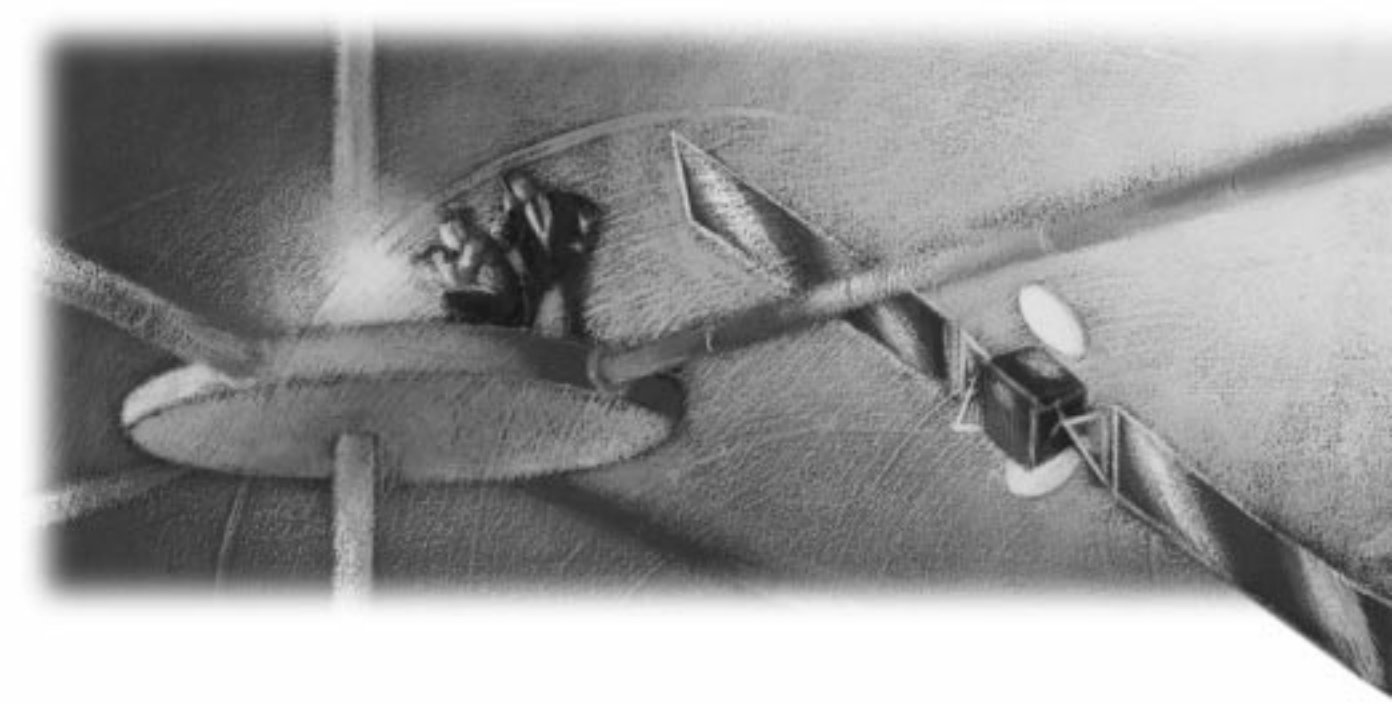

Objectivity/DB Administration

Release 6.0

Objectivity/DB Administration

Part Number: 60-DBA-0

Release 6.0, October 20, 2000

Contents

About This Book 11

Audience 11

Organization 11

Conventions and Abbreviations 12

Getting Help 13

Part 1 GUIDE

Chapter 1 Objectivity/DB Basics 17

Objectivity / DB System 17

System Database File 18

Database Files 18

Autonomous Partitions and Replicated Databases 19

Journal Files 19

Boot File 20

Lock Server 20

Application Processes 21

Pages and the Objectivity / DB Cache 21

Automatic Recovery 22

Distributed Objectivity / DB Systems 23

Administration Interface and Tools 24

Overview of Administration Tools 24

Chapter 2 Specifying Objectivity/DB Files 27

Filenames 27
- System-Database Files 27
- Database Files 28
- Journal Files 28
- Boot Files 29

File and Directory Access Permissions 29

Specifying Remote and Local Files 29
- Host and Path Formats 30
- Preserving Spaces in Pathnames 32
- Filename Case Sensitivity 33
- Setting a Boot File Environment Variable 33

Chapter 3 Federated Database Tasks 35

About Federated Databases 36
- Partitioned Federated Databases 36

Getting Federated Database Information 37
- Listing Current Attribute Values 37
- Listing All Associated Files 37
- Determining the File Type 38
- Summary of Tools That Display Attributes 38

Creating a Federated Database 38
- Examples 40

Copying a Federated Database 42

Changing Federated-Database Attributes 42
- Moving a Federated Database 43

Deleting a Federated Database 43

Dumping and Loading Federated-Database Objects 44
- Dumping Objects 44
- Loading Objects 44

Tidying a Federated Database 46
- Background 46
- How ootidy Works 46
- Guidelines for Using ootidy 47

Troubleshooting Access 47

Getting Transaction Information 48

Referencing Objects in a Federated Database 48
Estimating Disk Space Requirements 50
Estimating Initial Requirements 50
Estimating Maximum Federated Database Size 50

Chapter 4 Browsing Objects and Types 53

Information You Can Browse 54
Data Browser 54
Type Browser 55
Query Browser 56
Opening Browsers on Windows 57
Starting and Using oobrowse 57
Quitting oobrowse 57
Opening Browsers on UNIX 58
Starting and Using ootoolmgr 58
Quitting ootoolmgr 58

Chapter 5 Debugging a Federated Database 59

Inspecting and Editing a Federated Database 59
Starting oodebug as a Separate Process 60
Changing oodebug Modes 60
Performing Transactions With oodebug 61
Terminating oodebug 61
Running oodebug in a C++ Debugger 62
Terminating oodebug Within a Debugger 63
Using ooprint in a C++ Debugger 63

Chapter 6 Database Tasks 65

About Databases 66
Database Identifier Formats 66
Read-Only and Read-Write Databases 67
Replicated Databases 67
Getting Database Information 68
Getting a System Name or Database Identifier 68
Getting a Database's File Host and Path 68
Getting a Database's Page Size 68
Getting a List of Read-Only Databases 68

Creating a Database 69
Moving a Database File 70
Copying a Database File 70
Attaching a Database to a Federated Database 71
Moving a Database Between Federated Databases 71
Duplicating a Database Within a Federated Database 72
Guidelines for Attaching a Database 73
Consequences of Changing a Database Identifier 73
Attaching Multiple Databases 73
Changing Database Attributes 74
Changing the System Name or Database Identifier 74
Deleting a Database 75
Setting File Permissions on a Database 75
Tidying a Database 75
Troubleshooting Access Problems 76

Chapter 7 Using a Lock Server 77

About Lock Servers 77
Locks 78
Lock-Server Host 78
Types of Lock Server 79
Lock Servers on the Network 80
Required File and Directory 80
Deciding Whether to Use a Lock Server 80
Checking Whether a Lock Server is Running 81
Starting a Lock Server 81
Standard Lock Server 81
In-Process Lock Server 82
Stopping a Lock Server 83
Standard Lock Server 83
In-Process Lock Server 84
Changing Lock-Server Hosts 84
Listing Current Locks 85
Changing the TCP/IP Port for the Lock Server 85
Troubleshooting Problems With the Lock Server 87

Chapter 8 Advanced Multithreaded Server 91

About AMS 91
Deciding Whether to Use AMS 92
Comparing AMS to NFS 92
Guidelines for Choosing AMS 92
Checking Whether AMS is Running 93
Starting AMS 93
Stopping AMS 94
Setting AMS Usage in an Application 94
Changing the TCP / IP Port for AMS 94
Troubleshooting Problems With AMS 96

Chapter 9 Backup and Restore 97

About Backup and Restore 97
Backup Events and Backup Sets 98
Backup Medium and Backup Volumes 98
Backup Levels 99
Point of Restore 102
Full Restore 102
Backup Diary 102
User Access During Backup and Restore 102
Developing a Backup Strategy 103
Estimating the Disk Space Required for Backups 103
Defining a Backup Schedule 104
Backing Up Data 107
Creating a Backup Set 107
Performing a Backup 107
Obtaining a Federated Database's Backup History 109
Deleting a Backup Set 109
Restoring From a Backup 109
Restoring Files to Their Original Locations 110
Restoring Files to a Single New Location 111
Restoring Files to Multiple New Locations 112
Processing Backup Volumes 114
Processing Backup Volumes During a Backup 114
Processing Backup Volumes During a Restore 115

Backing Up to and Restoring From Tape 115
Configuring ootapebackup and ootaperestore 116
Backing Up to Tape 116
Restoring From Tape 117

Chapter 10 Automatic and Manual Recovery 119

About Recovery 119
Automatic Recovery From Application Failures 120
Automatic Recovery From Client-Host Failures 121
Windows Hosts 121
UNIX Hosts 122
Automatic Recovery From Lock-Server Failures 122
Performing Recovery at Lock-Server Startup 123
Performing Recovery When Locks are Requested 123
Access Required by the Lock Server 124
Setting Up Recovery in Mixed Environments 124
Performing Manual Recovery 125
Manual Recovery From Application Failures 126
Manual Recovery From Client-Host Failures 126
Manual Recovery From Lock-Server Host Failures 127
Manual Recovery From oocleanup Failures 127

Chapter 11 Working With Distributed Databases 129

Elements of a Distributed Environment 129
Using Windows Hosts 130
Windows Data-Server Hosts 130
Windows Client Hosts 131
Lock-Server Hosts 131
Boot-File Location 132
Mixed Environments: Summary 132

Chapter 12 Deploying to End Users 133

Building C++ Applications for End Users 133
Distributing Objectivity Executables 134
Executables You May Distribute 134
Executables You May Not Distribute 134

Distributing Libraries (Windows) 135
For Deployed Applications 135
For Redistributed Objectivity Executables 135
Distributing Libraries (UNIX) 136
For Deployed Applications 136
For Redistributed Objectivity Executables 136
Setting Up the End-User Site 137
Hardware Requirements 137
Software Requirements 137
Objectivity/DB Setup (Windows) 137
Objectivity/DB Setup (UNIX) 137
Installing a Federated Database 138

Part 2 REFERENCE

Chapter 13 Tools 141

Chapter 14 oodebug Commands 199

Appendix A Running Objectivity Servers on Windows 215
Starting and Stopping an Objectivity Server 215
Configuring an Objectivity Server 216
Specifying a Service's Logon Account 216
Uninstalling and Reinstalling an Objectivity Server 217

Index 219

About This Book

This book, *Objectivity/DB Administration*, describes how to administer Objectivity / DB in development and end-user environments. It discusses how to ensure optimum system performance and maintain the smooth, ongoing operation of Objectivity / DB.

Objectivity / DB provides tools and programming interfaces to help perform administration tasks. The tasks and tools are similar on all platforms; minor differences in usage, behavior, and naming that exist between different platforms are noted. The tools are described in this book; programming interfaces are described in the Objectivity / C++, Objectivity for Java, and Objectivity / Smalltalk documentation.

Audience

This book is intended for administrators who set up and maintain Objectivity / DB federated databases in either single- or mixed-platform environments.

Organization

- Part 1 is a guide to administering Objectivity / DB. Chapter 1 describes Objectivity / DB files and processes and gives an overview of the administration tools. Subsequent chapters describe the tasks involved in maintaining Objectivity / DB.
- Part 2 contains reference descriptions of the Objectivity / DB administration tools.
- Appendix A provides protocols for using the Objectivity Network Services tool to run Objectivity servers on Windows platforms.

Conventions and Abbreviations

Navigation

Table of contents entries, index entries, cross-references, and underlined text are hypertext links.

Typographical Conventions

`oobackup`	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
(*FTO*)	Feature of the Objectivity/DB Fault Tolerant Option product
(*DRO*)	Feature of the Objectivity/DB Data Replication Option product
(*IPLS*)	Feature of the Objectivity/DB In-Process Lock Server Option 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

We have done our best to make sure all the information you need to install and operate Objectivity products is provided in the product documentation. However, we also realize problems requiring special attention sometimes occur.

Technical Support Web Site

You can find answers to frequently asked questions, supported platforms, known bugs, and bug fixes on the Objectivity Technical Support web site. Send electronic mail or call Objectivity Customer Support to gain access to the site.

How to Reach Objectivity Customer Support

You can contact Objectivity Customer Support by:

- **Telephone:** Call 1.650.254.7100 *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 1.650.254.7171.
- **Electronic Mail:** Send electronic mail to *help@objectivity.com*.

Before You Call

If you need help from Customer Support, please have the following information ready before you contact Objectivity:

- Your name, company name, address, telephone number, fax number, and email address
- Description of your workstation environment, including the type of workstation, its operating system version, compiler or interpreter, and windowing environment
- Information about the Objectivity product you are using, including the version of the Objectivity / DB libraries
- Detailed description of the problem you have encountered

Part 1 GUIDE

1

Objectivity/DB Basics

This chapter describes the basic elements in an Objectivity / DB system and provides an overview of the Objectivity / DB tools you use to perform administration tasks.

Objectivity/DB System

An Objectivity / DB system consists of multiple processes and files that can be distributed across multiple host machines on a network (see Figure 1-1).

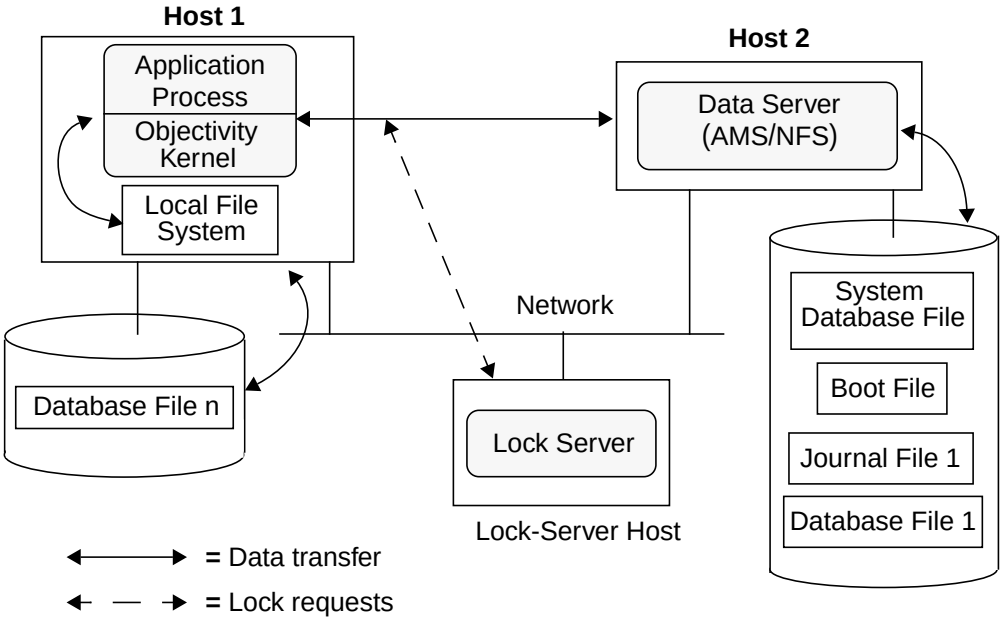

Figure 1-1 Example Objectivity/DB Configuration

System Database File

The *federated database* is the highest level in the Objectivity / DB logical storage hierarchy. Physically, each federated database exists as a file containing a *system database*, which stores the *schema* for the federated database, as well as a *catalog* of the additional databases that make up the federation. Each federated database is assigned a unique integer that identifies it to Objectivity / DB processes such as the lock server.

Every federated database has a *boot file*. Database applications and tools refer to a federated database using its boot-file name. The simple name of a boot file is sometimes called the federated database's *system name*.

As an administrator, you may need to change a federated database's attributes or location, or you may need to determine a federated database's memory and disk requirements; see Chapter 2, "Specifying Objectivity / DB Files," and Chapter 3, "Federated Database Tasks".

Schemas

A *schema* is a physical representation of the Objectivity / DB data model. It is a collection of type definitions and association definitions that allows a database to store and manage objects. The schema is stored in the system database of the federated database.

Schema creation depends on the programming interface:

- For Objectivity / C++, a schema is created by defining classes using the Objectivity / C++ Data Definition Language (DDL) and processing them with the DDL processor.
- For Objectivity for Java and Objectivity / Smalltalk, a schema is created automatically when persistent objects are created.
- For Objectivity / SQL++, a schema is created automatically when tables are created.

Database Files

A *database* is the second highest level in the Objectivity / DB logical storage hierarchy and is where your application's persistent data is stored. A database consists of one or more logical structures called *containers*, which in turn contain fundamental units of persistent data called *basic objects*. Containers determine the physical clustering of basic objects in memory and on disk. Each database contains a default container for any basic object not assigned to a user-created container.

A database is physically represented by a *database file*, which contains the database and all of its containers and basic objects. Each database is attached to exactly one federated database and is listed in that federated database's catalog. Database files may reside on different machines than the file for the federated database to which they are attached. In addition to having a physical filename, each database also has a user-specified *system name*, which is its logical name within the federated database.

As an administrator, you may need to change a database's attributes or location; see Chapter 2, "Specifying Objectivity / DB Files," and Chapter 6, "Database Tasks".

Autonomous Partitions and Replicated Databases

An *autonomous partition* is a mechanism for dividing a federated database into independent pieces, so that each autonomous partition is self-sufficient in case a network or system failure occurs in another partition. Although data physically resides in database files, each autonomous partition *controls* access to the databases and containers that belong to it. This capability is available through the Objectivity / DB Fault Tolerant Option product (Objectivity / FTO).

An autonomous partition is physically represented by a *system-database file*. This file contains a copy of the federation's system database, which stores the schema and a global catalog of all autonomous partitions and databases. Every autonomous partition has a *system name* and a *boot file*. When you create a federated database, it implicitly has a single initial autonomous partition.

Autonomous partitions are the foundation for using the Objectivity / DB Data Replication Option (Objectivity / DRO) product to create and manage multiple copies of a database, called *database images*. Database applications automatically access the closest image, reducing network traffic; if an image becomes unavailable due to network or system problems, work may continue with an available image.

As an administrator, you may need to perform the partition- and image-specific tasks described in the Objectivity / FTO and Objectivity / DRO book, and you may need to consider partitions and images for the administrative tasks in this book.

Journal Files

Whenever a transaction starts, it records update information in one or more *journal files*, which are used to return the federated database to its previously committed state if the transaction is aborted or terminated abnormally. Journal files enable Objectivity / DB to roll back changes made by incomplete transactions; they differ from relational-database transaction logs, which usually contain before and after images of the data and allow forward recovery up to the last commit. (Objectivity / DB supports forward recovery through its backup and restore tools.)

Journal files are written in a federated database's *journal directory*. In an Objectivity / FTO environment, each autonomous partition has a unique journal directory. A transaction that updates data in multiple partitions writes to multiple journal files, one in each journal directory. As an administrator, you may need to specify or change journal directories; see Chapter 3, "Federated Database Tasks".

Every single-threaded application normally creates one journal file per autonomous partition that will be updated by the transaction. This journal file is automatically reinitialized at the end of each committed transaction and is reused by the application's next transaction. Multithreaded applications normally create multiple journal files per autonomous partition, one for each thread that executes a separate series of transactions.

Objectivity / DB deletes journal files automatically when a process completes or when automatic recovery is performed after a process or system failure. In some cases, you should perform manual recovery when a journal file persists after a process has ended; see Chapter 10, "Automatic and Manual Recovery".

WARNING Never delete a journal file, because data corruption may result.

Boot File

A *boot file* contains information used by an application or tool to locate and open a federated database. A boot file is created automatically when you create a new federated database and contains entries specifying the various federated-database attributes. As an administrator, you may need to view or change these attributes; see Chapter 3, "Federated Database Tasks". You use Objectivity / DB tools to change a boot file; you never edit a boot file directly.

Most administration tools require that you refer to a federated database by specifying the path to its boot file. You can do this explicitly, or you can set the pathname in an environment variable.

In Objectivity / FTO environments, every autonomous partition has its own boot file, which is created automatically when the autonomous partition is created. Most tools allow you to refer to a partitioned federated database by specifying the boot file from any autonomous partition.

Lock Server

Objectivity / DB provides simultaneous multiuser access to data. To ensure that data remains consistent, database access is controlled through locks administered by a *lock server*. In a standard configuration, the lock server runs as a separate process from the applications that consult it. An alternative configuration is for a particular application to run the lock server internally (within the same process).

Such an application must use a separately purchased option to Objectivity / DB, namely, Objectivity / DB In-Process Lock Server Option (Objectivity / IPLS). In either configuration, the lock server can run on any of the workstations in a network.

A lock server can service multiple federated databases; each federated database (or each autonomous partition in a federated database) must be serviced by a single lock server.

For more information on lock servers, see Chapter 7, “Using a Lock Server”.

Application Processes

Objectivity / DB applications are C++, Java, or Smalltalk applications that store persistent data in Objectivity / DB databases. Such applications invoke database operations from the Objectivity / DB runtime library, or *kernel*. When your application starts a transaction to access persistent data, the Objectivity / DB kernel:

- Locates and opens the appropriate federated-database and database files.
- Requests an appropriate lock from the lock server.
- Reads the requested object from disk into memory.
- Writes updates to disk if requested.
- Records the transaction in the appropriate journal file.

Because the kernel runs in the same process as your application, there is no need for a separate database server process.

Pages and the Objectivity/DB Cache

Objectivity / DB stores persistent objects in *storage pages*. A storage page is the minimum unit of transfer to and from disk and across networks. The size of a page is configurable for each federated database and is set when the federated database is created; once set, the page size cannot be changed. See `oonewfd` (page 185) for information about setting the storage-page size for a federated database.

A federated database’s storage pages are usually sized so that one or more typical persistent objects will fit within a single storage page; each such *small object* occupies a *slot* on the page. A *large object* spans multiple storage pages, where one of these pages (the *header page*) has a slot containing overhead information and links to other pages containing data. Large-object header pages and the storage pages for small objects are sometimes called *logical pages*; these pages (and their slots) are assigned numbers that help identify persistent objects (see “Referencing Objects in a Federated Database” on page 48).

When Objectivity / DB reads a persistent object from disk, it places the storage page(s) containing the object into the *Objectivity/DB cache.* The cache is memory

allocated and managed by Objectivity / DB to provide fast access to persistent objects. The cache consists of *buffer pages*, which are the same size as the storage pages in the federated database. Figure 1-2 shows the role of the cache in the virtual address space of an application process.

In a multithreaded application, each thread that executes a separate series of transactions has an independent Objectivity / DB cache in the process's address space.

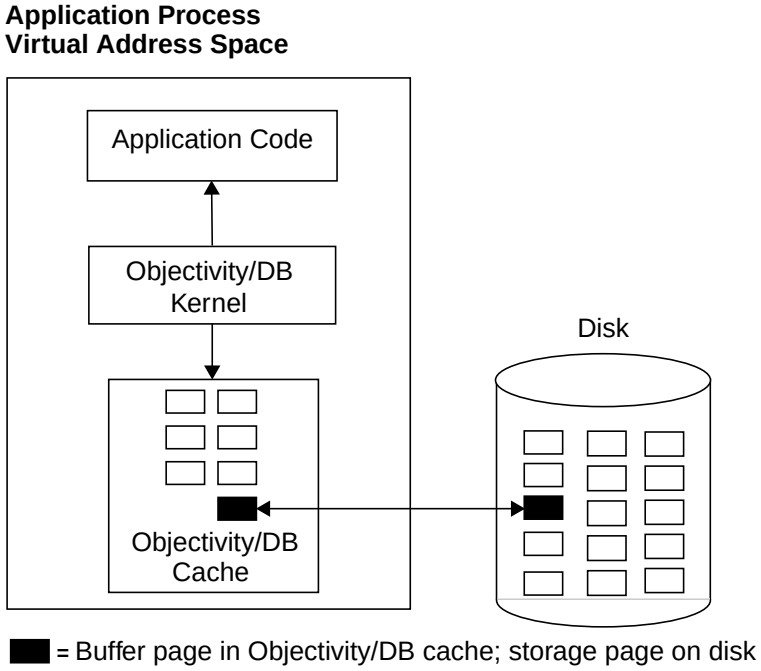

Figure 1-2 Objectivity/DB Cache in a Single-Threaded Process

Automatic Recovery

Objectivity / DB provides automatic recovery from most failures. Specifically, you can set up Objectivity / DB to automatically roll back incomplete transactions resulting from application-process failures, client-host failures, lock-server failures, and lock-server host failures. As an administrator, you need to know how to set up automatic recovery and how to perform manual recovery when necessary; see Chapter 10, "Automatic and Manual Recovery".

Distributed Objectivity/DB Systems

You can distribute an Objectivity / DB system across a network by placing the various Objectivity / DB files and processes on different hosts. Some important nodes in a distributed Objectivity / DB system are:

Client host	Network node that runs an Objectivity / DB application; sometimes called an application host
Data-server host	Network node that provides data storage; location of federated database, database, partition, and journal files
Lock-server host	Network node that runs an Objectivity / DB lock server

In a distributed Objectivity / DB system, an application running on a client host may request data that is either local (the data server and client are the same host) or remote (the data-server host is different than the client host). Note that a distributed system may, but need not, be partitioned through Objectivity / FTO.

In general, when servicing a request for data, the Objectivity / DB kernel finds:

- *Local databases* by directly accessing the local (client) file system
- *Remote databases* by contacting the specified host and contacting whichever data-server software is available:
  - Objectivity / DB's Advanced Multithreaded Server (AMS)
  - Network File System (NFS) server

However:

- If you use databases replicated with Objectivity / DRO, the kernel contacts only AMS to find those databases. Even if a replicated database is local to the client host, the kernel contacts AMS instead of the local file system.
- If all client and data-server hosts are Windows workstations, the kernel can contact Windows Network to find remote database, provided that all hosts use a common set of Universal Naming Convention (UNC) share names.

Accessing a database or federated database across a network may add significant amounts of I / O time and CPU time due to network overhead and contention. An application's speed is likely to improve significantly if it accesses local databases.

As an administrator, you may need to set up and run AMS, or help to choose between using AMS or another network file server (such as NFS) for remote data access; see Chapter 8, "Advanced Multithreaded Server". You may also need to decide where to locate the various elements of an Objectivity / DB system in a distributed, heterogeneous environment; see Chapter 11, "Working With Distributed Databases". You may also need to know how to reference remote and local files; see Chapter 2, "Specifying Objectivity / DB Files".

Administration Interface and Tools

Objectivity / DB provides a set of tools that support both administration and application development. Each Objectivity / DB tool returns status information using the conventions of the host operating system. For reference information on these tools, see Chapter 13, “Tools”.

On UNIX, you invoke Objectivity / DB tools from a command line or within a shell script.

On Windows:

- You can invoke tools either from a command prompt or from the ObjyTool graphical interface.
- You invoke Objectivity servers (such as the lock server and AMS) from the graphical interface of the Objectivity Network Services tool.

You can create your own tools for administration tasks using the Objectivity / C++, Objectivity for Java, or Objectivity / Smalltalk interfaces. For more information, see the documentation for these programming interfaces.

Overview of Administration Tools

Table 1-1 gives an overview of the kinds of tasks that you can perform using Objectivity / DB administration tools.

Table 1-1: Overview of Administration Tools

Creating and Modifying Federated Databases	`oochange`	Displays or changes the attributes of a federated database or autonomous partition.
	`oocopyfd`	Copies a federated database.
	`oodeletefd`	Deletes a federated database.
	`ooinstallfd`	Installs a remote federated database.
	`oonewfd`	Creates a federated database.
	`ooschemadump`	Writes a federated database’s evolved schema to a file.
	`ooschemaupgrade`	Applies an evolved schema to a federated database.

Table 1-1: Overview of Administration Tools (Continued)

Creating and Modifying Databases	ooattachdb	Attaches a database to a federated database.
	oochangedb	Displays or changes the attributes of a database or database image.
	oocopydb	Copies a database.
	oodeletedb	Deletes a database from a federated database.
	oonewdb	Creates a new database.
Getting Information	oobrowse	Browses objects and types, and makes queries (on Windows).
	oochange	Displays or changes the attributes of a federated database or autonomous partition.
	oochangedb	Displays or changes the attributes of a database or database image.
	oodumpcatalog	Lists all the files in a federated database.
	oofile	Displays information about a database or federated database.
	oolockmon	Lists all processes and locks currently managed by a lock server.
	oolistwait	Lists waiting transactions.
	ootoolmgr	Browses objects and types, and makes queries (on UNIX).
Backup and Restore	oobackup	Archives a federated database.
	oocreateset	Creates a backup set for a federated database.
	oodeleteset	Deletes a backup set.
	ooqueryset	Queries a federated database for existing backup sets.
	oorestore	Restores an archived federated database.

Table 1-1: Overview of Administration Tools (Continued)

Managing Objectivity Servers	Objectivity Network Services	Starts Objectivity servers (on Windows).
	`oocheckams`	Checks whether AMS is running on a system.
	`oocheckls`	Checks whether a lock server is running on a system.
	`ookillls`	Kills a lock server.
	`oolockmon`	Lists all processes and locks currently managed by a lock server.
	`oolockserver`	Starts a lock-server process for a federated database.
	`oostartams`	Starts AMS.
	`oostopams`	Terminates AMS.
Maintenance and Recovery	`oocleanup`	Rolls back transactions that have terminated abnormally.
	`ootidy`	Consolidates a fragmented federated database or database.
	`oogc`	Deletes unreferenced objects in a federated database (Objectivity for Java and Objectivity/Smalltalk only).
Miscellaneous	ObjyTool	Starts administration tools (on Windows).
	`ooconfig`	Creates a DDL processor for your compiler (Objectivity/DDL on UNIX only).
	`oodebug`	Provides commands for inspecting and editing a federated database.
	`oodump`	Creates a text file representing a federated database.
	`ooload`	Creates objects from an ASCII text file.

2

Specifying Objectivity/DB Files

Many administrative tasks involve creating or referencing Objectivity / DB files (system-database, database, journal, and boot files). This chapter provides important information about:

- Filenames
- Access permissions
- Pathnames for local and remote files

Filenames

System-Database Files

Federated Database

You specify the name of a federated database's system-database file when you create it or rename it (see Chapter 3, "Federated Database Tasks"). By convention, you construct the name of the system-database file from the federated database's system name plus an extension, typically `.fdb`, `.FDB`, or `.FDDB`:

`fdSysName``.FDB`

A federated database's system name is the name of the associated boot file.

Autonomous Partition

(*FTO*) You specify the system name of an autonomous partition when you create the partition (see the Objectivity / FTO and Objectivity / DRO book). At this time, you also specify the name of the partition's system-database file. By convention, you construct this filename from the partition's system name plus an extension, typically `.AP`:

`apSysName``.AP`

Because the first autonomous partition is created automatically when you create the federated database, the partition's system name and filename are those of the federated database.

Database Files

You specify the system name of a database when you create it or rename it (see Chapter 6, "Database Tasks"). You can optionally specify the name of the physical database file as well, or let it be automatically generated. Automatically generated database filenames are of the form:

dbSysName.*fdSysName*.DB

where *dbSysName* is the system name of the database and *fdSysName* is the system name of the federated database.

Journal Files

You specify the location (host and directory) for journal files when you create a federated database or change its attributes.

(*FTO*) You specify a unique journal host and directory for an autonomous partition when you create the partition (see the Objectivity / FTO and Objectivity / DRO book) or change its attributes.

The names of journal files are always automatically generated. They are of the form:

oo_*fdSysName_apId_hostName_processId_userName_contextID*.JNL

where

fdSysName	System name of the federated database.
apId	Autonomous-partition identifier. For single-partition federated databases, this identifier is 65535.
hostName	Name of the host on which the transaction is running.
processId	Process ID (pid) of the process in which the transaction occurs.
userName	Name of the user who started the process.
contextId	Context ID of the thread in which the transaction occurs.

For example: oo_testfd_6_machine12_12345_joe_1.JNL

Boot Files

You specify the name of a boot file when you create a federated database. A boot file does not require a filename extension. The simple name of the boot file is the system name for the federated database.

(*FTO*) When you create an autonomous partition, you may specify the name of its boot file, but you need not do so. If you do not specify a filename, the default filename is derived by appending the `.boot` extension to the system name of the autonomous partition. (See the Objectivity / FTO and Objectivity / DRO book for information about creating an autonomous partition.)

File and Directory Access Permissions

Access permissions are set on federated-database files, autonomous-partition files, database files, journal files, and boot files when they are created by Objectivity / DB tools or applications. You should ensure that every user account that runs such tools and applications sets permissions as appropriate to give other accounts the access they require. In particular, the accounts that run AMS or the lock server must be able to read and update all Objectivity / DB files. When an application uses AMS, any files created by the application are owned by the user account under which AMS was started. You typically create a special account for running AMS and always start AMS under that account.

Specifying Remote and Local Files

Many administration tools and programming interfaces require that you specify the names of Objectivity / DB files:

- Administration tools that create or relocate system-database, database, or journal files allow you to specify the desired locations for these files. The specified locations are generally stored in boot files or catalogs.
- Administration tools that operate on a federated database require you to specify a boot file for that federated database.
- In programming interfaces, functions that open a federated database require that you specify a boot file for that federated database.

Host and Path Formats

When your Objectivity / DB system is distributed among multiple hosts, you may need to specify remote files (files located on remote data-server hosts) in addition to local files (files located on host where you are running the tool or application). Objectivity / DB tools and functions use the following conventions for obtaining host and pathname information:

- Boot-file names are usually specified through a single argument or parameter value. Each such name can be specified in *host format*, where you explicitly specify the boot file host separated from the pathname by two colons:

  `host::path`

- Names of system-database files, database files, and journal files are usually specified as a pair of options specifying `host` and `path` values. This information is presented in host format when you display the contents of the boot file or catalog in which it is stored.

The following subsections describe how to specify `host` and `path` values (in host-format names or as separate options) to reference files that reside on the different kinds of data-server hosts.

Files on the Local Host

To designate a file that resides on the local host (the host where you are running the tool or application), you can omit the `host` value and specify a relative or absolute pathname in the format accepted by the local operating system. When you specify a local path, omitting the `host` value is the same as specifying the name of the local host. Note, however, that you *must* include a `host` value (even for a local host) when specifying a boot file path to the `oolockserver` tool (page 182).

Names that are stored in a boot file or catalog must be usable by every application that will consult that boot file or catalog, including applications running on remote client hosts. Take this into account when specifying files on Windows hosts, which allow files to have both local names (for example, `c:\project\myFD`) and network-visible names (for example, `\\mach33\c\project\myFD`). To make a local file visible to remote Windows clients, you must specify `host` and `path` values as described in “Files on Windows Network Data-Server Hosts” on page 32.

Files on AMS Data-Server Hosts

To designate a file on a remote data-server host running AMS, you specify host and pathnames as follows:

host The data-server host's TCP/IP network node name.

path The fully qualified path of the file on the specified host. The path is passed to AMS on that host. Because AMS passes the path to the specified host's file system, you use pathnames that are local to that host—for example, on Windows:
`c:\project\myFD`

When specifying a file on a Windows host running AMS, you must use the locally understood pathname; do not use a UNC share name.

Files on AMS data-server hosts are available to database applications running on Windows and UNIX client hosts.

Files on NFS Data-Server Hosts

To designate a file on a remote data-server host running NFS, you specify host and pathnames as follows:

host The data-server host's TCP/IP network node name.

path The fully qualified path of the file on the specified host. The path is passed to the NFS server on that host. The path must start with the name of the NFS-exported file system that contains the file—for example, on a UNIX platform where /usr is exported:
`/usr/project1/myFD`
The name format may vary among NFS products on Windows hosts.

Alternatively, you can omit the *host* value and specify a locally understood NFS mount name for the file (for example, `/net/machine33/usr/project1/myFD`). When you specify an NFS mount name, Objectivity/DB uses the mount table to translate the mount name into a host-format name whose *host* is the remote host and whose *path* is the file's name on that host.

Files on NFS data-server hosts are available to database applications running on Windows and UNIX client hosts.

Files on Windows Network Data-Server Hosts

To designate a file on a remote Windows Network data-server host, where that file is shared through a common Universal Naming Convention (UNC) share name, you specify host and pathnames (from a Windows client host) as follows:

host The literal string `oo_local_host`. This string causes the tool or application to use the local *client host* operating system to resolve the *path*. If you specify any other string for *host*, it will be converted to `oo_local_host` automatically.

path The fully qualified network path to the file—that is, a UNC share name for the file such as `\\mach33\c\project\myFD`.

All Windows client hosts and data-server hosts must have the same UNC share name definitions (see also “Using Windows Hosts” on page 130).

When specifying names to be entered in a catalog (for example, system-database names, database names, or journal-directory names), you should not mix UNC share names with other host and path formats. If any such name is specified as a UNC share name, you must specify all names that way, even if you are running AMS or NFS in addition to Windows Network.

Files on Windows Network data-server hosts are available only to database applications running on Windows client hosts.

WARNING Do not use UNC share names (for example, when creating a federated database) if the resulting files will be accessed by applications running on a UNIX client host. You must use AMS or NFS (and the corresponding naming conventions) to make files available to applications on UNIX client hosts.

Preserving Spaces in Pathnames

Avoid using spaces in the names of files and directories. If you do specify names with spaces in them, the names will be double-quoted in the boot file by `oonewfd`, `oochange`, and `ooinstallfd`. For example:

```
oonewfd -fdfilehost mach33 -fdfilepath
   "/test dir/test.fdb" -lock mach33 test
```

results in a boot file entry for the `ooFDDBFileName` that looks like:

```
ooFDDBFileName="/test dir/test.fdb"
```

WARNING When Objectivity / DB reads the boot file, it will ignore spaces in strings that are not surrounded by double quotes, with unpredictable results.

Filename Case Sensitivity

Objectivity / DB is sensitive to case when verifying filenames for certain operations. Therefore, when specifying the name of a file to Objectivity / DB, you must be careful to use the case that was used when the file was created.

You must specify Objectivity / DB files using the original case even on Windows, which otherwise allows you to access files using any case combination. For example, Windows allows you to access a file created as `HELLO` using `HELLO`, `Hello`, or `hello`. However, if this is an Objectivity / DB file, you may only use `HELLO`.

Setting a Boot File Environment Variable

If you will need to reference the same boot file many times or from many tools, you can specify its path as the value of the `OO_FD_BOOT` environment variable. You can then invoke many Objectivity / DB tools without specifying the boot file path on the command line (some tools, such as `ooinstallfd`, require an explicit boot file path). You can specify the path as a locally understood path or using the host format *`host`*`::`*`path`*.

Windows

To set the boot file environment variable in Windows, use the `set` command. For example:

```
set OO_FD_BOOT=c:\john\etc\infoNet
```

UNIX

To set the boot file environment variable in UNIX, use the appropriate command for your shell. For example, in the C shell:

```
setenv OO_FD_BOOT sys22::/usr/john/etc/infoNet
```

3

Federated Database Tasks

A federated database is the highest level in the Objectivity / DB storage hierarchy. It is the unit of administrative control for a "federation" of associated databases and contains the data model, or schema, that describes all classes of objects stored in these databases. Administering a federated database involves physical and logical restructuring, maintenance, and troubleshooting that affects the entire set of associated databases.

This chapter describes:

- Background information about federated databases
- Getting information about a federated database
- Creating, copying, restructuring, and deleting a federated database
- Dumping and loading federated database objects
- Tidying a federated database
- Troubleshooting access
- Getting transaction information
- Referencing objects in a federated database
- Estimating disk space requirements

Tasks that involve changing the number and distribution of databases and database files within a federated database are discussed in Chapter 6, "Database Tasks".

You can write applications to perform many routine federated database administration tasks. See Objectivity's language-specific programming interface documentation for more information.

About Federated Databases

Physically, a federated database consists of two files:

- The system-database file. This file stores the schema for the federated database and contains a *catalog* of member database system names and locations.
- The boot file. This file contains the configuration information used by applications and tools to locate and open the federated database.

A federated database has a number of *attributes* that describe its physical and logical structure:

- The federated database identifier (sometimes called the reference number)—the unique positive integer that identifies the federated database to the lock server.
- The federated database system name—the unique name that identifies the federated database to Objectivity / DB. The system name is the same as the boot file name.
- The system-database file host and file path—the host system and full directory path (including the filename) of the federated database's system-database file.
- The journal directory host and path—the host system and directory path for the journal files, which store information used by Objectivity / DB to roll back incomplete transactions for all member databases.
- The lock-server host—the host system on which the lock server for the federated database is running.
- The storage page size—the size (in bytes) of the unit of transfer to and from memory and across the network for the entire federated database.
- The boot file host and path—the host system and directory path for the boot file of the federated database.

The values for these attributes are set when a federated database is created. However, you can modify various attributes later using Objectivity / DB administration tools to accommodate system or network changes or to improve application performance.

Partitioned Federated Databases

In an Objectivity / FTO environment, a federated database is created with an implicit initial autonomous partition. When you add a second partition, the initial partition becomes explicit, and:

- The federated database's attributes (system name, identifier, lock-server host, and so on) become the attributes of the initial autonomous partition.

- The system-database file and boot file become the system-database file and boot file of the initial autonomous partition.

The tasks in this chapter apply to entire federated databases, regardless of the number of partitions. The exception is listing or changing attribute values, which affects only one partition at a time. Most Objectivity / DB tools allow you to specify a partitioned federated database using the boot file from any partition.

Getting Federated Database Information

Objectivity / DB provides three tools that you can use to get information about a federated database. As summarized in Table 3-1, these tools provide the current values for some or all federated-database attributes, and are therefore useful for obtaining the values necessary to invoke other federated database administration tools. Each tool has other administrative uses in addition to providing the attribute values.

Listing Current Attribute Values

You use oochange (page 149) with a boot-file name to list the current values of federated-database attributes. (*FTO*) You include the `-ap` option to list the attribute values for a specific autonomous partition.

You can also inspect the boot file to view the current values for most attributes. However, you must never edit this file directly.

Listing All Associated Files

You use `oodumpcatalog` (page 171) with a boot-file name to list the full pathnames of all files associated with a federated database:

- The system-database file
- The boot file
- The journal directory
- All database files
- (*FTO*) All system-database files, boot files, and journal directories of the autonomous partitions in a partitioned federated database
- (*DRO*) All database image files

In addition, `oodumpcatalog` lists all identifiers and system names.

Determining the File Type

You use `oofile` (page 172) with the name of an Objectivity/DB file to determine the file type and other associated information about that file. When you invoke this tool with a system-database filename, it lists the file type (federated database) and the attributes shown in Table 3-1. In addition, this tool reports the architecture (platform and operating system) on which the file was created and indicates the Objectivity/DB release with which the database format is compatible. The `oofile` tool terminates with an error message if a file is missing or corrupted.

Summary of Tools That Display Attributes

Table 3-1 compares the federated-database or autonomous-partition attributes that are displayed by `oochange`, `oodumpcatalog`, and `oofile`.

Table 3-1: Attribute Values Displayed By Tools

Attribute	`oochange`	`oodumpcatalog`	`oofile`
Identifier	✓	✓	✓
System name	✓	✓	✓
File host and path	✓	✓	—
Boot file host and path	✓	✓	—
Journal directory host and path	✓	✓	✓
Lock-server host	✓	✓	✓
Page size	✓	—	✓

Creating a Federated Database

To create a federated database, you:

1. Plan how your Objectivity/DB system will be configured around the new federated database. In particular, you should identify the data-server hosts, client hosts, and lock-server host, and answer the following questions:
   - Will your Objectivity/DB system be on a single host or distributed across multiple hosts?
   - In a distributed system, which type of platform (Windows or UNIX) will you use for each host?

- In a distributed system, what data-server software will you run on each data-server host?

The answers to these questions determine the format you will use to specify file locations when you create the federated database. To help you answer these questions, see "Distributed Objectivity/DB Systems" on page 23 and Chapter 11, "Working With Distributed Databases".

2. Identify the specific locations of system-database file, journal directory, and boot file of the new federated database. 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 federated database.

3. Decide how you want to identify the new federated database:
   - Choose the boot-file name to be used. The simple name of this file will be used as the federated database's system name.
   - Optionally choose a numeric identifier for the federated database.
4. Choose the storage-page size for the new federated database. Normally, you use the disk's page size as the storage-page size, although you might want to adjust the storage-page size according to the estimated size of the average object to be stored.

   For information about choosing an optimal page size, see the chapter on performance in the documentation for your Objectivity programming interface.

5. Verify that a lock server is running on the intended lock-server host for the new federated database; start it, if necessary (see "Starting a Lock Server" on page 81).
6. Invoke `oonewfd` (page 185) with appropriate options. At a minimum, you must specify the system-database file path, the lock-server host, and a boot file name; the other attributes have default values.

   The naming format you use to specify the system-database file (and, optionally, the journal directory) depends on the data-server software you chose in step 1 for the relevant data-server hosts. For complete details, see "Host and Path Formats" on page 30 and Table 11-1 on page 132. The examples below show naming formats in some common scenarios.

7. For Objectivity/C++ applications, you must run `ooddlx` to load a schema into the newly created federated database; see the Objectivity/C++ Data Definition Language book. (The schema is loaded automatically for Objectivity for Java and Objectivity/Smalltalk applications.)

(*FTO*) A federated database is created with an implicit initial autonomous partition. When you add a second partition, the initial partition becomes explicit.

Examples

The following subsections give examples of creating a federated database on a Windows data-server host and on a UNIX data-server host.

Windows

Assume you want to create a federated database on a Windows data-server host called `machine33`. Because the new federated database is to be accessed by applications and tools running on Windows and UNIX client hosts, `machine33` is running AMS as its data-server software. Furthermore, assume that:

- The lock-server host for the new federated database is `machine95`.
- The journal directory is to be created in the same directory as the system-database file on `machine33`.
- The boot file is to be created in the current directory on the host on which `oonewfd` is executed.
- The name of the boot file, `projectFD`, will also be the system name for the new federated database.
- The chosen federated-database identifier is 10, and the chosen page size is the default.

Because the system-database file is to be created on a Windows host running AMS, you must specify the file's location using the host and path format described in "Files on AMS Data-Server Hosts" on page 31.

EXAMPLE After the lock server is started on `machine95`, the `oonewfd` command is entered as shown. Because AMS is the data-server software, `oonewfd` specifies the hostname (`machine33`) and a fully qualified pathname that is local to that host.

```
oonewfd -fdfilehost machine33 -fdfilepath
   c:\project1\data\develop.fdb -lockserverhost machine95
   -fdnumber 10 projectFD
```

Now consider a variant of the previous example. Assume you want to create a federated database on `machine33` with the same attributes as the previous example, but because *all* the client hosts are Windows, the chosen data-server software is Windows Network. The decision is made to refer to files using a common set of UNC share names. Consequently, the command shown below specifies the system-database file and the journal directory using the host and path format described in "Files on Windows Network Data-Server Hosts" on page 32.

EXAMPLE After the lock server is started on machine95, the oonewfd command is entered as shown. Because a UNC share name is specified, the hostname can be omitted (the host value is automatically set to the special string oo_local_host, as is required for resolving UNC names).

```
oonewfd -fdfilepath \\project1\data\develop.fdb
   -lockserverhost machine95 -fdnumber 10 projectFD
```

UNIX

Assume you want to create a federated database on a UNIX data-server host called machine55. The new federated database will be accessed by applications and tools running on Windows and UNIX client hosts, so machine33 is running AMS as its data-server software. Furthermore:

- The lock-server host for the new federated database is machine95.
- The journal directory is to be created in its own directory on machine33.
- The boot file is to be created in the current directory on the host on which oonewfd is executed.
- The name of the boot file, projectFD, will also be the system name for the new federated database.
- The chosen federated-database identifier is 10, and the chosen page size is 4096.

Because the system-database file and the journal directory are to be created on a UNIX host running AMS, the oonewfd command specifies their locations using the host and path format described in “Files on AMS Data-Server Hosts” on page 31.

EXAMPLE After the lock server is started on machine95, the oonewfd command is entered as shown. Because AMS is the data-server software, oonewfd specifies the hostname (machine55) and fully qualified pathnames that are local to that host.

```
oonewfd -fdfilehost machine55 -fdfilepath
   /project1/data/development.FDB -lockserverhost machine95
   -jnldirhost machine55 -jnldirpath /project1/journalfiles
   -fdnumber 10 -pagesize 4096 projectFD
```

The same command could be used if NFS is used as the data-server software, provided that /project1 is the name of an NFS-exported file system.

Copying a Federated Database

You can copy a federated database—for example, to prepare it for end users or for archiving purposes. To do so, you invoke oocopyfd (page 162) with options specifying a new federated-database identifier that is different from the current identifier, and the path to the directory that will hold the copied files. You can optionally choose a new federated-database system name and lock-server host.

(*FTO*) Before using this tool on a partitioned federated database, you must first delete the autonomous partitions.

NOTE If you only want to move the system-database file from one location to another in a networked file system, *without* making a copy, it is simpler to use oochange (page 149).

EXAMPLE The following UNIX command copies a federated database named myFd to directory /test on host system machine55, sets the federated-database identifier to 2, and changes the federated-database attributes to use the new lock server on host system machine56. After the federated database is copied, the lock server must be started on host system machine56.

```
oocopyfd -fdnumber 2 -host machine55 -dirpath /test
   -lockserverhost machine56 myFd

# rlogin to machine56 at this point
oolockserver
```

Changing Federated-Database Attributes

You can use oochange (page 149) to change the following federated-database attributes:

- The system-database file host and path (to move the federated database)
- The lock-server host.
- The journal-directory host and path. You must make sure that no other federated database shares the same journal directory.
- The federated-database identifier.
- The boot file path. The oochange tool does not delete the old boot file. If you change the boot file location, you should delete the old boot file using the appropriate operating system commands.

You cannot change the storage-page size or system name of a federated database.

WARNING You must run oochange on the host where the federated database was created. If you need to run oochange on a different host, you use ooinstallfd (page 175) to install the federated database on the current host before running oochange.

(*FTO*) When a federated database is partitioned, you use oochange (with the -ap or -id option) to change the attributes of the specified autonomous partition. You cannot change the system name of an autonomous partition.

Moving a Federated Database

When you change the directory or host system of a federated database's system-database file (for example, to accommodate network or system configuration changes), oochange registers the move logically (that is, within the federated database's catalog), but does not *physically* move the file. To move the system-database file physically, you use the appropriate operating system command. For example, on UNIX you use mv.

EXAMPLE The following commands move the system-database file myFd.FDB to another host system (mach77) and directory (/data2):

```
oochange -sysfilehost mach77 -sysfilepath /data2/myFd.FDB myFd
mv myFd.FDB /net/machine77/data2/myFd.FDB
```

The following command changes the lock-server host name to sys10 and the federated-database identifier to 4 for the federated database named rdFD:

```
oochange -lockserverhost sys10 -fdnumber 4 rdFD
```

Deleting a Federated Database

To delete an entire federated database, including all its files, you use oodeletefd (page 166).

Dumping and Loading Federated-Database Objects

For development purposes (for example, creating a federated-database copy with an expanded schema or changing the page size), you can dump the contents of an entire federated database, a specific database, or a container into an ASCII text file and then load the objects from the text file into a different federated database. Dumping the contents of an entire federated database is also a useful technique for checking database integrity. Dumps of corrupted databases often fail because every object in a federated database must be accessed during the process.

The performance of the Objectivity/DB loading operation is determined by the number and size of the objects to be loaded and by the connectivity of the objects within the federated database. Most deployed federated databases are large and have many relationships (associations) between objects. Therefore, loading is not useful for creating copies of most deployed federated databases.

Dumping Objects

To dump the contents of an entire federated database, a specific database, or a container into an ASCII text file, you invoke `oodump` (page 168) with options specifying the identifier of the object to be dumped and the name of the output file.

The `oodump` tool *does not* dump the following information:

- Unidirectional relationships (associations) from not-dumped objects to a dumped object
- Persistent locks created with the checkout/checkin feature
- Keyed objects
- (*FTO*) Autonomous partition information

Loading Objects

All Objectivity/DB language interfaces support loading text-format objects into an existing federated database. Objectivity/C++ also supports loading text-format objects into newly created, empty federated databases.

Loading Objects Into an Existing Federated Database

To load text-format objects into an *existing* federated database, you invoke `ooload` (page 179) with options specifying the text file to load and the target federated database. The schema of the target federated database should be identical to (or a superset of) the schema of the federated database from which the file was originally dumped.

Loading Objects Into a New Federated Database

(Objectivity / C++) To load text-format objects into a newly created, empty federated database, follow these steps:

1. Invoke oonewfd (page 185) to create an empty federated database.
2. Run the DDL processor ooddlx on the new federated database to include the schema used by the objects in the dumped file. (See the Objectivity / C++ Data Definition Language book for information about ooddlx.)
3. Invoke ooload (page 179) with the name of the input text file and the path to the new boot file.

Limitations of ooload

You should be aware of the following ooload behavior:

- The ooload tool does not preserve object identifiers (OIDs).

  Objects created by ooload are not guaranteed to have the same object identifier specified in the input file, however, ooload does preserve the containment hierarchy. Object identifiers in the dumped file are used as temporary identification tags to indicate the containment hierarchy.

  The database and container information for short object identifiers is assumed to be the same as that for the object in which the short object identifier is defined.

- The ooload tool does not resolve external references (object identifiers of, and relationships or associations to, objects not included in the text file).

  By default ooload exits with an error if it encounters an external reference. Invoking the tool with the -external option instructs ooload to allow external references in a text file, issuing a warning message for each external reference it encounters. If the external reference is a relationship or association, it is not set. If the external reference is an object identifier, it is not remapped.

- The ooload tool does not load indexing information.

- The ooload tool does not call constructors when it creates objects.

  An object created by ooload initially has empty relationships (associations) and undefined values for all other fields. ooload assigns the values specified in the input file to the fields. To ensure database integrity, verify that all object fields in the input file have explicit values.

  This is not a concern if you load a text file created by oodump. Text files created by oodump specify values for all relevant fields.

- The ooload tool does not call destructors when it deletes objects.

  If ooload deletes a database or container whose object identifier or system name matches the object identifier or system name specified by the -db or -cont option, it does not call the object's destructor.

Tidying a Federated Database

Tidying a federated database consolidates data that has become fragmented over time and removes old container versions.

To tidy a federated database, you use ootidy (page 195). To tidy a specific database in a federated database, you invoke ootidy with the -db or -id option.

Knowing when *and when not* to use ootidy is important. This section provides guidelines for using ootidy and the background necessary to understand these guidelines.

Background

Objectivity / DB does not normally return freed disk space to the file system. Instead, it maintains per-database and per-container free lists. If you delete an object and there are no other objects on the page, Objectivity / DB places the page on the appropriate container's free list. If you delete a container, Objectivity / DB places the container's pages on the appropriate database's free list.

When you create a persistent object, Objectivity / DB first checks the container's free list. If it finds a page there, Objectivity / DB stores the new object on that page. If it does not find a page there, Objectivity / DB checks the appropriate database's free list. If the database's free list has no pages either, only then does Objectivity / DB go to the file system to obtain the space required to store the object.

This algorithm for recycling pages makes creating, deleting, and modifying objects much more efficient. An optimally efficient federated database will always have pages on the free lists of fast-evolving containers and databases.

How ootidy Works

When you tidy a federated database, ootidy tries to obtain an exclusive lock on each database in the federated database. An exclusive lock prevents all users, even multiple readers, one writer (MROW) readers, from concurrently accessing that database. If ootidy fails to obtain a lock for a particular database, it skips that database and continues on to the next. It does not return to a skipped database for the duration of that tidy operation.

After ootidy obtains an exclusive lock on a particular database in a federated database, it copies the database objects one by one to a temporary file. It then deletes the original file and renames the temporary file, which becomes the new database file.

Because ootidy creates temporary files and fills them with objects from the current database before deleting the old files, it requires free disk space approximately equal to the size of the largest database in the federation.

Guidelines for Using ootidy

In general, you should use `ootidy` in the following cases:

- Before distributing a copy of a federated database to an end user. The `ootidy` tool minimizes the disk space required to store the federated database.
- Following a massive update—for example, if you have just deleted a large portion of a container and do not plan to enlarge the container in the near future. If you do not use `ootidy` on the container's database, the pages will sit unused on the container's free list.
- On a periodic basis if space is at a premium.

Warning: You should *not* use `ootidy` in the following cases:

- While any other process is accessing the federated database, because database corruption could occur. One way to guarantee that no other processes can access the database is to kill the lock server and to run `ootidy` in standalone mode.

  You should especially avoid running `ootidy` during a backup because `ootidy` might delete objects that `oobackup` references.
- If you suspect that the federated database has been corrupted. In those cases, `ootidy` can make the problem significantly worse.

Troubleshooting Access

For an Objectivity / DB tool or application to access a federated database, all the following conditions must be true:

- The federated database boot file is readable at the pathname specified or through the `OO_FD_BOOT` environment variable.
- A lock server is running on the specified host system and is compatible with the system-database file (unless the tool is running in standalone mode, which does not require a lock server).
- The system-database file specified in the boot file is visible on the network.
- The journal directory is visible on the network.
- The federated database identifier is correct.
- Appropriate access permissions are set.

If a tool or application cannot access a federated database, first verify that these conditions are true. If they are, it is likely that a tool or application failed to release a lock on the federated database. To locate unexpected locks, you use `oolockmon` (page 181). To release them and roll back the transactions that started them, you then use `oocleanup` (page 156). For more about cleanup and recovery, see also "Performing Manual Recovery" on page 119.

Getting Transaction Information

You can list all active transactions that have locks on data in a federated database. To do so, you use `oocleanup` (page 156) with only the *`bootFilePath`* argument.

You can use `oolistwait` (page 178) to list transactions that are waiting on any lockable Objectivity/DB object (federated database, database, or container). This tool also finds out whether a specified transaction is waiting for a lockable object, and if so, which transactions currently hold the lock on that object. Table 3-2 summarizes the options for filtering `oolistwait` information.

Table 3-2: Listing Transactions Using `oolistwait`

To See	Use This Option
All waiting transactions	(no options)
Waiting transactions started on a specific host system	`-host`
Waiting transactions started by a specific user	`-user`
Waiting transactions started by a specific user on a specific host system	`-host` and `-user`
A specific transaction's status, and any transactions that are using resources that it needs	`-transaction`

NOTE The `oolistwait` tool lists transactions in the order in which they are retrieved, not in the order in which they will run (that is, *not* in a sequenced queue).

Referencing Objects in a Federated Database

Every object in a federated database can be referenced through a unique identifier that distinguishes it from other objects of the same type. A federated database's identifier (sometimes called a reference number) distinguishes it from the other federated databases using the same lock server; a database's identifier distinguishes it from the other databases in the same federated database; a container's identifier distinguishes it from the other containers in the same database; and so on.

For most types of Objectivity/DB objects, the identifier is a single integer that serves as a key for locating each object relative to the storage object that contains it. For example, within a federated-database catalog, each database identifier is

mapped to a particular database file; within a database catalog, each container identifier is mapped to a particular page within the database file.

The identifier of a basic object, called an *object identifier* or *OID*, contains enough information to distinguish it from every other basic object within the entire federated database, not just within the object's container. An object identifier is 64 bits long and is composed of four 16-bit fields in the following string format:

D-C-P-S

where

D Database identifier.

C Container identifier.

P Logical page number in the container. A logical page is a storage page containing one or more small objects or the header information for a large object.

S Slot number on the page. A slot is the portion of a storage page occupied by a single small object or the header information for a large object.

For example, 78-112-8-3 identifies the persistent object stored in slot 3 of page 8 in container 112 of the database whose identifier is 78. Objectivity/DB identifies objects using object identifiers instead of memory addresses because object identifiers provide interoperability across platforms, access to more objects than direct memory addresses permit, and runtime access to objects located anywhere in a network.

For uniformity, every integer identifier can be expressed in the *D-C-P-S* string format (that is, as a four-part object identifier):

- A database identifier *D* can be expressed as *D*-0-0-0. For example, the database identifier 78 can be expressed as 78-0-0-0.
- An autonomous-partition identifier *A* can be expressed in object identifier form as *A*-0-0-0. For example, the autonomous-partition identifier 12 can be expressed as 12-0-0-0.
- A container identifier *C* within a database *D* is expressed as *D-C-P*-1, where the page number *P* is 1 for an unhashed container and a low integer for a hashed container.

A persistent object's object identifier changes during the lifetime of the object only if the object is moved to a new container. When a persistent object is moved or deleted, its object identifier may be reused for a new persistent object. Application developers do not need to manage or access object identifiers. Object identifiers are reported in some errors to identify particular objects.

Estimating Disk Space Requirements

The following subsections describe how to estimate the initial and maximum disk space requirements for an Objectivity/DB federated database.

Estimating Initial Requirements

Objectivity/DB requires a certain amount of overhead disk space in addition to the disk space required by actual objects. Use the following formulas to estimate the initial amount of disk space required (before objects are created):

diskSpace ≈ *fdOverhead* + (*numberOfDbs* * *dbOverhead*)

where

fdOverhead	Approximately 100 kilobytes to 1 megabyte.
numberOfDbs	Number of databases in the federated database.

dbOverhead ≈ *numberOfConts* * *initPages*

where

numberOfConts	Number of containers in a database.
initPages	Initial number of pages allocated for a container (see the language-specific documentation for information about creating containers).

Estimating Maximum Federated Database Size

You can use the information in the following subsections to estimate the maximum size (in bytes) of a fully populated federated database and the approximate number of accessible objects it contains. The subsections describe:

- Basic values used in size estimates
- Formulas for making estimates on 64-bit systems
- Formulas for making estimates on 32-bit systems

Values Used in Size Estimates

Table 3-3 lists various values you will need for estimating federated-database size. Several of these values—the maximum number of databases, containers, and pages—are determined by the field sizes in an object identifier. Each 16-bit field can have up to 2^{16} values; this field size determines the addressing capability at each level of the storage hierarchy; some of these addresses are reserved for Objectivity/DB overhead.

Table 3-3: Values Required for Calculating Federated Database Size

Item	Value
Maximum number of databases in a federated database	2^{16} – 1 databases (65,535 databases).
Maximum number of containers in a database	2^{15} – 1 containers (32,767 containers). Objectivity/DB reserves the high-order bit for overhead. As containers get large, the number of containers is limited by the maximum database file size (in bytes), divided by the average container size: *(db size)* / (*average pages per cont* x *page size*). When Objectivity for Java or Objectivity/Smalltalk applications create persistent objects, an additional container is reserved for the roots container.
Maximum number of logical pages in a container	2^{16} – 1 logical pages (65,535 logical pages). A logical page is a storage page containing one or more small objects or the header information for a large object.
Minimum number of logical pages in a container	2 pages of overhead per unhashed container and 4 pages of overhead per hashed container.
Maximum physical size of a database file	Objectivity/DB uses 64-bit file offsets, so the maximum database size is constrained by the maximum file size of the operating system. For 32-bit file systems, this is generally 2^{31} bytes (2 gigabytes), or approximately 2 x 10^9 bytes, per file. Even if larger files are supported, the maximum useful file size is 2^{47} bytes (limit imposed by the addressing capability of an object identifier).
Maximum page size	2^{16} bytes (65,536 bytes).
Minimum page size	512 bytes.
Average number of objects (slots) per logical page	(*Page size* - 32) / (*Average object size* + 14). Although the theoretical maximum number of slots on a page is 2^{16}–1, Objectivity/DB reserves 32 bytes per page and 14 bytes per object for overhead. The available space for objects on a page is thus *page size* - 32 and the average slot size is *average object size* + 14. The larger the objects, the fewer slots per page.

Estimating Sizes on 64-Bit File Systems

For 64-bit file systems, the approximate maximum number of bytes in a federated database is as follows. This estimate assumes the maximum page size (2^{16} bytes) is used.

Maximum number of bytes ≈
2^{16} *db/fd* **x** **2^{15}** *cont/db* **x** **2^{16}** *pages/cont* **x** **2^{16}** *bytes/page* ≈
2^{63} (or approximately **10^{19}**) *bytes/fd*

The approximate number of accessible objects in a fully populated federated database is as follows. In this estimate, *N* represents the average number of objects per logical page, which is calculated as shown in Table 3-3. This estimate does not take into account the 2 or 4 pages of overhead per container.

Number of objects ≈
2^{16} *db/fd* **x** **2^{15}** *cont/db* **x** **2^{16}** *pages/cont* **x** *N objects/page* ≈
(2^{47} x *N***)** *objects/fd*

Estimating Sizes on 32-Bit File Systems

For 32-bit file systems, the maximum number of bytes in a federated database is approximately as follows. This estimate is limited by the 2-gigabyte (2^{31} bytes) file size limit.

Maximum number of bytes ≈
2^{16} *db/fd* **x** **2^{31}** *max bytes/db file* ≈
2^{47} (or approximately **10^{14}**) *bytes/fd*

The approximate number of accessible objects in a fully populated federated database is as follows. This estimate assumes the maximum page size (2^{16} bytes) is used. With the 2^{31}-byte file size limit and a 2^{16}-byte page size, the maximum number of pages per database is 2^{15}. In this estimate, *N* represents the average number of objects per logical page, which is calculated as shown in Table 3-3. This estimate does not take into account the number of containers or the 2 or 4 pages of overhead per container.

Maximum number of objects ≈
2^{16} *db/fd* **x** **2^{15}** *pages/db* **x** *N objects/page* ≈
(2^{31} x *N***)** *objects/fd*

4

Browsing Objects and Types

Objectivity/DB provides a tool for browsing objects and types in a federated database. You can also use this tool to make queries for objects in a federated database.

This chapter describes:

- Information you can browse
- How to use the Objectivity/DB browser tool (`oobrowse`) on Windows
- How to use the Objectivity/DB browser tool from the Tool Manager (`ootoolmgr`) on UNIX

Information You Can Browse

The Objectivity / DB browser tool provides three browsers—a data browser, a type browser, and a query browser.

Data Browser

Using the data browser, you can view objects in a federated database.

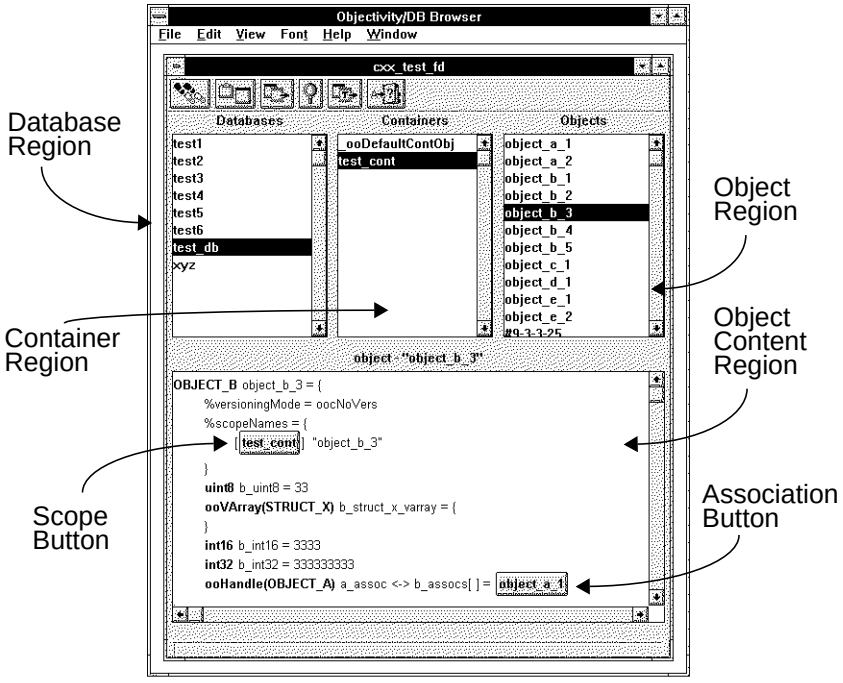

Figure 4-1 Objectivity/DB Data Browser

Type Browser

Using the type browser, you can view class definitions in a federated database schema.

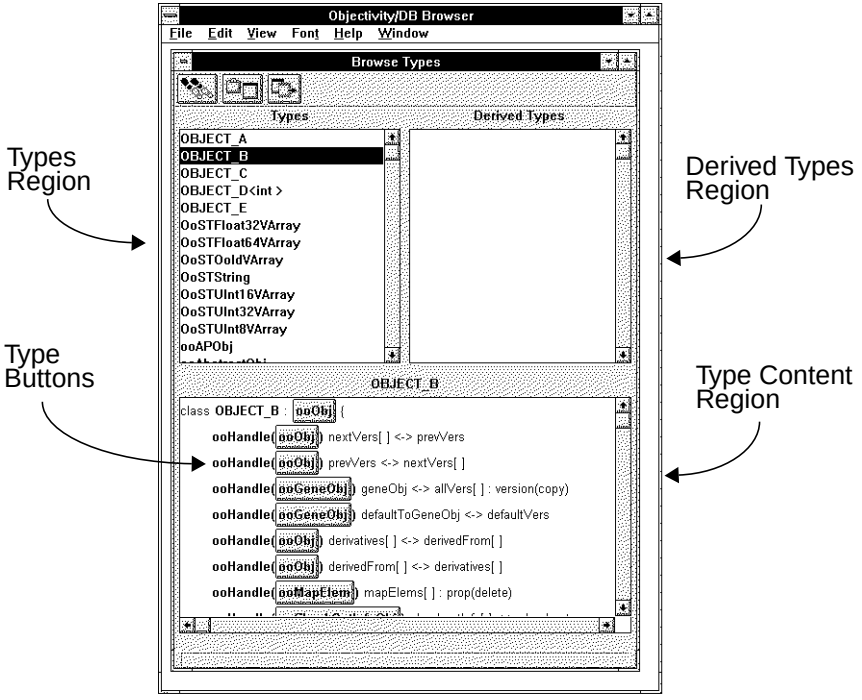

Figure 4-2 Objectivity/DB Type Browser

Query Browser

Using the query browser, you can perform ad hoc queries for objects in a federated database. When making queries, you can use regular expressions supported by the Objectivity / DB predicate query language. For information about using the predicate query language in a particular programming interface, see the documentation for that interface.

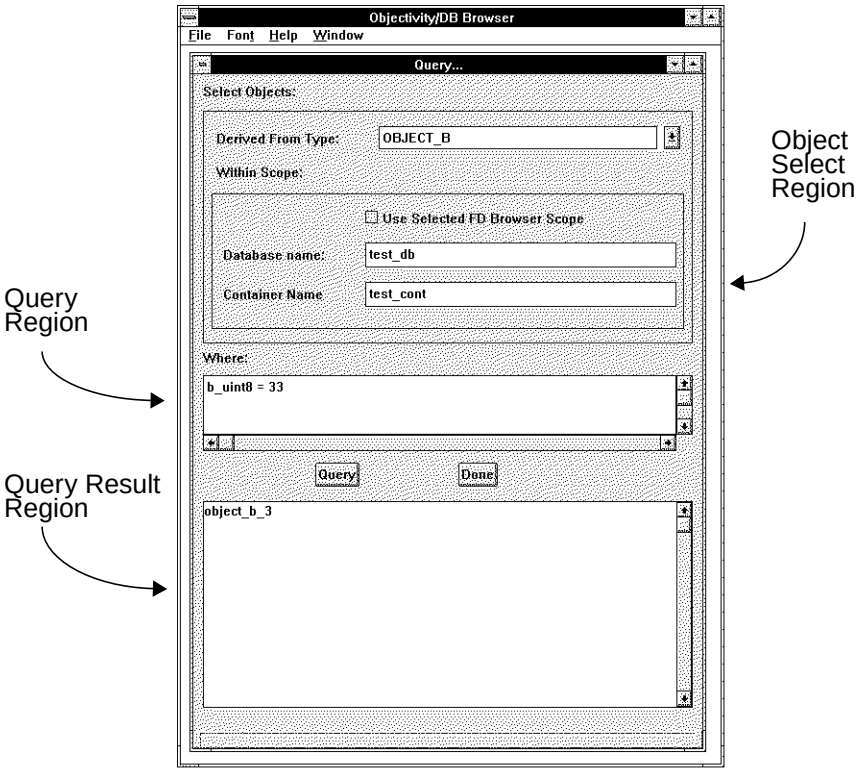

Figure 4-3 Objectivity/DB Query Browser

Opening Browsers on Windows

The Objectivity / DB browser tool on Windows platforms is oobrowse (page 149).

Starting and Using oobrowse

To start a browser on Windows:

1. Click **Start** and point to **Programs**. In the Objectivity submenu, select **oobrowse**. (Alternatively, you can run oobrowse from a command prompt).
2. From the main browser panel, open a data browser on a specific federated database. To do this, choose **Open** from the **File** menu and select a boot file.
3. Choose what the data browser displays by selecting from the **Options** menu:

Types Information	Type names of displayed objects
System-defined Fields	Data members defined by Objectivity/DB
User-defined Fields	User-defined data members
Inherited Fields	Data members inherited from other classes
Empty Fields	Fields containing no values
Object Names (not IDs)	Names or identifiers of displayed objects

4. If desired, start a type browser by choosing **Types** from the **View** menu.
5. Choose what the type browser displays by selecting from the **Options** menu:

System-defined Fields	Data members defined by Objectivity/DB
User-defined Fields	User-defined data members
Inherited Fields	Data members inherited from other classes
Persistent Types	Persistence-capable classes
Non-persistent Types	Non-persistence-capable classes

6. If desired, start a query browser:
   a. Visit the data browser by choosing **FD** from the **View** menu.
   b. Click **Query**.

Quitting oobrowse

To quit from oobrowse, choose **Exit** from the **File** menu.

Opening Browsers on UNIX

The Objectivity/DB browser tool on UNIX platforms is `ootoolmgr`.

Starting and Using ootoolmgr

To start a browser on UNIX:

1. Start a Tool Manager by executing the `ootoolmgr` command:
   `ootoolmgr [`*`bootFilePath`*`]`
   where

   `bootFilePath` Path to the boot file of the federated database. You can omit this parameter if you set the `OO_FD_BOOT` environment variable to the correct path.

2. If you omitted the boot file path in step 1 and the `OO_FD_BOOT` environment variable is not set, open a specific federated database by choosing **OpenFD** from the **File** menu.
3. Start the desired browser:
   - To open a data browser, choose **Browse FD** from the **Tools** menu.
   - To open a type browser, choose **Browse Types** from the **Tools** menu.
   - To open a query browser, open a data browser and then use the **Search** menu or click **Query**.

Quitting ootoolmgr

To quit from `ootoolmgr`, choose **Exit** from the **File** menu of the Tool Manager.

5

Debugging a Federated Database

When a database application produces incorrect results, you can inspect the objects in a federated database and change various aspects of their structure and contents.

This chapter provides information about:

- Inspecting and editing a federated database (all platforms)
- Inspecting and editing a federated database from within a C++ debugger (UNIX only)
- Viewing objects in a federated database from within a C++ debugger (Windows and UNIX)

If you just want to view objects in a federated database without modifying them, you should use the Objectivity / DB browser tool for your platform (see Chapter 4, “Browsing Objects and Types”).

Inspecting and Editing a Federated Database

You use the oodebug (page 164) tool on any platform to inspect and edit a federated database. The oodebug tool provides a set of commands for performing limited transactions on a federated database. You use these commands in one of two modes:

- In *read mode*, you can use the oodebug commands for printing an object’s contents and iterating through an object’s relationships (associations).
- In *update mode*, you can use the full set of oodebug commands to create or delete objects, change values in object fields, form or drop relationships (associations) between objects, and so on.

WARNING Update mode is dangerous—use it with care. Update operations do not have the built-in safeguards that your programming language provides. For example, oodebug commands for changing field values can access private data directly,

without going through access methods. Furthermore, oodebug commands for creating and deleting objects do not invoke object constructors and destructors; you must perform equivalent operations explicitly using appropriate oodebug commands.

Starting oodebug as a Separate Process

You can start oodebug as a separate process on any platform and use it to interact with a federated database that is not currently locked by other applications. Starting oodebug opens the specified federated database in read mode and initiates a transaction.

To start oodebug from a command line:

➤ Enter the following command:
oodebug *bootFilePath*
where

bootFilePath Path to the boot file of the federated database. You can omit this parameter if you set the OO_FD_BOOT environment variable to the correct path. (*FTO*) You can specify any autonomous partition boot file.

Changing oodebug Modes

The oodebug tool starts in *read mode*, which is indicated by the (read) command-line prompt. Read mode lets you safely view the structure and contents of a federated database without being able to edit it. In read mode, you are restricted to a subset of the oodebug commands.

If you want to edit a federated database, you can turn on *update mode*, which changes the oodebug command-line prompt to (update). Update mode allows you to perform all oodebug commands, including commands to change the structure and contents of a federated database.

NOTE Update mode is intended for use by database administrators and advanced users only.

You can switch between modes at any time while oodebug is running:

- To switch from read mode to update mode, you specify update at the oodebug command line.
- To switch from update mode to read mode, you specify read at the oodebug command line.

Performing Transactions With oodebug

When you start oodebug as a separate process, you can *start*, *commit*, or *abort* database transactions. Each transaction can comprise a single oodebug command or a series of logically related commands. When you enter a command that changes the structure or contents of the federated database, oodebug displays subsequent command prompts with a leading asterisk—for example, `(*read)` or `(*update)`.

A new transaction is started whenever you start oodebug as a separate process, invoke the `commit` command, or invoke the `abort` command.

When you invoke the `commit` command, oodebug:

- Ends the current transaction.
- Makes the transaction's changes permanent in the federated database.
- Frees any locks.
- Starts a new transaction.
- Displays subsequent command prompts without a leading asterisk (*).

When you invoke the `abort` command, oodebug:

- Ends the current transaction.
- Cancels any changes and maintains the structure and contents of the federated database in its state prior to the beginning of the current transaction.
- Frees any locks.
- Starts a new transaction.
- Displays subsequent command prompts without a leading asterisk (*).

Terminating oodebug

To terminate oodebug:

➤ Enter the following command:

```
quit
```

Terminating oodebug closes the federated database, and if no changes are pending, aborts the transaction that you started when you invoked oodebug or issued a previous `commit` or `abort` command.

If, however, changes are pending in the oodebug transaction, you must explicitly save or ignore these changes with the `commit` or `abort` commands. Pending changes are indicated by an asterisk in your command prompt: `(*read)` or `(*update)`.

If you enter `quit` without explicitly saving or ignoring changes, oodebug displays an error message.

EXAMPLE This example shows how to terminate oodebug if changes are pending.

```
(*update) quit
pending updates, please commit or abort
(*update) commit
transaction committed
(update) quit
%
```

Running oodebug in a C++ Debugger

On UNIX, you can run oodebug from your operating system's or compiler's C++ debugger (dbx and dbx variants). This allows you to view and alter a federated database while you are debugging a running C++ application, even if the federated database is locked by that application.

When you start oodebug from a debugger, you specify the handle of a persistent object that you want to view or change. Consequently, you must start oodebug after your application has started a transaction.

Once you start oodebug, the debugger's commands are not available until you terminate oodebug. While oodebug is running, its prompt replaces the debugger's prompt. oodebug is started in read mode.

You invoke oodebug from dbx using the oodebug convenience function. To do this:

1. Link your application with the debug-compatible Objectivity / DB library `liboo_dbx.a`. For more information on linking, see the *Installation and Platform Notes for UNIX*.
2. Define the oodebug convenience function to dbx using either of the following methods:
   - In your `.dbxinit` file, include the Objectivity / DB debugger convenience functions that are provided in *`installDir/arch`*`/etc/oo.dbxinit`.
   - From the dbx command line, invoke the `source` command to load the convenience functions from *`installDir/arch`*`/etc/oo.dbxinit`,

   where

`installDir`	Installation directory for Objectivity/DB
`arch`	Subdirectory for your architecture

3. In the debugger, run your application until it starts the desired transaction.
4. Invoke oodebug by entering:
   oodebug *ref_or_handle*
   where

   ref_or_handle Object reference or handle within the program you are debugging that identifies the object you want to view or change. *ref_or_handle* indicates the name of an active variable of the class ooRef(*className*) or ooHandle(*className*).

Terminating oodebug Within a Debugger

To terminate oodebug:

➤ Enter the following command:
   quit

Quitting oodebug returns you to the debugger.

Using ooprint in a C++ Debugger

On Windows and UNIX, you can use the ooprint (page 209) convenience function to view the contents of a persistent object while you are debugging a C++ database application.

You can run ooprint from the debugger without having to start the oodebug (page 164) tool (ooprint is functionally equivalent to oodebug's print command). The contents are displayed in the format shown by the Objectivity / DB data browser. When you start ooprint from a debugger, you specify the handle of a persistent object that you want to view. Consequently, you must start ooprint after your application has started a transaction.

Windows

To use the ooprint convenience function to view a persistent object while debugging a process with the Microsoft Visual C++ debugger:

1. Link your application with the appropriate debug-compatible Objectivity / DB library. For more information on linking, see the *Installation and Platform Notes for Windows*.
2. In the debugger, run your application until it starts the desired transaction.

3. Invoke `ooprint`:
   a. Select **View > Output** to display the debugger output window.
   b. Select **Debug > QuickWatch**.
   c. In the Expression field, enter the following and click **OK**:
   `ooprint(&`*`ref_or_handle`*`)`

   where

`ref_or_handle`	Object reference or handle within the program you are debugging that identifies the object you want to view. *`ref_or_handle`* indicates the name of an active variable of the class `ooRef(`*`className`*`)` or `ooHandle(`*`className`*`)`.

   The debugger output window displays the object that is referenced by the handle variable.

UNIX

To use `ooprint` to view a persistent object while debugging a process with `dbx`:

1. Link your application with the debug-compatible Objectivity / DB library `liboo_dbx.a`. For more information on linking, see the *Installation and Platform Notes for UNIX*.
2. Define the `ooprint` convenience function to `dbx` using either of the following methods:
   - In your `.dbxinit` file, include the Objectivity / DB debugger convenience functions that are provided in *`installDir`*`/`*`arch`*`/etc/oo.dbxinit`.
   - From the `dbx` command line, invoke the `source` command to load the convenience functions from *`installDir`*`/`*`arch`*`/etc/oo.dbxinit`,

     where

`installDir`	Installation directory for Objectivity/DB
`arch`	Subdirectory for your architecture

3. Invoke `ooprint` by entering:
   `ooprint` *`ref_or_handle`*

   where

`ref_or_handle`	Object reference or handle within the program you are debugging that identifies the object you want to view. *`ref_or_handle`* indicates the name of an active variable of the class `ooRef(`*`className`*`)` or `ooHandle(`*`className`*`)`.

6

Database Tasks

A *database* is the second highest level in the Objectivity / DB logical storage hierarchy, existing as a component of a federated database. Databases are logically and physically where persistent data is actually stored. Because each database is physically a separate file, you can use databases to distribute data across your network. Administering a database mainly involves updating its storage characteristics to help you better utilize your disk and network resources.

This chapter describes:

- General information about databases
- Getting information about a database
- Creating a database
- Relocating a database file
- Copying a database file and attaching it to a federated database
- Changing database attributes
- Deleting a database
- Setting access permissions on a database file
- Tidying a database
- Troubleshooting access problems

You can write applications to perform many routine administration tasks for databases. See Objectivity's language-specific programming interface documentation for more information.

About Databases

Like a federated database, a database has *attributes* that specify its various physical and logical characteristics. Physically, a database is a file; the attributes describing its physical location are:

- The file host—the host machine on which the database file resides
- The file path—the database file's pathname and filename

Logically, a database has an identity within a federated database; the attributes describing its identity are:

- The containing federated database—the federated database to which the database is *attached* and that serves as the point of administration for this database
- The system name—the unique logical name of the database within the scope of the containing federated database
- The database identifier—the unique numeric identifier within the scope of the containing federated database

A database also has the following attributes that identify:

- Whether a database is read-only or read-write
- (*FTO*) The controlling autonomous partition

Databases are normally created programmatically, so a database's attributes are usually set by the application that creates it. You can change any attribute value, including the database identifier, using various Objectivity/DB administration tools.

Database Identifier Formats

Many administration tools allow you to use a database identifier to specify a database. You find out a database's identifier by listing file or attribute information for the database.

A database identifier is usually a single, nonnegative integer—for example, 5. Sometimes, you will see database identifiers represented as object identifiers written in D-C-P-S format. For example, a database identifier of 5 is equivalent to an object identifier of 5-0-0-0. You can use either format when specifying a database to a tool.

Read-Only and Read-Write Databases

A database is either *read-only* or *read-write*. At creation, every database is read-write, so that persistent objects can be created in it. At any time, a database can be changed to read-only, so that it can be opened only for read; any attempt to open the database for update will fail as if there were a lock conflict. For example, a database that contains only archival information could be made read-only. Making a database read-only can improve the performance of an application that performs many read operations in the database; the application can grant read locks and refuse update locks on containers in the database without having to consult the lock server every time such containers are opened.

When a database is read-only, it can either be read or changed back to read-write. The database must be changed back to read-write before any other operations can be performed on it. You make a database read-only and change it back to read-write by changing its attributes.

Any number of databases can be read-only in a federated database. When a multiple read-only databases exist in a federated database, they are locked or unlocked as a group. Consequently, *no* read-only database can be changed back to read-write while *any* read-only database is being read by a tool or application.

Replicated Databases

In an Objectivity / DRO environment, a database can be replicated so that different images of the same database belong to different autonomous partitions. Physically, each image is a different database file. Logically, all images of a database share the same system name and database identifier, and each image is controlled by a different autonomous partition.

Each image of a replicated database has an additional attribute specifying its weight. Objectivity / DRO uses the weights of images to determine whether a quorum exists. In general, tasks affecting database images require that a quorum of the database's images be available (an image is available if the containing partition is available).

The tasks in this chapter apply to entire databases, regardless of the number of images. The exception is listing or changing attribute values, which affects only one database image at a time. A database is normally specified by its system name; a database image is normally specified by the database system name and the system name of the controlling autonomous partition.

If a database has multiple images, *all* images are either read-only or read-write. While a database is read-only, you cannot add, delete, or change the attributes of individual images.

For complete information about data replication, see the Objectivity / FTO and Objectivity / DRO book.

Getting Database Information

You can use Objectivity/DB tools to obtain information about databases:

- To list a database's current attribute values, you invoke oochangedb (page 152) with the -db (or -id) and *bootFilePath* options. (*DRO*) To list the attributes of a database image, you must also specify the -ap option.
- To list information about a particular database file, you invoke oofile (page 172) followed by the name of the file.
- (*DRO*) To obtain information about database images and weights, you invoke oodumpcatalog (page 171) followed by the boot-file name for any autonomous partition.

The following subsections describe how to find specific information about a database.

Getting a System Name or Database Identifier

A number of tools request a system name or database identifier:

- If you know only the name of a database file, you can use oofile to find the database's system name or identifier.
- If you know either the system name or database identifier, you can use oochangedb to find the other attribute.

Getting a Database's File Host and Path

A number of tools request a database file host or path:

- If you know the database system name or identifier, you can use oochangedb to find the file host and path.

Getting a Database's Page Size

For some tasks, you may need to find the size of the storage pages in a database. To do this:

1. Find the database's file path.
2. Using the database's file path, invoke oofile to display the storage-page size.

Getting a List of Read-Only Databases

You can find out which databases are currently read-only or read-write by invoking oodumpcatalog (page 171).

Creating a Database

You can use oonewdb (page 183) to create an empty database within a specified federated database. Using a tool to create a database is an alternative to creating databases from within an application, which is the normal practice. You must specify the new database's system name and file location.

The host and path format you use when specifying the file location depends on the type of host machine (Windows or UNIX) on which the new database is created, and the data-server software you are running on that host. For complete details, see "Host and Path Formats" on page 30 and Table 11-1 on page 132.

EXAMPLE The command in each of the following examples creates the `partsDb` database of the federated database `myFD`.

Windows

The database file `partsDb` is called `parts.DB` and is placed in the folder `c:\project1\data` on the Windows host that executed the `oonewdb` tool. This host is running AMS, so the command uses a fully qualified pathname that is local to that host.

```
oonewdb -db partsDb -filepath c:\project1\data\parts.DB myFD
```

UNIX

The database file `partsDb` is called `parts.DB` and is placed in the directory `/project1/data` on the UNIX host system `machine33`, which is running NFS:

```
oonewdb -db partsDb -host machine33
   -filepath /project1/data/parts.DB myFD
```

When you use `oonewdb`, you may optionally specify an identifier for the new database, for a reason such as the following:

- Application development is split across several teams, and each team by convention must assign database identifiers from within a certain range.
- You are recreating an existing federated database, and you need to make sure that the database with a given system name has the same identifier in both the original federated database and the new one.

(*DRO*) The `oonewdb` tool creates only the initial database image; you use `oonewdbimage` to create each additional image (see the Objectivity / FTO and Objectivity / DRO book).

Moving a Database File

You can move a database file from one location to another in a networked file system—for example, to take advantage of the disk space on a new host machine. To do this, you invoke oochangedb (page 152) with options specifying the new host and file path. This operation updates the catalog *and* physically relocates the file, but makes no change to the database's identity attributes—that is, the database keeps the same system name and identifier and remains within the same federated database.

EXAMPLE The command in each of the following examples moves the `partsDb` database of the federated database `myFD`.

Windows

The following command moves the file for the database `partsDb` to the local file `c:\project2\data\parts.DB`:

```
oochangedb -db partsDb -filepath c:\project2\data\parts.DB myFD
```

UNIX

The following command moves the file for the database `partsDb` to the file `/project2/data/parts.DB` on the host system `sys12`:

```
oochangedb -db partsDb -host sys12
   -filepath /project2/data/parts.DB myFD
```

If you want to update the catalog without physically moving the file, you invoke `oochangedb` with the `-catalogonly` option.

If the database you want to move is read-only, you must change it back to read-write before you can move it.

Copying a Database File

You can create a copy of a database file using oocopydb (page 160). This operation copies the contents of the specified database into a new file, preserving the same database system name and identifier. The resulting copy does not belong to any federated database, so the copy cannot be used until you attach it to a federated database. The original database is not affected by the operation.

(*DRO*) Copying a database creates only a single copy, regardless of the number of database images. The resulting copy cannot be reattached as a database image.

The oocopydb tool locks the database to be copied, providing a safe alternative to copying database files directly using operating system commands. The original database is inaccessible during the execution of oocopydb.

By default, the copy operation fails if objects within the database to be copied have relationships or associations to objects in another database. To copy a database with external relationships or associations, you specify the -external option.

EXAMPLE The command in each of the following examples copies the partsDb database of the federated database myFD. The unattached copy is stored in the stdparts.DB file in the specified location on the host that executed the oocopydb tool.

Windows

```
oocopydb -db partsDb -filepath c:\project2\data\stdparts.DB myFD
```

UNIX

```
oocopydb -db partsDb -filepath /project2/data/stdparts.DB  myFD
```

Attaching a Database to a Federated Database

When you copy a database using oocopydb (page 160), the resulting copy does not belong to any federated database. To attach the copy to a federated database, you use ooattachdb (page 144). Thus, you can combine oocopydb and ooattachdb to:

- Move a database to another federated database.
- Duplicate a database within a federated database.
- Change the system name or identifier of a database.

Moving a Database Between Federated Databases

You can move a database from one federated database to another, provided that the two federated databases have the same storage-page size and compatible schemas. To do this:

1. Invoke oocopydb to create a copy of the database to be moved.
2. (Optional) Invoke oodeletedb (page 165) to delete the original database.
3. Invoke ooattachdb to attach the database copy to the target federated database. Note that:
   - You must specify the -db and -id options; you can use these options to specify a new a system name and database identifier.

- You can specify the `-readonly` option to make the attached database read-only. By default, the database is attached as a read-write database.

See "Guidelines for Attaching a Database" on page 73 and "Consequences of Changing a Database Identifier" on page 73.

4. (*DRO*) Recreate database images as desired in the target federated database.

EXAMPLE The commands in the following examples:

- Copy the `partsDb` database into a new file (`stdparts.DB`) in the specified location.
- Delete the original database `partsDb` from the federated database `myFD`.
- Attach the copied database with the system name `stdParts` and a database identifier of `15` to the target federated database `newFD`.

Windows

```
oocopydb -db partsDb -host machine5
   -filepath c:\project2\data\stdparts.DB myFD
oodeletedb -db partsDb myFD
ooattachdb -db stdParts -id 15 -host machine5
   -filepath c:\project2\data\stdparts.DB newFD
```

UNIX

```
oocopydb -db partsDb -host machine33
   -filepath /project2/data/stdparts.DB myFD
oodeletedb -db partsDb myFD
ooattachdb -db stdParts -id 15 -host machine33
   -filepath /project2/data/stdparts.DB newFD
```

Duplicating a Database Within a Federated Database

You can duplicate a database within the same federated database. To do this, you follow the steps for moving a database, except that you attach the copy to the same federated database. You *must* attach the copy with a different system name and database identifier from the original database (see "Consequences of Changing a Database Identifier" on page 73).

(*DRO*) Duplicating a database results in a database copy that can be updated independently from the original. In contrast, when a database is replicated, updates are propagated across its images to keep them identical.

Guidelines for Attaching a Database

To execute successfully, `ooattachdb` requires that:

- The database file is recognizable as an Objectivity / DB database file. To check whether this condition holds, you use `oofile` (page 172).
- No database with the same system name, database identifier, or database file name already exists in the target federated database.

  Note: You must specify the `-db` and `-id` options, even if you want to preserve the original system name and database identifier.
- The storage-page size of the target federated database is the same as the storage-page size of the database being attached. You can use `oofile` to determine the page size of a federated database or database.
- The schema of the target federated database is identical to (or a superset of) the schema of the database being attached.

  Note: It is your responsibility to ensure that the two schemas are compatible.

Consequences of Changing a Database Identifier

When you change a database identifier using the `-id` option of `ooattachdb`, Objectivity / DB automatically adjusts:

- The object identifiers (OIDs) of the objects in the database
- Any relationships (associations) within the attached database
- Any relationships (associations) that exist among a group of databases attached through the `-dbmap` option

However, Objectivity / DB does not attempt to adjust any references from objects in the target federated database to objects in the database being attached.

Attaching Multiple Databases

You can attach a group of databases to a federated database in a single operation. To do this, you invoke `ooattachdb` with the `-dbmap` option specifying the name of an ASCII text mapping file.

The mapping file specifies the system name, the database identifier, and file location of each database to be attached. Each line has the format:

`targetDbID targetDbSysName hostName filepath`

where

`targetDbID`	Database identifier with which the database is to be attached. If this is a new *`targetDbID`*, see “Consequences of Changing a Database Identifier” above.
`targetDbSysName`	System name with which the database is to be attached.

hostName	Host system where the database file is located.
filepath	Pathname (including the filename) where the database file is located.

If an error is detected in the line entry for any of the databases in the mapping file, *none* of the databases are attached and ooattachdb terminates after issuing an error message.

EXAMPLE This command attaches three databases to the newFD federated database:

```
ooattachdb -dbmap /tmp/file.map newFD
```

The mapping file /tmp/file.map reads as follows:

```
3 parts machine33 /project1/parts/parts.DB
4 newParts machine33 /project1/testData/newParts.DB
5 oldParts machine33 /project1/oldParts/oldParts.DB
```

Changing Database Attributes

If a database is read-only, you must change it back to read-write before you can change any other attributes. You can change a read-only database back to read-write only if no other tool or application is currently reading either that database or any other read-only database in the same federated database.

You can use oochangedb (page 152) to change these database (or database-image) attributes:

- The file host and file path (this moves the database file)
- Whether the database is read-only or read-write
- (*FTO*) The containing autonomous partition
- (*DRO*) The weight of a database image

Other tools are required to change a database's system name or database identifier as shown in the following subsection.

Changing the System Name or Database Identifier

Changing a database's system name or database identifier is similar to duplicating a database. That is, you copy the database, delete the original database, and reattach the copy with the new identifier or system name. (*DRO*) If the database is replicated, you must recreate any desired images from the reattached copy.

A database identifier is a component of the object identifiers (OIDs) of objects in the database. To see how such OIDs are affected, see “Consequences of Changing a Database Identifier” on page 73.

Deleting a Database

You can delete a database from a federated database by using `oodeletedb` (page 165). Objectivity / DB deletes the database file, updates the federated database catalog, and removes all bidirectional relationships (associations) and all unidirectional relationships (associations) to objects in other databases.

If the database you want to delete is read-only, you must change it back to read-write before you can delete it.

(*DRO*) If the database is replicated, `oodeletedb` deletes all images; use `oodeletedbimage` to delete an individual image.

If the database file no longer exists, you delete the database from the federated database catalog by invoking `oodeletedb` with the `-catalogonly` option.

EXAMPLE This command deletes the database `partsDb` from the federated database `myFD`:

```
oodeletedb -db partsDb myFD
```

Setting File Permissions on a Database

You can prevent unauthorized users from writing to or deleting a database file by using operating system commands to set appropriate permissions on that file and, if necessary, the directory that contains it. Be sure to grant adequate permissions to the account(s) that run AMS or the lock server so that these servers can read and update all Objectivity / DB files.

Tidying a Database

To consolidate a database that has become fragmented over time, you invoke `ootidy` (page 195) with the `-db` option. This tool transfers database objects to a temporary database file while it is running. Therefore it requires free disk space approximately equal to the size of the database you are tidying. See “Tidying a Federated Database” on page 46 for a detailed discussion of `ootidy` operation. (*DRO*) The `ootidy` tool tidies all images of a replicated database.

Warning: You should *not* use `ootidy`:

- While any other process is accessing the database, because database corruption could occur. One way to guarantee that no other processes can access the database is to kill the lock server and to run `ootidy` in standalone mode.

  In particular, you should avoid running `ootidy` during a backup because `ootidy` might delete objects that `oobackup` references.
- If you suspect that the database has been corrupted. `ootidy` can make the problem significantly worse.

Troubleshooting Access Problems

Any of the following conditions can prevent access to a single database within a federated database:

- The database file is missing or corrupted.
- The federated database catalog is incorrect or corrupted.
- The database file protections are incorrect.
- The database host system is down.
- The database is read-only, and the federated database contains other read-only databases that are currently being accessed. (A read-only database cannot be changed back to read-write if any other read-only database in the federation is currently being read.)

A workstation or network error can cause you to lose data. If you determine that the database file is missing or corrupted, restore the file from a backup.

For troubleshooting access to an entire federated database, see “Troubleshooting Access” on page 47.

7

Using a Lock Server

Objectivity/DB provides concurrent multiuser access to data. To ensure that data remains consistent, database access is controlled through locks granted by a *lock server.*

This chapter describes:

- General information about lock servers
- Deciding whether to use a lock server
- Checking whether a lock server is running
- Starting and stopping a lock server
- Changing the lock-server host for a federated database or autonomous partition
- Listing the locks that are currently managed by a lock server
- Changing the lock-server port
- Troubleshooting problems with the lock server

About Lock Servers

A *lock server* manages concurrent access to persistent objects by granting or refusing locks to requesting transactions. When a transaction requests data from a federated database, Objectivity/DB locates the lock server that services the federated database and then contacts the lock server to obtain a lock on the requested data. The lock is granted only if it is compatible with existing locks. Obtaining a lock prevents multiple concurrent transactions from performing incompatible operations on the same data, whether these transactions belong to different applications or to different threads of the same application.

Locks

A lock server grants locks to transactions requesting data from a federated database or partition. Locks prevent multiple concurrent transactions from performing incompatible operations on objects. Depending on the request, the lock server may grant the transaction a read lock, an update lock, or an exclusive lock for a requested object.

Read lock — When a transaction has a read lock for an object, other transactions can concurrently obtain read locks for that object. If the read lock was obtained by a standard transaction, no other transaction can obtain an update lock for the object. If the read lock was obtained by a transaction that uses the multiple readers, one writer (MROW) concurrency mechanism, at most one other transaction can obtain an update lock.

Update lock — When a transaction has an update lock for an object, no other transactions can concurrently obtain an update lock for that object, although transactions using MROW may obtain read locks.

Exclusive lock — When a transaction has an exclusive lock for an object, no other transaction can concurrently obtain any kind of lock on the object. Objectivity / DB obtains exclusive locks for operations such as creating or deleting a container.

Locking a basic object causes its container to be locked.

Read and update locks exist only while the lock server is running. If the lock server or its host fails during a transaction, any locks held by the transaction cease to exist. For information about recovering transactions in this situation, see “Automatic Recovery From Lock-Server Failures” on page 122 and “Manual Recovery From Lock-Server Host Failures” on page 127.

Lock-Server Host

A lock server is identified by its location—that is, by the workstation, or *lock-server host*, on which it is running. Every federated database or autonomous partition stores the name of a lock-server host as an attribute. When a transaction requests data from a federation or partition, Objectivity / DB inspects the corresponding boot file to identify the relevant lock-server host; the lock server running on that host is then contacted.

A single lock server may service multiple federated databases and autonomous partitions. In this situation, each federated database or partition specifies the same workstation as the lock-server host.

Because a federated database can have only one lock-server host, it can be serviced by only one lock server. (*FTO*) When a federated database is partitioned, each autonomous partition may specify its own lock-server host, so different partitions can be serviced by different lock servers.

You can change the lock server that services a federated database or partition; see “Changing Lock-Server Hosts” on page 84.

Types of Lock Server

A *standard lock server* normally runs as a separate process on a lock-server host, as illustrated in Table 1-1 on page 24. A standard lock server is external to all applications that consult it, even those running on the same host. Most Objectivity / DB installations are configured to use one or more standard lock servers. You normally run a standard lock server when you use administration tools such as `oonewfd` or the Objectivity / DDL tool `ooddlx`.

An *in-process lock server* is just like a standard lock server, except that it runs as part of an application process. This enables the application to request locks through simple function calls without having to send these requests to an external process. An in-process lock server can improve the runtime speed of the application that starts it, provided that most or all of the serviced lock requests are from that application.

A C++, Java, or Smalltalk application starts an in-process lock server using the interface provided with Objectivity / DB In-Process Lock Server Option (Objectivity / IPLS), which is a separately purchased product. An application that starts an in-process lock server is called an *IPLS application*. To find out more information about IPLS applications, see the documentation for your Objectivity programming interface.

From an administration point of view, the main difference between a standard lock server and an IPLS application is the way they are started and stopped; otherwise, you can treat them the same way. When an in-process lock server is started, the IPLS application becomes the lock server for the workstation on which it is running. Consequently, if a federated database names this workstation as its lock-server host, all applications accessing that federated database will send their lock requests to the IPLS application. The in-process lock server uses a separate thread to service requests from external applications.

In this book, the term “lock server” refers to either a standard lock server or an in-process lock server. Unless specified otherwise, the lock-server tasks in this chapter apply both to standard lock servers and in-process lock servers.

Lock Servers on the Network

You may run multiple lock servers on the same network, depending on the number of federated databases or autonomous partitions to be supported. A lock server may, but need not, run on the same workstation as the federated databases or partitions it serves.

You cannot run multiple lock servers from the same release of Objectivity / DB on the same workstation. Although it is possible for the same workstation to run both a current lock server and a lock server from certain older releases, this is not recommended practice. If you want to run lock servers from different Objectivity / DB releases (for example, because you are maintaining applications built with different Objectivity / DB releases), you should arrange for the relevant federated databases to use different lock-server hosts.

Required File and Directory

A lock server must have access to files named `ooRsvTb`*x* and `ools-x.VPL`, where *x* represents the file's current version number. You must not move, modify, or delete these files.

The required files are in the *Objectivity server system directory* on the lock-server host when the lock server is first accessed by an application. You must run the lock server under an account that has read and write access to the Objectivity server system directory. The location of the directory depends on the platform:

- On Windows, the Objectivity server system folder is the Windows folder, which is typically `c:\windows` for Windows 98 and `c:\winnt` for Windows NT and Windows 2000. The location of this folder, especially the drive name, may be different on your system.
- On UNIX, the Objectivity server system directory is `/usr/spool/objy`. You normally create this directory when you install Objectivity / DB.

The Objectivity server system directory also contains a file called `oolsrec.log`, which records any errors from automatic recovery triggered by the lock server.

Deciding Whether to Use a Lock Server

Locking is required whenever multiple concurrent transactions (in multiple applications or in multiple concurrent threads) can access the same federated database. However, locking can be disabled for an application that has exclusive access to a federated database and requires maximum performance—generally, a single-user, single-threaded application run by a user with exclusive file permissions. For information about disabling locking in an application, see the documentation for your Objectivity programming interface.

WARNING If an application disables locking, you *must* guarantee that only one thread has access to the federated database at any time, or data corruption may occur.

Checking Whether a Lock Server is Running

You can check whether a lock server is running on a particular workstation. To do this, use `oocheckls` (page 155).

Starting a Lock Server

Before you start a lock server, you must determine whether to start a standard lock server or an in-process lock server (“Types of Lock Server” on page 79). A standard lock server is used during application development and with most deployed federated databases at end-user sites. An in-process lock server is normally used only as necessary to improve the runtime speed of a particular deployed application.

Standard Lock Server

When you install Objectivity / DB, you normally configure a workstation to start a standard lock server automatically every time the machine reboots. However, there may be situations that require you to start (or restart) a lock-server process.

You can start the lock server with or without arguments, depending on the kind of automatic recovery you want.

Windows

On Windows platforms, you start the lock server using the Objectivity Network Services tool that is provided with Objectivity / DB. This causes the lock server to run even when no user is logged on and to start automatically whenever the system boots. See “Starting and Stopping an Objectivity Server” on page 215.

You can optionally specify arguments to the lock server for automatic recovery. See “Configuring an Objectivity Server” on page 216.

On Windows NT or Windows 2000, you must make sure that the lock server is started under a logon account that has the necessary access permissions. See “Specifying a Service’s Logon Account” on page 216.

The account under which the lock server runs must have:

- Read and write permissions to the Objectivity server system directory; see “Required File and Directory” on page 80.
- Read permission to the boot file for each federated database or autonomous partition to be serviced.
- Read and write permissions to all system-database, database, and journal files to be serviced, and to the directories containing them.
- Permissions to use any required UNC network share names.

UNIX

On UNIX, you start a lock server using `oolockserver` (page 182), optionally specifying arguments for automatic recovery.

You must start the lock server under a user account that belongs to a group with read and write permissions to the Objectivity server system directory and to all system-database, database, and journal files to be serviced.

In-Process Lock Server

You start an in-process lock server by running an IPLS application—that is, by running an application in which Objectivity/IPLS functions are called.

An in-process lock server triggers automatic recovery as if it were a standard lock server started without arguments—the recovery of each serviced federated database is delayed until data is requested from it.

An in-process lock server cannot be started on a workstation that is already running a standard lock server or an IPLS application.

Windows

On Windows, you start an in-process lock server on a workstation as follows:

1. If necessary, reconfigure the workstation so that rebooting it does *not* automatically start a standard lock server. To do this, you use the Objectivity Network Services tool to uninstall the standard lock server as a system service.
2. Start a single IPLS application on the workstation. On Windows NT or Windows 2000, you must make sure that the IPLS application is started under a logon account that has:
   - Read and write permissions to the Objectivity server system directory

- Read permission to the boot file for each federated database or autonomous partition to be serviced
- Read and write permissions to all system-database, database, and journal files to be serviced, and to the directories containing them
- Permissions to use any required UNC network share names

UNIX

On UNIX, you start an in-process lock server on a workstation as follows:

1. If necessary, reconfigure the workstation so that rebooting it does *not* automatically start a standard lock server. To do this, you remove the `oolockserver` command from the workstation's startup script.
2. Start a single IPLS application on the workstation. You must start the IPLS application under a user account that belongs to a group with read and write permissions to the Objectivity server system directory and to all system-database, database, and journal files to be serviced.

Stopping a Lock Server

The mechanism for stopping a lock server depends on whether it is a standard lock server or an in-process lock server.

Standard Lock Server

You can stop a standard lock server at any time, provided that it is not currently servicing any active transactions. An active transaction is any transaction that holds a lock, even if the process that started it is no longer running. Thus, an active transaction may be currently in progress or waiting for recovery.

NOTE If you stop a lock server while a database application is still running (for example, while the application is between transactions), the application will encounter an error the next time it tries to start a transaction.

Windows

On Windows, you stop a standard lock server from the Objectivity Network Services tool.

Alternatively, you can enter `ookillls.exe` at a command prompt. On Windows 98, *do not* choose **End Task** while the lock server is selected in the Task Manager.

UNIX

On UNIX, you stop a standard lock server using `ookillls` (page 177).

If You Cannot Stop a Standard Lock Server

A standard lock server cannot be stopped while servicing an active transaction. If an application is using a lock server that you want to stop, you:

1. Run `oolockmon` (page 181) to determine which transactions are currently using the lock server.
2. If necessary, use `oolistwait` (page 178) to determine which processes are using the lock server; notify the process owners to commit or abort their transactions.
3. If any locks belong to processes that are no longer running, use `oocleanup` (page 156) to recover the incomplete transactions.
4. When you are certain that no active transactions are using the lock server, terminate the lock-server process as you normally would on your platform.

In-Process Lock Server

An in-process lock server can only be stopped by the IPLS application that started it.

WARNING Terminating an IPLS application before it stops its in-process lock server is equivalent to an abnormal lock-server failure, and any incomplete transactions will require recovery.

Changing Lock-Server Hosts

You may need to change the lock server that services a federated database or autonomous partition. For example, if the current lock-server host for a federated database has become permanently unavailable, you must use a lock server on a different host. Or, to reduce the number of federated databases or partitions serviced by a particular lock server, you can assign some of them to a different lock server.

To change the lock-server host for a federated database or autonomous partition:

1. Ensure that the new lock-server host can access the journal directory and all files associated with the federated database or partition, using the pathnames registered in the boot file and the system catalogs. Use oochange (page 149) with just the boot-file name (and `-ap` option, if necessary) to view these pathnames.
2. Use oochange to specify the new lock-server host as an attribute of the federated database or autonomous partition. If the old lock server is not running, use the `-standalone` option.

You can now restart the lock server on the new lock-server host.

EXAMPLE ***UNIX***

This example sets name of the lock-server host of the federated database `myFD` to `mach44` (assuming that the old lock server is still running):

```
oochange -lockserverhost mach44 myFD
```

Listing Current Locks

You can list all locks and processes currently managed by the lock server. To do this, use `oolockmon` (page 181). You can use this information to determine the locking status on objects and to locate unexpected locks. If you find locks that are held by transactions belonging to terminated application processes, you can use `oocleanup` (page 156) to release them.

Changing the TCP/IP Port for the Lock Server

The lock server is assigned a default TCP/IP port number by Objectivity/DB. This port is used by remote processes (such as Objectivity/DB applications or AMS) for communicating with the lock server.

On rare occasions, you may be prevented from starting the lock server because another service is already using the default lock-server port. If possible, you should assign the other service to a different TCP/IP port. If you cannot do this, you can assign the lock server to a nondefault port; however, you will have to make this change on *every* host that runs a process that interacts with this lock server.

Windows

If you cannot start the lock server, you can determine whether a port conflict exists:

1. Inspect the lock server output:
   - On Windows NT or Windows 2000, open the Event Viewer from the Administrative Tools folder of the Control Panel, and look in the Application Log.
   - On Windows 98, inspect the `ooLog.txt` file in the Windows folder.
2. If a port conflict exists, the message `10048` (`WSAEADDRINUSE`) is displayed.

To assign a different lock-server port:

1. Log on as administrator (Windows NT and Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. Select the Objectivity Lock Server and click the **Configure** button.
4. In the TCP/IP Port Number field, enter the new port number and click **OK**. To avoid conflicts, enter a number greater than 1035.

Note: You must make this change on *every* host that runs the lock server or a process that uses the lock server (an Objectivity/DB application or AMS).

UNIX

To assign a lock-server port on a UNIX platform:

1. Log in as `root`.
2. Add the following entry to the TCP/IP `services` file (typically, `/etc/services`):

   `ools-`*`xx`* *`portNumber`*`/tcp # Objectivity/DB lock server`

   where

`xx`	The current version of the lock server. (For the current version, see the Objectivity Technical Support web site; call Objectivity Customer Support to get access to the site.)
`portNumber`	TCP/IP port number (a number greater than 1024).

Note: You must make this change on *every* host that runs the lock server or a process that uses the lock server (an Objectivity/DB application or AMS).

If you are using the Network Information Service (NIS), you should ask your system administrator to perform the equivalent operation for your NIS configuration.

Troubleshooting Problems With the Lock Server

Because lock servers control distributed systems, problems with lock servers may differ across operating systems.

Windows and UNIX

Lock-Server Timeout

By default, an application or tool waits 25 seconds for the lock server to respond to a request. However, a lock server running on a busy machine may need more time to respond. If an application or tool consistently signals a lock-server timeout error you can increase the timeout period by setting the `OO_RPC_TIMEOUT` environment variable to the desired number of seconds—for example:

```
set OO_RPC_TIMEOUT=50                  # Windows
setenv OO_RPC_TIMEOUT 50               # UNIX
```

Alternatively, you can consider running the lock server on a less congested host.

Windows

Consider the following information if you encounter problems with the lock server on a Windows platform. On Windows NT or Windows 2000, you can use the Windows EventViewer to check the Application Log for any relevant messages.

If You Cannot Start a Lock Server

If you cannot start a lock server on a Windows platform, it may be due to one of the following reasons:

- You do not have read and write access to the Windows folder or the required files in it.
- Required services have not initialized.

  When Windows starts or reboots, many services required by the lock server are initialized, including TCP/IP. It is possible for initialization to take time. If the lock server fails to start immediately after the system reboots, try starting it again after a minute or so.
- TCP/IP is not installed.

  The lock server requires that a Winsock-compatible TCP/IP stack is installed and correctly configured. See “TCP/IP Configuration Problems” below for more information.
- A port conflict exists between the lock server and another running process.

If You Cannot Connect to the Lock Server

If your applications cannot connect to the lock server in a Windows environment, it may be due to one of the following:

- Lock server is not running.

  Make sure the lock server is running on the machine specified by your boot file.

- TCP / IP is not installed.

  Objectivity / DB applications require that a Winsock-compatible TCP / IP stack be installed and correctly configured. See "TCP / IP Configuration Problems" below for more information.

- Different hosts have different TCP / IP ports assigned to the lock server.

TCP/IP Configuration Problems

Objectivity / DB applications require that a Winsock-compatible TCP / IP stack be installed and correctly configured. In particular:

- The TCP / IP hosts configuration file should contain entries for your workstation and any other machines you wish to access, even if you are using DNS. Entries should include the host name and IP address.
- If you are using DHCP, your system administrator should set up a DHCP reservation for your host to ensure that the same IP address will always be assigned to your host.

NOTE The TCP / IP hosts configuration file is typically `C:\windows\hosts` on Windows 98, and `C:\winnt\system32\drivers\etc\hosts` on Windows NT or Windows 2000.

Some common TCP / IP configuration problems are listed below. You should consult your system administrator before changing your TCP / IP configuration:

- Incorrect IP address or host name is specified.

  Make sure that the IP addresses and host names of all the machines are consistent on all hosts. This may involve checking each workstation's TCP / IP hosts configuration file, verifying that DNS is configured properly, or verifying that DHCP is configured properly.

- Host name is missing.

  Make sure that an entry for your workstation is included in the TCP / IP hosts configuration file.

UNIX

Consider the following information if you encounter problems with the lock server on a UNIX platform. You can check the system log file for any relevant messages.

If You Cannot Start the Lock Server

A port conflict may exist between the lock server and another running process.

File Access Requirements

To start a lock server, you must have read and write access to the Objectivity server system directory (`/usr/spool/objy`) or the required files in it.

Database Files Not Exported by NFS

The lock server requires that any user directories that contain, or will contain, Objectivity/DB files be exported by NFS. If they are not exported, refer to the documentation for your operating system to export these directories. To export these directories, place them in the `/etc/exports` file.

8

Advanced Multithreaded Server

In a distributed system, an Objectivity / DB application may request data that is *local* (on the same host as the application) or *remote* (on a different host). When servicing a request for remote data, Objectivity / DB contacts *data-server software* to obtain that data. If you choose, you can use Objectivity's Advanced Multithreaded Server (AMS) as your data-server software.

This chapter describes:

- General information about AMS
- Deciding whether to use AMS
- Checking whether AMS is running
- Starting and stopping AMS
- Setting AMS usage in an application
- Changing the AMS port
- Troubleshooting problems with AMS

About AMS

AMS is data-server software that is provided with Objectivity / DB. You can run AMS on remote data-server hosts to make the system-database, database, or journal files on those hosts available to Objectivity / DB applications. AMS is an optional alternative to native file servers (commonly, NFS or Microsoft Windows Network), which means you can use AMS on any or all remote data-server hosts in a distributed Objectivity / DB system. You *must* run AMS on each data-server host that is to contain a replicated database.

When you use AMS, you refer to each Objectivity / DB file by specifying the data-server host on which the file resides and the pathname of the file on that host (for more information, see "Files on AMS Data-Server Hosts" on page 31). You do not need to export any file systems or use special network share names or mount names.

You can run AMS on multiple hosts in a network. On a single workstation, however, you may run only one AMS process per version of AMS.

Deciding Whether to Use AMS

In most cases, AMS is recommended over NFS or Microsoft Windows Network because of performance advantages, flexibility, and ease of use. AMS is required if you are using Objectivity / DB Data Replication Option.

Comparing AMS to NFS

Compared to NFS, AMS provides improved write (update) performance. AMS uses Objectivity / DB's caching and locking services to provide safe asynchronous writing to disk. Because NFS provides only synchronous writing, performance may be slowed by applications with many remote updates.

Under certain circumstances, NFS may provide some advantages compared to AMS. For example, if the server machine has hardware that optimizes NFS performance (such as an Auspex file server with NFS in firmware), you will not need the improved write performance of AMS.

Guidelines for Choosing AMS

You must run AMS on data-server hosts if:

- You are using Objectivity / DB Data Replication Option. Specifically, you must run AMS on every data-server host that is to store a replicated database. AMS need not be running when you create an original database image; however, you must start AMS before you can create additional database images.

You should run AMS on a data-server host if:

- Your Objectivity / DB application performs many update transactions, or modifies or creates many objects per update transaction.
- Your Objectivity / DB application performs a moderate number of update transactions and is connected to the server machine via a WAN or large LAN.
- NFS is not already in use on the data-server host.
- You prefer not to export complete NFS file systems to all users.
- You prefer not to run NFS.
- You need interoperability between Windows and UNIX platforms. While this is possible with other networking software, AMS offers a simpler solution.
- You want a simpler alternative to setting up common UNC network share names in a Windows environment.

Checking Whether AMS is Running

You can check whether AMS is running on a particular workstation. To do this, use oocheckams (page 155).

Starting AMS

You can start AMS on a Windows or UNIX data-server host. On a given host, you can run only one AMS process per version of AMS.

Windows

On Windows platforms, you start AMS using the Objectivity Network Services tool that is provided with Objectivity/DB. This causes AMS to run even when no user is logged on and to start automatically whenever the system boots. See "Starting and Stopping an Objectivity Server" on page 215.

On Windows NT or Windows 2000, you should start AMS under a special-purpose logon account; see "Specifying a Service's Logon Account" on page 216. For security reasons, you should set up the AMS logon account to have just the minimum necessary access permissions. The AMS logon account must have read and write access to:

- All system-database, database, and journal files to be accessed
- The Objectivity server system directory; see "Required File and Directory" on page 80.

UNIX

On UNIX, you start AMS using oostartams (page 194).

For security reasons, you should start AMS under a special-purpose user account that has just the minimum necessary access permissions. The AMS user account must belong to a group with read and write permissions to:

- All system-database, database, and journal files to be accessed
- All directories in which new system-database, database, and journal files will be created
- The Objectivity server system directory (`/usr/spool/objy`)

Stopping AMS

You can stop AMS if no database applications are currently using it. An error message is displayed if you try to stop AMS while database applications are using it.

Windows

On Windows, you stop AMS from the Objectivity Network Services tool.

Alternatively, you can run `oostopams.exe` (page 194) from a command prompt. On Windows 98, do not choose **End Task** while AMS is selected in the Task Manager.

UNIX

On UNIX, you stop AMS using `oostopams`.

Setting AMS Usage in an Application

By default, Objectivity/DB applications use AMS whenever possible. You can override the default behavior either to prevent or to require AMS usage. For a discussion of setting AMS usage in an application, see the documentation for the appropriate Objectivity programming interface.

Changing the TCP/IP Port for AMS

AMS is assigned a default TCP/IP port number by Objectivity/DB. This port is used by remote processes (such as Objectivity/DB applications) for communicating with AMS.

On rare occasions, you may be prevented from starting AMS because another service is already using the default AMS port. If possible, you should assign the other service to a different TCP/IP port. If you cannot do this, you can assign AMS to a nondefault port; however, you will have to make this change on *every* host that runs a process that interacts with AMS.

Windows

If you cannot start AMS, you can determine whether a port conflict exists:

1. Inspect AMS output:
   - On Windows NT or Windows 2000, open the Event Viewer from the Administrative Tools folder of the Control Panel, and look in the Application Log.
   - On Windows 98, inspect the `ooLog.txt` file in the Windows folder.
2. If a port conflict exists, the message `10048` (`WSAEADDRINUSE`) is displayed.

To assign a TCP/IP port for AMS on a Windows platform:

1. Log on as administrator (Windows NT or Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. Select AMS and click **Configure**.
4. In the TCP/IP Port Number field, enter the new port number and click **OK**. To avoid conflicts, enter a number greater than 1035.

Note: You must make this change on *every* host that runs either AMS or a process that uses AMS (an Objectivity/DB application, a lock server, or Objectivity/DB tools).

UNIX

To assign a TCP/IP port for AMS on a UNIX platform:

1. Log in as `root`.
2. Add the following entry to the TCP/IP `services` file (typically, `/etc/services`):

   `ooams-`*`xx`* *`portNumber`*`/tcp # Objectivity/DB AMS`

   where

`xx`	The current version of AMS. (For the current version, see the Objectivity Technical Support web site; call Objectivity Customer Support to get access to the site.)
`portNumber`	TCP/IP port number (a number greater than 1024).

Note: You must make this change on *every* host that runs either AMS or a process that uses AMS (an Objectivity/DB application, a lock server, or Objectivity/DB tools).

If you are using the Network Information Service (NIS), you should ask your system administrator to perform the equivalent operation for your NIS configuration.

Troubleshooting Problems With AMS

Windows and UNIX

AMS Timeout

By default, an application or tool waits 25 seconds for AMS to respond to a request. However, when running on a busy machine, AMS may need more time to respond. If an application or tool consistently signals an AMS timeout error, you can increase the timeout period by setting the `OO_RPC_TIMEOUT` environment variable to the desired number of seconds—for example:

```
set OO_RPC_TIMEOUT=50                # Windows
setenv OO_RPC_TIMEOUT 50             # UNIX
```

9

Backup and Restore

This chapter describes how to use Objectivity/DB administration tools to back up and restore a federated database. You can invoke these tools either from the command line or within a shell script, and they include options that allow you to customize backup and restore tasks for special needs. Objectivity/DB also supports full user access (read and update) during backup.

This chapter describes:

- General information about backup and restore
- Developing a backup strategy
- Backing up data
- Restoring from a backup
- Processing data during backup and restore
- Backing up to and restoring from tape

About Backup and Restore

You need to back up critical user data so it can be restored if the original data becomes unusable. Possible scenarios include:

- A catastrophic event (such as an accidental erasure or a disk crash) destroys part or all of the federated database.
- The logical state of the federated database is corrupted. Data in the federated database is incorrect and the transaction that created the problem has already committed.
- The physical state of the federated database is corrupted, making user data inaccessible.
- You want to go back to using a previous version of Objectivity/DB software after upgrading to a new release of Objectivity/DB that has a different database format.

The following sections describe important terms and concepts you should know before backing up and restoring your data.

Backup Events and Backup Sets

A *backup event* is a logical construct that represents one backup of a federated database.

A *backup set* is a separately administered group of backup events. For example, one backup set can represent an entire month of federated database backup events.

The Objectivity/DB backup and restore tools require you to reference backup events by their encompassing set names. Each backup set must have a unique name and each backup event must have a unique name within the scope of its encompassing set.

Backup Medium and Backup Volumes

The *backup medium* is the medium used to store the physical backup of a federated database. Objectivity/DB supports backup to hard disk on all platforms and backup to tape on UNIX platforms. (For non-UNIX platforms, Objectivity/DB provides a mechanism for executing your own scripts to perform tape backup.)

Each backup event is stored on one or more *backup volumes*. A backup volume is a defined portion of the backup medium that contains archived data. For example, a file that contains archived data is a backup volume. When you initiate a backup event, you specify both a volume name prefix and the backup volume capacity. The name of each backup volume consists of the volume name prefix plus a sequential numeric value. For example, if you assign the volume name prefix `myVol`, the Objectivity/DB backup tool assigns the first volume of a federated database backup the name `myVol_1`. If a second volume is generated, it is given the name `myVol_2`. Multiple volumes are generated only if the backup size exceeds the backup volume capacity.

During a backup, you cannot append data to an already existing backup volume. Consequently, each backup volume contains data from at most one federated database and at most one backup event.

Figure 9-1 shows the relationship between the medium, sets, events, and volumes. Keep in mind that sets and events are only logical constructs. The physical entities are the medium and the volumes.

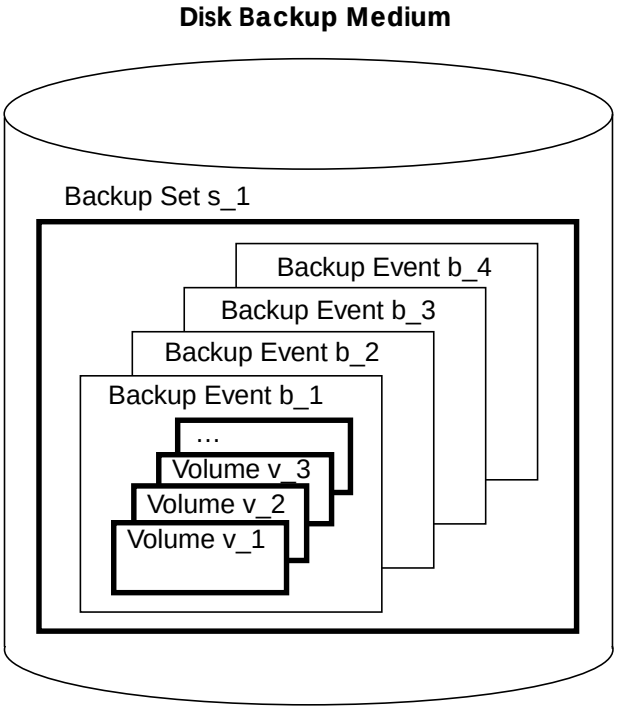

Figure 9-1 Backup Medium, Sets, Events, and Volumes

Backup Levels

Objectivity / DB supports both full and incremental backups with a system of ten user-specified *backup levels*. You specify the level of backup needed when you invoke the Objectivity / DB backup tool.

During a *full (*or *level-0) backup*, the entire contents of a federated database are archived to the backup medium. During an *incremental backup* (backup levels 1 through 9), only a specific portion of the data is archived to the backup medium: those containers that have changed since the most recent lower level backup. (The smallest portion of data that can be archived to a backup medium is a container.)

Table 9-1 summarizes which containers are backed up at each backup level.

Table 9-1: Backup Levels

Use This Level	To Back Up
0	Entire federated database (including all databases and all containers within each database regardless of whether they have been updated).
1	All containers updated since the most recent level-0 backup
2	All containers updated since the most recent backup of level-1 *or* level-0
3	All containers updated since the most recent backup of level-2, level-1, *or* level-0
n	All containers updated since the most recent backup of level-n–1, level-n–2, …, *or* level-0

Understanding Backup Levels

Figure 9-2 shows what happens if each succeeding backup has a level greater than the previous backup.

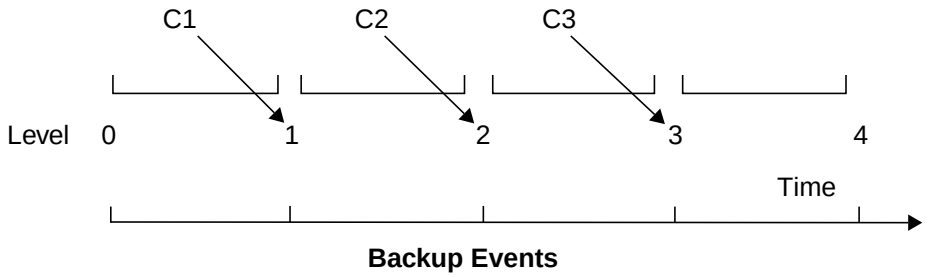

Figure 9-2 Successive Backups at Increasing Levels

In Figure 9-2, any container updated within one of the bracketed time spans is archived only during the next incremental backup. For example, container C1 is updated between the start of the time line and the time when the level-1 backup is executed. The updated contents of container C1 appear only on the backup volumes generated by the level-1 backup. Container C2 is updated in the time span between the level-1 and the level-2 backups. The contents of container C2 appear only on the backup volumes generated by the level-2 backup.

Figure 9-3 shows what happens if all incremental backups are at the same level.

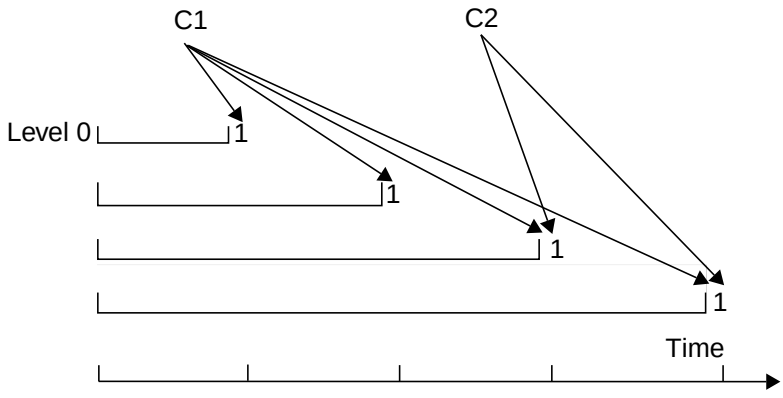

Figure 9-3 Successive Incremental Backups at the Same Level

In Figure 9-3, every incremental backup includes the contents of all containers updated since the last full backup. For example, container C1 is updated in the time period between the level-0 backup and the first level-1 backup. The contents of container C1 are included in every subsequent incremental backup.

Table 9-2 shows in another form the results of combining the strategies of Figure 9-2 and Figure 9-3. The table assumes that the backups occur after business hours on the days indicated.

Table 9-2: Successive Backups Using a Combined Strategy

Day	Level	Days' Changes Backed Up
Sunday	0	Entire federated database
Monday	1	Monday
Tuesday	1	Monday and Tuesday
Wednesday	1	Monday–Wednesday
Thursday	2	Thursday
Friday	3	Friday
Saturday	3	Friday and Saturday

Point of Restore

The *point of restore* is the point in time to which the federated database is restored. Each backup event represents a possible point of restore.

Full Restore

Whenever you initiate a restore, Objectivity / DB always restores the entire federated database even if the point of restore is represented by an incremental backup. If you specify an incremental backup as the point of restore, Objectivity / DB automatically accesses and restores data from multiple backup events (including, but not limited to, the point of restore and the last full backup before the point of restore). Enforcing a single level of restore guarantees the referential integrity of the relationships (associations) between databases and the logical integrity of the data within and across databases.

Backup Diary

When you first back up a federated database, Objectivity / DB creates a container and classes within the federated database to help administer data backups and restorations. These classes are collectively called the *backup diary*.

Objectivity / DB uses the diary to keep track of all backup and restore events, creating objects to represent each event. The diary is used internally by Objectivity / DB tools such as `ooqueryset` (page 187), and is not available for direct user access. It is archived during each backup to ensure the ability to recover from its accidental deletion or corruption.

User Access During Backup and Restore

Objectivity / DB allows full user access (both read and write) during backups. By contrast, once you initiate a restore, Objectivity / DB prevents other users from accessing the federated database until it is entirely restored.

NOTE Even if you restore the federated database to an entirely new location (in effect, making a copy), both the copy and the original are inaccessible for the duration of the restore. For directions on how you can work around this restriction, see “Allowing Restricted User Access During a Restore” on page 111.

Increased Space Requirements Due to User Access

To allow full access during a backup while maintaining data consistency, Objectivity / DB retains the old versions of any containers modified while the backup is in progress. (Normally, old versions of database containers are deleted as soon as transactions commit.) Other users access the most recent versions of the containers while the Objectivity / DB backup tool records the state of the federated database as it was when the backup began.

This has important implications for the amount of free disk space required to execute a backup. Namely, while a backup is in progress, the federated database grows in size as users update its containers. After the backup is complete, the extra versions are purged and the space is recycled for reuse the first time a process updates a class and finds that no other users have locked the container for update.

Developing a Backup Strategy

To ensure the safety of critical data, you should develop a backup strategy before problems arise. Developing a backup strategy involves:

- Estimating the disk space required to perform and store backups
- Defining a backup schedule

NOTE Always perform a full backup before a software upgrade in case you need to restore the data to its prior format because of problems with the software upgrade.

Estimating the Disk Space Required for Backups

The amount of free disk space required to execute a full, online backup of a federated database can be as much as two times (2X) the current size of the database. This figure has two components: 1X represents the space required to store the backup volumes and 1X represents the potential increase in database size while the backup is in progress.

NOTE The 1X figure given for database growth during a backup assumes that every database container is updated while the backup is in progress. This may or may not be a reasonable assumption for your federated database backup. For a discussion of database growth during an online backup, see “Increased Space Requirements Due to User Access” on page 103.

You can decrease the space required to store a backup in two ways:

- Process the backup volumes as they are produced to compress them or move them to another medium.

  Note: Objectivity / DB allows you to override the default storage capacity (1 MB) of the backup volumes. When moving backup volumes from disk to another backup medium, you can decrease the amount of disk space required for temporary storage by decreasing the backup volume capacity.
- Execute an incremental backup. Containers that have not been updated are not archived.

You can decrease the amount of database growth during a backup in two ways:

- Perform the backup when usage is light. This slows the rate of database growth.
- Execute an incremental backup. This decreases the total time required to perform the backup and thus the time during which the database can grow.

Defining a Backup Schedule

A backup schedule defines how much data is archived at what times. It provides a means of administering data backups and informs users when backups will occur. Informed users can request changes to the backup schedule to improve the security of critical data given their pattern of usage.

For a small federated database, daily full backups can be practical. For a large federated database, however, daily full backups are impractical because of the time required to complete a full backup, and the space needed to execute and store full backups. Therefore, most backup schedules involve:

- A full backup at the beginning of the week (or some other defined cycle)
- Incremental backups during the week (or cycle)

For example, as a normal routine, you might perform a full backup each Sunday and incremental backups every weekday.

Guidelines for Defining a Backup Schedule

To help you choose the length of the cycle, and the frequency and level, of the incremental backups, observe the following guidelines:

- Match the backup frequency to the rate that critical data is updated. Ideally, you should make sure that every modified container with critical user data appears on at least one backup volume.
- Start incremental backups at fairly high levels (for example, level 6) so that later in the backup cycle the lower levels (1 through 5) are available to fine tune the amount of data archived.
- Avoid frequent full backups if the amount of data to be archived is large.

Example Backup Schedules

Table 9-3 shows a sample schedule for a federated database that is updated daily during the week. The schedule reduces the risk of losing many days' updates if one of the backup volumes is damaged (This is because most modified containers appear on multiple backup volumes and because the number of events that Objectivity / DB must access during a restore is minimized.) Depending upon the size of the database, a level-0 backup could occur either at the beginning of each month or at the beginning of each week.

Table 9-3: Backup Schedule Minimizing the Risk of Losing Data

Day	Level	Days' Changes Backed Up
Sunday	1 (*or* 0)	Since last full backup (*or* full backup)
Monday	6	Monday
Tuesday	6	Monday–Tuesday
Wednesday	6	Monday–Wednesday
Thursday	6	Monday–Thursday
Friday	6	Monday–Friday

Table 9-4 shows a moderately low-risk backup schedule for an entire month under the assumption that archiving once a week to a previous level-0 backup is too time consuming. In this example, the schedule in Table 9-3 has been modified so that Sunday backups archive only the previous week's changes.

Table 9-4: Modified Low-Risk Backup Schedule Decreasing the Time Required to Perform Sunday Backups

Day	Level	Days' Changes Backed Up
End of month	0	Full backup
Weekdays of 1st week (daily)	6	1st weekday–day of backup
1st Sunday	2	Since last full backup
Monday—Friday of 2nd week (daily)	6	Monday–day of backup
2nd Sunday	3	Since 1st Sunday

Table 9-4: Modified Low-Risk Backup Schedule Decreasing the Time Required to Perform Sunday Backups (Continued)

Day	Level	Days' Changes Backed Up
Monday—Friday of 3rd week (daily)	6	Monday–day of backup
3rd Sunday	4	Since 2nd Sunday
Monday—Friday of 4th week (daily)	6	Monday–day of backup
4th Sunday	5	Since 3rd Sunday

If user access is heavy at all times during weekday business hours, then archiving the updates occurring over smaller periods of time may be necessary in order to decrease the time and space devoted to the weekday backups. The schedule in Table 9-5 illustrates such a strategy.

Table 9-5: Backup Schedule for a Heavily Used Federated Database

Day	Level	Days' Changes Backed Up
Sunday	1 (*or* 0)	Since last full backup (*or* full backup)
Monday	2	Monday
Tuesday	4	Tuesday
Wednesday	6	Wednesday
Thursday	8	Thursday
Friday	2	Monday–Friday

This schedule entails more risk than the schedule presented in Table 9-3. For example, to restore from a backup performed on Tuesday, Wednesday, or Thursday, Objectivity / DB must access backup volumes from three (Tuesday) to five (Thursday) backup events, as opposed to two for the schedule in Table 9-3. Moreover, until the Friday backup is complete, an updated container appears on only one backup volume during the week.

To contrast the safety of the two schedules, consider the following scenario:

- Federated database corruption is detected during a failure of the Friday backup.
- One of the Tuesday backup volumes is damaged.

For the schedule in Table 9-3, the restore from the Thursday backup event is successful. Only Friday updates are lost. For the schedule in Table 9-5, you must

restore from the Monday backup. Updates made from Tuesday through Friday are lost.

Backing Up Data

You back up a federated database by creating a backup set and then invoking the Objectivity / DB backup tool, as described in the following subsections.

Creating a Backup Set

All backup events must be associated with a particular backup set. You can associate a backup event with an existing backup set or you can create a new backup set using `oocreateset` (page 163).

The name of each backup set for a particular federated database must be unique. The backup set information is maintained in the backup diary as part of the federated database and is archived during each backup event to protect against its accidental deletion or corruption.

EXAMPLE This example creates a backup set named `septemberBackups` in the federated database named `mfgFd`:

```
oocreateset -set septemberBackups mfgFd
```

To execute this command when a lock server is not running or to bypass a running lock server, you use the `-standalone` option.

Performing a Backup

You back up a federated database using the Objectivity / DB backup tool, `oobackup` (page 147), with options specifying where to store the physical backup and how to identify the backup for future reference. You must include the backup set name, the backup event name, the volume name, and the full directory pathname where the backup volumes are to be stored. For an incremental backup, you must specify the backup level and you must place it in the same backup set as the associated full backup.

EXAMPLE This example invokes a full backup (level-0) of the `mfgFd` federated database to the `mfgset` backup set and names the backup event `weeklyBackup`. The storage location for the backup volumes is indicated using the `-device` option followed by the pathname `/dba/mfgFd/backups`.

```
oobackup -set mfgset -backup weeklyBackup -volume vol020492
-level 0 -device /dba/mfgFd/backups mfgFd
```

Backup Boot File

The first time you back up a federated database, you establish the *backup boot file*, which is the boot file you must specify in *all* subsequent backup and restore operations on the same federated database. When a federated database has only one boot file, that boot file is the backup boot file. When a federated database has multiple boot files, you must choose one specific boot file to be the backup boot file.

For example, you must choose a specific backup boot file for:

- A distributed federated database that has multiple copies of the same boot file on different client hosts. Only the chosen backup boot file is saved in the backup.
- (*FTO*) A partitioned federated database, where each autonomous partition has its own boot file. One boot file per autonomous partition is saved in the backup.

If a Backup Fails

Because backups of corrupted federated databases often fail, you can use routine backups as opportunities to check for database corruption. If you run `oobackup` from a backup script, make sure that the script checks the tool's return status.

The `oobackup` tool cannot recover from errors that occur during its execution. If a backup fails at any time during the backup process, you must restart the backup. An aborted backup event has no effect on the existing federated database.

Restrictions on Using oobackup

Do not run `oobackup` and `ootidy` concurrently—`ootidy` may delete objects that `oobackup` references.

Obtaining a Federated Database's Backup History

You can list all backup sets and backup events for a specified federated database. To do this, you use `ooqueryset` (page 187). To list only those backup events belonging to a particular backup set, you specify the backup set name with the `-set` option.

EXAMPLE This example lists the backup events in the backup set named `septemberBackups` for the federated database named `mfgFd`:

```
ooqueryset -set septemberBackups mfgFd
```

To execute this command when a lock server is not running or to bypass a running lock server, you use the `-standalone` option.

Deleting a Backup Set

You can delete a backup set and all the information in the backup diary about the backups associated with that set. To do this, you use `oodeleteset` (page 167).

EXAMPLE This example deletes a backup set named `septemberBackups` for the `mfgFd` federated database:

```
oodeleteset -set septemberBackups mfgFd
```

You can invoke a program or script before `oodeleteset` deletes each file in the backup set. To do this, use `oodeleteset` with the `-procfiles` option followed by the program or script name. The name of the file about to be deleted is passed to the program or script as a command-line argument. The status returned by the program or script is ignored.

Restoring From a Backup

You restore a federated database by invoking the Objectivity / DB restore tool, `oorestore` (page 188), with options that specify the point of restore and the locations to which the federated database and its associated files are to be restored. You can restore files to their original locations or to a new location. A restore operation fails if the intended file destinations are unavailable.

If the point of restore is an incremental backup, Objectivity / DB automatically accesses all the backup volumes necessary to perform the restore. However, the

necessary volumes must reside in the same location as they did when they were first created, because Objectivity/DB consults the archived backup diary to find the locations of the previous backup volumes. A restore operation fails if it is unable to locate one or more required backup volumes.

Usually you restore from the most recent backup event. However, if you want to restore a federated database because you have detected corruption, try to determine when the corruption occurred and restore from the most recent backup before the time of corruption. If you cannot determine when the corruption occurred, you can restore (to a new location so that you do not overwrite the current system-database file) from a recent backup and then test the restored federated database for corruption by dumping its contents.

If a Restore Fails

The `oorestore` tool cannot recover from errors that occur during its execution—for example, if backup volumes cannot be located or if a destination directory does not exist. If a restore fails at any time during the restore process, you must restart the restore. However, a failed restore can leave behind both files and locks. Therefore, before you restart the restore, delete any files that were restored before the failure occurred and restart the lock server.

Restoring Files to Their Original Locations

The basic way to restore a federated database is to restore its associated files—the system-database files, the backup boot file, and the database files—to the same physical locations (host and directory) where they resided when they were archived. To restore a federated database's files to their original locations:

1. Verify that the backup volumes and the original directories exist and are accessible.
2. Invoke `oorestore` (page 188) with options for specifying the backup set name, the backup event name, the volume name, and the full directory pathname of the backup event representing the point of restore. You must also specify the path of the backup boot file (the boot file that was used to create the backup).

EXAMPLE This example restores the `mfgFd` federated database from the `Monday` backup, which is a member of the `dailyBackups` backup set. The restored files are placed on the hosts and directories from which they were archived.

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups mfgFd
```

Restoring Files to a Single New Location

Sometimes you cannot restore a federated database's files to their original locations—for example, when the original hosts or directories no longer exist. In such cases, you can specify a new location for the restored federated database. When you do this:

- The backup boot file is restored to the current working directory.
- The system-database file(s) and all database files are restored to the specified location. All catalogs are updated accordingly.
- Every journal-directory attribute (one for each autonomous partition) is reset to the same specified directory.

To specify a new location for restored files:

1. Verify that the required backup volumes and the destination directory exist and are accessible.
2. Invoke oorestore (page 188) with the `-newhost` and `-newdirectory` options in addition to the options specifying the point of restore. You must also specify the path of the backup boot file (the boot file that was used to create the backup).
3. If the federated database is partitioned, move each partition's restored system-database file and boot file to an appropriate location, and set a unique journal directory for each autonomous partition (use oochange, page 149).

You can specify the `-failonly` option to restore files to the new location only if the files' original locations are inaccessible. The backup boot file is restored to the current working directory if its original location is inaccessible.

Allowing Restricted User Access During a Restore

For development purposes, you may want to restore a federated database to a new location without locking users out of the original. To do this you use oorestore with the `-newhost`, `-newdirectory`, and `-standalone` options. In effect, you make a copy that you can modify without endangering the original.

EXAMPLE This example restores the `mfgFd` federated database from the `Monday` backup, which is a member of the `dailyBackups` backup set, to directory `/mnt/newfdloc` on host `machine42`.

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups -newhost machine42
   -newdirectory /mnt/newfdloc mfgFd
```

This example restores the `mfgFd` federated database from the `Monday` backup, which is a member of the `dailyBackups` backup set. The files that cannot be restored to their original locations are restored to directory `/mnt/newfdloc` on host `machine42`.

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups -newhost machine42
   -newdirectory /mnt/newfdloc -failonly mfgFd
```

Restoring Files to Multiple New Locations

You can provide file destinations explicitly in an ASCII-text mapping file to restore the files of a federated database to multiple new locations. Files not specified in the mapping file are restored to their original locations, except for the backup boot file, which is placed in the current working directory. Normally, you restore to multiple new locations only if the original locations are unavailable and the files to be restored are too large to fit into a single directory.

To restore to multiple new locations:

1. Create a mapping file specifying the desired destination locations.
2. Verify that the required backup volumes and the destination directories exist and are accessible.
3. Invoke `oorestore` with the `-dbmap` option followed by the name of the mapping file. You cannot use this option with the `-newhost`, `-newdirectory`, or `-failonly` option.

Creating a Mapping File

To create the mapping file, you need the system names of the federated database, autonomous partitions, and databases that you want to restore to new locations. If the federated database has been corrupted or destroyed, you may need to obtain catalog information from the backup volumes themselves.

Each line in the mapping file specifies the destination location of a system-database or boot file, or a new value for a journal directory attribute. Each line consists of four strings, separated by spaces or tab characters, as follows:

```
FD fdSysName targetHost targetPath
FDJNL fdSysName targetHost targetPath
AP apSysName targetHost targetPath
APBOOT apSysName targetHost targetPath
APJNL apSysName targetHost targetPath
DB dbSysName targetHost targetPath
```

where

fdSysName	Federated database system name (*FTO*) Name of the original autonomous partition in a partitioned federation
apSysName	(*FTO*) Autonomous partition system name
dbSysName	Database system name
targetHost	Target host name
targetPath	Target path for directory or filename

EXAMPLE This example restores the `mfgFd` federated database from the `Monday` backup, which is a member of the `dailyBackups` backup set. The files are restored to the locations specified in the mapping file `/tmp/file.map`. If a file is not specified in the mapping file, it is restored to its previous location.

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups -dbmap /tmp/file.map mfgFd
```

The mapping file `/tmp/file.map` reads as follows:

```
FD mfgFd machine42 /mnt/newfdloc/fd/mfgFd.FDB
FDJNL mfgFd machine42 /mnt/newfdloc/fd
AP mfgAp1 machine42 /mnt/newfdloc/ap/mfgAp1.AP
AP mfgAp2 machine42 /mnt/newfdloc/ap/mfgAp2.AP
APJNL mfgAp2 machine42 /mnt/newfdloc/ap
APBOOT mfgAp2 machine42 /mnt/newfdloc/ap/mfgAp2
DB partsDb fafhrd /newdbloc/partsDb.mfgFd.DB
```

Obtaining Catalog Information

To display the catalog contents, including all system names, for an archived federated database, you use `oorestore` (page 188) with the `-dumpcatalog` option. You can use this information when creating a mapping file.

EXAMPLE This example displays the catalog files for the `mfgFd` federated database from the `Monday` backup, which is a member of the `dailyBackups` backup set.

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups -dumpcatalog mfgFd
```

WARNING *Do not* run `oorestore` in the directory that contains the federated database's system-database or other files. Using `oorestore` with the `-dumpcatalog` option overwrites and then deletes any system-database and related files in the current working directory.

Processing Backup Volumes

You can run custom programs or shell scripts to pre- or post-process backup volumes while a backup or restore is in progress. To do this you invoke `oobackup` (page 147) and `oorestore` (page 188) with various `-procfiles` options followed by the name of your program or shell script. Although you can use this feature with any program or shell script you choose, the most common application involves conserving disk space by:

- Compressing backup volumes as `oobackup` produces them
- Uncompressing backup volumes before `oorestore` reads them and recompressing them after `oorestore` restores them

Another common use for pre- and post-processing is to back up to and restore from tape. This conserves disk space by temporarily storing only one backup volume at a time. On UNIX systems, you should initially use the provided direct-to-tape backup scripts. Later you may want to write customized scripts. On non-UNIX systems, you need to write your own scripts for backing up to and restoring from tape.

Your custom program or script should exit with a value of zero upon successful completion, and nonzero otherwise. `oobackup` and `oorestore` terminate with an error message if the system call to run the program or shell script returns a nonzero value.

Processing Backup Volumes During a Backup

To invoke a program or script after `oobackup` completes writing each backup volume, you use `oobackup` with the `-procfiles` option followed by the program or script name. During the execution of `oobackup`, the name of the volume just written and the total size in bytes of the backup volumes written so far are passed to the program or script as command-line arguments.

Processing Backup Volumes During a Restore

To invoke a program or script before `oorestore` opens each backup volume for restore, use `oorestore` with the `-procfilesbef` option followed by the program or script name. The name of the volume about to be restored is passed to the program or script as a command-line argument.

You can also invoke the same or a different program or script after `oorestore` finishes restoring each backup volume by using the `-procfilesaft` option followed by the program or script name. The name of the volume just restored is passed to the program or script as a command-line argument.

EXAMPLE In this example, `mycompress` is a user-defined program that compresses a single file, and `myuncompress` is a user-defined program that uncompresses a single file.

The following command directs `oobackup` to compress each backup volume as soon as it is produced:

```
oobackup -set mfgset -backup weeklyBackup
   -volume vol020492 -capacity 1000 -level 0
   -device /dba6/mfgFd/backups -procfiles mycompress mfgFd
```

The following command directs `oorestore` to uncompress each backup volume before it is restored and to recompress it after it is restored:

```
oorestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups -procfilesbef myuncompress
   -procfilesaft mycompress mfgFd
```

Backing Up to and Restoring From Tape

On UNIX platforms, Objectivity/DB provides a direct-to-tape backup and restore capability using two driver programs (`ootapebackup` and `ootaperestore`), which invoke several Bourne shell scripts and either `oobackup` or `oorestore`.

The syntax and command-line options for `ootapebackup` and `ootaperestore` are identical to those for `oobackup` and `oorestore`, respectively, except that the `-procfiles` options cannot be used directly with the driver programs.

Backing up to tape requires sufficient free disk space to hold each backup volume from the time it is created to the time it is written to tape. If space is limited, you can use `ootapebackup` with the `-capacity` option to decrease the size of the backup volumes. (The default size is 1 MB.)

Configuring ootapebackup and ootaperestore

Before backing up to and restoring from tape, you must configure `ootapebackup` and `ootaperestore` to fit your UNIX environment. To do this, you must modify the following shell scripts provided by Objectivity/DB:

- `oobackf`
- `ooendb`
- `ooendr`
- `oorestfa`
- `oorestfb`
- `oostrtb`
- `oostrtr`

These shell scripts (as well as `oobackup` and `oorestore`) must reside in a directory in your path so that `ootapebackup` and `ootaperestore` can find them.

Perform the following steps in each script:

1. Locate the configuration variables for your environment. These are defined near the beginning of each file between two comment blocks that delineate the configuration section.
2. Set the variable `dev` equal to the pathname of the tape device. Note that the name of the tape device is *not* the same as the path to the disk directory where the backup volumes are stored temporarily, which is specified by the `-device` option to `ootapebackup` and `ootaperestore`.
3. Uncomment the configuration information for your platform.

Backing Up to Tape

First, `ootapebackup` invokes the Bourne shell script `oostrtb` to prepare the tape device for the backup. Next, `ootapebackup` invokes `oobackup`, which creates a series of backup volumes, writing each first to disk, then moving each to tape using the Bourne shell script `oobackf`. After `oobackup` finishes, `ootapebackup` resets the tape drive by invoking the Bourne shell script `ooendb`. If `oobackup` or one of the shell scripts terminates with an error, `ootapebackup` exits immediately.

When backing up to tape, you must place the backup volumes for all incremental backups on the same tape as the backup volumes for the associated full backup. The volume name prefixes for distinct backup events stored on the same tape must be different.

EXAMPLE This command backs up the `mfgFd` federated database to tape.

```
ootapebackup -set mfgset -backup weeklyBackup
   -volume vol020492 -capacity 1000 -level 0
   -device /dba/mfgFd/backups mfgFd
```

Restoring From Tape

First, `ootaperestore` invokes the Bourne shell script `oostrtr` to prepare the tape device for the restore. Next, `ootaperestore` invokes `oorestore` with the `-procfilesbef` and the `-procfilesaft` options designating the `oorestfb` and `oorestfa` shell scripts, respectively. During the execution of `oorestore`, each backup volume is read unopened from tape to disk by `oorestfb`, restored by `oorestore`, and then removed from disk by `oorestfa`. After `oorestore` finishes restoring all volumes, `ootaperestore` resets the tape drive and cleans up by invoking the Bourne shell script `ooendr`.

If `oorestore` or one of the shell scripts terminates with an error, `ootaperestore` exits immediately.

EXAMPLE The following command restores the `mfgFd` federated database from tape.

```
ootaperestore -set dailyBackups -backup Monday -volume fdbVol
   -device /fdb/backups mfgFd
```

10

Automatic and Manual Recovery

Recovery is the process of restoring a federated database to a consistent state after a transaction fails to commit. Depending on the nature of the failure, recovery is performed by the application that started the transaction or through one of several automatic recovery mechanisms that you enable.

This chapter describes:

- General information about Objectivity / DB recovery mechanisms
- Enabling automatic recovery from application, client-host, and lock-server failures
- Performing manual recovery when automatic recovery is not possible

You can also create a special-purpose recovery application using the Objectivity / C++ programming interface. For more information, see the Objectivity / C++ programmer's guide.

About Recovery

Recovery is required when a transaction updates data and then terminates without committing the changes. Objectivity / DB performs recovery by rolling back the transaction's uncommitted changes. This restores the federated database to the logical state it was in before the transaction started. Objectivity / DB uses the information recorded in one or more journal files to roll back changes. If the lock server is still running, Objectivity / DB instructs it to release all locks held by the transaction; if the lock server has stopped, its locks are lost, so there are no locks to release.

An executing Objectivity / DB application initiates recovery automatically whenever it aborts a transaction. An application may abort a transaction directly through a function call or indirectly through a signal or exception handler. Thus, if an Objectivity / C++ or Objectivity for Java application catches an interrupt during a transaction (for example, a user enters Control-c on UNIX), the default Objectivity / DB signal handler aborts the transaction and rolls it back. An

Objectivity / Smalltalk application must provide an exception handler that aborts the transaction.

Sometimes a transaction terminates due to an application, client-host, or lock-server failure that does not result in an abort. For example, the failure may produce a signal or exception that the application cannot handle, or it may simply stop execution, preventing the abort from taking place. After such a failure, recovery must be performed by an Objectivity / DB process other than the one that started the transaction.

You can enable various Objectivity / DB processes to initiate recovery automatically after application, client host, and lock-server failures, as described in the following sections. When automatic recovery is enabled, manual recovery is normally necessary only when an Objectivity / DB host becomes permanently unavailable after a failure.

Automatic Recovery From Application Failures

Objectivity / DB applications can fail in a number of ways, leaving incomplete transactions on a federated database. Such failures are caused by:

- Network interruption
- A Windows application issuing an `ExitProcess` or `TerminateProcess`
- Quitting a debugging session during a transaction
- A UNIX `SIGKILL` signal, for example, from a `kill -9` command

You set up automatic recovery from application failures by programmatically enabling each Objectivity / DB application to initiate recovery at the start of its first transaction. That is, the first time a recovery-enabled application tries to open a federated database, Objectivity / DB determines whether any incomplete transactions have been left on that federated database by previous application failures on the same client host. If so, Objectivity / DB automatically rolls back the leftover transactions before opening the federated database.

During the rollback, other active transactions using the same lock server may experience delays, or may be unable to connect to the lock server. Once the rollback is complete, access to all Objectivity / DB data is restored and the federated database is in a consistent state.

You explicitly enable automatic recovery in your application. For a discussion of automatic recovery from application failures, see the documentation for the appropriate Objectivity programming interface.

Automatic Recovery From Client-Host Failures

Client hosts can fail in ways that prevent an application's `exit` or termination signal from being caught—for example, when:

- The client host runs out of disk space.
- The client host loses power.

When a client host fails in one of these ways, Objectivity / DB applications running on the host may leave incomplete transactions on one or more federated databases.

You set up automatic recovery from client-host failures using the techniques described in the following subsections, as appropriate to your host (Windows or UNIX).

If a client host becomes permanently unavailable, incomplete transactions cannot be recovered automatically. To recover from this situation, see "Manual Recovery From Client-Host Failures" on page 126.

Windows Hosts

You set up automatic recovery from a Windows client-host failure by using one or both of the following techniques:

- Enabling automatic recovery in each Objectivity / DB application that you run on the client host (see page 120). Recovery is initiated by the first application you restart after the client host reboots.
- Configuring the client host to run `oocleanup` (page 156) whenever a user logs in.

To configure a Windows client host to run `oocleanup`:

➤ For each federated database that is accessed by applications running on the client host, add the following command to the Startup program folder:
`oocleanup -local` *`bootFilePath`*

The `oocleanup` tool causes Objectivity / DB to roll back any incomplete transactions that exist on the specified federated databases. If the lock server is still running, the locks held by these transactions are released. If the lock server is not running, the `oocleanup` command will fail because it requires the `-standalone` option to run without a lock server; in this case, you can run `oocleanup` from a command prompt.

If the client host is also the lock-server host, then rebooting the host restarts the lock server automatically. You should configure the lock server to initiate recovery for every federated database accessed by an application that runs on the host (see "Performing Recovery at Lock-Server Startup" on page 123). Adding `oocleanup` to the Startup program folder is redundant but not harmful.

UNIX Hosts

You set up automatic recovery from a UNIX client-host failure by configuring the host to run `oocleanup` (page 156) whenever the system reboots. To do this:

- For each federated database that is accessed by applications running on the client host, add the following command to the startup script (usually `/etc/rc.local`):

  `oocleanup -local` *`bootFilePath`*

The `oocleanup` tool causes Objectivity/DB to roll back any incomplete transactions that exist on the specified federated database. If the lock server is still running, the locks held by these transactions are released; if the lock server is not running, the `oocleanup` command will fail because it requires the `-standalone` option to run without a lock server.

If the client host is also the lock-server host, you should check whether the host's startup script runs the lock server:

- If the startup script runs the lock server, you should configure the lock server to initiate recovery for every federated database accessed by an application that runs on the host (see "Performing Recovery at Lock-Server Startup" on page 123). You do not need to add the `oocleanup` command to the startup script.
- If the startup script does not run the lock server, you should either add the lock server to the script or add the `oocleanup` command with the `-local` and `-standalone` options.

Automatic Recovery From Lock-Server Failures

If a lock server or its host fails while transactions are in progress, the failure leaves incomplete transactions on one or more federated databases. You initiate automatic recovery from lock-server failures by restarting the lock server. The way you start the lock server determines when recovery is performed, as described in the following subsections.

If a lock-server host becomes permanently unavailable, incomplete transactions cannot be recovered automatically. To recover from this situation, see "Manual Recovery From Lock-Server Host Failures" on page 127.

When a lock server stops, the locks it manages are lost, so recovery just rolls back uncommitted changes.

Performing Recovery at Lock-Server Startup

If you are using a standard lock server (not an in-process lock server), you can cause automatic recovery to be performed immediately after the lock server restarts. To do this:

- Start the standard lock server with one or more boot-file names as arguments (see page 81).

  You can specify boot files for any or all of the federated databases that are serviced by the lock server. You must specify each boot-file name in full host format, even if it is local—that is, you must specify fully qualified paths of the form *`hostName::fullLocalPath`*.

When the lock server starts, Objectivity / DB rolls back all incomplete transactions on the specified federated databases. New transactions may experience a delay until recovery is complete.

This technique is recommended in a distributed environment because you can ensure that the lock server gets boot file pathnames it can resolve. Furthermore, if the lock server cannot resolve a pathname, you get an error message when the lock server starts.

Performing Recovery When Locks are Requested

You can delay the automatic recovery of each serviced federated database until data is requested from it. To do this:

- Start the standard lock server *without* specifying boot-file names (see page 81) *or* start an in-process lock server (see page 82).

Recovery is initiated on a particular federated database when its boot-file name is passed by the application to the lock server. Thus, the first time a federated database is opened by an application after the lock server restarts, Objectivity / DB rolls back any incomplete transactions on that federated database.

If the lock server services multiple federated databases and you specify only some of them explicitly, the specified federated databases are recovered when the lock server is restarted. Each of the remaining, unspecified federated databases is recovered the next time an application opens it for a transaction.

Each federated database is recovered only once during the lifetime of the lock-server process. If a federated database is recovered when the lock server starts, it will not be recovered again when an application opens it.

Access Required by the Lock Server

The lock server must be able to access all the files for each federated database being serviced. Therefore, you must ensure that:

- The lock-server host can access all file systems containing the relevant files.
- The lock server has appropriate permissions to the relevant files—specifically:
  - Read access to the boot file
  - Read and write access to the system-database and database files
  - Read and write access to the journal directory and journal files

Setting Up Recovery in Mixed Environments

The lock server must be able to resolve the boot file pathnames that you specify as arguments or that an application passes to it. The boot file pathname passed by an application is taken from the member function or method that opens the federated database, and expanded, if necessary, to the form *host*::*full_path*. In all cases, the lock server must be able to resolve the name of the federated database listed in each of the specified boot files, as well as the names of the database files listed in the federated database catalogs and journal directory.

In general, the lock server consults the local file system to resolve the names of local files (or remote Windows Network files referenced by UNC share names). The lock server contacts either AMS or NFS on remote hosts to find all other remote files.

When Objectivity/DB is distributed among network nodes running different operating systems, you should consider the following guidelines to enable the lock server to access all of the files it needs:

- Always start a standard lock server with one or more explicitly specified boot file pathnames. This way you can ensure that the lock server gets a pathname it can resolve. Furthermore, if the lock server cannot resolve a pathname, you get an error message when the lock server starts.
- Set up all data servers to run either AMS or NFS.
  - If data servers run AMS (recommended), you can use host-specific pathnames in your application and federated database catalogs.
  - If data servers run NFS, you can use NFS pathnames uniformly in all applications and federated database catalogs.
- If you set up Windows data servers to use Windows Network without AMS or NFS:
  - Run the lock server on a Windows node (preferably Windows NT or Windows 2000) that recognizes the UNC share names used by your Windows applications. Do not use a UNIX node as the lock-server host because it will not be able to resolve the UNC share names.

- When running the lock server on a Windows NT or Windows 2000 node, be sure to start the lock server under a logon account that has permissions to use the UNC share names.

Performing Manual Recovery

In rare circumstances, Objectivity / DB may not be able to recover a federated database automatically. In these situations, you can recover a federated database manually using the procedures described in this section. Possible scenarios that require manual recovery after a failure are summarized in Table 10-1.

Table 10-1: Manual Recovery Scenarios

Scenario	Manual Recovery Needed
An application opened the federated database with automatic recovery disabled, and a failure occurs.	Run `oocleanup` on the client host (see page 126).
A client host with no Objectivity/DB files becomes permanently unavailable (perhaps due to a disk failure or other hardware problem).	Run `oocleanup` from another host, or stop and restart the lock server (see page 126).
The lock-server host becomes permanently unavailable.	Run `oocleanup` (see page 127), or start a new lock server on a host that you have renamed to the hostname of the failed lock-server host.
A data-server host with Objectivity/DB files becomes permanently unavailable.	Restore the federated database from a backup archive (see page 109).

For most manual recovery tasks, you will run `oocleanup` (page 156), which uses journal files to roll back the uncommitted changes made by incomplete transactions. If the lock server is still running, `oocleanup` also releases all the locks held by the incomplete transactions.

You must run `oocleanup` with a user identifier that has read access to the boot file, and read / write access to:

- Journal files and the directories that contain them
- All system-database and database files

Whenever possible, you should run `oocleanup` on the same machine as the process that started the transaction to be recovered. This allows Objectivity / DB to

verify that the process is not active, which prevents recovering an active transaction.

WARNING `oocleanup` may not successfully roll back a transaction if the transaction terminated while databases were being deleted. Under these circumstances, the federated database may remain in an inconsistent state. Currently, the only way to recover from such a situation is to restore the federated database from backup archives.

Manual Recovery From Application Failures

If automatic recovery is enabled when an application leaves incomplete transactions after terminating abnormally, the next transaction started on the client host automatically rolls back the incomplete transactions. If, however, automatic recovery is not enabled in the applications you run on that client host, you must manually recover all incomplete transactions that were started by local processes.

To manually recover all incomplete transactions that were started by local processes, use `oocleanup` (page 156) with the `-local` option. To recover a specific incomplete transaction, use `oocleanup` with the `-transaction` option.

EXAMPLE In this UNIX example, `oocleanup` recovers the transaction `357254` for the federated database named `mfgFD`.

```
oocleanup -transaction 357254 mfgFD
```

`oocleanup` checks to make sure the process that owns the transaction is terminated before performing the recovery.

Manual Recovery From Client-Host Failures

If a client host fails, but not permanently, you can manually recover incomplete transactions using `oocleanup`, or you can initiate automatic recovery by starting recovery-enabled applications on the client host.

If the host becomes permanently unavailable, and the host does not contain Objectivity/DB files, you can run `oocleanup` (page 156) from another host, with the `-deadowner` and `-transaction` options. Alternatively, you can stop and restart the lock server with one or more boot-file names to trigger automatic recovery (see “Performing Recovery at Lock-Server Startup” on page 123).

If the host becomes permanently unavailable, and the host contains federated-database files, you must restore the federated database from a backup archive.

Manual Recovery From Lock-Server Host Failures

If a lock-server host fails, but not permanently, you do not need to perform manual recovery; you should restart the lock server with one or more boot-file names to trigger automatic recovery (see "Performing Recovery at Lock-Server Startup" on page 123).

If a lock-server host becomes permanently unavailable, you must recover incomplete transactions manually. To recover when a lock-server host becomes permanently unavailable, you:

1. Use `oocleanup` (page 156) with the `-standalone` option. This option allows `oocleanup` to run without a lock server.

   Note: When a lock server stops, the locks it manages are lost, so recovery just rolls back uncommitted changes.
2. Change the lock-server host for the federated database (see page 84).
3. If necessary, start the lock server on the new lock-server host (see page 81).

Alternatively, you can start a new lock server on a host that you have renamed to the hostname of the failed lock-server host.

Manual Recovery From oocleanup Failures

Objectivity / DB allows only one recovery activity to occur against any given federated database at a time. To accomplish this, Objectivity / DB places a *recovery lock* on a federated database—each time a recovery operation is run, a file named `oorecvr.LCK` is placed in the journal directory and is deleted when recovery is complete. While this file exists, all other attempts to run recovery against the same federated database will fail.

If an `oocleanup` process fails, the `oorecvr.LCK` file may be left behind. To recover from this failure, you delete this file by running `oocleanup` with the `-resetlock` option.

11

Working With Distributed Databases

You can distribute an Objectivity / DB system among the nodes in various kinds of network environments. This chapter provides information you should consider when setting up a distributed Objectivity / DB system, including:

- Elements of a distributed environment
- Considerations for using Windows hosts
- Summary of Objectivity / DB usage in a mixed network environment

Elements of a Distributed Environment

When you distribute Objectivity / DB applications and databases in a network environment, you make choices about where to put various Objectivity / DB elements. This chapter uses the following terms to refer to the various nodes in a distributed Objectivity / DB system:

Client host	Network node that runs an Objectivity / DB application; sometimes called an application host
Data-server host	Network node that provides data storage; location of system-database, database, and journal files
Lock-server host	Network node that runs an Objectivity / DB lock server

Accessing a database or federated database across a network may add significantly to response time because of network overhead and contention. Regardless of the size and configuration of a database, an application's speed is likely to improve significantly if you store databases locally to your application.

For information about specifying local and remote Objectivity / DB files, see "Specifying Remote and Local Files" on page 29.

Using Windows Hosts

You can run Objectivity / DB applications on a TCP / IP network that includes nodes running Windows 98, Windows NT, and Windows 2000.

TCP / IP is included in Windows. See the *Installation and Platform Notes for Windows* for information about configuring TCP / IP in preparation for Objectivity / DB.

Windows Data-Server Hosts

Windows nodes can be data-server hosts for one or more Objectivity / DB files (system-database, database, and journal files). Various restrictions apply, depending on the kind of client hosts you plan to use.

The way you share files determines how filenames should be specified to tools and applications that access the federated database and update its catalogs. For information about the formats for specifying files on Windows data-server hosts, see “Specifying Remote and Local Files” on page 29.

Serving Windows and UNIX Client Hosts

You can run either of the following on a Windows data-server host to make Objectivity / DB files available to both Windows and UNIX client hosts:

- Objectivity / DB’s Advanced Multithreaded Server (AMS). AMS is required for accessing replicated databases created with Objectivity / DRO.
- Network File System (NFS). Call Objectivity Customer Support for a list of NFS products that have been tested with Objectivity / DB.

Any Windows or UNIX client host can access Objectivity / DB files on AMS or NFS data-server hosts. AMS is recommended over NFS on Windows data servers because it improves update performance and it simplifies the specification of pathnames for distributed Objectivity / DB files.

Serving Only Windows Client Hosts

If Objectivity / DB files will be accessed exclusively by applications on Windows client hosts, you can choose AMS, NFS, or Microsoft Windows Network to make these files available from Windows data-server hosts. AMS is recommended over Windows Network because it simplifies the pathname specification for Objectivity / DB files.

Sharing Files Using UNC Names

If you choose to use Windows Network to share Objectivity / DB files residing on Windows data-server hosts, you must refer to these files using Universal Naming Convention (UNC) network share names. You may not use virtual drive mappings. UNC names are of the form `\\`*`host`*`\`*`sharedDirectoryName`*`\`*`path`*.

When you use UNC names in a tool or application, you must also specify the file host using the literal string `oo_local_host` (see page 32).

When specifying names to be entered in a catalog (for example, system-database names, database names, or journal-directory names), you may not mix UNC share names with non-UNC names. If any such name is specified as a UNC share name, you must specify all names that way, even if you are running AMS or NFS in addition to Windows Network.

Windows Client Hosts

You can run Objectivity / DB database applications on Windows nodes.

Access to Data-Server Hosts

Applications running on a Windows client host can access data on:

- Any kind of data-server host running AMS or NFS
- A Windows data-server host running Windows Network instead of AMS or NFS

Restriction on Windows Applications Accessing NFS

Every application that accesses an NFS data-server host from a Windows client host has a user ID of 100. You must ensure that user 100 has the desired access permissions for Windows client host access.

Lock-Server Hosts

You can run the lock server on any Objectivity / DB-supported platform, including Windows hosts. However, when Objectivity / DB is distributed among network nodes running different operating systems, you should follow the guidelines for locating the lock server in “Setting Up Recovery in Mixed Environments” on page 124. These guidelines enable the lock server to locate Objectivity / DB files when performing automatic recovery.

Boot-File Location

Database applications and lock servers use boot files to find a federated database's (or autonomous partition's) system-database files. In principle, only one boot file is necessary per federated database or partition. However, in networks that contain both Windows and non-Windows client hosts, it is sometimes more convenient to maintain several copies of the boot file—for example, one copy on a UNIX machine for UNIX client hosts to share, and one copy on a Windows machine for Windows client hosts to share. This allows applications to use client-specific filenames to refer to the boot file.

Mixed Environments: Summary

Table 11-1 summarizes the possible combinations of platforms (Windows and UNIX) in a mixed Objectivity/DB environment. Use this table to choose appropriate hosts for Objectivity/DB applications, lock servers, and files (system-database, database, and journal files).

When creating files and managing the Objectivity/DB catalog, be sure to use the host and path formats specified in the table for each particular file server platform.

Table 11-1: Mixed Environment Platform Combinations

Data-Server Hosts	Catalog Entry Format	Allowed Client Hosts	Allowed Lock-Server Hosts
Windows using AMS[a]	`host, c:\pathName`	Any	Any[b]
Windows using NFS[c]	`host, /c/pathName`	Any	Any[b]
Windows using Windows Network	`oo_local_host, \\host\pathName`	Windows only	Windows only
UNIX using AMS[a] or NFS	`host, /pathName`	Any	Any[b]

a. AMS is recommended.
b. In a mixed environment, you should enable automatic recovery by starting the lock server with the boot file path as an argument. See "Setting Up Recovery in Mixed Environments" on page 124.
c. The pathname to access a file varies among NFS vendors.

12

Deploying to End Users

Deploying an application is the process of making the application available to end users. Deploying an Objectivity / DB application typically results in the production of distribution media containing an Objectivity / DB application executable, various runtime libraries and database administration tools, the federated database to be used by the application, and an installation program.

Because Objectivity / DB applications vary widely, this chapter provides general guidelines and topics for consideration, rather than prescribing a single definitive set of steps. In particular, this chapter describes:

- Building the application executable (C++ applications)
- Distributing appropriate Objectivity / DB executables
- Distributing appropriate Objectivity / DB and third-party libraries for deployment on Windows and UNIX platforms
- Setting up the end-user site
- Installing a federated database

This chapter does not describe general deployment tasks, such as how to package software on a distribution medium or how to write an installation program.

Building C++ Applications for End Users

To build C++ applications for end users, follow the compiling and linking guidelines described in the *Installation and Platform Notes* for your operating system.

Distributing Objectivity Executables

You typically deploy an Objectivity / DB application with a number of additional executables, generally including custom administration tools, redistributed Objectivity / DB administration tools, and redistributed Objectivity / DB servers. If you plan to distribute any Objectivity executables, you must do so in accordance with your Objectivity runtime-licensing agreement.

Executables You May Distribute

The table in "Reference Index" on page 142 lists executables for Objectivity / DB tools. Depending on your application's requirements, you may redistribute these tools except as noted in the table and summarized in the following sections. You may consider redistributing the Objectivity / DB executables for:

- Administrative tools
- The lock server
- The Advanced Multithreaded Server (AMS)

NOTE The lock server is not required for standalone applications (single-user applications providing database access through a single thread). However, if you do not deploy the lock server, you *must* guarantee that only one thread has access to the federated database at any time, or data corruption may occur.

If the deployed application also uses:

- Objectivity / FTO, you should consider distributing that product's executables for tools that manage autonomous partitions.
- Objectivity / DRO, you *must* distribute the Objectivity / DB executables for AMS, and you should consider distributing the executables for Objectivity / DRO tools.
- Objectivity for Java or Objectivity / Smalltalk, you should distribute the Objectivity for Java or Objectivity / Smalltalk garbage-collection tool.

Executables You May Not Distribute

Because you cannot license your end users for database development, you are not permitted to distribute the Objectivity / DB executables for:

- Browsing a federated database: `oobrowse`, `ootoolmgr`
- Debugging a federated database: `oodebug`
- Creating a new federated database or schema: `oonewfd`, `ooconfig`, `ooddlx`
- Dumping or loading a federated database's data: `oodump`, `ooload`

Distributing Libraries (Windows)

For Deployed Applications

All Objectivity / DB applications deployed on Windows platforms are linked dynamically with the Objectivity / DB kernel. This means all applications written using the Objectivity / C++, Objectivity for Java, and Objectivity / Smalltalk interfaces.

When deploying a dynamically linked Objectivity / DB application on a Windows platform, you must:

- Redistribute the Objectivity / DB DLL `oodb`*xx*`.dll` (the characters *xx* represent version numbers). You must purchase a runtime license from Objectivity to redistribute this library.
- Redistribute the version of `oochk`*xx*`.dll` that enables the features used by the deployed application:
  - The version installed with Objectivity / DB enables Objectivity / DB and Objectivity / C++ features.
  - The version installed with Objectivity / FTO enables Objectivity / DB, Objectivity / C++, and Objectivity / FTO features.
  - The version installed with Objectivity / DRO enables Objectivity / DB, Objectivity / C++, Objectivity / FTO, and Objectivity / DRO features.

For Redistributed Objectivity Executables

Objectivity executables for tools and servers are dynamically linked with the Objectivity / DB kernel. Therefore, any Objectivity executables you redistribute require the same DLLs used by deployed applications.

Distributing Libraries (UNIX)

For Deployed Applications

Most Objectivity / DB applications deployed on UNIX platforms are linked dynamically with the Objectivity / DB kernel. These include:

- All applications written against the Objectivity for Java and Objectivity / Smalltalk interfaces
- Dynamically linked applications written against the Objectivity / C++ interface

When deploying a dynamically linked Objectivity / DB application on a UNIX platform, you must:

- Redistribute the shared Objectivity / DB library for the deployment architecture. This library is provided in the *installDir/arch/*`lib` directory of your Objectivity / DB development environment. You redistribute this library under the terms negotiated in your Objectivity runtime-licensing agreement.
- Ensure that the end-user environment has an appropriate C++ runtime library. You normally do not need to redistribute this library, because it is usually provided with the operating system.

For Redistributed Objectivity Executables

Objectivity executables for tools and servers are dynamically linked with the Objectivity / DB kernel. Therefore, if you redistribute any Objectivity executables with your application, you must:

- Redistribute the shared Objectivity / DB tools library for the deployment architecture. This library has `_tools` embedded in its name and is provided in the *installDir/arch/*`lib` directory of your Objectivity / DB development environment. You redistribute this library under the terms negotiated in your Objectivity runtime-licensing agreement.
- Ensure that the end-user environment has an appropriate C++ runtime library. You normally do not need to redistribute this library, because it is usually provided with the operating system.

Setting Up the End-User Site

Setting up an end-user site for a deployed Objectivity / DB application is similar to setting up an Objectivity / DB development site. You may find it helpful to consult the *Installation and Platform Notes* for the appropriate operating system.

Hardware Requirements

Before deploying your application, you should establish the hardware requirements through testing and performance tuning.

Software Requirements

The end-user environment must have Winsock-compatible TCP / IP software installed, even for standalone (single-user) Objectivity / DB applications. On Windows platforms, Microsoft TCP / IP is normally installed with the operating system.

If NFS has been chosen as the data-server software for accessing data on remote Windows hosts, the end user may need to install separately purchased NFS software on those hosts.

Objectivity/DB Setup (Windows)

You can create a setup program similar to the setup program used for installing Objectivity products (which is created with InstallShield). If you are installing the lock server or AMS at your end-user site, the appropriate executables (`ools.exe` or `ooams.exe`) must be installed as services. You can accomplish this using the standard capabilities of a commercial setup product such as InstallShield. If you are creating your own setup program on Windows NT or Windows 2000, you can use the Service Controller (`sc`) tool provided with the Windows Resource Kit.

Objectivity/DB Setup (UNIX)

When you install your application and any Objectivity executables on a UNIX platform, you may need to perform additional setup steps:

- On some architectures, you may need to set environment variables to enable redistributed Objectivity / DB tools to operate correctly in an end-user environment. See *Installation and Platform Notes for UNIX*.
- If you are installing the lock server at the end-user site, see the steps for setting up the lock server in *Installation and Platform Notes for UNIX*.
- If you are installing AMS or using NFS at the end-user site, see the steps for setting up data-server software in *Installation and Platform Notes for UNIX*.

Installing a Federated Database

To install an existing federated database at an end-user site:

1. Use `oocopyfd` (page 162) to copy all the files of the federated database to one directory.
2. Copy the files to the distribution media (CD-ROM or tape) and distribute them to your end users. No preparation of the federated database is required for the installation procedure.
3. At the end-user site:
   a. Load all executables and libraries from the distribution media; install and start the lock server.
   b. Load the federated database's files from the distribution media and run `ooinstallfd` (page 175) with the appropriate options.

The `ooinstallfd` tool requires that all files of the federated database be located in the same directory. When database files are very large, however, this may not be possible. In this case, you can run `ooinstallfd` with the `-nocheck` option. This allows `ooinstallfd` to run to completion, printing a warning for each database file it could not locate. You must then update the catalog for each of the missing files using `oochangedb` (page 152) with the `-catalogonly` option.

Part 2 REFERENCE

13

Tools

This chapter describes the Objectivity/DB administration tools. See:

- "Overview of Administration Tools" on page 24 for a summary of the kinds of tasks you can perform using Objectivity/DB administration tools
- "Reference Index" on page 142 for an alphabetical list of tools
- "Reference Descriptions" on page 144 for complete syntax and usage descriptions of tools

Tool Names

The names of the tools are the same on Windows and UNIX, with the exception that the filenames of the executables have an extension (`.exe`) on Windows 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, and in most cases, options and arguments are lower case. Be sure to type options and arguments using the correct case.

Reference Index

Tool	Brief Description
Objectivity Network Services	Starts Objectivity servers (on Windows).
ObjyTool	Starts administration tools (on Windows).
`ooattachdb`	Attaches a database to a federated database.
`oobackup`	Archives a federated database.
`oobrowse`	Browses objects and types, and makes queries (on Windows). *For development only; you may not redistribute.*
`oochange`	Displays or changes the attributes of a federated database or autonomous partition.
`oochangedb`	Displays or changes the attributes of a database or database image.
`oocheckams`	Checks whether AMS is running on a system.
`oocheckls`	Checks whether a lock server is running on a system.
`oocleanup`	Rolls back transactions that have terminated abnormally.
`ooconfig`	Creates a DDL processor for your compiler (Objectivity/DDL on UNIX only). *For development only; you may not redistribute.*
`oocopydb`	Copies a database.
`oocopyfd`	Copies a federated database.
`oocreateset`	Creates a backup set for a federated database.
`oodebug`	Provides commands for inspecting and editing a federated database. *For development only; you may not redistribute.*
`oodeletedb`	Deletes a database from a federated database.
`oodeletefd`	Deletes a federated database.
`oodeleteset`	Deletes a backup set.
`oodump`	Creates an ASCII text file representing a federated database. *For development only; you may not redistribute.*
`oodumpcatalog`	Lists all the files in a federated database.
`oofile`	Displays information about a database or federated database.

Tool	Brief Description
oogc	Deletes unreferenced objects in a federated database (Objectivity for Java and Objectivity/Smalltalk only).
ooinstallfd	Installs a remote federated database.
ookillls	Kills a lock server.
oolistwait	Lists waiting transactions.
ooload	Creates objects from an ASCII text file. *For development only; you may not redistribute.*
oolockmon	Lists all processes and locks currently managed by a lock server.
oolockserver	Starts a lock-server process for a federated database.
oonewdb	Creates a new database in a federated database.
oonewfd	Creates a new federated database. *For development only; you may not redistribute.*
ooqueryset	Queries a federated database for existing backup sets.
oorestore	Restores an archived federated database.
ooschemadump	Writes a federated database's evolved schema to a file. *For development only; you may not redistribute.*
ooschemaupgrade	Applies an evolved schema to a target federated database.
oostartams	Starts the Advanced Multithreaded Server (AMS).
oostopams	Terminates AMS.
ootidy	Consolidates a fragmented federated database or database.
ootoolmgr	Browses objects and types, and makes queries (on UNIX). *For development only; you may not redistribute.*

Reference Descriptions

Objectivity Network Services

A graphical interface on Windows platforms for starting, stopping, and configuring Objectivity servers, such as the lock server and AMS.

Discussion

To invoke the Objectivity Network Services tool:

1. Log on as administrator (Windows NT or Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. See Appendix A, "Running Objectivity Servers on Windows," for information about using this tool.

ObjyTool

A graphical interface on Windows platforms for starting administration tools.

Discussion

To use ObjyTool:

1. Click **Start** and point to **Programs**. In the Objectivity submenu, select **ObjyTool**.
2. Click to open a menu of tools. For example:
   - To run a backup / restore tool, click **Backup**.
   - To run an administration tool, click **Administration**.
3. Choose a tool from the menu.
4. In the dialog box that appears, type the desired options and arguments for the selected tool. For help with syntax, type `-help` in the dialog box. To save tool output to a file, select **File > Save Buffer** to specify the filename.

ooattachdb

Attaches a database to a federated database.

```
ooattachdb
     (-db dbSysName -id oid [-host hostName] -filepath path)|
                 (-dbmap mapFile)
     [-standalone]
     [-readonly]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *dbSysName*

System name with which the database is to be attached. If a database with this system name already exists in the federated database, the `ooattachdb` tool terminates after issuing an error message.

`-id` *oid*

Identifier with which the database is to be attached, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

If a database with this identifier already exists in the federated database, the `ooattachdb` tool terminates after issuing an error message.

`-host` *hostName*

Name of the host system where the database file is stored. The default value is the current host.

If the `-filepath` option specifies a Windows UNC share name, the *hostName* value is automatically set to the literal string `oo_local_host`.

`-filepath` *path*

Pathname (including the filename) of the database file on host *hostName*. If the `-host` option is used to designate a remote system, *path* must be fully qualified, not relative. The database file remains in the directory specified by this option.

`-dbmap` *mapFile*

ASCII text mapping file that specifies a group of databases to be attached. This option replaces the `-db`, `-id`, `-host`, and `-filepath` options. The mapping file contains one line for each database to be attached. Each line has the format:

targetDbID targetDbSys hostName filepath

where

targetDbID	Database identifier with which the database is to be attached
targetDbSys	System name with which the database is to be attached
hostName	Host system where the database file is located
filepath	Pathname (including the filename) where the database file is located

The `ooattachdb` tool leaves each database file in the directory specified by *filepath*.

If an error is detected in the line entry for any of the databases in the mapping file, *none* of the databases are attached and `ooattachdb` terminates after issuing an error message.

`-readonly`

Attaches the database file as a read-only database. When a database is read-only, all requests for read locks are automatically granted and all requests

for update locks are automatically refused, independently of the lock server. Omitting this option causes the database to be attached as a read-write database. After the database is attached, you can change its access status using `oochangedb`.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database to which the database is to be attached. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.
(*FTO*) Specify the boot file of the autonomous partition that is to control the attached database.

Discussion

Objectivity / DB checks whether the storage-page sizes of the database and the target federated database are the same. However, it is your responsibility to ensure that the schema of the target federated database is identical to (or a superset of) the schema of each database being attached.

A database identifier is a component of the object identifiers (OIDs) of objects in the database. If you attach one or more databases with new database identifiers, Objectivity / DB automatically adjusts:

- The object identifiers of all objects within each attached database
- Any relationships (associations) within each attached database
- Any relationships (associations) that exist among a group of databases attached through the `-dbmap` option

Objectivity / DB does not attempt to find and adjust references that may exist from objects in the target federated database to objects in the databases being attached.

Objectivity / DB does not update keyed objects during `ooattachdb`.

See also

`oochangedb`
`oocopydb`

oobackup

Archives a federated database to a backup medium.

```
oobackup
     -set setName
     -backup backupName
     -volume volumeName
     -device deviceSpecifier
     [-procfiles programName]
     [-capacity size]
     [-level backupLevel]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-set` *`setName`*

Name of the backup set that is to contain the backup event.

`-backup` *`backupName`*

Name of the backup event to be executed. The name must be unique within the scope of the backup set specified by *`setName`*.

`-volume` *`volumeName`*

Volume name prefix. Each volume name consists of the volume name prefix plus a sequential numeric value. For example, if *`volumeName`* is `myVol`, the first volume of a federated database backup has the name `myVol_1`. The second volume has the name `myVol_2`. Multiple volumes are generated only if the backup size exceeds the backup capacity value.

`-device` *`deviceSpecifier`*

Full pathname of the disk directory where the backup volumes are to be stored. For example, if the value for *`deviceSpecifier`* is `/dba/backups` and the value for *`volumeName`* is `fdb020492`, then the actual disk filename for the first backup volume is `/dba/backups/fdb020492_1`.

`-procfiles` *`programName`*

Full or relative pathname of the shell script or program to be executed after each backup volume is written.

During the execution of `oobackup`, the name of the backup volume just written and the total size in bytes of the backup volumes written so far are passed to the script as command-line arguments. If the script exits with a nonzero status, `oobackup` issues an error message and terminates immediately.

`-capacity` *size*

Capacity of each backup volume in kilobytes. The default is 1000 (1 MB).

`-level` *backupLevel*

Backup level. Valid values are integers 0 through 9. The default level is 0, which executes a full backup.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 backup boot file for the federated database to be archived. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. You must use the same boot file when restoring the federated database.

Discussion

Some federated databases have multiple boot files—for example, a distributed federated database may require multiple copies of the boot file. Similarly, in an Objectivity/FTO environment, each autonomous partition has its own boot file. When a federated database has multiple boot files, you must choose one specific boot file, called the *backup boot file*, to specify the federated database for both backup and restore operations.

If the size of the backup event exceeds the volume capacity, multiple backup volumes with the same volume name prefix are generated.

Do not run `ootidy` and `oobackup` concurrently—`ootidy` may delete objects that `oobackup` references.

See also

`oorestore`

oobrowse

Graphical interface for browsing objects, browsing types, and making queries on Windows.

```
oobrowse
     [-standalone]
     [bootFilePath]
```

Options

`-standalone`

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

`bootFilePath`

Path to the boot file of the federated database to be browsed. If you omit this argument, you can open the federated database from within the tool. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

To invoke `oobrowse`:

1. Click **Start** and point to **Programs**. In the Objectivity submenu, select **oobrowse**.
2. See Chapter 4, “Browsing Objects and Types,” for information about using the browser.

See also

`ootoolmgr`

oochange

Lists and optionally changes the attributes of a federated database or, in Objectivity/FTO environments, an autonomous partition.

```
oochange
     [-ap apSysName | -id oid]
     [-lockserverhost newLockServerHost]
     [-fdnumber newFdId]
     [[-bootfilehost newBootFileHost]
                -bootfilepath newBootFilePath]
     [[-sysfilehost newFileHost] -sysfilepath newFilePath]
     [[-jnldirhost jnlDirHost] -jnldirpath jnlDirPath]
     [-offline | -online]
     [-nowait | -standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-ap` *apSysName*

System name of the autonomous partition whose attributes are to be changed or listed.

`-id` *oid*

Identifier of the autonomous partition whose attributes are to be changed or listed, specified in D-C-P-S format (for example, 12-0-0-0).

`-lockserverhost` *newLockServerHost*

New lock-server host name. (*FTO*) Requires the `-ap` or `-id` option to change the lock-server host for an autonomous partition.

`-fdnumber` *newFdId*

New federated-database identifier.

`-bootfilehost` *newBootFileHost*

New host for the boot file. Omit this option to leave the host unchanged. (*FTO*) Requires the `-ap` or `-id` option to change the boot-file host for an autonomous partition.

If the `-bootfilepath` option specifies a Windows UNC share name, the *newBootFileHost* value is automatically set to the literal string `oo_local_host`.

`-bootfilepath` *newBootFilePath*

New path to the boot file. If the `-bootfilehost` option designates a remote system, *newBootFilePath* must be fully qualified, not relative. (*FTO*) Requires the `-ap` or `-id` option to change the boot-file path for an autonomous partition.

`-sysfilehost` *newFileHost*

New host for the system-database file. Omit this option to leave the host unchanged. (*FTO*) Requires the `-ap` or `-id` option to change the file host for an autonomous partition.

If the `-sysfilepath` option specifies a Windows UNC share name, the *newFileHost* value is automatically set to the literal string `oo_local_host`.

`-sysfilepath` *newFilePath*

New path to the system-database file. If the `-sysfilehost` option designates a remote system, *newFilePath* must be fully qualified, not relative. (*FTO*) Requires the `-ap` or `-id` option to change the file path for an autonomous partition.

`-jnldirhost` *jnlDirHost*

New host where the federated database's journal files are to be written. Omit this option to leave the host unchanged. (*FTO*) Requires the `-ap` or `-id` option to change the journal-directory host for an autonomous partition.

If the `-jnldirpath` option specifies a Windows UNC share name, the *`jnlDirHost`* value is automatically set to the literal string `oo_local_host`.

`-jnldirpath` *`jnlDirPath`*

New directory where the federated database's journal files are to be written. If the `-jnldirhost` option designates a remote system, *`jnlDirPath`* must be fully qualified, not relative. (*FTO*) Requires the `-ap` or `-id` option to change the journal-directory path for an autonomous partition.

`-offline`

(*FTO*) Sets the status of the specified autonomous partition to offline. Requires the `-ap` or `-id` option.

`-online`

(*FTO*) Sets the status of the specified autonomous partition to online. Requires the `-ap` or `-id` option.

`-nowait`

Instructs the tool to terminate immediately if the federated database or the autonomous partition (specified by the `-ap` or `-id` option) is being accessed by another process.

`-standalone`

Nonconcurrent mode. Use this option if no lock server is running. You may also use this option to bypass a running lock server when changing any attribute except the lock-server host. In this case, the tool must be allowed to interact with the original lock server, if it is still running.

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database to be changed or listed. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file; the `-ap` or `-id` option specifies the autonomous partition to be changed or listed.

Discussion

To list the current attribute values of a federated database, enter oochange with just the *bootFilePath* argument. (*FTO*) To list the current attribute values of an autonomous partition, enter oochange with the -ap or -id option and the *bootFilePath* argument.

If you use the -bootfilepath argument to specify a new boot file location, oochange writes an updated boot file in the specified location. After oochange exits, however, the old boot file remains. You must delete the old boot file using the suitable operating system commands.

Warning: You must run this tool on the host where the federated database was created. If you cannot access the original host, you first use ooinstallfd to install the federated database on the host where you will run this tool.

See also

oofile

oochangedb

Displays and optionally changes the attributes of a database or, in Objectivity / DRO environments, a database image.

```
oochangedb
     (-db dbSysName) | (-id oid)
     [-ap apSysName]
     [[-host newDbHost] -filepath newDbFilePath [-catalogonly]]
     [-movetoap newapSysName]
     [-weight weight]
     [-exists  ask | delete | quit]
     [-readonly | -readwrite]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

-db *dbSysName*

System name of the database to be changed. If you use this option, you cannot use the -id option.

-id *oid*

Identifier of the database to be changed, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78). If you use this option, you cannot use the -db option.

`-ap` *`apSysName`*

(*DRO*) System name of the autonomous partition containing the database image to be changed. This option is not required if there is only one image of the database.

`-host` *`newDbHost`*

Name of the host system where the database is to be relocated. The default is the host on which you are running this tool.

If the `-filepath` option specifies a Windows UNC share name, the *`newDbHost`* value is automatically set to the literal string `oo_local_host`.

`-filepath` *`newDbFilePath`*

Path (including the database filename) where the database is to be relocated. If the `-host` option designates a remote system, *`newDbFilePath`* must be fully qualified, not relative.

`-catalogonly`

Updates the catalog only, without physically relocating the database file. This option is valid only in combination with the `-host` and `-filepath` options.

`-movetoap` *`newapSysName`*

(*FTO*) System name of the autonomous partition where the database is to be moved. The `oochangedb` tool issues an error message if the partition already contains an image of the database.

`-weight` *`weight`*

(*DRO*) Weight of the designated database image. *`weight`* must be a positive integer. If it is 0, `oochangedb` issues an error message.

`-exists ask | delete | quit`

Action to take if the file specified by the `-host` and `-filepath` options already exists.

`ask`	Prompts whether to overwrite the existing file. If the answer is `No`, the program terminates. `No` is the default.
`delete`	Overwrites any existing file.
`quit`	Terminates without changing the database if the file currently exists.

The default value is `ask`.

`-readonly`

Makes the database a read-only database. When a database is read-only, all requests for read locks are automatically granted and all requests for update locks are automatically refused, independently of the lock server.

While a database is read-only, you can either read its contents or change it back to read-write. You must change the database back to read-write before you can perform any other operations on it.

(*DRO*) If one image of a database is made read-only, all images are automatically made read-only.

`-readwrite`

Changes a read-only database back to a read-write database. When a database is read-write, all lock requests are serviced by the lock server. All databases are created as read-write databases.

You can change a read-only database back to read-write only if no application or tool is currently reading either that database or any other read-only database in the same federated database.

(*DRO*) If one image is changed back to read-write, all images are changed to back read-write.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database containing the database to be copied. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

To relocate the database file and update the catalog, you use both the `-host` and `-filepath` options. To update the catalog without physically relocating the database file, you use the `-catalogonly` option with the `-host` and `-filepath` options.

If a database is read-only, you must change it back to read-write with the `-readwrite` option before you can change any other attributes. You can change a read-only database back to read-write only if no application or tool is currently

reading either that database or any other read-only database in the same federated database.

If you specify only the `-db` (or `-id`) and *`bootFilePath`* options, `oochangedb` provides a report of the current attribute values.

See also `ooattachdb`
`oofile`

oocheckams

Checks whether AMS is running on a particular host system.

```
oocheckams
     [hostName]
     [-notitle]
     [-quiet]
     [-help]
```

Options *`hostName`*

Name of the host system to be checked for AMS. Omitting this option causes the current host to be checked.

`-notitle`

Suppresses the 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.

oocheckls

Checks whether a lock server is running on a particular host system.

```
oocheckls
     [hostName]
     [-notitle]
     [-quiet]
     [-help]
```

Options *`hostName`*

Name of the host system to be checked for a lock server. Omitting this option causes the current host to be checked.

`-notitle`

Suppresses the 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.

Discussion

This tool checks whether the specified host is running either a standard lock server (which runs as a separate process) or an in-process lock server (which runs as part of an IPLS application process). The `oocheckls` output identifies the running lock server by its process name; a standard lock server is identified as `ools` or `ools-xx`, where `xx` is a number.

oocleanup

Lists the active transactions for a federated database or autonomous partition; recovers the specified abnormally terminated transactions.

```
oocleanup
     [[-local] | -transaction tId [-deadowner] [-resetlock]]
     [-force]
     [-standalone]
     [-allpart | -onepart]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-local`

Recovers transactions started by local processes. When you specify this option, `oocleanup` identifies local transactions and checks the status of the processes that started them; if a process is no longer active, its transactions are recovered.

`-transaction` *`tId`*

Transaction identifier of a specific transaction to be recovered. By default, the transaction is recovered only if the process that started it is no longer active (see the `-deadowner` option).

`-deadowner`

Used with the `-transaction` option; recovers the specified transaction without checking the status of the process that started it. If you omit the `-deadowner` option, `oocleanup` checks the process status and performs recovery only if the process is no longer active.

Warning: You can corrupt the federated database by using this option on an active transaction.

`-resetlock`

Used with the `-transaction` option; resets the recovery lock before performing recovery.

Use this option only if you are *sure* that the process owning the lock no longer exists, for example, if a previous invocation of `oocleanup` terminated abnormally while recovering the transaction.

`-force`

Performs the cleanup operation without requesting verification. Useful when invoking the tool from a script or application.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-allpart`

Inspects the journal files of all autonomous partitions in the federated database to identify the transactions to be recovered. When used with the `-local` option, `-allpart` causes the recovery of all local, incomplete transactions against the entire federation. However, inspecting all journal files is time-consuming, even if only few transactions actually require recovery.

This option should be considered only if *`bootFilePath`* specifies a partitioned federated database or an autonomous partition.

`-onepart`

Inspects just the journal files of the autonomous partition specified by *`bootFilePath`* to identify the transactions to be recovered. When used with the `-local` option, `-onepart` causes the recovery of just the local, incomplete transactions listed in the inspected journal files. Recovery of these transactions may affect other partitions—specifically, if a transaction updated multiple partitions, the transaction's changes are rolled back in every affected partition. However, the journal files of other partitions are not inspected for transactions to be recovered, so transactions requiring recovery could exist elsewhere in the federated database.

If you know that only a small subset of partitions require recovery, you can use this option to clean up each partition individually. This is faster than using `-allpart` to clean up transactions in all partitions; however, you must determine which partitions require recovery and ensure that they are recovered.

This option should be considered only if *`bootFilePath`* specifies a partitioned federated database or an autonomous partition.

`-notitle`

Suppresses the 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 the federated database or autonomous partition whose transactions are to be listed or recovered. If you are recovering a transaction that holds locks in multiple autonomous partitions, you can specify the boot file of any of these partitions. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

You can use `oocleanup` to accomplish these tasks:

- To display a list of active update transactions on a particular federated database or autonomous partition, specify just the *`bootFilePath`* argument.
- To recover any incomplete transactions started by local processes that are no longer running, use the `-local` option. For a partitioned federated database, you can further control the scope of recovery by adding either the `-allpart` or the `-onepart` option.
- To recover a specific transaction, use the `-transaction` option.
- To recover a transaction started by a remote process, run `oocleanup` on the remote machine so the process status can be verified. If you cannot access the remote system (for example, if it fails to reboot) and you know that the process has terminated, run `oocleanup` locally using the `-transaction` and `-deadowner` options.
- To recover transactions while the lock server for *`bootFilePath`* is stopped, use the `-standalone` option along with any other required `oocleanup` options. Do not use `-standalone` while the lock server is running.

When the `oocleanup` tool recovers a transaction, it rolls back the transaction's uncommitted changes, restoring the federated database to the logical state it was in before the transaction started. If a transaction left uncommitted changes in multiple autonomous partitions, the changes are rolled back in all of the available partitions. In particular, if the transaction left uncommitted changes in a replicated database, the changes are rolled back in all of the available images.

The `oocleanup` tool uses the journal files of the specified federated database or autonomous partition to identify the transactions to be listed or recovered. When just the *`bootFilePath`* argument is specified, the `oocleanup` tool lists all active transactions that opened the specified federated database or partition for update, including transactions started by remote processes. From this list, `oocleanup` with the `-local` option recovers any transactions started by local processes that are no longer running.

If *bootFilePath* specifies an autonomous partition, and the journal files for this partition contain at least one incomplete transaction that affects another partition, then `oocleanup` with the `-local` option inspects the journal files of *all* partitions in the federated database; all incomplete local transactions against the entire federation are recovered. Otherwise, if the specified partition's journal files contain no multipartition transactions requiring recovery, then `oocleanup` recovers only the incomplete local transactions affecting the specified partition. You can guarantee one behavior or the other by specifying either the `-allpart` option or the `-onepart` option.

If the lock server for the specified federated database or autonomous partition is still running, `oocleanup` causes it to release all locks held by the recovered transaction. If the lock server has stopped—or stopped and restarted—the transaction's locks are lost, so `oocleanup` just rolls back changes.

By default, `oocleanup` acquires a *recovery lock* to make sure that no recovery is being performed by another `oocleanup` process. If `oocleanup` fails to acquire a recovery lock, it displays an error message.

ooconfig

Creates a DDL processor (`ooddlx`) that is compatible with your C++ compiler, deleting any existing DDL processor, on UNIX.

```
ooconfig
```

Discussion

When you install the Objectivity/C++ Data Definition Language product, the installation program automatically runs `ooconfig` to create a DDL processor for your platform. The DDL processor creates schemas for Objectivity/C++ applications. For information about using the DDL processor, see the Objectivity/C++ Data Definition Language book.

If you change the version or location of your C++ compiler, you must run `ooconfig` again to update your DDL processor. When you run `ooconfig`, it prompts you for:

- The name and version of your compiler
- Whether the C++ preprocessor is ANSI
- The compiler's include path
- The compiler's predefined preprocessor variables

oocopydb

Copies a database file.

```
oocopydb
     (-db dbSysName) | (-id oid)
     [-host hostName] -filepath path
     [-exists  ask | delete | quit]
     [-external]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *`dbSysName`*

System name of the database whose file is to be copied.

`-id` *`oid`*

Identifier of the database to be copied, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

`-host` *`hostName`*

Name of the host system where the database copy is to be created. The default is the host on which you are running this tool.

If the `-filepath` option specifies a Windows UNC share name, the *`hostName`* value is automatically set to the literal string `oo_local_host`.

`-filepath` *`path`*

Path (including the filename) where the database copy is to be created. If the `-host` option is used to designate a remote system, *`path`* must be fully qualified, not relative.

`-exists ask | delete | quit`

Action to take if a filename clash exists between the original database file and a file in the target directory.

`ask`	Prompts whether to overwrite the existing file. If the answer is `No`, the program terminates. `No` is the default.
`delete`	Overwrites the file in the target directory.
`quit`	Terminates without copying the database to the target directory.

The default value is `ask`.

`-external`

Allows and copies external relationships (associations) in the specified database. By default, `oocopydb` terminates if it encounters external relationships.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database containing the specified database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

The database copy is not attached to any federated database. The `oocopydb` tool provides a safe alternative to copying database files directly using operating system commands. The original database is inaccessible for the duration of the copy operation.

See also

`ooattachdb`

oocopyfd

Creates a complete copy of a federated database, placing all the copied files in the specified directory.

```
oocopyfd
     -fdnumber fdId
     [-host hostName] -dirpath path
     [-lockserverhost lockServerHost]
     [-fdname fdSysName]
     [-exists  ask | delete | quit]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-fdnumber` *`fdId`*

New federated-database identifier. This number must not be the same as the original federated-database identifier (in the boot file for the original federated database).

`-host` *`hostName`*

Name of the host system where copies of the files are to be written. The default is the current host.

If the `-dirpath` option specifies a Windows UNC share name, *`hostName`* is automatically set to the literal string `oo_local_host`.

`-dirpath` *`path`*

Path on the host system to the directory that will hold the copied files. If the `-host` option is used to designate a remote system, the *`path`* value must be fully qualified, not relative.

`-lockserverhost` *`lockServerHost`*

Name of the lock-server host for the new federated database.

`-fdname` *`fdSysName`*

System name of the new federated database.

`-exists ask | delete | quit`

Action to take if a filename clash exists between the original system-database file and a file in the target directory.

`ask`	Prompts whether to overwrite the existing file. If the answer is `No`, the program terminates. `No` is the default.
`delete`	Overwrites the file in the target directory.

`quit` Terminates without copying the database to the target directory.

The default value is `ask`.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database to be copied. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool is useful for preparing to deploy an existing federated database. It locates all the files of the federated database, copies them to one directory, and writes a new boot file. The copy is fully operational.

(*FTO*) Before copying a partitioned federated database, you must delete the autonomous partitions.

This tool obtains an exclusive write lock on the federated database for the duration of the copy operation.

oocreateset

Creates a backup set for the specified federated database.

```
oocreateset
     -set setName
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-set` *`setName`*

Name of the backup set to be created.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 for the federated database to be archived. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

oodebug

Tool for debugging a federated database that is not currently locked for update by another application.

```
oodebug
     [-standalone]
     [-help]
     [bootFilePath]
```

Options

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-help`

Prints the tool syntax and definition to the screen.

`bootFilePath`

Path to the boot file of the federated database to be debugged. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

This tool provides a set of commands for inspecting and editing the contents of a federated database. See Chapter 14, “oodebug Commands”.

You can use this tool on any platform to inspect or edit objects that were created by applications written in any of the Objectivity programming interfaces.

On UNIX, you can use the oodebug alias to run this tool from within a C++ debugger.

oodeletedb

Deletes a database from a federated database. (*DRO*) Deletes all images of a replicated database.

```
oodeletedb
     (-db dbSysName) | (-id oid) | -all
     [-catalogonly]
     [-force]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *`dbSysName`*

System name for the database to be deleted.

`-id` *`oid`*

Identifier of the database to be deleted, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

`-all`

Removes all databases from the specified federated database.

`-catalogonly`

Removes the database from the federated-database catalog only.

`-force`

Deletes the database without requesting confirmation. Useful when invoking the tool from another tool or product.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database from which the specified database is to be deleted. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

If the database file exists, all bidirectional relationships (associations) and unidirectional relationships (associations) to objects in other databases are removed and the file is deleted. If the database file does not exist, you can delete the database from the federated-database catalog by using the `-catalogonly` option.

(*DRO*) To delete just a single image of a replicated database, you must use the `oodeletedbimage` tool; see the Objectivity / FTO and Objectivity / DRO book.

oodeletefd

Deletes a federated database.

```
oodeletefd
     [-force]
     [-notitle]
     [-quiet]
     [-help]
     bootFilePath
```

Options

`-force`

Deletes the federated database without requesting verification. Useful when invoking the tool from another tool or product.

`-notitle`

Suppresses the 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 the federated database to be deleted. (*FTO*) You can specify any autonomous-partition boot file. *This is a required argument; there is no default value.*

Discussion

You may not delete a federated database that has journal files in its journal directory. Before deleting a federated database, you must:

- Ensure that no active transactions exist against the federated database.
- Recover any abnormally completed transactions (for example, using `oocleanup`).

oodeleteset

Deletes a backup set and all information in the backup diary about the backups associated with that set.

```
oodeleteset
     -set setName
     [-procfiles programName]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-set` *`setName`*

Name of the backup set to be deleted.

`-procfiles` *`programName`*

Executes the shell script or program *`programName`* before deleting the backup volume. *`programName`* can contain a full or relative pathname.

During the execution of `oodeleteset`, the name of the file about to be deleted is passed to the script as a command-line argument. If the script exits with a nonzero status, `oodeleteset` issues an error message, but continues processing.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 for the federated database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

oodump

Creates a text file containing a representation of the contents of a federated database to be loaded later using `ooload`.

```
oodump
     [[-id oid] |([-cont contSysName] -db dbSysName)]
     [-range  full | one | down | up]
     [-format  dec | oct | hex]
     [-outfile filename]
     [-exists  ask | delete | quit]
     [-line length]
     [-compress [tool]]
     [-empty]
     [-ignore]
     [-standalone]
     [-noheader]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-id` *`oid`*

Identifier of the database to be dumped, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

By default, this tool dumps all databases within the specified federated database. If you use this option, you cannot use the `-cont` or `-db` option.

`-cont` *`contSysName`*

System name of a particular container to dump. If you use this option, you must also use the `-db` option to specify the database that contains this

container. If you use this option, you cannot use the `-id` option. If *`contSysName`* contains spaces, you must enclose it in double-quote (`" "`) characters.

`-db` *`dbSysName`*

System name of a particular database to be dumped. By default, this tool dumps all databases within the specified federated database. If you use this option, you cannot use the `-id` option.

`-range full | one | down | up`

Range of objects to dump.

`full`	Dumps the specified object, all objects it contains, and all objects that contain it.
`one`	Dumps only the specified object.
`down`	Dumps the specified object and all objects it contains.
`up`	Dumps the specified object and all objects that contain it.

The default value is `full`.

`-format dec | oct | hex`

Output format of integers.

`dec`	Decimal format
`oct`	Octal format
`hex`	Hexadecimal format

The default value is `dec`.

`-outfile` *`filename`*

Name of the output file, which contains the textual representation in Object Text Format of the dumped objects. By default, all output is sent to `stdout`.

`-exists ask | delete | quit`

Action to take if the file specified with the `-outfile` option exists.

`ask`	Prompts whether to overwrite the existing file. If the answer is `No`, the program terminates. `No` is the default.
`delete`	Deletes any existing file.
`quit`	Quits the program if the file currently exists.

The default value is `ask`.

`-line` *`length`*

Line length in characters for string fields. *`length`* can be any nonnegative integer. 0 means that there is no restriction on line length. The default value is 0.

`-compress [`*`tool`*`]`

Pipes the output through the specified data compression tool. If *`tool`* is not specified, output is sent to the standard UNIX `compress` tool.

`-empty`

Includes empty relationships (associations) in the output. If you omit this option, empty relationships are not included in the output.

`-ignore`

Ignores errors when possible and continues processing.

Warning: Use of the `-ignore` option can result in text files that `ooload` cannot load.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-noheader`

Suppresses the header in the output. If you omit this option, a header is included at the beginning of output.

`-notitle`

Suppresses the 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 the federated database containing the objects to be dumped. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

You can use `oodump` to dump a federated database, a database, or a container. This tool does *not* dump the following information:

- Unidirectional relationships (associations) from a not-dumped object to a dumped object
- Persistent locks set using the checkout / checkin feature
- Keyed objects
- (*FTO*) Autonomous partitions

oodumpcatalog

Lists all the files in a federated database.

```
oodumpcatalog
     [-outfile filename]
     [-exists  ask | delete | quit]
     [-format  hostlocal | native]
     [(-ap apSysName) | (-id apOID)]
     [-nolabel]
     [-control]
     [-standalone]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-outfile` *`fileName`*

Filename of an optional output file that will contain the names of all of the files in the federated database. You can edit this file to drive a standard backup tool.

`-exists ask | delete | quit`

Action to take if a filename clash exists between the file specified with the `-outfile` option and a file in the target directory.

`ask`	Prompts whether to overwrite the existing file. If the answer is `No`, the program terminates. `No` is the default.
`delete`	Overwrites the file in the target directory.
`quit`	Terminates without creating the file.

The default value is `ask`.

`-format hostlocal | native`

Format for printing filenames.

`hostlocal`	Filename printed in host format, as *`hostName`*`::`*`localPath`*—for example, `mach3::/mnt/fred/project/myfd.FDB`
`native`	Filename printed as full pathname using the native operating system's format—for example, `/net/mach3/usr/mnt/project/myfd.FDB`

The default value is `hostlocal`.

`-ap` *`apSysName`*

(*FTO*) Lists only the files controlled by the autonomous partition with the system name *`apSysName`*.

`-id` *`apOID`*

(*FTO*) Lists only the files controlled by the autonomous partition with identifier *`apOID`*.

`-nolabel`

Suppresses the labeling of files in the output. By default, each filename in the output is labeled.

`-control`

(*FTO*) Lists the controlling autonomous partition for each file.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database containing the files to be listed. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

You can use `oodumpcatalog` to find out which databases (if any) have been made read-only.

(*FTO*) The output includes autonomous-partition files. (*DRO*) The output lists the autonomous-partition files under each database for which there are multiple images.

oofile

Displays the file type and other information about the specified Objectivity / DB file.

```
oofile
     fileName
     [-notitle]
     [-help]
```

Options

fileName

Filename of the Objectivity/DB file to be described.

`-notitle`

Suppresses the 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.

Discussion

This tool displays the following information about the specified file:

- The file type—Database file, system-database file for a federated database or autonomous partition, or boot file.
- The attributes for the type of file, such as the identifier, system name, journal-directory host and path, lock-server host, storage-page size, and containing federated database.
- The architecture (platform and operating system) on which the file was created.
- The Objectivity/DB release with which the database format is compatible. This information is useful when you are upgrading to a new release.

See also

oochange
oochangedb

oogc

Deletes unreferenced objects from garbage-collectible containers in a federated database. Garbage-collectible containers can be created and used only by Objectivity for Java or Objectivity/Smalltalk applications.

```
oogc
     [-askroots]
     [-nodelete]
     [-verbose]
     [-statistics]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-askroots`

Asks interactively for special root objects.

`-nodelete`

Displays the object identifiers of containers and objects that would be deleted, but does not delete them.

`-verbose`

Displays all roots, followed references, and deleted objects.

`-statistics`

Displays data about the number of roots, visited containers, visited objects, deleted containers, and deleted objects.

`-notitle`

Suppresses the 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 the federated database containing the containers to be garbage-collected. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

This tool can delete unreferenced objects from a garbage-collectible container only if it can get an update lock on that container. When a garbage-collectible container is unnamed and contains only unreferenced persistent objects, this tool deletes the container. A persistent object is considered referenced only if it can be reached from an Objectivity for Java or Objectivity / Smalltalk named root.

Garbage collection does not apply to non-garbage-collectible containers, which exist primarily for interoperating with applications written in a non-garbage-collected language, such as C++. All objects in a non-garbage-collectible container are assumed to be valid and remain in the database until explicitly deleted by an application.

The `oogc` tool can run concurrently with other database processes.

Substantial virtual memory can be required for `oogc`'s temporary data, depending on how much garbage needs to be collected. Processing a large container with references to many other large containers also increases the memory requirement.

ooinstallfd

Installs a fully operational federated database at a remote site.

```
ooinstallfd
     [-lockserverhost lockServerHost]
     [-fdname fdSysName]
     [-fdnumber fdId]
     [[-fdfilehost fdFileHost] -fdfilepath fdFilePath]
     [[-dbdirhost dbDirHost] -dbdirpath dbDirPath]
     [[-jnldirhost jnlDirHost] -jnldirpath jnlDirPath]
     [-nocheck]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     bootFilePath
```

Options

`-lockserverhost` *`lockServerHost`*

Lock-server host for the newly installed federated database. The default is the host on which you are running this tool.

`-fdname` *`fdSysName`*

System name of the newly installed federated database. The default is the system name specified in the boot file.

`-fdnumber` *`fdId`*

New identifier of the federated database. The default is the identifier specified in the boot file.

`-fdfilehost` *`fdFileHost`*

New host where the system-database file is to be located. The default is the host on which you are running this tool, if the `-fdfilepath` option is specified; otherwise the location is completely determined by *`bootFilePath`*.

If the `-fdfilepath` option specifies a Windows UNC share name, the *`fdFileHost`* value is automatically set to the literal string `oo_local_host`.

`-fdfilepath` *`fdFilePath`*

New path on *`fdFileHost`* where the system-database file is to be located. If the new path is not specified, this tool locates the system-database file in the directory indicated by *`bootFilePath`*. If the `-fdfilehost` option is used to designate a remote system, *`fdFilePath`* must be fully qualified, not relative.

`-dbdirhost` *`dbDirHost`*

New host where the database files are to be located. The default is the host on which you are running this tool, if the `-dbdirpath` option is specified; otherwise, the location is completely determined by *`bootFilePath`*.

If the `-dbdirpath` option specifies a Windows UNC share name, the *`dbDirHost`* value is automatically set to the literal string `oo_local_host`.

`-dbdirpath` *`dbDirPath`*

New path where the database files are to be located. The default is the directory part of *`bootFilePath`*. If the `-dbdirhost` option is used to designate a remote system, *`dbDirPath`* must be fully qualified, not relative.

`-jnldirhost` *`jnlDirHost`*

New host where the federated database's journal files are to be written. The default is the host on which you are running this tool, if the `-jnldirpath` option is specified; otherwise the location is completely determined by *`bootFilePath`*.

If the `-jnldirpath` option specifies a Windows UNC share name, the *`jnlDirHost`* value is automatically set to the literal string `oo_local_host`.

`-jnldirpath` *`jnlDirPath`*

New directory where the federated database's journal files are to be written. The default is the directory part of *`bootFilePath`*. If the `-jnldirhost` option is used to designate a remote system, *`jnlDirPath`* must be fully qualified, not relative.

`-nocheck`

Continues the installation even if a file is missing from the install directory (the current working directory). By default, `ooinstallfd` terminates if a file is missing.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database to be installed. *This argument is required.*

Discussion

No preparation of the federated database is required. This tool only works on nonpartitioned federated databases.

All files to be installed must be located in the current working directory. To install a federated database that cannot fit in a single directory, you use the `-nocheck` option. You must then correct the file locations in the catalog using `oochangedb` with the `-catalogonly` option.

ookillls

Kills a standard lock server (a lock server that is running as a separate process).

```
ookillls
     [lockServerHost]
     [-notitle]
     [-help]
```

Options

`lockServerHost`

Host system where the lock server is running. If you do not specify *`lockServerHost`*, the local lock server is killed.

`-notitle`

Suppresses the 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.

Discussion

If the lock server is currently servicing active transactions, the lock server refuses to terminate and an error message is issued.

You cannot use this tool to kill an in-process lock server. An in-process lock server can only be stopped by the IPLS application that started it. Terminating an IPLS application before it stops its in-process lock server is equivalent to an abnormal lock-server failure, and any incomplete transactions will need recovery.

On Windows, you normally kill a standard lock server from an Objectivity Network Services tool, rather than using `ookillls.exe` from a command prompt. On Windows NT or Windows 2000, you require the administrator's permission to use this tool because the lock server is started as a service, not as a user program.

oolistwait

Displays information about transactions waiting on any lockable Objectivity / DB object, which can be a federated database, a database, or a container.

```
oolistwait
     [-transaction id] | ([-host hostName] [-user userId])
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-transaction id`

Checks the status of the specified transaction.

`-host hostName`

Lists the status of all waiting transactions started on the specified host, subject to filtering based on other options. This option defaults to all nodes.

`-user userId`

Lists the status of all waiting transactions started by the specified user, subject to filtering based on other options. This option defaults to all users.

`-notitle`

Suppresses the 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 a boot file specifying the lock server to be queried. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

This tool queries the specified lock server to check for waiting transactions started by a specified user or host. You can also use this tool to find out whether a specified transaction is waiting for a lockable object, and if so, which transactions currently hold the lock on that object. The displayed transactions may be waiting for resources in any of the federated databases or autonomous partitions serviced by the specified lock server.

The following table summarizes how to use the various options to view different types of transaction information.

To View	Use This Option
All waiting transactions	(no options)
Waiting transactions started on a specific host system	`-host`
Waiting transactions started by a specific user	`-user`
Waiting transactions started by a specific user on a specific host system	`-host` and `-user`
A specific transaction's status, and any transactions that are using resources that it needs	`-transaction`

You should ignore any *latches* reported in the output. Latches are internally used locks on containers whose identifiers appear to be out of range.

ooload

Creates persistent objects in a federated database, using information from a text file created by `oodump`.

```
ooload
     [-abort | -db]
     [-cont]
     [-uncompress [tool]]
     [-external]
     [-resize]
     [-hash hashFactor]
     [-grow growthPercent]
     -infile fileName
     [-verbose]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-abort`

Terminates without applying changes. You can use this option (with or without the `-verbose` option) to test the load operation without committing the transaction. If you use this option you cannot use the `-db` option.

`-db`

Deletes any databases whose system name matches a database in the input text file. If you use this option you cannot use the `-abort` option. This option supersedes the `-cont` option when both are specified.

`-cont`

Deletes containers from the database whose object identifier or system name matches a container in the text file.

`-uncompress [`*`tool`*`]`

Pipes the input file through the specified data-decompression tool. If *`tool`* is not specified, the standard UNIX `uncompress` tool is used.

`-external`

Allows external references within the text file, generating a warning for each external reference it encounters.

Warning: Use of this option can introduce semantic inconsistencies into the federated database.

`-resize`

Ignores container size information in the input file and resizes containers as necessary. Use of this option can result in a more compact federated database.

`-hash` *`hashFactor`*

Overrides the hash-factor information in the input file with the value *`hashFactor`* for all containers. *`hashFactor`* can be any nonnegative integer.

`-grow` *`growthPercent`*

Overrides the growth-factor information in the input file with the value *`growthPercent`* for all containers. *`growthPercent`* can be any nonnegative integer.

`-infile` *`fileName`*

Input text file (previously created by `oodump`).

`-verbose`

Prints full status information during processing and a summary message after processing terminates. This tool sends all status messages to `stdout` and all error messages to `stderr`.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database where the objects will be loaded; by default the federated database specified in the text input file. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

If invoked without the `-external` option, `ooload` terminates upon encountering an external reference. External references are either relationships (associations) with, or object identifiers for, objects not in the federated database.

The federated database referenced by *`bootFilePath`* must have a schema identical to (or a superset of) that of the federated database from which the text input file was dumped.

oolockmon

Lists all processes and locks currently managed by a lock server.

```
oolockmon
     [-detail]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-detail`

Displays intention locks. An intention lock is a special kind of lock placed on a database or federated database when you open it. An intention lock simply indicates that the transaction may also hold a read, update, or exclusive lock lower in the storage hierarchy.

`-notitle`

Suppresses the 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 the federated database or autonomous partition whose lock server is to be queried. You can omit this argument if you set the OO_FD_BOOT environment variable to the correct path.

Discussion

This tool reports the state of the lock server that services the specified federated database. The lock server may be either a standard lock server (which runs as a separate process) or an in-process lock server (which runs as part of an IPLS application process). The oolockmon tool displays the requested information in a table that includes:

- The transaction identifier of the transaction that obtained the lock
- The lock mode—read or update
- The type of locked object—federated database, database, or container
- The federated database, autonomous partition, database, and container identifiers (if relevant) of the locked object

You should ignore any *latches* reported in the output. Latches are internally used locks on containers whose identifiers appear to be out of range.

oolockserver

Starts a standard lock server (a separate lock-server process) on the current workstation.

```
oolockserver
     [{bootFilePath}]
     [-notitle]
     [-help]
```

Options

bootFilePath

Path to the boot file of the federated database or autonomous partition to be recovered when the lock server is started. You can specify one or more paths. If you omit a federated database or autonomous partition that is serviced by the lock server, that federated database or autonomous partition is recovered the first time it is opened by an application.

You must specify each boot file name in host format, even if it is local—that is, you must specify fully qualified paths of the form:

hostName::*fullLocalPath*

-notitle

Suppresses the 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.

Discussion

Restarting the lock server after a lock-server failure performs automatic recovery on the federated databases specified by *`bootFilePath`*.

On Windows, you normally start the lock server from an Objectivity Network Services tool, rather than using `oolockserver.exe` from a command prompt. On Windows NT or Windows 2000, you require the administrator's permission to use this tool because the lock server is started as a service, not as a user program.

A lock server grants locks on resources (containers, databases, or federated databases) to requesting transactions. Each transaction can lock multiple resources, and a given resource can be locked by multiple transactions. A single lock server can support:

- A maximum of 1031 concurrent transactions
- A large number of concurrently held locks, limited only by available virtual memory

In general, a lock server cannot be started on a workstation that is already running a lock server (either a standard lock server or an IPLS application—see "Types of Lock Server" on page 79). However, it is possible for a lock server from the current release of Objectivity / DB to run on the same workstation as a lock server from certain older releases of Objectivity / DB. If a federated database specifies such a workstation as its lock-server host, you must guarantee that all applications accessing that federated database have been built with the same release of Objectivity / DB (so that they will all contact the same lock server).

Warning: Data corruption will occur if two applications built with different releases contact different lock servers while accessing data in the same federated database.

oonewdb

Creates a new database in a federated database.

```
oonewdb
     -db dbSysName
     [-id oid]
     [[-host hostName] -filepath path]
     [-weight weight]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-db` *dbSysName*

System name of the database to be created.

`-id` *oid*

Identifier of the database to be created, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

`-host` *hostName*

Name of the host system where the database file is to be created.

If the `-filepath` option specifies a Windows UNC share name, the *hostName* value is automatically set to the literal string `oo_local_host`.

`-filepath` *path*

Path to the directory where the database file is to be created. Optionally, *path* can include the filename for the database; if you omit the filename, it is generated automatically. If the `-host` option is used to designate a remote system, *path* must be fully qualified, not relative.

`-weight` *weight*

(*DRO*) Weight of the database image. *weight* must be a positive integer. The default is 1. If the value is 0, `oonewdb` issues an error message.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database to which the new database is to be attached. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) Specify the boot file of the autonomous partition that is to control the new database.

Discussion

If neither the `-host` nor the `-filepath` option is used, the database is created in the same directory as the federated database's system-database file. You can use `oochangedb` to move the new database into a different autonomous partition.

(*DRO*) You use oonewdb to create the first or only image of a database in a federated database. To replicate the database (that is, to create additional database images), you must use oonewdbimage; see the Objectivity / FTO and Objectivity / DRO book.

oonewfd

Creates a federated database, including the system-database file and the boot file.

```
oonewfd
     [-fdfilehost fdFileHost] -fdfilepath fdFilePath
     -lockserverhost lockServerHost
     [[-jnldirhost jnlDirHost] -jnldirpath jnlDirPath]
     [-fdnumber fdId]
     [-pagesize pageSize]
     [-bootonly]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-fdfilehost` *`fdFileHost`*

Host where the system-database file is to be located. If you omit this option, the default value is the current host.

If the `-fdfilepath` option specifies a Windows UNC share name, the *`fdFileHost`* value is automatically set to the literal string `oo_local_host`.

`-fdfilepath` *`fdFilePath`*

Path (including the filename) of the system-database file on host *`fdFileHost`*. If the `-fdfilehost` option is used to designate a remote system, *`fdFilePath`* must be fully qualified, not relative.

`-lockserverhost` *`lockServerHost`*

Host where the lock server servicing the new federated database is located.

`-jnldirhost` *`jnlDirHost`*

Host where the federated database's journal files are to be written. The default value is the current host if the `-jnldirpath` option is specified, and *`fdFileHost`* if it is not.

If the `-jnldirpath` option specifies a Windows UNC share name, the *`jnlDirHost`* value is automatically set to the literal string `oo_local_host`.

`-jnldirpath` *`jnlDirPath`*

Directory where the federated database's journal files are to be written. The default value is the directory part of *`fdFilePath`*. If the `-jnldirhost` option is

used to designate a remote system, *jnlDirPath* must be fully qualified, not relative.

`-fdnumber` *fdId*

Federated-database identifier that identifies the federated database to the lock server. The default value is 1.

`-pagesize` *pageSize*

Storage-page size (the size of the unit of storage transferred between memory and disk) in bytes. Allowable page sizes are integer multiples of 8 between 512 and 65,536, inclusive. The default page size is 8192 bytes.

`-bootonly`

Creates only the boot file, not the system-database file.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the new federated database is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 (including the filename) of the boot file of the new federated database. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path.

Discussion

The federated database's system name is the simple name of the boot file specified in *bootFilePath*. Once a federated database is created, you cannot change its system name.

A lock server must be running on *lockServerHost* while you run `oonewfd`.

Normally, you set the storage-page size to be the disk's page size. However, you might want to set the storage-page size to be slightly larger than the size of common objects. For example, you might increase the page size if common objects are larger than the default, or decrease the page size if common objects are smaller than the default. For information about choosing an optimal page size, see the chapter on performance in the documentation for your Objectivity programming interface.

ooqueryset

Displays information about the backup events in a specified backup set or for an entire federated database.

```
ooqueryset
     [-set setName]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-set setName`

Name of the backup set to query. If you omit this option, `ooqueryset` displays information about the backup events in all backup sets.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 the federated database whose backup set is to be queried. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

For each backup event listed, `ooqueryset` displays the backup event name, the level, the backup volume capacity, the time stamp, the volume name prefix, and the path to the disk directory where the backup volumes were generated.

oorestore

Restores a federated database previously archived using `oobackup`.

```
oorestore
     -set setName
     -backup backupName
     -volume volumeName
     -device deviceSpecifier
     [-procfilesbef befProgName]
     [-procfilesaft aftProgName]
     [[-newhost targetHost] [-newdirectory targetDir]
     [-failonly] | [-dbmap mapFile]]
     [-dumpcatalog]
     [-exists ask | delete | quit]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-set` *`setName`*

Name of the backup set containing the backup event representing the point of restore.

`-backup` *`backupName`*

Name of the backup event representing the point of restore.

`-volume` *`volumeName`*

Volume name prefix of the backup volumes containing the data to be restored.

`-device` *`deviceSpecifier`*

Full pathname of the disk directory where the backup volumes are stored. For example, if the value for *`deviceSpecifier`* is `/dba/backups` and the value for *`volumeName`* is `fdb020492`, then the actual disk filename for the first backup volume is `/dba/backups/fdb020492_1`.

`-procfilesbef` *`befProgName`*

Executes the shell script or program *`befProgName`* before each backup volume is opened for read access by `oorestore`. *`befProgName`* can contain a full or relative pathname. The name of the volume to be read is passed to the script as a command-line argument. If *`befProgName`* exits with a nonzero status, `oorestore` issues an error message and terminates immediately.

`-procfilesaft` *`aftProgName`*

Executes the shell script or program *`aftProgName`* after `oorestore` finishes reading each backup volume. *`aftProgName`* can contain a full or relative pathname. If *`aftProgName`* exits with a nonzero status, `oorestore` issues an error message and terminates immediately.

`-newhost` *`targetHost`*

Restores files to the new host *`targetHost`*, except for the backup boot file, which is restored to the host on which you are running this tool. The journal-directory attribute of the federated database or every autonomous partition is set to *`targetHost`*.

`-newdirectory` *`targetDir`*

Restores files to the new directory *`targetDir`*, except for the backup boot file, which is restored to the current working directory. The journal-directory attribute of the federated database or every autonomous partition is set to *`targetDir`* (as a separate operation, you should reset each attribute to an appropriate unique directory).

If *`targetHost`* designates a remote system, *`targetDir`* must be fully qualified, not relative. *`targetDir`* must already exist (`oorestore` will not create it).

`-failonly`

Restores files to the location specified by `-newhost` and `-newdirectory` only if the files' original locations are inaccessible. The specified backup boot file is restored to the current working directory if its original location is inaccessible.

`-dbmap` *`mapFile`*

ASCII text mapping file that specifies the destination hosts and directories for system-database files, boot files, journal files, and database files. The mapping file must be local to the directory in which `oorestore` is running. You cannot use this option with the `-newhost`, `-newdirectory`, or `-failonly` option.

The mapping file contains one line for each file or journal directory to be remapped. Each line has the format:

`keyword systemName targetHost targetPath`

where

`keyword`	Kind of file or directory to be remapped; either: `FD`—federated-database system-database file `FDJNL`—federated-database journal directory `DB`—database file `AP`—autonomous-partition system-database file `APBOOT`—autonomous-partition boot file `APJNL`—autonomous-partition journal directory
`systemName`	System name of a federated database, database, or autonomous partition.

targetHost Target host name.

targetPath Target path for directory or filename. The directory portion of each path must exist (`oorestore` does not create new directories).

Any file not specified in the mapping file is restored to its original location. The backup boot file is always restored to the current working directory.

`-dumpcatalog`

Displays the system names and original locations of all system-database files, boot files, journal files, and database files. The files are first restored to the current working directory (overwriting any files already there), and then deleted.

`oorestore` always runs in standalone mode when this option is used since no usable federated database is generated.

Warning: Invoking `oorestore` with this option deletes any system-database files in the current working directory. Do not use this tool in a directory that already contains such files.

`-exists ask | delete | quit`

Action to take if a database in the archive file already exists.

`ask` Prompts whether to overwrite the existing database. If the answer is `No`, the program terminates. `No` is the default.

`delete` Overwrites the existing database.

`quit` Terminates if the database already exists.

The default value is `ask`.

`-standalone`

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

Warning: Corruption can occur if concurrent access to the federated database is attempted while any process is using this mode.

`-notitle`

Suppresses the 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 backup boot file for the federated database to be restored. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the

correct path. The backup boot file is the boot file that was used to create the backup.

Note: You must specify the original backup boot file path to `oorestore`, even if the backup boot file no longer exists or is no longer accessible. As long as the necessary backup volumes are accessible and undamaged, the condition of the boot file or the boot file host does not affect the restore.

Discussion

When you restore, you must include the backup set name (`-set`), the backup event name (`-backup`), the volume name (`-volume`), and the full directory pathname (`-device`) of the backup event representing the point of restore.

Unless you specify otherwise, `oorestore` restores files to their original locations. If the original locations are not available, you can restore files to a single location by specifying the *`-newhost`* and *`-newdirectory`* options, or you can restore files to multiple locations by specifying the `-dbmap` option with a mapping file (in either case, the backup boot file is always restored to the current working directory). If files are restored to one or more nonoriginal locations, `oorestore` automatically adjusts all journal-directory attributes as specified; if you have multiple autonomous partitions, you should reset each journal file attribute to a unique directory. Journal files themselves are not restored.

See also

`oobackup`

ooschemadump

Writes a representation of a federated database's evolved schema to the specified output file.

```
ooschemadump
     [-encode]
     [-outfile fileName]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-encode`

Encodes the output file so that it cannot be read by end users. If you omit this option, the schema change information is written as text.

`-outfile` *`fileName`*

Name of the file to which the schema representation is to be written. If you omit this option, the output will be written to a default file called `schema.dmp` in the current directory.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database with the evolved schema. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

You normally perform schema evolution on a source federated database at your development site. When you are ready to test or deploy the evolved schema, you can use `ooschemadump` to write the schema changes to an output file. You (or your end users) then use `ooschemaupgrade` to apply the changes in this file to the target federated database (for example, your end user's production federated database).

(Objectivity/C++) Some complex schema evolution operations require that you run a conversion or upgrade application between uses of the DDL processor. When this is the case, you need to use `ooschemadump` after *each* use of the DDL processor, and before you run a conversion or upgrade application or continue with any other schema evolution operations. When you deploy the schema changes, you must distribute all such `ooschemadump` output files to your end user.

ooschemaupgrade

Upgrades the schema of the specified federated database by applying the schema changes contained in the specified file.

```
ooschemaupgrade
     [-infile fileName]
     [-standalone]
     [-notitle]
     [-quiet]
     [-help]
     [bootFilePath]
```

Options

`-infile` *`fileName`*

Name of the file containing the schema changes to be applied. The file must be the output of `ooschemadump`. If you omit the `-infile` option, `ooschemaupgrade` looks for a default file named `schema.dmp` in the current directory.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the target federated database whose schema is to receive the schema changes. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

You normally perform schema evolution on a source federated database at your development site. When you are ready to test or deploy the evolved schema, you use `ooschemadump` to write the schema changes from the source federated database to an output file. You (or your end users) then use `ooschemaupgrade` to apply the changes in this file to the target federated database (for example, your end user's production federated database).

The schema of the target federated database must be identical to the schema that existed in the source federated database before schema evolution was performed.

The `ooschemaupgrade` tool enables you to deploy schema changes to production federated databases without disclosing the schema to end users—that is, without:

- (*Objectivity/C++*) Running the DDL processor at end user sites
- (*Objectivity for Java* and *Objectivity/Smalltalk*) Including a compiler

oostartams

Starts the Advanced Multithreaded Server (AMS) on the current system.

```
oostartams
     [-numthreads numthreads]
     [-notitle]
     [-help]
```

Options

`-numthreads` *`numthreads`*

Number of threads (or processes) the server should use to process concurrent client requests. The default value is `8`.

`-notitle`

Suppresses the 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.

Discussion

If AMS is already running when this tool is invoked, an error message is displayed. Only one AMS process per version of AMS can run on any one system.

On UNIX, this command is typically placed in a startup script that runs when the server system is booted. However, it can be executed at any time.

On Windows, you normally start AMS from an Objectivity Network Services tool. On Windows NT or Windows 2000, this command requires the administrator's permission because AMS is started as a network service, not as a user program.

See also

`oostopams`

oostopams

Terminates the Advanced Multithreaded Server (AMS) on the specified host system, provided that there are no client applications using it. If client applications are running, an error message is displayed.

```
oostopams
     [-notitle]
     [-help]
     [hostName]
```

Options

`-notitle`

Suppresses the 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.

`hostName`

Name of the host system on which the AMS process is to be terminated. The default is to terminate any AMS process running on the local host (the host on which you are running this tool).

See also `oostartams`

ootidy

Consolidates a fragmented database or federated database.

```
ootidy
     [(-db dbSysName) | (-id oid)]
     [[-tmpdirhost hostName] -tmpdirpath dirPath]
     [-standalone]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options `-db` *`dbSysName`*

System name of the database to be tidied. By default, this tool tidies all databases within the specified federated database.

`-id` *`oid`*

Identifier of the database to be tidied, specified in D-C-P-S format (for example, 78-0-0-0). This option also accepts the single-integer database identifier format (for example, 78).

By default, `ootidy` tidies all databases within the specified federated database.

`-tmpdirhost` *`hostName`*

Name of the host system where temporary files are to be created. If you use this option, you must also use the `-tmpdirpath` option.

`-tmpdirpath` *`dirPath`*

Local path of the directory where temporary files are to be created. If the `-tmpdirhost` option is used to designate a remote system, *`dirPath`* must be fully qualified, not relative.

`-standalone`

Nonconcurrent mode. Use this option only if the lock server for the specified federated database or autonomous partition is stopped. If the lock server is running, the tool terminates after issuing an error message.

`-notitle`

Suppresses the 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 the federated database to be tidied. You can omit this argument if you set the `OO_FD_BOOT` environment variable to the correct path. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

To hold temporary database files created during its execution, `ootidy` requires free disk space equal to the size of the federated database or database you are tidying.

Warning: To prevent potential database corruption, make sure that no other processes access a database or federated database being tidied. One way to guarantee that no processes access the database is to kill the lock server and to run `ootidy` in standalone mode.

Warning: Do not run `ootidy` and `oobackup` concurrently—`ootidy` might delete objects that `oobackup` references. Do not run `ootidy` if you suspect that the federated database has been corrupted. It can make the problem significantly worse.

ootoolmgr

Tool for browsing objects, browsing types, and making queries on UNIX.

```
ootoolmgr
     [-standalone]
     [-notitle]
     [-help]
     [bootFilePath]
```

Options

`-standalone`

Nonconcurrent mode. No lock server is required.

`-notitle`

Suppresses the 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 the federated database to be browsed. If you omit this argument, you can open the federated database from within the tool. (*FTO*) You can specify any autonomous-partition boot file.

Discussion

See Chapter 4, “Browsing Objects and Types,” for information about using the browser.

See also

`oobrowse`

14

oodebug Commands

This chapter describes the commands you can use in an oodebug session. You can run oodebug as a separate process on any platform, or you can run it within a C++ debugger (dbx and its variants) on UNIX.

See:

- "Reference Summary" on page 201 for an overview of oodebug command capabilities described in this chapter
- "Reference Index" on page 202 for an alphabetical list of oodebug commands
- "Reference Descriptions" on page 203 for complete syntax and usage descriptions of oodebug commands

For more information about oodebug, see Chapter 5, "Debugging a Federated Database"; see also oodebug in Chapter 13, "Tools".

Using oodebug Commands

Modes

oodebug has two modes: read and update. These modes determine which oodebug commands are available for use. You can safely use read mode to view the structure and contents of your federated database.

However, update mode can be dangerous—use it with care. For example, a common technique used in C++ to maintain consistency between fields in an object is to declare the fields as private and allow access to the fields only through access methods. In update mode, oodebug allows direct access to these fields, bypassing the protection of the access methods.

Abbreviating Command Names

You can abbreviate an oodebug command by typing the first characters of the command name. You need type only enough characters to distinguish your

intended command from all available commands. For example, you can run the `whatis` command by typing `what`, `wh`, or `w`.

Command Parameters

`oodebug` commands use these parameters:

`object`

Object on which to operate, specified in any of the following forms:

- Object identifier (OID)—Enter the object identifier in the form: *`D-C-P-S`*. Optionally, you can use this form: *`#D-C-P-S`*.
- Object name in the federated database scope—Enter the name in the form *`fdbSysName.objScopeName`*, where *`objScopeName`* is the scope name of the object.
- Object name in a database scope—Enter the object scope name in the form *`dbSysName.objScopeName`*, where *`dbSysName`* is the database system name and *`objScopeName`* is the object's scope name.
- Object name in a container or basic object scope—Enter the object in the form *`D-C-P-S.objScopeName`*, where *`D-C-P-S`* is the object identifier of the scope object, and *`objScopeName`* is the object's scope name. Optionally, you can use this form: *`#D-C-P-S.objectName`*.
- Last object used specification—When you perform a series of commands on the same object, use the exclamation point character (`!`) instead of repeating the object identifier or object name.

`container`

Container on which you want the function to operate.

`nameOfLink`

Name of a relationship (association) between two objects.

`persistentClass`

Name of a class in the schema that inherits from class `ooObj`.

`fieldExpression`

Field on which to operate, specified in one of the following forms:

- Field name—Enter the name of the field.
- Field of a nested class—Enter in the form *`y.z`*, where *`y`* is the class and *`z`* is the field name.
- Array element—Enter in the form *`variable[index]`*, where *`variable`* is the array's name and *`index`* is an element's position in the array.

`dbSysName`

System name (as a character string) of a valid database.

Reference Summary

The following table provides an overview of oodebug commands. Availability of these commands depends on which mode is turned on and whether you are running oodebug as a separate process or within a C++ debugger (dbx and its variants) on UNIX.

Function	Command	Mode		dbx
		Read	Update	
Show object contents	`print` `whatis`	✓ ✓	✓ ✓	✓ ✓
Follow relationships (associations)	`iter` `next`	✓ ✓	✓ ✓	✓ ✓

List contained objects	`listconts`
	`listdbs`
	`listobjs`
		✓
		✓
		✓
			✓
			✓
			✓
				✓
				✓
				✓

Change mode	`read` `update`	✓ ✓	✓ ✓	—
Manage transactions	`abort` `commit`	✓ ✓	✓ ✓	—

Manipulate object contents	`add`
	`assign`
	`del`
	`sub`
		—	✓
			✓
			✓
			✓
				✓
				✓
				✓
				✓

Create and delete objects	`delete` `new`	—	✓ ✓	✓ ✓

Other commands	`help`
	`quit`
	`stats`
		✓
		✓
		✓
			✓
			✓
			✓
				✓
				✓
				✓

Reference Index

abort	Cancels the current transaction's changes, then starts a new transaction.
add	Adds a relationship (association) between two objects.
assign	Assigns a constant value to a field.
commit	Makes the current transaction's changes permanent in the federated database, then starts a new transaction.
del	Deletes an entire to-one, to-many, unidirectional, or bidirectional relationship (association).
delete	Deletes a basic object or container from the federated database.
help	Describes and lists syntax for oodebug commands.
iter	Initializes an iterator for a to-many relationship (association).
listdbs	Displays a list of the databases contained in the federated database.
listconts	Displays a list of the containers in a database.
listobjs	Displays a list of the basic objects in a container.
new	Creates an object.
next	Iterates to the next object in an iterator and displays its contents.
oodebug	Alias for invoking the oodebug tool within a UNIX C++ debugger (dbx and its variants).
ooprint	Alias for viewing the contents of an object within a C++ debugger, without invoking the oodebug convenience function.
print	Displays an object's contents in the same format used by the Objectivity/DB data browser (with default settings for the View menu). Displays all fields of this object unless you specify the name of a particular field to display.
quit	Terminates oodebug.
read	Sets the mode to read mode.
stats	Displays statistics about the federated database that are generated during the current session. For information about stats output, see the chapter on performance in the Objectivity/C++ programmer's guide.

sub	Removes a relationship (association) between two objects.
update	Sets the mode to update mode.
whatis	Displays an object's class or struct declaration.

Reference Descriptions

abort

oodebug command

Cancels the current transaction's changes, then starts a new transaction.

```
abort
```

Discussion

Use `commit` or `abort` when in read mode to free any locks acquired during the current transaction.

Mode

Available in read and update mode, but *not* from within `dbx`.

Example

This example aborts the current transaction and starts a new one.

```
(*update) abort
transaction aborted
(update)
(read)
```

add

oodebug command

Adds a relationship (association) between two objects.

```
add object nameOfLink objectToAdd
```

Parameters

See "Command Parameters" on page 200.

Discussion

You can use this command to add an object to any type of relationship.

- Used on a to-many relationship, it is functionally equivalent to the `add_`*`nameOfLink`* C++ member function.
- Used on a to-one relationship, it is functionally equivalent to the `set_`*`nameOfLink`* C++ member function.

Mode

Available in update mode only.

assign

oodebug command

Assigns a constant value to a field.

```
assign object fieldExpression = constantValue
```

Parameters

See “Command Parameters” on page 200.

Discussion

Assigns a constant value to a field of any of the following types:

- `char`
- `float32`, `float64`
- `int16`, `int32`
- `unit8`, `uint16`, `uint32`
- `enum` (integer value)
- `ooRef(ooObj)` (using format *`#D-C-P-S`* or *`D-C-P-S`*)
- `ooShortRef(ooObj)` (using format *`#P-S`* or *`P-S`*)

You can assign values to fundamental types that belong to aggregate types, such as individual array elements or fields of structures. However, you need to fully qualify the fundamental type name. For example, if `w` is an array of `int16` within an object named `schedule`, then the following is a valid assignment:

```
assign schedule w[1] = 1234 // Valid assignment
```

If `y` is a structure that contains field `z` within an object named `schedule`, the following is also a valid assignment:

```
assign schedule y.z = 1234 // Valid assignment
```

The following types of assignments are *not* valid:

```
assign schedule w = … // Not valid assignments
assign schedule y = …
```

Mode

Available in `update` mode only.

Example

This example assigns the value `12` to the *`numberOfFunctionsCalled`* field of type `int32`, in the object `main` with name scope `Functions`.

```
(update) assign Functions.main numberOfFunctionsCalled = 12
assignment succeeded
(*update)
```

commit

oodebug command

Makes the current transaction's changes permanent in the federated database, then starts a new transaction.

```
commit
```

Discussion

Use `commit` or `abort` when in read mode to free any locks acquired during the current transaction.

Mode

Available in read and update mode, but *not* from within `dbx`.

Example

This example shows how to commit a transaction.

```
(*update) commit
transaction committed
(update)
```

del

oodebug command

Deletes an entire to-one, to-many, unidirectional, or bidirectional relationship (association).

```
del object nameOfLink
```

Parameters

See "Command Parameters" on page 200.

Mode

Available in update mode only.

delete

oodebug command

Deletes a basic object or container from the federated database.

```
delete object
```

Parameters

See "Command Parameters" on page 200.

Discussion

Warning: The `delete` command does not invoke the destructor before deleting the object. Thus, before you use this command, you must manually perform the operations that a destructor would have performed: `oodebug` commands `assign`, `add`, `sub`, and `del`.

Mode

Available in update mode only.

Example This example shows how to delete an object by object identifier.

```
(update)delete 4-19-11-2
4-19-11-2 deleted
(*update)
```

This example shows how to delete an object by object name.

```
(update) delete theObject
4-19-11-2 deleted
(*update)
```

help

oodebug command

Describes and lists syntax for oodebug commands.

`help [commandName]`

iter

oodebug command

Initializes an iterator for a to-many relationship (association).

`iter object nameOfLink`

Parameters See “Command Parameters” on page 200.

Discussion Use this command with the next command to follow an object’s relationships.

Example This example shows how to initialize an iterator for an object identified by the object identifier 4-3-42-11, in order to iterate through its unidirectional relationships named calledBy.

```
(read) iter 4-3-42-11 calledBy
iterator initialized
(read)
```

listdbs

oodebug command

Displays a list of the databases contained in the federated database.

`listdbs`

Example This example lists the databases in the federated database.

```
(read) listdbs
Functions
Files
(read)
```

listconts

oodebug command

Displays a list of the containers in a database.

```
listconts dbSysName
```

Parameters See “Command Parameters” on page 200.

Example This example shows how to display a list of the containers in database `MyDB`.

```
(read) listconts MyDB
#2-2-1-1 (class ooDefaultContObj)
#2-3-3-1 (class ooContObj)
(read)
```

listobjs

oodebug command

Displays a list of the basic objects in a container.

```
listobj container
```

Parameters See “Command Parameters” on page 200.

Example This example shows how to display a list of the basic objects in a container identified by the object identifier `2-3-3-1`.

```
(read) listobjs 2-3-3-1
#2-3-3-3 (class ObjectA)
#2-3-3-5 (class ObjectB)
(read)
```

new

oodebug command

Creates an object.

```
new persistentClassName nearObject
```

Parameters See “Command Parameters” on page 200.

Discussion Use to create any object in the schema, except for `ooFDObj` or `ooDBObj`, which require special creation semantics.

Warning: The `new` command does not invoke a constructor after creating an object. After the `new` command has been used to create an object, VArray fields are of size 0, associations are empty, and all other fields are undefined. Therefore, when you use this command, you must manually perform the operations that a

constructor would have performed: oodebug commands assign, add, sub, and del.

Mode Available in update mode only.

next

oodebug command

Iterates to the next object in an iterator and displays its contents.

```
next
```

Discussion Use in conjunction with the iter command to follow an object's relationships (associations).

Example Within a CASE federated database, the database called Functions contains a container called Main of class FunctionRep, which inherits from ooContObj. The Main function represents the main function of a C++ program, and calls three functions, openFiles, processTransactions, and closeFiles. These functions are also stored as containers of the same class. A relationship named Calls helps locate the three functions called by the main program. To ask the database to display all of the functions called by container Main, you initialize an iterator, and iterate through all relationships.

```
(read) iter Functions.main Calls
iterator initialized

(read) next
(read) next
FunctionRep #2-3-3-1 = {
    %systemName = "openFiles"
    …
}
(read) next
FunctionRep #2-4-3-1 = {
    %systemName = "processTransactions"
    …
}
(read) next
FunctionRep #2-5-3-1 = {
    %systemName = "closeFiles"
    …
}
…
(read) next
end of iteration

(read)
```

oodebug

convenience function

Alias for invoking the oodebug tool within a UNIX C++ debugger (dbx and its variants).

In a UNIX debugger, enter the following at the debugger command prompt:

```
oodebug objectRef_or_Handle
```

Parameters

objectRef_or_Handle

Object reference or handle within the program you are debugging that identifies the object you want to view or change. *objectRef_or_Handle* indicates the name of an active variable of the class ooRef*(className)* or ooHandle*(className)*.

Discussion

The oodebug convenience function makes all of the oodebug commands available from within the debugger except for those that manage transactions and change modes (such operations are controlled by the application you are debugging). Once you start oodebug, the debugger's commands are not available until you terminate oodebug.

Before you can invoke oodebug, you must link the application you are debugging with the debug-enabled Objectivity / DB library and define the convenience function to your debugger (see "Running oodebug in a C++ Debugger" on page 62).

ooprint

convenience function

Alias for viewing the contents of an object within a C++ debugger, without invoking the oodebug convenience function.

- In the Windows Visual C++ debugger, use the following function prototype:

  ```
  ooprint(&objectRef_or_Handle)
  ```

- In a UNIX debugger (dbx and its variants), enter the following at the debugger command prompt:

  ```
  ooprint objectRef_or_Handle
  ```

Parameters

objectRef_or_Handle

Object reference or handle within the program you are debugging that identifies the object you want to view. *objectRef_or_Handle* indicates the name of an active variable of the class ooRef*(className)* or ooHandle*(className)*.

Discussion

The ooprint convenience function is identical to the print command used within oodebug, except ooprint does not have the optional *fieldExpression* argument.

Before you can invoke `ooprint`, you must link the application you are debugging with the debug-enabled Objectivity / DB library and perform setup steps appropriate to your platform (see “Using ooprint in a C++ Debugger” on page 63).

print

oodebug command

Displays an object’s contents in the same format used by the Objectivity / DB data browser (with default settings for the **View** menu). Displays all fields of this object unless you specify the name of a particular field to display.

`print` *`object [fieldExpression]`*

Parameters

See “Command Parameters” on page 200.

Example

This example displays the contents of an object identified by object identifier `2-3-3-3`.

```
(read) print 2-3-3-3
OBJECT_A #2-3-3-3 = {
    %scopeNames = {
        [#2-3-3-1] "object_a_1"
    }
    int32 a_int = 10
    STRUCT_W a_struct_w = {
        uint8 w_uint8 = 100
        int16 w_int16 = -1000
        pointer w_ptr = 0x0
    }
    ENUM_Z a_enum_z = F_ZERO 0
    ooHandle(OBJECT_B) b_assocs[] <-> a_assoc = {
        #2-3-3-7
        #2-3-3-9
        #2-3-3-11
    }
    ooHandle(OBJECT_B) uni_b_assocs[]
    : prop(lock,delete), inhibit(delete), version(move) = {
        #2-3-3-7
        #2-3-3-9
    }
    ooHandle(OBJECT_C) uni_c_association
    : version(copy) = #2-3-3-17
    ooHandle(OBJECT_D) d_assoc <-> a_assoc
    : prop(lock), version(drop) = #2-3-3-19
}
```

```
(read)
```

This example shows how to display the contents of the x field of the object identified by object identifier 5-3-3-3.

```
(read) print 5-3-3-3 x
55

(read)
```

quit

oodebug command

Terminates oodebug.

```
quit
```

Discussion

If no changes are pending, quit aborts the transaction that originated when you started oodebug or issued a previous commit or abort command. If changes are pending in the current transaction, you must explicitly save or ignore changes with the commit or abort commands. Pending changes are indicated by an asterisk in your command prompt: (*read) or (*update).

Example

This example shows how to terminate oodebug if changes are pending.

```
(*update) quit
pending updates, please commit or abort
(*update) commit
transaction committed
(update) quit
%
```

read

oodebug command

Sets the mode to read mode.

```
read
```

Discussion

Read mode allows you to enter all oodebug commands, except those that change the structure or contents of the federated database. Read mode is indicated by one of two prompts: (read), or (*read), depending on whether there are pending changes to commit to the federated database.

Mode

Available in read and update mode, but *not* from within dbx.

Example

This example shows how to select read mode.

```
(*update) read
```

```
(*read)
```

stats

oodebug command

Displays statistics about the federated database that are generated during the current session. For information about `stats` output, see the chapter on performance in the Objectivity/C++ programmer's guide.

```
stats
```

Mode

Available in update mode only.

Example

This example shows sample Database Statistics.

```
 (read) stats

************************************************
Object Manager  Statistics Wed Aug 09 21:21:22 2000

** Number of federated DBs created  => 0
** Number of federated DBs opened   => 1
** Number of federated DBs closed   => 1
** Number of federated DBs deleted  => 0
**
** Number of databases created      => 0
** Number of databases opened       => 2
** Number of databases closed       => 0
** Number of databases deleted      => 0
**
** Number of containers created     => 0
** Number of containers opened      => 9
** Number of containers closed      => 11
** Number of containers deleted     => 0
**
** Number of objects Created        => 0
** Number of objects opened         => 8390304
** Number of multiple opens         => 892
** Number of new versions           => 0
** Number of objects closed         => 8390304
** Number of multiple  closes       => 892
** Number of objects deleted        => 0
**
** Number of objects named          => 0
** Number of new OCBs               => 256
** Number of new associations       => 0
** Number of disassociations        => 0
** Number of associations resized   => 0
```

```
** Number of transactions  started  => 1
** Number of transaction commits    => 1
** Number of commit and holds       => 0
** Number of transaction aborts     => 0
** Number of system aborts          => 0
************************************************
************************************************
Storage Manager  Statistics Wed Aug 09 21:21:22 2000

** Page size                        => 32768
** Number of buffers used           => 500
** Number of large buffer entries   => 200
** Number of SM objects opened      => 8390241
** Number of SM objects created     => 0
** Number of objects still opened   => 0
** Number of buffers read           => 5
** Number of disk  reads            => 33865
** Number of old pages written      => 0
** Number of new pages written      => 0
** Number of openHash calls         => 1
** Number of hash overflows         => 0
** Number of times OCs extended     => 0
** Number of Pages added to OCs     => 0
** Number of SM objects resized     => 0
*********************************************
(read)
```

sub

oodebug command

Removes a relationship (association) between two objects.

`sub` *`object nameOfLink objectToSubtract`*

Parameters — See “Command Parameters” on page 200.

Discussion — You can use this function to subtract an object from any type of relationship.

- Used on a to-many relationship, this is functionally equivalent to the `sub_`*`nameOfLink`* C++ member function.
- Used on a to-one relationship, this is functionally equivalent to the `del_`*`nameOfLink`* C++ member function.

Mode — Available in update mode only.

update

oodebug command

Sets the mode to update mode.

```
update
```

Discussion

Update mode allows you to enter commands that change the structure and contents of the federated database. Update mode is indicated by one of two prompts: `(update)`, or `(*update)`, depending on whether there are pending changes to commit to the federated database. *Update mode is intended for use by database administrators and advanced users only.*

Mode

Available in read and update modes, but *not* from within `dbx`.

Example

This example shows how to select update mode.

```
(read) update
(update)
```

whatis

oodebug command

Displays an object's class or `struct` declaration.

```
whatis object
```

Parameters

See "Command Parameters" on page 200.

Discussion

The format is the same as shown by the Type Browser.

Example

This example shows how to display the class or `struct` declaration of an object with an object identifier of 5-3-3-1.

```
(read) whatis 5-3-3-1
class filetree : ooContObj {
 int32 numfiles;
};
```

A

Running Objectivity Servers on Windows

Objectivity servers include the lock server, the Advanced Multithreaded Server (AMS), and the Objectivity/SQL++ ODBC server. On Windows platforms, you manage these servers using the Objectivity Network Services tool that is provided with Objectivity/DB. This causes each server to run even when no user is logged on, and to start automatically whenever the system boots.

This appendix describes steps for:

- Starting and stopping Objectivity servers
- Configuring Objectivity servers
- Uninstalling and reinstalling Objectivity servers

Starting and Stopping an Objectivity Server

You use the Objectivity Network Services tool to start and stop Objectivity servers. To start or stop a server:

1. Log on as administrator (Windows NT or Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. In Objectivity Network Services, select the desired server and either:
   - Click **Start** to start the server. At this point, the server will run while your system is running, and will continue to run until stopped.
   - Click **Stop** to stop the server. Before you stop an Objectivity server, consult the documentation for that server.

Configuring an Objectivity Server

You use the Objectivity Network Services tool to configure an Objectivity server. To do this:

1. Log on as administrator (Windows NT or Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. In Objectivity Network Services, select the desired server and click **Configure**. Different options are available, depending on the server you selected:
   - You can specify boot-file names as arguments to the lock server for recovery.
   - You can assign a different TCP / IP port to the lock server and AMS.

     Note: If you change the TCP / IP port for a server, you must assign the *same* TCP / IP port to that server on every other host that runs an Objectivity / DB process or hosts an Objectivity / DB file.

NOTE See the Objectivity / SQL++ book for further information about configuring an Objectivity / SQL++ ODBC server.

---

4. When you have finished entering options for the selected server, click **OK**.

Specifying a Service's Logon Account

On Windows NT or Windows 2000, you must start each server under a logon account that has appropriate access permissions.

Windows NT

To specify a logon account on Windows NT:

1. Log on as administrator.
2. If necessary, stop the lock server using the Objectivity Network Services tool.
3. Open the Control Panel and double-click **Services**.
4. In Services, select the desired Objectivity service, and click **Startup**.
5. Specify the desired logon account.
6. Use the Objectivity Network Services tool to restart the service.

Windows 2000

To specify a logon account on Windows 2000:

1. Log on as administrator.
2. If necessary, stop the lock server using the Objectivity Network Services tool.
3. Open the Control Panel; double-click **Administrative Tools** and then double-click **Services**.
4. In Services, double-click the desired Objectivity service.
5. In Properties, click **Log On** and specify the desired logon account.
6. Use the Objectivity Network Services tool to restart the service.

Uninstalling and Reinstalling an Objectivity Server

Objectivity servers are installed automatically during product installation. If you do not plan to run a particular server on your machine, you can uninstall it. To uninstall and subsequently reinstall an Objectivity server, you:

1. Log on as administrator (Windows NT or Windows 2000 only).
2. Click **Start** and point to **Programs**. In the Objectivity submenu, select **Objectivity Network Services**.
3. In Objectivity Network Services, select the desired server and either:
   - Click **Uninstall** to uninstall the server.
   - Click **Install** to reinstall the server.

Index

A

abbreviating
- oodebug commands 199
- tool options 141

abort, oodebug command 203

access permissions 29
- AMS 29, 93
- database file 75
- lock server 29, 82, 83

access status of database 67

accessing federated database 47

add, oodebug command 203

administration tasks
- backup and restore 25, 97
- creating and modifying
  - databases 25, 65
  - federated databases 24, 35
- getting information 25, 37, 53, 68
- maintenance and recovery 26, 119
- managing Objectivity servers 26, 77, 91, 215
- overview 24

administration tools (see tools)

Advanced Multithreaded Server (see AMS)

AMS 23, 91
- access permissions 29, 93
- changing TCP/IP port
  - on UNIX 95
  - on Windows 95, 216
- checking if running 93, 155
- compared to NFS 92
- configuring on Windows 216
- guidelines for choosing 92, 130
- logon account on Windows 216
- oostartams 93, 194
- oostopams 94, 194
- output on Windows 95
- setting timeout period 96
- specifying files 31
- starting 194
  - on UNIX 93
  - on Windows 93, 215
- stopping 194
  - on UNIX 94
  - on Windows 94, 215
- timeout errors 96

application
- failures, recovery from 120
- IPLS application 79
- processes 21

argument, tool 141

assign, oodebug command 204

assigning
- AMS port
  - on UNIX 95
  - on Windows 95, 216
- lock-server port
  - on UNIX 86
  - on Windows 86, 216

attaching
- database file 71
- multiple database files 73

attributes
- autonomous partition
  - changing 43
  - listing 37

database 66
changing 74, 152
listing 68
federated database 36
changing 42, 149
listing 37
automatic recovery (see recovery)
autonomous partition 19
attributes 36
changing 43
listing 37
boot file 19, 29, 37
copying a federated database and 42
file 19, 37
filename 27
identifier 49
initial 36, 39
listing files 37
lock server for 79
system name 19, 27
system-database file 19, 37

B

backup 97, 147
(see also restoring)
before software upgrade 103
boot file 108, 148, 191
creating a backup set 107
database corruption and 108
deleting a backup set 109
diary 102
event
defined 98
listing 109
failures 108
history, obtaining 109
incremental 99
levels 99
medium 98
performing 107
processing volumes during 114
querying a backup set 109, 187
schedules
defining 104
examples 105
guidelines 104
low- and high-risk 106
scripts on UNIX 116
set
creating 107, 163
defined 98
deleting 109, 167
strategies 104
to tape 116
tools for 25
user access during 102
volume 98
boot file 20
autonomous partition 19, 29, 37
backup 108, 148, 191
federated database 18
name 29
format for automatic recovery 124
placement for Windows clients 132
browser 53, 149, 196
data 54
opening
on UNIX 58
on Windows 57
query 56
type 55
buffer page (see page)

C

cache, see Objectivity/DB cache
case sensitivity 33
changing
AMS port
on UNIX 95
on Windows 95
autonomous-partition attributes 43
database attributes 74, 152
database identifier 74
database system name 74
federated-database attributes 42, 149

lock-server host 84
lock-server port
on UNIX 86
on Windows 86
checking
AMS 93, 155
lock server 81, 155
client host 23, 129
failures, recovery from 121
commit, oodebug command 205
configuring
AMS on Windows 216
backup scripts on UNIX 116
lock server on Windows 216
consolidating
databases 75
federated database 46
container
identifier 49
identifier appears out of range 179, 182
maximum number in database 51
number of logical pages in 51
convenience function
oodebug 62
ooprint 63
copying
database file 70, 160
federated database 42, 162
corruption
detecting during a backup 108
detecting during a dump 44
ootidy 47, 76
restoring after detecting 110
creating
backup set 163
database 25, 69, 183
federated database 24, 38, 185
mapping file for restore 112
unattached database 70
customer support 13

D

data browser (see browser)
data-server host 23, 129
UNIX 132
Windows 130
data-server software 23, 91
AMS 91, 130
best for automatic recovery 124
choosing 92
local files 23
NFS 91, 130
remote files 23
Windows Network 91, 130
database
access
permissions 75
troubleshooting 76
attaching
file 71
guidelines 73
multiple files 73
attributes 66
changing 74
listing 68
consolidating 75
copying to file 70, 160
creating 69, 183
deleting 75, 165
distributed (see distributed databases)
duplicating in a federated database 72
file 19, 66
filename 28
format, finding 173
getting attribute values 68
getting file information 68
identifier
changing 74
displaying 68
format 49, 66
setting 69
image (see database image)
maximum number 51
maximum size 51
moving file 70

moving to another federated database 71
page size, displaying 68
read-only 67, 153
restricting access 67, 75
system name 19, 28
changing 74
displaying 68
tidying 75
tools 141
for creating and modifying 25
for getting information 25
database image 19, 67
D-C-P-S format 49
DDL processor
creating 159
debugging
federated database 164, 199
ooprint 63
defragmenting (see tidying)
del, oodebug command 205
delete, oodebug command 205
deleting
backup set 109, 167
database 75, 165
federated database 43, 166
deploying Objectivity/DB applications 133
distributing libraries
UNIX 136
Windows 135
distributing Objectivity executables 134
installing a federated database 138
diary, backup 102
directory
AMS 93
journal 20
Objectivity server 80
disk space requirements 50
displaying
autonomous-partition attributes 37
database attributes 68
database file information 68
federated-database attributes 37
federated-database file information 38
distributed databases 23, 129
recovery considerations 124
distributing at deployment
libraries
UNIX 136
Windows 135
Objectivity executables 134
drive mapping, virtual 131
DRO abbreviation 12
dumping
database corruption and 44
evolved schema 191
failure 44
federated database 44, 168
information not dumped 44
duplicating database in a federated database 72

E

end-user tasks
distributing runtime tools 134
installing a federated database 138
environment variable
OO_FD_BOOT 33, 58
OO_RPC_TIMEOUT 87, 96
setting 33
exclusive lock 78
external references, ooload does not resolve 45

F

failures
application 120
client host 121
lock server 122
federated database
access, troubleshooting 47
attributes 36
changing 42
listing values 37
backing up 97
catalog information 113

changing
attributes 42
lock-server host 84
copying 42, 162
creating 38, 185
database format 173
debugging 164, 199
defragmenting 195
deleting 43, 166
dumping 44, 168
evolved schema 191
file 18, 36
filename 27
garbage collection 173
getting file information 38, 172
identifier 48
setting 39
installing 175
journal directory 39
listing files 37, 171
loading 44, 179
lock server for 77
moving 43
OO_FD_BOOT environment variable 33
reference number 48
restoring from backup 109
storage-page size
changing 43, 44
choosing 39
displaying 38
system name 18, 27
system-database file 18, 36
tidying 46
tools
for creating and modifying 24
for getting information 25
file
(see also boot file)
(see also journal file)
database 19, 66
listing
autonomous partition 37
federated database 37
Objectivity / DB 17
placement in mixed OS environment 132
specifying local 30
specifying remote
AMS 31
NFS 31
Windows Network 32, 131
system-database
of autonomous partition 19, 37
of federated database 18, 36
filename
boot file 29
case sensitivity 33
database file 28
host format 30
journal file 28
spaces in 32
system-database file
of autonomous partition 27
of federated database 27
FTO abbreviation 12

G

garbage collection tool 173
generated filename
database 28
journal file 28
getting information, tools for 25

H

header page (see page)
help, oodebug command 206
host format filename 30

I

identifier
autonomous partition 49
getting 38
database 49
getting 68
D-C-P-S format 49
federated database 48
getting 38
setting 39

image (see database image)
incremental backup 99
indexes, ooload does not preserve 45
in-process lock server
(see lock server, in-process)
installing
end-user federated database 138
federated database 175
IPLS abbreviation 12
iter, oodebug command 206

J

journal directory 20
specifying 39
journal file 19
automatic recovery and 119
federated database 39
name 28

K

kernel, Objectivity/DB 21, 23
killing a lock server (see stopping)

L

large objects 21
latches 179, 182
listconts, oodebug command 207
listdbs, oodebug command 206
listing
active transactions 48
autonomous-partition attributes 37
autonomous-partition files 37
database attributes 68, 152
federated-database attributes 37, 149
federated-database files 37, 171
locks 181
transaction information 48
waiting transactions 48, 178
listobjs, oodebug command 207
loading
federated database 44, 179
information not loaded 45
local data server (see Objectivity/DB kernel)
lock file, recovery 127, 159
lock server 20, 77
access permissions 29, 82, 83, 124
applications cannot connect 88
automatic recovery 216
initiating 122
changing TCP/IP port
on UNIX 86
on Windows 86, 216
checking if running 81, 155
configuring on Windows 216
crash recovery 127
enabling automatic recovery 122
failures, recovery from 122
host 23, 78, 129
changing 84
UNIX 132
Windows 131
in-process 79
logon account on Windows 216
output on Windows 86
setting timeout period 87
standard 79
starting 182
on UNIX 82, 83
on Windows 81, 82, 215
problems 87
stopping 177
on UNIX 84
on Windows 83, 215
problems 84
system directory 80
timeout errors 87
uninstalling on Windows 82

locks
exclusive lock 78
intention lock 181
listing 181
problems 87
read lock 78
update lock 78
logical page (see page)

M

manual recovery 125, 156
mapping drives, virtual 131
mapping file
attaching multiple databases 73, 145
specifying locations for restore 112, 189
Microsoft Windows (see Windows)
moving
database file 70
database to another federated database 71
federated database 43

N

network
considerations 23
impact on performance 129
share names 131
Network File System (see NFS)
new, oodebug command 207
next, oodebug command 208
NFS 23, 91
compared to AMS 92
specifying files 31
user ID of Windows application 131
Windows client 131

O

object
browsing 53
iterating 208
large 21
object identifier (OID)
autonomous partition 49
container 49
database 49
D-C-P-S format 49
ooload does not preserve 45
persistent object 49
Objectivity Network Services 24, 26, 142, 144
using 215
Objectivity server system directory 80
Objectivity servers
AMS 91
lock server 77
Objectivity / SQL++ ODBC server 215
tools for managing 26, 215
Objectivity/DB
backup 97
basics 17
cache 21
files 17
kernel 21, 23
naming files 29
processes 17
recovery 119
release and database format 173
tools 141
Objectivity/SQL++ ODBC server 215
ObjyTool 24, 26, 142, 144
ODMG abbreviation 12
OID (see object identifier)
OO_FD_BOOT environment variable 33, 58
oo_local_host 32
OO_RPC_TIMEOUT environment variable 87, 96
ooams-xx service 95
ooattachdb 25, 71, 142, 144
oobackf script 116
oobackup 25, 107, 142, 147
ootidy and 108
oobrowse 25, 57, 142, 149
oochange 24, 25, 37, 42, 85, 142, 149
oochangedb 25, 68, 70, 74, 138, 142, 152
oocheckams 26, 93, 142, 155

oocheckls 26, 81, 93, 142, 155
oochkxx.dll 135
oocleanup 26, 48, 84, 85, 121, 125, 142, 156
ooconfig 26, 142, 159
oocopydb 25, 70, 71, 142, 160
oocopyfd 24, 42, 138, 142, 162
oocreateset 25, 107, 142, 163
oodbxx.dll 135
ooddlx 39, 45
  created by ooconfig 159
  schema, loading a 39
oodebug 26, 142, 164, 199
  aliases 209
  command parameters 200
  convenience function 62, 209
  modes
    read 60, 199
    update 60, 199, 214
  quitting 61, 63
  running
    as separate process 60
    from UNIX C++ debugger 62
oodebug commands
  abort 203
  add 203
  assign 204
  commit 205
  del 205
  delete 205
  help 206
  iter 206
  listconts 207
  listdbs 206
  listobjs 207
  new 207
  next 208
  oodebug convenience function 209
  ooprint convenience function 209
  print 210
  quit 211
  read 211
  stats 212
  sub 213
  terminate 63
  update 214
  whatis 214
oodeletedb 25, 71, 75, 142, 165
oodeletefd 24, 43, 142, 166
oodeleteset 25, 109, 142, 167
oodump 26, 44, 142, 168
oodumpcatalog 25, 37, 68, 142, 171
ooendb script 116
ooendr script 116
oofile 25, 38, 68, 142, 172
oogc 26, 143, 173
ooinstallfd 24, 43, 143, 175
ookillls 26, 84, 143, 177
oolistwait 25, 48, 84, 143, 178
ooload 26, 44, 45, 143, 179
oolockmon 25, 26, 84, 85, 143, 181
oolockserver 26, 82, 143, 182
ools-xx service 86
oonewdb 25, 69, 143, 183
oonewfd 24, 39, 45, 143, 185
ooprint convenience function 63, 209
ooqueryset 25, 109, 143, 187
oorecvr.LCK file 127, 159
oorestfa script 116
oorestfb script 116
oorestore 25, 109, 143, 188
ooschemadump 143, 191
ooschemaupgrade 24, 143, 192
oostartams 26, 93, 143, 194
oostopams 26, 94, 143, 194
oostrtb script 116
oostrtr script 116
ootapebackup 116
  configuring scripts 116
ootaperestore 117
  configuring scripts 116
ootidy 26, 46, 143, 195
  guidelines for using 47
  oobackup and 47, 76
ootoolmgr 25, 58, 143, 196
option, tool 141

P

page
buffer page 22
header page, for large object 21
logical page 21
in object identifier (OID) 49
page size
changing 43, 44
displaying 38, 68
storage page 21
federated-database attribute 36
partition (see autonomous partition)
permissions (see access permissions)
point of restore 102
port
AMS 94
conflict 85, 94
lock server 85
print, oodebug command 210
printing object contents in debugger 63

Q

query browser (see browser)
quit, oodebug command 211

R

read lock 78
read, oodebug command 211
read-only database 67
recovery 119
application failures 120, 126
automatic 119, 156
boot-file name format 124
client-host failures 121, 126
lock file for 127, 159
lock-server failures 122, 127
manual 125, 156
oocleanup failure 127
reference number for federated database 48
remote data server (see AMS)
replicated database (see database image)
restoring 188
(see also backup)
after detecting corruption 110
entire federated database 102
failure 110
from backup 109
from tape 117
mapping file 112
point of restore 102
processing volumes during 115
scripts on UNIX 116
to multiple locations 112
to original location 110
to single location 111
user access while 102, 111
restricting access
database file 75
read-only database 67
return status 24
rolling back incomplete transactions 119
runtime tools 134

S

schedule, defining backup schedule 104
schema 18
dumping after evolution 191
loading with ooddlx 39
upgrading after evolution 192
scripts for backup 116
server
(see data-server software)
(see Objectivity servers)
setting
AMS timeout period 96
lock server timeout period 87
software upgrade, backing up before 103
spaces in filenames 32

specifying
filenames 27
local files 30
remote files
AMS 31
NFS 31
Windows Network 32
standard lock server 79
starting
AMS 194
on UNIX 93
on Windows 93, 215
in-process lock server
on UNIX 83
on Windows 82
lock server 182
on Windows 215
standard lock server
on UNIX 82
on Windows 81
stats, oodebug command 212
stopping
AMS 194
on UNIX 94
on Windows 94, 215
lock server 177
on UNIX 84
on Windows 83, 215
storage page (see page)
sub, oodebug command 213
system name
(see autonomous partition)
(see database)
(see federated database)
system-database file
filename 27
of autonomous partition 19, 37
of federated database 18, 36

T

TCP/IP
configuration problems 88
port
AMS 94
conflict 85, 94
lock server 85
tidying
database 75
federated database 46, 195
timeout period
AMS 96
lock server 87
tool
argument 141
option 141
overview 24
return status 24
Tool Manager 58, 196
tools 85
graphical interface
Objectivity Network Services 144
ObjyTool 144
oobrowse 149
Objectivity Network Services 24, 144, 215
ObjyTool 24, 144
ooattachdb 71, 144
oobackup 107, 147
oobrowse 149
oochange 37, 42, 85, 149
oochangedb 68, 70, 74, 138, 152
oocheckams 155
oocheckls 81, 93, 155
oocleanup 48, 85, 121, 125, 156
ooconfig 159
oocopydb 70, 71, 160
oocopyfd 42, 138, 162
oocreateset 107, 163
oodebug 164
oodeletedb 71, 75, 165
oodeletefd 43, 166
oodeleteset 109, 167
oodump 44, 168
oodumpcatalog 37, 68, 171

oofile 38, 68, 172
oogc 173
ooinstallfd 43, 175
ookillls 84, 177
oolistwait 48, 84, 178
ooload 44, 45, 179
oolockmon 84, 181
oolockserver 82, 182
oonewdb 69, 183
oonewfd 39, 45, 185
ooqueryset 109, 187
oorestore 109, 188
ooschemadump 191
ooschemaupgrade 192
oostartams 93, 194
oostopams 94, 194
ootapebackup 116
ootaperestore 117
ootidy 46, 195
ootoolmgr 58, 196
transaction
aborting 203
committing 205
handling 61
listing 48, 178
recovering incomplete 119, 156
troubleshooting
database access 76
federated database access 47
listing active transactions 48
listing transaction information 48
listing waiting transactions 48
solving lock problem 87
starting lock server 87
stopping a lock server 84
type browser (see browser)

U

UNC names
for remote Objectivity/DB files 32, 131
uninstalling a server on Windows 217
Universal Naming Convention (see UNC names)
UNIX
browser 58
clients 31, 130, 132
setting up automatic recovery 122
data servers 132
setting up automatic recovery 124
lock-server host 132
running oodebug in C++ debugger 62
starting server
AMS 93
lock server 82, 83
stopping server
AMS 94
lock server 84
user account for Objectivity servers 82, 83, 93
viewing objects in debugger 63
update lock 78
update, oodebug command 214
upgrading
applying schema changes 192
getting database format 173
utilities (see tools)

V

viewing
class definitions 55
object contents in debugger 63
objects 54
virtual drive mapping 131

W

whatis, oodebug command 214
Windows
AMS output 95
browser 57
clients 31, 130
setting up automatic recovery 121
data servers 130
setting up automatic recovery 124
lock-server host 131
lock-server output 86

logon account for Objectivity servers 82, 93, 216
running Objectivity servers 215
starting
AMS 215
lock server 215
starting server
AMS 93
lock server 81, 82
stopping
AMS 215
lock server 215
stopping server
AMS 94
lock server 83
UNC names and Objectivity / DB 32, 131
uninstalling an Objectivity server 217
viewing objects in debugger 63
virtual drive mapping 131
Windows Network
for accessing Objectivity / DB files 130
specifying files 32
WSAEADDRINUSE error message 86