system on which the lock server for the partition is running.
- Boot-file host and path—the host system and directory path for the boot file of the partition.

The following properties specify the partition's:

- Offline status—whether this partition is to be considered private except to tools and privileged applications. By default, this property is set to online, which makes the partition accessible to any application or tool.
- Weight—integer value of this partition to be used in quorum calculations. By default, the weight of a partition is 1.

The values for these properties are set when an autonomous partition is created. Because the initial autonomous partition is created implicitly when the federated database is created, the properties of the initial partition are those of the federated database. You can modify various properties of an existing autonomous partition using administrative tools; "Changing Autonomous-Partition Properties" on page 59.

Required Access to Autonomous Partitions

The description of each task in this chapter indicates which autonomous partitions must be accessible to the tool that performs the task. A partition is accessible to a running tool if the tool can successfully connect to the partition's system-database file (and AMS on the file's data-server host), lock server, and journal directory. This means that all of the relevant network links, hosts, and server processes must be operational.

NOTE The boot partition that you specify to the tool must be accessible so that the tool can start.

- The following tools can be used in tasks that change the global catalog or schema. Such tasks require that a *quorum* of partitions be accessible. The quorum must include the partition whose boot file is specified to the tool, and any specific partition that is affected by the operation:
  - `oochange`
  - `oochangedb`
  - `ooddlx`
  - `oodeleteap`
  - `ooimport`
  - `oonewap`
  - `ooschemaupgrade`

- The following tools can be used in tasks that simply read global catalog information. Such tasks require that at least one partition be accessible—namely, the partition whose boot file is specified to the tool:
  - `oochange`
  - `oochangedb`
  - `oodumpcatalog`
  - `oosupportinfo`
  - `ooexportfd`
  - `ooexportcatalog`
  - `ooexportschema`
  - `ooexportdata`

General Administrative Tasks

You use standard Objectivity/DB tools to perform general administrative tasks on a partitioned federated database. These tools are described in *Objectivity/DB Administration*.

Backing Up a Partitioned Federated Database

Must have access to:

- Quorum of autonomous partitions, including the boot partition for the tool

  The boot partition must be the partition to which the *backup boot file* belongs. The backup boot file is the boot file that was used the first time a backup was performed on the federated database.
- Quorum of images for each database

You back up a partitioned federated database by following the procedures described in "Performing a Backup" in Chapter 12 of *Objectivity/DB Administration*.

NOTE For referential integrity reasons, you cannot back up just an individual partition.

Restoring a Partitioned Federated Database

You can restore a partitioned federated database either with its autonomous partitions or without them.

Restoring With Partitions

You can restore a federated database with its partitions in the following cases:

- You are restoring to the original physical locations (hosts and directories). See "Restoring Files to Their Original Locations" in Chapter 12 of *Objectivity/DB Administration*.
- You are using a mapping file to restore to multiple new locations. See "Restoring Files to Multiple New Locations" in Chapter 12 of *Objectivity/DB Administration*.

You cannot use the `-standalone` option when restoring a federated database with its partitions.

Restoring Without Partitions

You must restore the federated database *without* its partitions if you are restoring to a single new location—or example, because the original hosts or directories no longer exist. To do so:

1. Follow the procedure in "Restoring Files to a Single New Location" in Chapter 12 of *Objectivity/DB Administration*, adding the `-purgeAps` option to the `oorestore` tool.

   The restored federated database consists of the archived boot partition with a single image of every database in the federation. Partitions other than the boot partition are purged from the global catalog.
2. Re-create any desired partitions.

Recovering Incomplete Transactions

Must have access to: The autonomous partitions that were involved in the transaction being recovered

Recovery for a partitioned federated database is similar to recovery for an unpartitioned federated database. See Chapter 13, "Automatic and Manual Recovery," and "Getting Transaction Information" in Chapter 4 of *Objectivity/DB Administration*.

The main differences are:

- The entire federated database does not need to be accessible—only the partitions that were involved in the transaction being recovered. A partition is involved in a transaction if the partition controls a quorum image of at least one database that was open for update during the transaction.

- You can recover a transaction using the boot file of *any* partition that was involved in the transaction.

Installing a Partitioned Federated Database

The purpose of installing a federated database is to update its properties and global catalog after placing its files in a new operating environment. Because autonomous partitions are highly specific to a particular operating environment, federated databases must be installed without partitions. One way to arrange for this is to delete the partitions from the federated database *before* you place its files in the new operating environment; see “Deleting an Autonomous Partition” on page 63.

However, deleting the partitions in the original federated database may be undesirable (for example, because you want to keep the original partitions to support continuing work) or impossible (for example, because you don’t have access to the federated database in its original operating environment). In this case, you must purge the partitions as part of the install operation.

To purge autonomous partitions while installing a federated database:

1. Choose one partition to be the “main partition” (the one that best represents the entire federated database). Place its system-database file and boot file in the desired locations; place all its database files in a single directory.
2. Run the `ooinstallfd` tool with the `-purgeAps` option and the options you would normally use for an unpartitioned federated database; see “Installing a Federated Database” in Chapter 4 of *Objectivity/DB Administration*.

   This operation installs the main partition as an unpartitioned federated database, removes the other partitions from the global catalog, and detaches any database that does not have an image in the boot partition.
3. Reattach databases to the federated database as necessary; see “Attaching a Database to a Federated Database” in Chapter 6 of *Objectivity/DB Administration*.
4. Re-create any desired partitions; see “Creating an Autonomous Partition” on page 58.
5. Re-create any desired database images in the appropriate partitions; see “Creating a Database Image” on page 68.

Exporting From a Partitioned Federated Database

Must have access to:

- Boot autonomous partition for the tool
- Quorum of images for each database from which data is to be exported

You export a partitioned federated database as you would an unpartitioned federated database; see "About Exporting" in Chapter 14 of *Objectivity/DB Administration*. Autonomous partitions and database images are represented in an exported global catalog. An error message reports any partition that is inaccessible when a global catalog is exported.

Importing Into a Partitioned Federated Database

Must have access to:

- Quorum of autonomous partitions, including the boot partition for the tool
- Quorum of images for each database to be imported

You can import an XML document representing a partitioned federated database or any portion of one; see "About Importing" in Chapter 14 *Objectivity/DB Administration*.

By default, the import operation preserves a source federated database's autonomous partitions, although you can eliminate these partitions by adding the `-collapsepartitions` option when you run the `ooimport` tool.

Imported partitions are merged with any existing destination partitions; so a new partition is created only if no existing partition has a matching system name. If an imported partition's system name matches that of an existing partition, any database images controlled by the imported partition are placed under the control of the existing partition.

By default, all imported partitions are configured to use the boot-file location, system-database location, lock server host, and journal-directory host and path of the boot partition; in this case, you must perform separate operations to redistribute files or reconfigure Objectivity/DB system resources. As a convenience, you can instruct `ooimport` to reconfigure the Objectivity/DB system resources of one or more partitions. To do so, you edit the *install file* corresponding to the imported XML document. You can also use this technique to skip or rename partitions, or to change the controlling partition for one or more database images.

Enabling Objectivity/Assist to Show Partitions

Objectivity/Assist (Assist) is a tool for viewing and managing the data, files, and system resources of an Objectivity/DB federated database. You can use Assist to perform various administrative and development tasks pertaining to autonomous partitions. For basic information, including how to start Assist and set up an FD project, see Chapter 3, "Working With Objectivity/Assist," in *Objectivity/DB Administration*.

By default, Assist displays a federated database without its partitions. To enable Assist to display autonomous partitions:

1. In the Assist menu bar, click **Window >Preferences**.
2. In the Preferences dialog, click the **+** symbol next to **Objectivity/Assist**.
3. Click **Catalog**.
4. Select **show APs** and click **OK**.

Now you can expand the federated database's **Catalog** in **FD Project Explorer** to show the following items:

- **Autonomous Partitions**—Expands to show one **AP** entry for each partition in the federated database.
- **Database Images**—Expands to show a single **DB Image** entry for each unreplicated database, and multiple **DB Image** entries for each replicated database in the federation.
- **Databases—**Expands to show one **DB** entry for each database in the federation. Each **DB** entry expands to show the **DB Image** entries for just the images of that database.
- **Hosts—**Expands to show a **Host** entry for each lock-server or data-server host used by the partitions in the federated database.

You can double-click or right-click an **AP**, **DB Image**, **DB**, or **Host** entry to work with a particular partition, image, database, or host.

When partitions are shown, Assist reorganizes database properties so you:

- Select a **DB Image** entry to work with the location and replication properties of a database file or image file.
- Select a **DB** entry to work with the database properties that are invariant among all images of a particular database.

(As always, you work with the contents of a particular database by expanding **Data** in **FD Project Explorer** and selecting a **Database** entry.)

Getting Autonomous-Partition Information

You can get information about an autonomous partition, including the current values of its properties. Such values are typically used as input to other tools.

Listing All Partitions and Their Properties

Must have access to: Boot autonomous partition for the tool

To list all partitions in a federated database, along with their properties, either:

➤ Run `oodumpcatalog` (see *Objectivity/DB Administration*) with the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and double-click **Autonomous Partitions**.

Listing the Properties of a Partition

Must have access to: Boot autonomous partition for the tool

To list the properties of a particular autonomous partition, either:

➤ Run `oochange` (see *Objectivity/DB Administration*) with the `-ap` or `-id` option specifying the partition of interest, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Autonomous Partitions**. Double-click on the **AP** entry for the partition of interest.

Listing Control Relationships for a Partition

Must have access to: Boot autonomous partition for the tool

To list the files controlled by a particular autonomous partition, run `oodumpcatalog` (see *Objectivity/DB Administration*) with the `-ap` or `-id` option specifying the partition of interest, and the *`bootFilePath`* for any partition.

Creating an Autonomous Partition

- **Must have access to:** Quorum of autonomous partitions, including the boot partition for the tool
- The initial partition in a federated database is created implicitly when you create the federated database. You must create the second and subsequent partitions explicitly.

Planning for the New Partition

Before you create a new autonomous partition.

1. Identify the partition's lock-server host.
2. Identify the specific locations of system-database file, journal directory, and boot file of the new partition. These locations can be on the same data-server host or on different data-server hosts.

   For best performance, the journal directory should not contain the system-database file and should not be the journal directory of any other partition.
3. Choose the partition's system name. The system name will be used as the simple name of the partition's boot file.

Procedure for Creating the New Partition

To explicitly create an autonomous partition:

1. Verify that a lock server is running on the intended lock-server host for the new partition; start it, if necessary.
2. Verify that AMS is running on the data-server host that is to store the new partition's system-database file; start it, if necessary.

   Note: If you are creating the *second* partition in a federated database, you must *also* start AMS on the host that stores the original system-database.
3. Run the `oonewap` tool (page 82), with the `-ap` option, the `-lockserverhost` option, the `-apfilepath` option, and the *`bootFilePath`* for any partition. (Specify additional options to set nondefault values for the other properties.)

NOTE *(On selected platforms)* You can use Objectivity/Assist (page 56) instead of running `oonewap`. In **FD Project Explorer**, click to expand **Catalog**. Right-click **Autonomous Partitions**, click **Add AP**, and fill in the dialog.

A newly created partition has no databases under its control; see "Transferring Control of a Database to Another Partition" on page 62.

Changing Autonomous-Partition Properties

You can change various properties of an autonomous partition, usually to accommodate system or network changes or to improve application performance. The following properties can be changed:

- Host and path of any system resource
- Offline status
- Weight

NOTE You cannot change a partition's system name or identifier.

Moving a Partition's System Resources

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the partition of interest.

You can move one or more system resources of a partition by changing the corresponding host or path properties. To do so, either:

➤ Run the oochange tool (see *Objectivity/DB Administration*) with the `-ap` or `-id` option specifying the partition of interest, the *`bootFilePath`* for any partition, and the appropriate options from the following table.

To Change This Resource's Host or Path	Use These oochange Options
System-database file	`-sysfilehost` `-sysfilepath`
Boot file	`-bootfilehost` `-bootfilepath`
Journal directory	`-jnldirhost` `-jnldirpath`
Lock server (see Note, below)	`-lockserverhost`

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Autonomous Partitions**. Right-click on the **AP** entry for the partition of interest, click **Edit**, and fill in the dialog.

The change operation updates the specified location properties in the boot file and global catalog. In addition:

- Changing the properties of the boot file creates a new copy of the file in the file system. You delete the old boot file using an operating-system command.
- Changing the properties of the system-database file physically moves (or renames) the file in the file system. **Note:** If you do not want Objectivity/DB to move the file, add the `-catalogonly` option to `oochange`, and use an operating-system command to physically move or rename the file.

NOTE If you change the property for a partition's lock-server host, you must also restart the lock server on the new host; see "Changing Lock-Server Hosts" in Chapter 8 of *Objectivity/DB Administration* for the complete procedure.

Changing the Offline Status

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool

To change the offline status of an autonomous partition, either:

➤ Run the `oochange` tool (see *Objectivity/DB Administration*) with the `-ap` or `-id` option specifying the partition of interest, the `-offline` or `-online` option, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Autonomous Partitions**. Right-click on the **AP** entry for the partition of interest, click **Mark Offline** or **Mark Offline**.

Changing the Weight

Must have access to: Quorum of autonomous partitions (both before and after the change), including the boot partition for the tool

To change the weight of an autonomous partition, either:

➤ Run the `oochange` tool (see *Objectivity/DB Administration*) with the `-ap` or `-id` option specifying the partition of interest, the `-weight` option, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Autonomous Partitions**. Right-click on the **AP** entry for the partition of interest, click **Edit** and fill in the dialog.

You may not change the weight of a partition if doing so would eliminate the quorum of partitions.

EXAMPLE Assume that a federated database has four partitions (A, B, C, and D), where partitions A, B, and C each have a weight of 1, and partition D has a weight of 5. Only partitions C and D are accessible to the `oochange` tool, but they constitute a quorum, because their combined weight (6) is greater than half of the total (8).

With this quorum, you can change D's weight to 2, because the new combined weight of C and D (3) is still greater than half the new total of (5).

However, attempting to change D's weight to 1 would fail, because the combined weight (2) of partitions C and D would now be exactly half of the total weight (4), so the quorum would be eliminated.

Assume again that partitions A, B, and C each have a weight of 1, partition D has a weight of 5, and partitions C and D are both accessible. Partitions C and D constitute a quorum, because their combined weight (6) is greater than half of the total (8).

With this quorum, you can change A's weight to 3, because the combined weight of C and D (6) is still greater than the new total (10).

However, attempting to change A's weight to 7 would fail, because the combined weight (6) of the partitions C and D would now be less than half of the new total weight (14), so the quorum would be eliminated.

Managing Controlled Databases

You can create a database directly in the autonomous partition that is to control it, and you can transfer the control of a database from one partition to another.

Creating a Database in the Controlling Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the partition of interest.

To create a database directly in the partition that is to control it, either:

- ➤ Run `oonewdb` (see *Objectivity/DB Administration*) with the `-db` option, the `-ap` option specifying the partition of interest, the `-host` and `-filepath` options, and the *`bootFilePath`* for any partition.
- ➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, right-click **Data**, click **Add DB**, and fill in the dialog (including the AP field).

If the database is to be replicated, you create additional images as described in “Creating a Database Image” on page 68.

Transferring Control of a Database to Another Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the source and destination partitions

To transfer the control of a database from one partition to another, either:

- ➤ Run `oochangedb` (see *Objectivity/DB Administration*) with `-db` or `-id` option, the `-ap` option specifying the source partition, the `-movetoap` option specifying the destination partition, and the *`bootFilePath`* for any partition.
- ➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for the database of interest. Right-click this entry, click **Edit**, and choose a value for the dialog’s AP field.

This operation updates the global catalog to reflect the new controlling partition for the database.

If the database is replicated, you move its images individually; see “Transferring Control of an Image to Another Partition” on page 71.

Deleting an Autonomous Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool

To delete an autonomous partition, either:

- ➤ Run `oodeleteap` (page 79) with the `-ap` or `-id` option specifying the partition of interest, and the *`bootFilePath`* for any partition.

  To delete a partition that controls the *only* image of a database, you must either add the `-deleteLastImage` option or move the image to another partition before running `oodeleteap`.

- ➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Autonomous Partitions**. Right-click on the **AP** entry for the partition of interest, click **Delete.**

Deleting a partition removes its entry from the federated database's global catalog. In addition, if the partition is accessible, the operation physically deletes the following files from the file system:

- ■ The partition's system-database file.
- ■ The partition's boot file.
- ■ The database images controlled by the partition.

If the partition is inaccessible, you must delete the files with operating-system commands.

Purging Autonomous Partitions

Must have access to: Quorum of autonomous partitions that are *not* being purged; quorum must include the boot partition for the tool

If you want to remove one or more autonomous partitions, but you cannot delete them because you cannot access a quorum of partitions, you can *purge* the unwanted partitions from the global catalog.

To purge one or more autonomous partitions, run `oopurgeaps` (page 87) with the `-aps` or `-apids` option specifying the partitions of interest, and the *`bootFilePath`* for any partition.

A purge operation succeeds only if a quorum of partitions will be accessible *after* the operation is performed. (This quorum is calculated relative to the reduced set of partitions that remain after the unwanted partitions are removed.) In contrast, a delete operation succeeds only if a quorum of partitions is accessible *before* the operation is performed—that is, before any partitions are removed.

A purge operation removes the following references from the global catalog:

- All references to the partitions being purged.
- All references to the individual database images controlled by the purged partitions.
- All references to any database whose quorum was in the purged partitions. That is, if a quorum of images for a database can no longer be accessed after the purge, all references to all images of that database are removed—in effect, removing the database from the federated database.

No files are deleted by a purge operation.

4

Data Replication Tasks

NOTE This chapter applies only to *non-placement-managed* federated databases.

You can improve the availability of a particular database replicating it—that is, by creating and distributing managed copies (called *images*) of the database among multiple autonomous partitions.

This chapter describes:

- General information about database images
- Access required to perform replication tasks
- Getting information about database images, such as its properties and its controlling partition
- Creating a database image
- Changing the properties of a database image
- Managing the tie-breaker partition for a database
- Deleting a database image
- Checking the accessibility of database images in the federated database
- Restoring database images into service

About Database Images

A database image is a managed copy of a database; every image of a particular database is controlled by a different autonomous partition. The number of images a database can have is thus limited by the number of autonomous partitions in the federated database.

Physically, every image is a separate database file. Thus, a replicated database exists as a set of equivalent files, with no intrinsic difference between the original database file and any replicated image of it. An unreplicated database is simply a database with a single image.

NOTE Before you can work with replicated databases, you must install and run AMS on every data-server host where a database image is to reside. For more information, see Chapter 9, “Advanced Multithreaded Server,” of *Objectivity/DB Administration*.

A database image’s *properties* specify its various physical and logical characteristics. Some properties belong to the database as a whole and so are shared by all images of the same database. Other properties are unique to each individual image.

The following properties describe a database image’s logical identity within a federated database:

- Database system name—the logical name of the database (*shared by all images of the same database*)
- Database identifier—the numeric identifier of the database (*shared by all images of the same database*)
- Controlling autonomous partition—partition to which the image belongs (*unique for each image of a database*)

The following properties describe a database image’s physical file location:

- File host—the host on which the image’s file resides
- File path—the directory pathname and filename of the image’s file

The following property describes a database image’s:

- Weight—integer value to be used in quorum calculations. By default, the weight of a partition is 1.

The properties of a database image are set when the image is created. You can change various properties of an image; see “Changing Database-Image Properties” on page 69).

Required Access to Autonomous Partitions

The description of each task in this chapter indicates which autonomous partitions must be accessible to the tool that performs the task. A partition is accessible to a running tool if the tool can successfully connect to all of the Objectivity/DB system resources that are associated with the partition (system database file, boot file, lock server, journal directory). This means that all of the relevant network links, hosts, and server processes must be operational.

NOTE The boot partition that you specify to the tool must be accessible so that the tool can start.

In general, if a replication task:

- Changes the global catalog, the relevant tool must be able to access a quorum of partitions.
- Reads the data in a database, the relevant tool must be able to access a quorum of the database's images. (An image is accessible if its controlling partition is accessible).
- Affects a particular database image, the relevant tool must be able to access the partition that controls the image of interest.

Getting Database-Image Information

You can get information about a database image, including the current values of its properties. Such values are typically used as input to other administration tools.

Listing Images of All Databases

Must have access to: Boot autonomous partition for the tool

To see a list of all images of every database in a federated database, either:

➤ Run `oodumpcatalog` (see *Objectivity/DB Administration*) with the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and double-click **Database Images**.

The `oodumpcatalog` output groups the entries for database images by controlling partition. The information shown for each database image includes its database system name, database identifier, weight, and database file location. Images of the same database in different partitions all have the same database system name and database identifier.

Listing Images of a Particular Database

Must have access to: Boot autonomous partition for the tool

To see a list of all images of a particular database, either:

➤ Run the `oochangedb` tool (see *Objectivity/DB Administration*) with the `-db` or `-id` option, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Databases**. Click to expand the **DB** entry for the database of interest to see the list of **DB Image** entries.

Listing the Properties of a Database Image

Must have access to: Boot autonomous partition for the tool

To list the properties of a particular database image, either:

➤ Run the `oochangedb` tool (see *Objectivity/DB Administration*) with the `-db` or `-id` option, the `-ap` option, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, click to expand **Catalog** and **Database Images**. Double-click the **DB Image** entry for the image of interest.

Creating a Database Image

Must have access to:

- Quorum of autonomous partitions, including the boot partition for the tool, and the partition that will control the new image
- Quorum of the database's images

You can create an image of a particular existing database in any autonomous partition that does not already control an image of that database.

To create the initial image of a database, see "Creating a Database in the Controlling Partition" on page 62.

To create an image of an existing database:

1. Verify that AMS is running on the data-server host that is to store the new image; start it, if necessary.

   Note: If you are creating the *second* image of a database, you must *also* start AMS on the host that stores the initial image of the database.

2. Run the `oonewdbimage` tool (page 85) with the `-db` or `-id` option and the `-ap` option specifying the partition to control the new image, and the *`bootFilePath`* for any partition.

   Specify additional options to set nondefault values for the other properties. For example, add the `-host` and `-filepath` options to specify the image's location; otherwise, the image is created with a generated filename in the current directory.

NOTE *(On selected platforms)* You can use Objectivity/Assist (page 56) instead of running `oonewdbimage`. In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for an existing image of the database. Right-click this entry, click **Replicate**, and fill in the dialog.

EXAMPLE This command creates a new image for database `DB5` in partition `AP1` on host `sys22`, with the path `/mnt/objyAP1/DB5Image.DB` and with a weight of 3.

```
oonewdbimage -db DB5 -ap AP1 -host sys22
    -filepath /mnt/objyAP1/DB5Image.DB -weight 3 partsFD
```

You cannot replicate a database that has external containers (containers stored in separate container files). Furthermore, after a database is replicated, an external container cannot be added to it.

Changing Database-Image Properties

You can change the following properties of a database image:

- Weight
- Host and path of the image file
- Controlling partition

You can also change the images of a database to read-only or read/write.

NOTE If a database is read-only, you must change it to read/write before you can change the properties of any of its images; see "Making Database Images Read-Only or Read/Write" on page 71.

Changing a Database Image's Weight

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool.

To change a database images's weight, either:

➤ Run oochangedb (see *Objectivity/DB Administration*) with the `-db` or `-id` option, the `-ap` option specifying the controlling partition, the `-weight` option, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for the image of interest. Right-click this entry, click **Edit**, and specify the weight in the dialog.

Moving a Database Image's File

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the partition controlling the image of interest

To move a database image's file to a new location, either:

➤ Run oochangedb (see *Objectivity/DB Administration*) with the `-db` or `-id` option, the `-ap` option specifying the controlling partition, the `-host` and `-filepath` options specifying the file's new location, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for the image of interest. Right-click this entry, click **Edit**, and specify the host and directory in the dialog.

EXAMPLE This command changes the location properties of the image of database `DB5` in partition `AP1`. The moved image is `/mnt/objyAP1/DB5Image.DB` on host `sys22`.

```
oochangedb -db DB5 -ap AP1 -host sys22
   -filepath /mnt/objyAP1/DB5Image.DB FD1
```

Moving a database image's file:

- Updates the file's location properties in the global catalog.
- Physically relocates the file in the file system.

Transferring Control of an Image to Another Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the source and destination partitions

To transfer the control of a database image from one partition to another, either:

➤ Run oochangedb (see *Objectivity/DB Administration*) with the `-db` or `-id` option, the `-ap` option specifying the source partition, the `-movetoap` option specifying the destination partition, and the *`bootFilePath`* for any partition.

➤ *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for the image of interest. Right-click this entry, click **Edit**, and choose a value for the dialog's AP field.

The destination partition may not already control an image of the database.

EXAMPLE This command moves an image of database `DB5` from partition `AP1` to partition `AP2`.

```
oochangedb -db DB5 -ap AP1 -movetoap AP2
```

Making Database Images Read-Only or Read/Write

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool

To make the images of a replicated database read-only or read/write, either:

➤ Run oochangedb (see *Objectivity/DB Administration*) with the `-db` or `-id` option, the `-ap` option, the `-readonly` or `-readwrite` option, and the *`bootFilePath`* of any partition. The `-ap` option must specify a partition that controls an image of the database.

➤ *(On selected platforms)* Use Objectivity/Assist (page 57). In **FD Project Explorer**, browse **Data** to find the **Database** entry for the database of interest; right-click this entry and click **Set Read Only** or **Set Read-Write**.

Making a replicated database read-only (or read/write) makes *all* of its images read-only (or read/write).

While a replicated database is read-only, you can read its contents, add or delete images, or change its images back to read/write; you cannot change the properties of an individual image. For more information, see "Read-Only and Read/Write Databases" in Chapter 6 of *Objectivity/DB Administration*.)

Managing Tie-Breaker Partitions

A replicated database in a hot-failover configuration can have a tie-breaker partition to establish a quorum of images. See “Tie-Breaker Partitions” on page 43.

Listing the Tie-Breaker Partition

Must have access to: Boot autonomous partition for the tool

To see which partition (if any) is the tie-breaker for a database, run the `oodumpcatalog` tool (see *Objectivity/DB Administration*) with the *`bootFilePath`* for any partition. Every entry for the database lists the identifier of the tie-breaker partition for that database.

Alternatively, you can run the `oochangedb` tool (see *Objectivity/DB Administration*) with the `-db` or `-id` option and the *`bootFilePath`* for any partition. If the specified database has a tie-breaker partition, `oochangedb` gives the name of that partition.

Setting the Tie-Breaker Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool

To set a tie-breaker partition for a database, or to change the tie-breaker to be a different partition, run the `oonewdbimage` tool (page 85) with the `-db` or `-id` option, the `-ap` option, the `-tiebreaker` option, and the *`bootFilePath`* for any partition.

If necessary, you must create the desired partition before setting it as a tie-breaker. A lock server must be running on the lock-server host for the specified partition. For more information on lock servers, see *Objectivity/DB Administration*.

EXAMPLE This command sets partition `AP3` as the tie-breaker partition for database `DB5`.

```
oonewdbimage -db DB5 -ap AP3 -tiebreaker FD1
```

Removing the Tie-Breaker Partition

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool

To remove the tie-breaker partition for a database, run the `oodeletedbimage` tool (page 81) with the `-db` or `-id` option, the `-tiebreaker` option, and the *`bootFilePath`* for any partition.

EXAMPLE This command deletes the tie-breaker partition for database `DB5`.

```
oodeletedbimage -db DB5 -tiebreaker FD1
```

Deleting a Database Image

Must have access to: Quorum of autonomous partitions, including the boot partition for the tool, and the partition controlling the image of interest

To delete an individual database image from the autonomous partition that controls it, either:

- Run `oodeletedbimage` (page 81) with the `-db` or `-id` option, the `-ap` option, and the *`bootFilePath`* for any partition.

  To delete the last image of a database (and therefore, the database itself), you must also specify the `-deleteLastImage` option.
- *(On selected platforms)* Use Objectivity/Assist (page 56). In **FD Project Explorer**, browse **Catalog** to find a **DB Image** entry for the image of interest. Right-click this entry and click **Delete**.

EXAMPLE This command deletes the image of database `DB5` from partition `AP1`. The deletion is allowed, even if this is the last image of the `DB5` in the federated database.

```
oodeletedbimage -db DB5 -ap AP1 -deleteLastImage FD1
```

To delete all images of a database in a single operation, see “Deleting a Database” in Chapter 6 of *Objectivity/DB Administration*.

Checking the Accessibility of Database Images

Must have access to: Boot partition's system-database file

To check whether the images of one or more databases are accessible, you use `ooreplstat` (page 88). This tool tests whether an image is accessible by identifying the autonomous partition that controls it, and then attempting to connect to that partition's system resources, including its lock server and the associated AMS processes. An image is considered inaccessible if `ooreplstat` cannot connect to the partition's lock server or to AMS on the data-server host(s) for the image file and the partition's system-database file. The host of an Objectivity server is considered inaccessible if the server fails to respond within a specified timeout period.

- The `-db` or `-id` option identifies a single database whose images are to be checked; you can omit these options if you want to check the images of every replicated database in the specified federated database.
- The `-timeout` option specifies how long (in seconds) to wait for a response from an Objectivity server, before concluding that its host is inaccessible.

You can run the `ooreplstat` tool periodically to list:

- Any host or Objectivity server that is inaccessible
- Any partition that has been set offline
- Any database for which the tool cannot access a quorum of images
- Any database for which the quorum should be recalculated (for example, because a partition controlling an image has become accessible since the last quorum calculation)

If the entire system is operational and fully utilized, the following message is printed: `No problems found`.

Restoring Database Images into Service

To recalculate a database's quorum and restore its images into service after a previously inaccessible partition comes back online, use `ooreplstat` (page 88) with the `-renegotiate` option. You can use the `-db` or `-id` option to specify a particular database; otherwise, the tool restores images for any replicated database that is found to have at least one newly accessible image not currently in the quorum. The restored images are synchronized with images already in the quorum.

You should provide your deployment site with a mechanism for determining when it is necessary to resynchronize database images.

Part 3 REFERENCE

5

Tools

This chapter lists the administration tools for Objectivity/HA. The syntax descriptions of these tools appear in alphabetical order by tool name, beginning on page 79.

Tool Names

The names of the tools are the same on Windows and UNIX, with the exception that, on Windows, the filenames of the executables have an extension (`.exe`) that is not required on UNIX.

Tool Options and Arguments

The command-line syntax for most tools includes either or both of the following:

- *Options*, which modify the way the tool works. Syntactically, options are characters prefixed with a hyphen and set off with spaces—for example, `-help`. Some options are followed by *values*—for example, `-host` *`hostName`*.
- *Arguments*, which specify values directly to the tool. For example, many tools accept a *`bootFilePath`* argument.

When specifying options for an Objectivity/DB tool, you need to type only as many letters of the option as are necessary to identify it uniquely. This is also true for the fixed values sometimes associated with a command name or option.

Most options and arguments to Objectivity/DB tools are case sensitive; most options and arguments are lower case. Be sure to type options and arguments using the correct case.

Tool Return Status Conventions

Each tool described in this book returns status information using the standard tool return status conventions of the operating system you use.

Reference Index

The following table lists tools that are described in this book. These tools are provided with the Objectivity/HA product. General-purpose tools are described in *Objectivity/DB Administration*.

Tool	Description
`oodeleteap`	Deletes an autonomous partition from a federated database.
`oodeletedbimage`	Deletes an image or tie-breaker of the specified database.
`oonewap`	Creates a new autonomous partition in a federated database.
`oonewdbimage`	Creates a new image of the specified database in the specified autonomous partition.
`oopurgeaps`	Purges autonomous partitions from a federated database.
`ooreplstat`	Reports the status of database images in the specified federated database, optionally calculating the quorum for one or more replicated databases.

Reference Descriptions

oodeleteap

Deletes an autonomous partition from a federated database.
Not for use with a placement-managed federated database.

```
oodeleteap
     (-ap apSysName) | (-id apId)
     [-standalone]
     [-notitle]
     [-help]
     [-quiet]
     [-deleteLastImage]
     [bootFilePath]
```

Options

`-ap` *`apSysName`*

System name of the autonomous partition to delete.

`-id` *`apId`*

Integer identifier of the autonomous partition to delete (for example, 18).

`-standalone`

Nonconcurrent mode. Use this option if no lock server is running or to bypass a running lock server.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-help`

Prints the tool syntax and definition to the screen.

`-quiet`

Suppresses all normal program output.

`-deleteLastImage`

Deletes the autonomous partition even if it controls the last image of a read/write database. If you omit this option and the partition controls the last image of a database, the tool reports an error and terminates without deleting the partition.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database; the `-ap` or `-id` option specifies the particular partition to be deleted. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

This tool removes the specified autonomous partition from the federated database's global catalog. In addition, if the tool can access the specified partition, all files related to the partition are deleted from the file system.

If you attempt to delete a partition that controls the last image of a database, an error is reported. Use the `-deleteLastImage` option to force the deletion.

Before deleting a partition, you must ensure that no locks are being held in any database images controlled by the partition. Consequently, you should:

- Ensure that no active transactions exist against the partition.
- Recover any abnormally completed transactions against the partition (for example, using the `oocleanup` tool). For information about recovery, see *Objectivity/DB Administration*.

oodeletedbimage

Deletes an image or tie-breaker of the specified database.
Not for use with a placement-managed federated database.

```
oodeletedbimage
     -db dbSysName | -id dbId
     (-tiebreaker [-ap apSysName] |
       -ap apSysName [-catalogonly] [-deleteLastImage])
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *`dbSysName`*

System name of the database to be deleted.

`-id` *`dbId`*

Integer identifier of the database to be deleted (for example, 78). This option also accepts the identifier specified in D-C-P-S format (for example, 78-0-0-0). For a discussion of the format, see *Objectivity/DB Administration*.

`-ap` *`apSysName`*

System name of the autonomous partition controlling the database to be deleted. This option is required unless the `-tiebreaker` option is specified.

`-catalogonly`

Removes the specified database image from the global catalog but does not physically delete its database file.

`-deleteLastImage`

Deletes the database if the specified image is the last image of that particular database. This tool deletes the last image of a database only if this option is included *and* the database is read/write.

`-tiebreaker`

Deletes the tie-breaker for the specified database. If the `-ap` option is given, deletes tie-breaker only if *`apSysName`* matches the name of the database's tie-breaker partition.

`-standalone`

Nonconcurrent mode. Use this option if no lock server is running or to bypass a running lock server.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-quiet`

Suppresses all normal program output.

`-help`

Prints the tool syntax and definition to the screen.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

If the `-tiebreaker` option is specified, this tool deletes the tie-breaker partition for the specified database. Otherwise, it deletes the image of the database in the specified partition; an error is reported if the specified partition does not control an image of the specified database.

oonewap

Creates a new autonomous partition in a federated database.
Not for use with a placement-managed federated database.

```
oonewap
     -ap apSysName
     -lockserverhost lockServerHost
     [-lockwait seconds]
     [-apfilehost apFileHost] -apfilepath apFilePath
     [[-bootfilehost bootFileHost] -bootfilepath bootFilePath]
     [[-jnldirhost jnlDirHost] -jnldirpath jnlDirPath]
     [-weight weight]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-ap` *`apSysName`*

System name of the new autonomous partition. The specified name:

- Must be a valid filename in your operating system.
- Must be unique among the system names of all autonomous partitions in the federated database.
- Must be between 1 and 127 characters long inclusive, unless the boot file name is specified with *`bootFilePath`*.

The system name implicitly specifies the name of the autonomous-partition boot file.

`-lockserverhost` *`lockServerHost`*

Host of the lock server servicing the new autonomous partition.

`-lockwait` *`seconds`*

Number of seconds to wait to obtain a lock. If you omit this option, the default is 10 seconds.

`-apfilehost` *`apFileHost`*

Data-server host of the new system-database file. When you omit this option, the default host is:

- The current host, if `-apfilepath` specifies a local path.
- The host implied by `-apfilepath`, if an NFS mount name is specified.

`-apfilepath` *`apFilePath`*

Path to the new system-database file on the designated host. Because that host must be running AMS, *`apFilePath`* must be a name that is understood locally by the host's file system; you may not specify a Windows UNC share name. If *`apFilePath`* is an NFS mount name, it is automatically converted to the corresponding local pathname on the implied host.

If the `-apfilehost` option specifies a remote system, *`apFilePath`* must be full, not relative.

The path may optionally include the filename for the system database; if you omit the filename, it is generated automatically. The generated filename is of the form described in "System-Database Files" in Chapter 2 of *Objectivity/DB Administration*.

`-bootfilehost` *`bootFileHost`*

Host of the new autonomous partition's boot file. When you omit this option, the default host is:

- The current host, if `-bootfilepath` specifies a local path.
- The host implied by `-bootfilepath`, if an NFS mount name is specified.
- The same as *`apFileHost`*, if `-bootfilepath` is also omitted.

If `-bootfilepath` specifies a Windows UNC share name, *`bootFileHost`* is set to the literal string `oo_local_host`; any value you specify is ignored.

`-bootfilepath` *`bootFilePath`*

Path to the new autonomous partition's boot file on the designated host. If the `-bootfilehost` option specifies a remote system, *`bootFilePath`* must be full, not relative.

The path may optionally include the filename for the boot file; if you omit the filename, the default value is *`apSysName`*`.boot` in the *`bootFilePath`*

The name of the boot file (not including the extension) must be between 1 and 127 characters long inclusive or an error is reported.

If you omit both the `-bootfilepath` and `-bootfilehost` options, the default path is `*apSysName*.boot` in the *`apFilePath`* directory.

`-jnldirhost` *`jnlDirHost`*

Host of the new autonomous partition's journal directory. When you omit this option, the default host is:

- The current host, if `-jnldirpath` specifies a local path.
- The host implied by `-jnldirpath`, if an NFS mount name is specified.
- The same as *`apFileHost`*, if `-jnldirpath` is also omitted.

If `-jnldirpath` specifies a Windows UNC share name, *`jnlDirHost`* is set to the literal string `oo_local_host`; any value you specify is ignored.

`-jnldirpath` *`jnlDirPath`*

Path to the new autonomous partition's journal directory on the designated host. If the `-jnldirhost` option specifies a remote system, *`jnlDirPath`* must be full, not relative.

If you omit both the `-jnldirhost` and `-jnldirpath` options, the default path is the directory part of *`apFilePath`*.

`-weight` *`weight`*

Weight of the new autonomous partition. *`weight`* must be a positive integer. The default is 1. If you specify 0, `oonewap` reports an error.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-quiet`

Suppresses all normal program output.

`-help`

Prints the tool syntax and definition to the screen.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

Creating an autonomous partition creates the partition's system-database file and boot file.

oonewdbimage

Creates a new image of the specified database in the specified autonomous partition.
Not for use with a placement-managed federated database.

```
oonewdbimage
     (-db dbSysName | -id dbId)
     -ap apSysName
     (-tiebreaker |
         [[-host hostName] -filepath path][-weight weight])
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *`dbSysName`*

System name of the database.

`-id` *`dbId`*

Integer identifier of the database (for example, 78). This option also accepts the identifier specified in D-C-P-S format (for example, 78-0-0-0). For a discussion of the format, see *Objectivity/DB Administration*.

`-ap` *`apSysName`*

System name of the controlling autonomous partition for the new database image; if `-tiebreaker` is also specified, *`apSysName`* is the system name of the partition to be set as the tie-breaker partition for the specified database.

`-host` *`hostName`*

Data-server host of the new database image. The specified host must be running AMS or an error is reported and the image is not created.

When you omit this option, the default host is:

- The current host, if `-filepath` specifies a local path.
- The host implied by `-filepath`, if an NFS mount name is specified.
- The host of the controlling partition's system-database file, if `-filepath` is also omitted.

`-filepath` *`path`*

Path to the new database-image file on the designated host. Because that host must be running AMS, *`path`* must be a name that is understood locally by locally by the host's file system; you may not specify a Windows UNC share name. If *`path`* is an NFS mount name, it is automatically converted to the corresponding local pathname on the implied host.

If the `-host` option specifies a remote system, *`path`* must be full, not relative.

The path may optionally include the filename for the database image; if you omit the filename, it is generated automatically. The generated filename is of the following form described in “Database Files” in Chapter 2 of *Objectivity/DB Administration*.

If you omit both the `-filepath` and `-host` options, the default path consists of a generated filename in the directory of the controlling partition’s system-database file.

`-weight` *`weight`*

The weight of the new database image. The default is 1; must be an integer greater than 0 or an error is reported.

`-tiebreaker`

Sets the specified autonomous partition by `-ap` as the tie-breaker for the specified database; a new image is not created. If the specified partition already controls an image of the specified database, an error is reported and the tie-breaker is not set. If the specified database already has a tie-breaker partition, the specified partition replaces it as the tie-breaker.

`-standalone`

Nonconcurrent mode. Use this option if no lock server is running or to bypass a running lock server.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-quiet`

Suppresses all normal program output.

`-help`

Prints the tool syntax and definition to the screen.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database; the `-ap` or `-id` option specifies the particular partition to which the new database image is to be added. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

This tool reports an error and quits if either of the following is true:

- An image of the database already exists in the autonomous partition; a second image is not created.
- The database has one or more external containers (containers stored in separate container files). **Note:** After a database is replicated, an external

container cannot be added to it. For information about external containers, see *Objectivity/DB Administration*.

oopurgeaps

Purges autonomous partitions from a federated database.
Not for use with a placement-managed federated database.

```
oopurgeaps
     (-aps ap1SysName, ap2SysName, ...) | (-apids id1, id2, ...)
     [-force]
     [-standalone]
     [-notitle]
     [-help]
     [-quiet]
     [bootFilePath]
```

Options

`-aps` *`ap1SysName`*`,` *`ap2SysName`*`, ...`

System names of the autonomous partitions to purge.

`-apids` *`id1`*`,` *`id2`*`, ...`

Integer identifiers of the autonomous partitions to purge, separated by commas—for example, 18, 25, 30.

`-force`

Purges without prompting the user for verification.

`-standalone`

Nonconcurrent mode. Use this option if no lock server is running or to bypass a running lock server.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-help`

Prints the tool syntax and definition to the screen.

`-quiet`

Suppresses all normal program output.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database; the `-aps` or `-apids` option specifies the particular partitions to be purged. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

Use this tool to remove one or more autonomous partitions if you cannot delete them with `oodeleteap`.

A purge operation succeeds only if a quorum of partitions will be accessible *after* the operation is performed. (This quorum is calculated relative to the reduced set of nonpurged partitions that remain after the unwanted partitions are removed.) In contrast, a delete operation succeeds only if a quorum of partitions is accessible *before* the operation is performed—that is, before any partitions are removed.

Purging a partition removes all references to it from the global catalog, although the partition's files are not deleted.

When you purge a set of partitions, a new quorum calculation is performed for each database with an image in a purged partition. Each quorum calculation considers just the images in the nonpurged partitions that remain after the purged partitions are removed. After the quorum calculation for a database, one of the following actions is taken:

- If a quorum of images is accessible, then the database remains in the federated database, and any reference to an image or tie-breaker in the purged partitions is removed from the global catalog.
- If a quorum of images is not accessible, then the database is completely removed from the federated database—that is, all references to all images of the database are removed from the global catalog. The database file, however, is not deleted from the file system.

To use this tool safely, you must be *absolutely certain* that no Objectivity/DB process is active in any of the partitions to be purged.

ooreplstat

Reports the status of database images in the specified federated database, optionally calculating the quorum for one or more replicated databases. *Not for use with a placement-managed federated database.*

```
ooreplstat
     [(-db dbSysName | -id dbId)]
     [-renegotiate]
     [-timeout seconds]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-db` *`dbSysName`*

System name of the database whose images are to be considered.

If you omit both this option and the `-id` option, status is reported for all replicated databases in the federated database.

`-id` *`dbId`*

Integer identifier of the database whose images are to be considered (for example, 78). This option also accepts the identifier specified in D-C-P-S format (for example, 78-0-0-0). For a discussion of the format, see *Objectivity/DB Administration*. The identifier 1 specifies the system database that is replicated in each autonomous partition.

If you omit both this option and the `-db` option, status is reported for all replicated databases in the federated database.

`-renegotiate`

Calculates the quorum for the database(s) being examined.

Calculating the quorum for a database can momentarily reduce the runtime performance of every application that is accessing the database. This option limits the performance impact by calculating a database's quorum only if the quorum would actually be changed—that is, only if the database is found to have at least one accessible image that is not currently in the quorum. If calculation would not affect a database's quorum, the `-renegotiate` option is ignored for that database.

`-timeout` *`seconds`*

Number of seconds to wait for a response from a lock server or AMS, before concluding that the server host is inaccessible. If you omit this option, the timeout period is 25 seconds.

`-notitle`

Suppresses copyright notice and program title banner. Useful when invoking the tool from another tool or product.

`-help`

Prints the tool syntax and definition to the screen.

`bootFilePath`

Path to the boot file of any autonomous partition in the federated database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is valid only for non-placement-managed federated databases.

You can use `ooreplstat` to accomplish tasks such as the following:

- Check the status of the network to find out whether any Objectivity servers are inaccessible.

- Calculate the quorum for one or more databases, to ensure that all accessible images are included and resynchronized. To do so, specify the `-renegotiate` option.
- Diagnose the failure of an Objectivity/DB application. For example, if a failed application reports that it cannot connect to a lock server, but does not specify which one, you can run `ooreplstat` with the *`bootFilePath`* argument to determine which lock server is currently inaccessible.

When you omit the `-db` (or `-id`) option, this tool:

1. Identifies every autonomous partition that controls an image of a replicated database in the specified federation.
2. Attempts to perform the following:
   - Make a network connection to each partition's lock server.
   - Read each partition's journal directory.
   - Make a network connection to the AMS process(es) associated with every partition. (AMS must be running on every data-server host that has a system-database file or a database image file.)
3. Waits for a response from each Objectivity server within the timeout period specified by the `-timeout` option.
4. Prints messages reporting:
   - Any Objectivity server that isn't running (but could be restarted because its host is still operational and accessible over the network)
   - Any Objectivity-server host that is inaccessible (inferred when the relevant Objectivity server fails to respond within the timeout period)
   - Any partition that has been set offline
   - Any database for which the tool cannot access a quorum of images
   - Any database for which the quorum should be calculated (for example, because an image of the database has become accessible since the last quorum calculation)

If the entire system is operational and fully utilized, the following message is printed: `No problems found`.

You can specify the `-db` or `-id` option if you want to limit the status report to the partitions that control the images of a particular database.

You can specify the `-id` option with the value 1 to find out whether schema and global catalog information is up-to-date in the accessible partitions. (The identifier 1 refers to the system database that is replicated in every autonomous partition.) You can specify both the `-id 1` and `-renegotiate` options to resynchronize schema and global-catalog information in all accessible system-database images.

Index

A

abbreviation, tool options 77
access permissions 21, 34
accessibility, database
  about 20
  individual images 33
  replicated 33
administration tasks
  creating and modifying
    autonomous partition 58, 59
    database images 68, 69
  getting information
    autonomous partition 57
    database image 67
administration tools 77
AMS
  autonomous partitions 13, 17
  replicated data 29
application performance 22, 30
argument, tool 77
Assist 56
autonomous partitions
  about 14
  access to global information 26
  AMS required 50
  basics 13
  boot autonomous partition 19
  boot file 17, 84
  changing offline status 60
  configuring with import 55
  control vs. logical containment 15
  creating 58, 82
  data replication 31
  database images
    controlling 31
    transferring control 71
  deleting 63, 79
  example 22
  exporting 55
  guidelines for partitioning 15, 25
  identifier 50
  importing 55
  initial 14, 18
  journal directory 17
  listing all in federated database 57
  listing controlled files 57
  location of files and processes 20
  location properties 50
  lock server 17
  merging with existing partitions 55
  moving resources 59
  Objectivity/DB system resources 17
  offline 27
  offline status 27
  online 27
  private 27
  properties
    about 50
    changing 59
    listing 57
    setting 58
  purging 37, 63, 87
  quorum 26, 46
  system name 50
  system-database file 17, 50

tasks 49
tie-breaker
about 43
getting name 72
removing 73
setting 72
troubleshooting and recovery 53
unreplicated databases
controlling 15
transferring control 62
usage guidelines 15, 25
weight
assigning 26
changing 60

B

backup and restore 52, 53
boot autonomous partition 19
boot file
autonomous partitions 17, 19, 50
federated database 16
length of filename 84
bootFilePath 19

C

container files and replication 69, 86
containment
logical storage hierarchy 15, 32
synonym for control 15
control
compared with logical containment 15
databases 15
transferring 62, 71
Customer Support 9

D

data replication
about 30
AMS requirements 29
autonomous partitions 31
data integrity 37
database images 31
external containers and 69, 86
hot-failover configuration 42
quorum of database images 34, 35
synchronizing images 74
tie-breaker partition 43
two-machine handler function 44
database
accessibility
about 20
when replicated 33
availability when replicated 34
back up and restore 52, 53
control 15
setting at creation 62
transferring 62
images, *see* database images
logical entity represented by images 32
organized by partitions 15
read-only 71
database images
about 31
accessibility
about 33
checking 74
control
about 31
transferring 71
creating 62, 68, 85
deleting 73, 81
importing 55
listing all in federated database 67
listing properties 68
location 32
making read-only 71
moving files 70
moving to another partition 71
nonquorum 36
physical representation of logical database 32
properties
about 66
changing 69
setting 69
pseudo images 43
quorum, *see* quorum of database images

read image 34, 37
restoring into service 45, 74
stale data 36
status 88
synchronizing 45, 74
tie-breaker partitions
getting name 72
removing 73
setting 72
weight
about 35
changing 70
strategies for assigning 38

distributed files 19
DRO 7

E

end-user site deployment 54
external containers and replication 69, 86

F

fault tolerance
data replication 30
improving 21
federated database
access to global information 26
boot file 16
distributed 19
global catalog 16
guidelines for partitioning 25
initial autonomous partition 14, 18
journal directory 16
lock server 16
Objectivity/DB system resources 16
partitioned
about 14
back up and restore 52
exporting and importing 55
installing 54
recovery 53
schema 16
system-database file 16
unpartitioned 14
files
boot file
autonomous partitions 17, 19, 50
federated database 16
length of filename 84
distributed 19
location 19
permissions 21, 34
system-database file
autonomous partitions 17, 50
federated database 16
FTO 7

G

global catalog
about 16
configuring with import 55

H

HA abbreviation 8
hot-failover configuration 42

I

images, *see* **database images**

J

journal directory
autonomous partitions 17
federated database 16
file location 19
host connections 20
partition planning 58

L

location of files and processes 19
lock server
autonomous partitions 17
federated database 16
logical storage hierarchy 15

N

nonquorum images 36
nonquorum reading 34

O

Objectivity/Assist (Assist) 56
Objectivity/DB system resources 16
  decentralized by partitions 21
  distribution on network 19
  effect on database access 20
  partitioned federated database 17
  unpartitioned federated database 16
Objectivity/DRO 7
Objectivity/FTO 7
ODMG abbreviation 8
oochange 51, 52, 57, 59, 60
oochangedb 51, 52, 62, 68, 70, 71, 72
ooddlx 51
oodeleteap 51, 63, 79
oodeletedbimage 73, 81
oodumpcatalog 52, 57, 67, 72
ooexportcatalog 52
ooexportdata 52
ooexportfd 52
ooexportschema 52
ooimport 51, 55
oonewap 51, 58, 82
oonewdb 62
oonewdbimage 69, 72, 85
oopurgeaps 63, 87
ooreplstat 74, 88
oorestore 53
ooschemaupgrade 51
oosupportinfo 52
options, tool 77

P

partitioned federated database, *see* **federated database, partitioned**
partitions, *see* **autonomous partitions**
properties
  autonomous partition 50
  database image 66
pseudo images, *see* **tie-breaker partitions**

Q

quorum of autonomous partitions 26, 46
quorum of database images
  about 36
  access 34
  calculation
    about 35
    automatic 36
    requesting 74, 88
    tie-breaker partition 43
    two-machine handler function 44
  data integrity ensured by 37
  hot-failover configuration 42
  negotiation 35
  reading without 34
  weight assignment 38
  working with 35

R

read image 34, 37
read-only database 71
recovery 53

S

schema 16
servers
  AMS, autonomous partitions 17
  lock server
    autonomous partitions 17
    federated database 16
synchronizing database images 45
system-database file
  autonomous partitions 17, 50
  federated database 16

T

tie-breaker partitions
about 43
getting name 72
removing 73
setting 72
tools
abbreviations 77
about 77
argument 77
Assist 56
oochange 51, 52, 57, 59, 60
oochangedb 51, 52, 62, 68, 70, 71, 72
ooddlx 51
oodeleteap 51, 63, 79
oodeletedbimage 73, 81
oodumpcatalog 52, 57, 67, 72
ooexportcatalog 52
ooexportdata 52
ooexportfd 52
ooexportschema 52
ooimport 51, 55
oonewap 51, 58, 82
oonewdb 62
oonewdbimage 69, 72, 85
oopurgeaps 63, 87
ooreplstat 74, 88
oorestore 53
ooschemaupgrade 51
oosupportinfo 52
options and arguments 77
return status 78
troubleshooting 53
two-machine handler function 44
typographical conventions 8

W

weight
autonomous partitions
assigning 26
changing 60
database images
about 35
changing 70
strategies for assigning 38