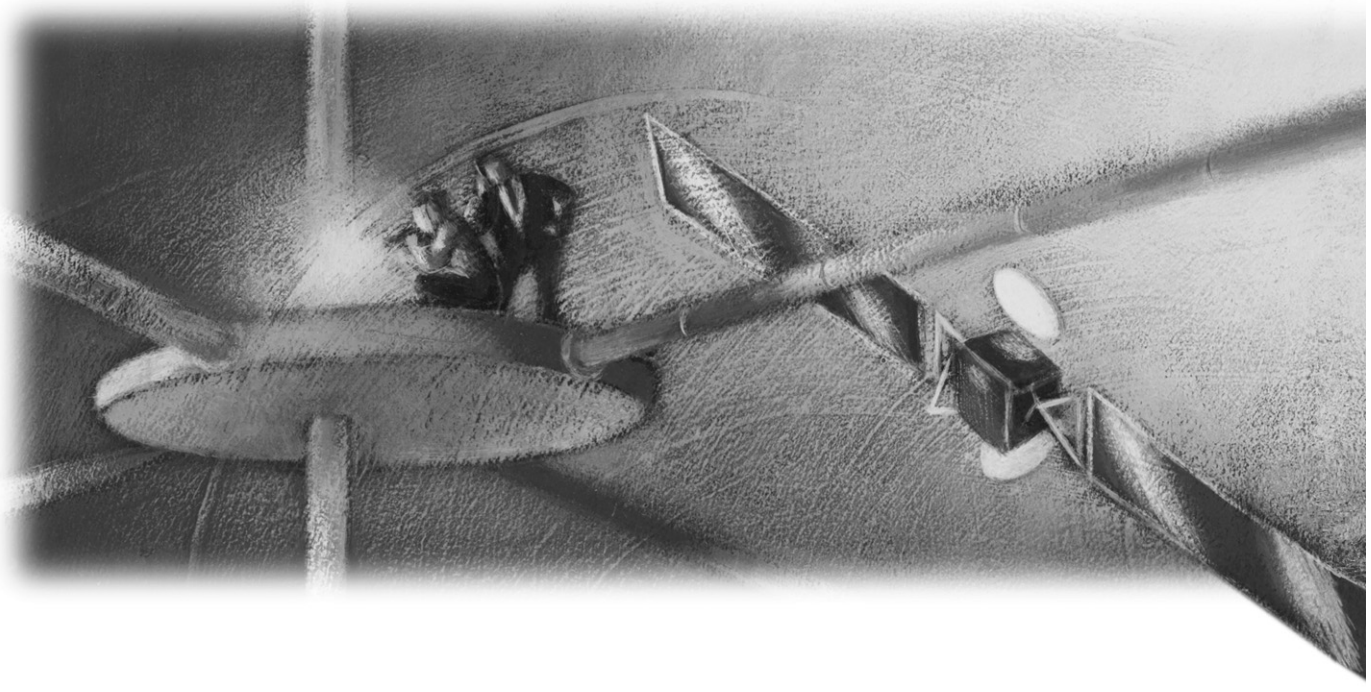

# Objectivity Release Notes

Release 10.0

# Objectivity Release Notes

Part Number: 10-RN-0

Release 10.0, February 24, 2010

# Contents

# Getting Help

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

## How to Reach Objectivity Customer Support

You can contact Objectivity Customer Support by:

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

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

## Before You Call

Please be ready to submit the following to Objectivity Customer Support:

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

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

# 1

# Release Overview

This release note describes the additions and changes made to Objectivity products and documentation in Release 10.0. This chapter provides an overview of these changes. This chapter summarizes:

- New products and features
- Updated products
- New architectures
- New and updated books

---

***For Additional Release Information***

Consult the *detailed release notes* (a series of plain text files), for the latest information about supported platforms and compilers, required operating-system patches and compiler patches, the open and fixed software problems in Release 10.0, and documentation errata and corrections.

You can find the detailed release notes for Release 10.0 on your distribution CD and on the Objectivity Technical Support web site. Call Objectivity Customer Support to get access to this web site.

You can also find the detailed release notes and the *Objectivity Release Notes* files for previous releases on the Objectivity Technical Support web site.

---

# New Products and Features

The following table lists the major new Objectivity products and features in this release.

| Product or Feature | Description | See |
|---|---|---|
| Objectivity/.NET for C# | .NET/C# programming interface to Objectivity/DB. | page 57 |
| Objectivity Developer Network | Web site providing technical information and resources for Objectivity/DB developers and integrators. | page 17 |
| Objectivity/DB performance analyzer | Tool for evaluating your application and detecting parameter settings and code constructs that might compromise performance. | page 24 |
| Session snapshot for debugging | Method for capturing a set of information about the state of an application in the current session. | page 17 |
| Faster ooupgrade tool | Improved speed and reduced requirements for free disk space. Very large databases have been observed to upgrade 30 times faster. | page 25 |
| Performance optimization mechanisms | Support for changing whether a database grows as a sparse file when extending a container in the database.<br><br>Support for optimizing a container's size when clustering new objects in it.<br><br>Support for optimizing the order for scan iterations.<br><br>Mechanism for maximizing the performance benefit of comparison arrays in a sorted collection by creating the collection so that it uses comparison arrays alone to determine the equality of two persistent objects.<br><br>Method for enabling system-managed creation of an I/O read-ahead thread.<br><br>(*Objectivity for Java*) Ability to enforce the standard relationship-persistence policy for the lifetime of a session. | page 23<br><br>page 18<br><br>page 19<br><br>page 23<br><br>page 20<br><br>page 49 |

| Product or Feature | Description | See |
|---|---|---|
| Objectivity/DB enhanced object qualification | Support for more data types, for more operators, and for qualifying objects according to conditions on related objects. | page 21 |
| | (*Objectivity/C++*) Enhanced support for qualifying objects one at a time. | page 35 |
| | (*Objectivity/C++*) Access to the underlying expression tree interface. | page 19 |
| | (*Objectivity/C++*) Support for creating custom operators. | page 21 |
| Extended support for C++ date and time data | New Objectivity/C++ classes for representing general-purpose date and time information. | page 31 |
| Objectivity/DB plugins | Mechanism for extending or customizing certain features of Objectivity servers or Objectivity/DB applications. | page 20 |
| Enhanced tool for disk-format conversion | Support for converting storage pages between 32-bit and 64-bit architectures. | page 25 |
| Extended support for Unicode strings | (*Objectivity/C++*) Ability to iterate over complete characters in a Unicode string using instances of character iterator classes. | page 32 |
| | (*Objectivity/C++*) Ability to use Unicode literal strings (wide character strings) as predicate strings in `scan` and `parallelScan` methods as well as in object qualifiers. | page 34 |
| | (*Objectivity/C++*) Ability to use Unicode strings as keys in name maps. | page 36 |
| | (*Objectivity/C++*) Ability to use Unicode strings as key field values for string typed attributes in indexes. | page 36 |
| | (*Objectivity for Java*) Ability to use Unicode strings as file and directory names on Windows platforms. | page 50 |
| Internet protocols | Support for Internet Protocol version 6 (IPv6) networks. | page 20 |

# Updated Products

## Products With New, Changed, or Removed Features

| Product Name | Product Description | See |
|---|---|---|
| Objectivity/DB | Distributed object database. | page 17 |
| Objectivity/DB High Availability (Objectivity/HA) | Objectivity/DB option supporting autonomous partitions and data replication. | page 26 |
| Objectivity/C++ | C++ programming interface to Objectivity/DB. | page 29 |
| Objectivity/DB Active Schema for C++ | C++ programming interface for reading and modifying an Objectivity/DB schema dynamically. | page 38 |
| Objectivity for Java | Java programming interface to Objectivity/DB. | page 43 |
| Objectivity/DB Active Schema for Java | Java programming interface for reading and modifying an Objectivity/DB schema dynamically. | page 54 |
| Objectivity/.NET for C# | C# programming interface to Objectivity/DB. | page 57 |

## Products Rebuilt for Compatibility

| Product Name | Product Description |
|---|---|
| Objectivity/DB In-Process Lock Server (Objectivity/IPLS) | Objectivity/DB option that enables a C++, Java, or Smalltalk database application to run a lock server within the same process to improve performance. |
| Objectivity/DB Open File System (Objectivity/OFS) | Customizable interface between Objectivity/DB and hierarchic storage systems. |
| Objectivity/DB Parallel Query Engine (Objectivity/PQE) | Objectivity/DB option enabling applications to find persistent objects by searching separate portions of a federation in parallel. |
| Objectivity/DB Secure Framework (Objectivity/Secure) | Customizable interface between Objectivity/DB and standard security solutions. |
| Objectivity/C++ Data Definition Language (Objectivity/DDL) | Objectivity/C++ option for creating and maintaining a schema of persistence-capable class definitions. |
| Objectivity/Smalltalk for VisualWorks | Smalltalk programming interface to Objectivity/DB. |
| Objectivity/Python | Python programming interface to Objectivity/DB. |

| Product Name | Product Description |
|---|---|
| Objectivity/SQL++ | Server and tools providing ANSI-standard SQL access to Objectivity/DB with object-oriented extensions to SQL. |
| Objectivity/SQL++ ODBC Driver (Objectivity/ODBC) | Objectivity/SQL++ option that enables ODBC-compliant client applications to access an Objectivity/DB federated database. |

# New Platforms

The following table lists the new combinations of architecture, operating system, and addressing mode supported by Objectivity/DB Release 10.0.

| Architecture Name | Operating System | Addressing Mode |
|---|---|---|
| x86 | Windows Server 2008 R2<br>Windows 7 | 32-bit |
| x86-64 | Windows Vista<br>Windows Server 2008 R2<br>Windows 7 | 64-bit |

# Obsolete Platforms

Support has been removed for the following Objectivity UNIX platforms:

| Hardware | Operating System | Abbreviation |
|---|---|---|
| Compaq AlphaServer | all | `alphaosf1` |
| Sun SPARC | Solaris 2.6 | `solaris4` |

Support for the Sun SPARC platform now requires Solaris 8 or later.

Support has been removed for the following Objectivity Windows platforms:

| Hardware | Operating System | Abbreviation |
|---|---|---|
| Intel Pentium or greater | Windows 2000 Professional | Windows 2000 |
| | Windows 2000 Server | Windows 2000 |
| | Windows 98 | Windows 98 |
| | Windows NT Workstation<br>Windows NT Server | Windows NT |

# New and Updated Books

During installation, the online books for Objectivity products are placed in the documentation subdirectory of your Release 10.0 Objectivity/DB installation directory.

***NOTE*** All Objectivity online books are available on the Objectivity Technical Support web site. Contact Objectivity Customer Support to get access to this web site.

The following sections list the books delivered with this release.

## Books in PDF

This section lists Objectivity books in Portable Document Format (PDF).

- New or updated for Release 10.0:
  - *Objectivity Release Notes, Release 10.0* (this document)
  - *Installation and Platform Notes for Windows, Release 10.0*
  - *Installation and Platform Notes for UNIX, Release 10.0*
  - *Objectivity/DB Administration, Release 10.0*
  - *Objectivity/DB High Availability, Release 10.0*
  - *Monitoring Lock Server Performance, Release 10.0*
  - *Objectivity/C++ Programmer's Guide, Release 10.0*
  - *Objectivity/C++ Programmer's Reference, Release 10.0*
  - *Objectivity/C++ Data Definition Language, Release 10.0*

- *Objectivity/DB Active Schema for C++, Release 10.0*
- Objectivity for Java *Programmer's Guide, Release 10.0*
- *Objectivity/DB Schema Development, Release 10.0*
- *Objectivity/SQL++, Release 10.0*
- *Objectivity/SQL++ ODBC Driver User's Guide, Release 10.0*

- Earlier books used with Release 10.0:
  - *Objectivity/Smalltalk for VisualWorks, Release 8*

### Accessing PDF Books

After you install Objectivity/DB, you can access Objectivity books by clicking links from the following PDF file:

`installDir\doc\ObjyBooks.pdf` on Windows

`installDir/arch/doc/ObjyBooks.pdf` on UNIX

where:

| | |
|---|---|
| `installDir` | Release 10.0 Objectivity/DB installation directory. |
| `arch` | UNIX architecture name. |

You can view the books in PDF using the freely available Acrobat Reader software from Adobe Systems, Inc. You can obtain Acrobat Reader for your platform from Adobe's online services. Use your Web browser to access the web site `www.adobe.com`.

## Books in HTML

The following books are available in HTML format:

- New or updated for Release 10.0:
  - *Objectivity for Java Programmer's Guide, Release 10.0*
  - *Objectivity for Java Programmer's Reference, Release 10.0*
  - *Objectivity/DB Active Schema for Java Programmer's Reference, Release 10.0*
- Earlier books used with Release 10.0:
  - *Getting Started With Objectivity/Python, Release 9.2*
  - *Objectivity/Python Programmer's Reference, Release 9.2*

You can use your Web browser to access these books from the following HTML index files:

*installDir*\doc\java\index.html on Windows
*installDir*\doc\python\index.html

*installDir*/*arch*/doc/java/index.html on UNIX
*installDir*/*arch*/doc/python/index.html

where:

| | |
|---|---|
| *installDir* | Release 10.0 Objectivity/DB installation directory. |
| *arch* | UNIX architecture name. |

## Books in Compiled HTML Help

The *Objectivity/.NET for C# Programmer's Reference, Release 10.0* is provided as a Microsoft Compiled HTML Help file on the Windows platform.

- To open the help file, double-click on it:
  *installDir*\doc\ObjyNETcsharp.chm

# 2

# Base Products

This chapter describes new and changed features of Objectivity/DB and other language-independent Objectivity products in Release 10.0.

# Objectivity/DB (Database Services)

This section describes new, changed, and deprecated features of Objectivity/DB. See the Technical Support web site for software or documentation problems that have been fixed in this release.

## New Features

### Developer Network Web Site

The new Objectivity Developer Network web site provides technical information and resources, such as tutorials, FAQs, code examples, and sample applications, for Objectivity/DB developers and integrators.

http://devnet.objectivity.com

### Capturing Snapshots of Objectivity/DB Applications

You can capture a set of information (called a *snapshot)* about the state of an application with a method call in the current session. You can use this method to capture a snapshot at the time the method is called, or you can specify that the capture occur each time an exception is encountered. By default, a snapshot is emitted as plain text, but you can specify a comma-separated-values spreadsheet format.

A snapshot can be particularly useful when debugging a problem that occurred remotely in a conventional debugging environment.

For programming-interface details, see page 30 (Objectivity/C++) and page 43 (Objectivity for Java).

## Calculating the Optimal Container Size for Fast Access

You can now calculate the maximum number of logical pages that a container should have in order to optimize the performance of applications that make frequent updates to the basic objects in the container. Limiting a container's size to the calculated number of logical pages keeps the container's page map from becoming a large object. When a page map is a large object, it spans multiple storage pages, causing certain internal housekeeping operations to require more memory than they would if the page map could fit entirely within a single page.

A different number may be calculated for different containers, because the number is based on the size of a container's storage pages. The container-fill policy of a clustering strategy uses this calculation in its implementation; see "Container-Fill Policy in Clustering Strategies" on page 18.

For programming-interface details, see page 31 (Objectivity/C++) and page 46 (Objectivity for Java).

## Container-Fill Policy in Clustering Strategies

Clustering strategies now provide a policy for limiting the number of new logical pages that can be added to a container to accommodate new basic objects.

A clustering strategy may be permitted by its clustering priorities to add new logical pages to a container whenever such pages are needed for creating new basic objects. Prior to Release 10.0, the clustering strategy could potentially fill up a container by adding as many logical pages as the container would hold. However, filling up a container can cause overflow errors later if variable-size arrays or persistent-collection nodes need extra logical pages on which to expand. Furthermore, filling up a container causes its page map to become a large object, which can adversely affect the performance of an application that makes frequent updates to the container's basic objects.

In Release 10.0, you can set a clustering strategy's container-fill policy to specify a limit on how full a container can get. The new policy enables a clustering strategy to add new logical pages until one of the following is true:

- The container has the maximum possible number of logical pages, so all of the possible logical-page identifiers are in use.
- The container has 65,000 logical pages, which leaves some logical pages in reserve for future expansion. This limit is the new default for a standard clustering strategy.

- The container has as many logical pages as can be listed on its page map without turning the page map into a large object. This limit optimizes a container for fast update access to the basic objects it. (See "Calculating the Optimal Container Size for Fast Access" on page 18.)

For programming-interface details, see page 30 (Objectivity/C++) and page 46 (Objectivity for Java).

## Optimizing Scan Order

You can now optimize the order in which an object iterator scans a container or database for basic objects. You typically do so in response to advice provided by the performance analyzer; see "Performance Analyzer" on page 24.

Requesting the optimization may change the order in which the object iterator finds basic objects while scanning one or more containers. Changing the scan order can speed up the search significantly, but only if the containers being scanned are in databases that need to be tidied. There is no performance benefit, and possibly a small performance penalty, if the containers being scanned are in recently tidied databases.

For programming-interface details, see page 33 (Objectivity/C++) and page 49 (Objectivity for Java).

## Enhanced Object-Qualification Interface

There is now full access to the expression-tree representation underlying object qualification, so a non-Objectivity/DB query front-end may be adapted to perform an Objectivity/DB object scan. An expression tree may be programmatically validated in an iterative query-building approach, and the result saved in portable form—either for re-use or for parallelized use with the Objectivity/DB Parallel Query Engine.

---

***NOTE*** For information about using the expression-tree representation, see the Technical Support web site.

---

Several additional capabilities are built on the new object-qualification interface, including an enhanced predicate query language (page 21) and enhanced support for custom operators (page 21). Using the new interface, or any of the capabilities it supports, requires the following new shared libraries:

**Table 2-1:** Libraries Used by Enhanced Object Qualification Features

| Description | Windows | UNIX |
|---|---|---|
| Object qualification library | `ooObjectQualification.dll` | `libooObjectQualification.so` or `libooObjectQualification.sl` |
| Regular expression library | `oopcre.dll` | `libPCRE.so` or `libPCRE.sl` |
| Active schema library | `ooas100.dll` | `liboo_as.so` or `liboo_as.sl` |
| Internal support library | `ooObjectModel.dll` | `libooObjectModel.so` or `libooObjectModel.sl` |
| Internal support library | `ooSchemaModelInt.dll` | `libooSchemaModelInt.so` or `libooSchemaModelInt.sl` |

## I/O Read-Ahead Thread

Enabling the system-managed creation of an I/O read-ahead thread may improve performance when iterating over ordered collections or performing scan operations, especially for applications that access persistent data from remote file hosts. In parallel with accessing the data on a given current page, the I/O thread loads the next data page to be accessed.

For programming-interface details, see page 34 (Objectivity/C++), and page 50 (Objectivity for Java).

## Compatibility With Internet Protocol Version 6 (IPv6)

Objectivity/DB clients and servers will automatically adapt to networks using IPv6. Objectivity/DB continues to support Internet Protocol version 4 (IPv4) networks.

## Support for Plugins

You can now extend or customize certain features of Objectivity servers or Objectivity/DB applications by providing *plugins* for them. A plugin is an auxiliary executable component that is loaded on demand into an executing server or application. Plugins must be written in C++.

The performance analyzer in Release 10.0 in an example of a plugin; see "Performance Analyzer" on page 24. In addition, custom operators can be implemented as plugins; see "Enhanced Predicate Query Language" on page 21.

# Changed Features

## Enhanced Predicate Query Language

The predicate query language (PQL) that is used to create conditions for qualifying persistent objects now lets you qualify objects according to:

- Conditions on objects across to-many relationships or associations.
- Conditions on referenced or embedded objects.
- Conditions on objects in an array of referenced objects.
- Conditions on objects in a referenced collection.
- Conditions specified with path-based query criteria.

Release 10.0 expands the support for data types that can be used in queries to qualify objects. In addition to the data types available prior to Release 10.0, you can query for objects based on attribute values representing dates, times and all the Unicode strings supported by your Objectivity programming language interface. More operators are also provided in Release 10.0.

---

***NOTE*** The Objectivity/Assist query builder does not support the enhanced PQL functionality.

---

The enhanced PQL is a superset of the pre-Release 10.0 PQL, so predicate strings used in existing applications are fully compatible with the enhanced PQL and are therefore still valid.

The enhanced PQL is built on the expression-tree interface described in "Enhanced Object-Qualification Interface" on page 19.

For programming-interface details, see page 34 (Objectivity/C++) and page 50 (Objectivity for Java).

## Enhanced Support for Custom Operators

In previous releases, support for application-defined operators was limited to relational operators only. In Release 10.0, applications can create custom operators as substitutions for *any* of the predefined PQL operators provided by Objectivity. These custom operators have the same capabilities provided to the built-in ones.

**NOTE** For information about creating custom operators, see the Technical Support web site.

The enhanced custom operators feature is built on the expression-tree interface described in "Enhanced Object-Qualification Interface" on page 19.

For programming-interface details, see page 34 (Objectivity/C++).

## Default Clustering Priorities

In Release 10.0, a standard clustering strategy is initialized with the default clustering priorities, which now consist of the entire set of clustering rules. Consequently, a standard clustering strategy will attempt to add new containers or databases as necessary whenever existing containers or databases run out of space for new persistent objects.

As in prior releases, Objectivity/DB attempts to place a new basic object on the same logical page, if the page has enough space; otherwise, the new object is put on an existing logical page in the same container, or, if necessary, on a new logical page in that container. In Release 10.0, Objectivity/DB now:

- Creates a new container, if a new logical page cannot be added to the attempted container.
- Creates a new database, if a new container cannot be added to the attempted database.

**NOTE** Because of the new default clustering priorities, a standard clustering strategy will no longer ensure that the nodes of a short-reference ordered collection are all created in a single container.

## Clustering of Hash Buckets in Unordered Collections

When new hash buckets are added to an unordered persistent collection, Objectivity/DB now distributes them among multiple containers only if the clustering priorities of the current session's clustering strategy includes the clustering rule that allows a new container to be created. In previous releases, the new-container rule was enforced regardless of which priorities were set for the clustering strategy. If the clustering strategy uses the new default clustering priorities (see page 22), it allows hash buckets to be distributed among multiple containers in multiple databases.

As in previous releases, every hash bucket is clustered on its own logical page, regardless of whether other page-level clustering priorities have been set.

## Unique Comparison Arrays in Sorted Collections

You can now maximize the performance benefit of comparison arrays in a sorted collection by creating the collection so that it uses comparison arrays alone to determine the equality of two persistent objects. When such a collection encounters two identical comparison arrays, it automatically considers the corresponding persistent objects to be equal *without* opening those objects to obtain further comparison data for the comparator to evaluate. You should therefore use this feature only if you can guarantee that every element of a particular sorted collection will have a unique comparison array.

For programming-interface details, see page 31 (Objectivity/C++) and page 48 (Objectivity for Java).

## Longer Boot File Names Allowed

The simple name of a federated-database boot file can now contain from 1 to 127 characters, not including a filename extension. Prior to this release, the length could range from 2 to 31 characters.

## Optimized Process for Growing Database Files

Release 10.0 optimizes the default process by which Objectivity/DB adds new storage pages to a database file when extending a container in that database. On most supported Objectivity/DB platforms, the default is to grow database files as nonsparse files; on selected platforms, the default is to grow them as sparse files. Prior to this release, all database files were grown as sparse files.

When a database file grows as a nonsparse file, the space for each newly added storage page is reserved immediately on disk. This optimizes the file system's ability to allocate contiguous disk space for newly added pages and reduces the likelihood that a database file will be fragmented. Reducing file fragmentation, in turn, can significantly improve the runtime speed of operations that access the database's persistent objects.

When a database file grows as a sparse file, each newly added storage page is represented by file-system metadata until it is used for storing new persistent objects. A sparse database file with many unused pages therefore occupies less disk space than it would if most or all of its pages contained persistent objects. However, a sparse database file is likely to become fragmented as applications write new persistent objects on unused storage pages over time, especially when multiple transactions add objects to different containers of the same database.

An application can now specify whether to grow database files as sparse or non-sparse files. For programming-interface details, see page 33 (Objectivity/C++) and page 48 (Objectivity for Java).

# Objectivity/DB (Tools)

This section describes new, changed, deprecated, and obsolete features of Objectivity/DB tools. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed tools, see the updated online book *Objectivity/DB Administration*.

***NOTE*** Any changes to tools delivered with Objectivity/DB High Availability are described in "Objectivity/DB In-Process Lock Server" on page 28.

## New Tools

Tools with new or changed features are listed in alphabetical order by tool name.

### Performance Analyzer

The *Objectivity/DB performance analyzer* is a tool that lets you evaluate your application and receive suggestions about parameter settings and code constructs that might compromise your application's performance. You can also use the performance analyzer to temporarily override parameters in your application to test the suggested improvements without rebuilding your application.

The performance analyzer can work in conjunction with an Objectivity/DB default performance tuner, providing access to a greater number of performance-related parameters that can be temporarily overridden.

The performance analyzer is provided as a plugin, so you do not need to modify your source code in order to use it. To enable the performance analyzer and modify its settings, you edit an XML plugin specification file included in the installation hierarchy.

You can deploy the performance analyzer, which can be used to enable logging at a deployment site.

# Changed Tools

## Enhancements to ooupgrade

The `ooupgrade` tool has been reimplemented for faster performance and is approximately 10 times faster in most cases; sufficiently large databases have been observed to upgrade 30 times faster. You typically use this tool to upgrade a pre-Release 9.0 federated database to take advantage of the current internal database format; see "Upgrading to the Release 9.0 Internal Database Format" on page 68.

The following are additional changes to this tool's behavior:

- Now requires less free disk space, because temporary XML files are no longer used as intermediate representations of databases, with the exception of the system database. The tool now requires free space equal to slightly more than the size of the largest database.
- Now leaves the storage pages of an upgraded user database in their original owning architecture unless you specify a different architecture using `-format`.
- Now preserves redirection if any basic object was redirected as a result of schema evolution.
- Now automatically performs redirection as needed to account for larger sized basic objects if the storage pages of the database are owned by a 32-bit architecture and the upgrade operation will cause them to be owned by a 64-bit architecture; to avoid this redirection, you can specify a larger page size for the upgrade.
- Now preserves external containers instead of embedding them in their parent databases.

## Enhancements to ooconvertformat

The `ooconvertformat` tool can now convert between 32-bit and 64-bit architectures. When converting from a 32-bit to a 64-bit architecture, any basic object that no longer fits in its original storage space due to an increase in the class's size is automatically redirected to a page with more space. A link to the object's new location is maintained by a redirection stub left in the original location.

The `ooconvertformat -check` option now provides detailed information about how many pages need to be converted and shows their current architecture. When this option is combined with `-verbose`, the output includes information about each page's type as well. `The -check` option also detects when your federation has multiple architectures. It is recommended that you run `ooconvertformat` with the `-check` option before performing any conversion.

### Enhancements to oocheck and oofix

By default, `oocheck` now reports all errors encountered instead of only the first 100. Likewise, by default, `oofix` now prompts you to examine each error instead of only the first 100.

## Obsolete Tools (Removed in This Release)

### Objectivity/DB ooAssistant

The ooAssistant tool was deprecated in Release 9.0 and has been removed from the product in Release 10.0. If Objectivity/Assist is available on your platform, you should use it instead of ooAssistant.

---

***NOTE*** Assist is available on Windows platforms and on selected UNIX architectures. A UNIX architecture *`arch`* has Assist if the Objectivity/DB installation directory *`installDir/arch`* contains a subdirectory called `eclipse`. If you cannot run Assist, see `ootoolmgr` (UNIX platforms) or `oobrowse` (Windows platforms).

---

# Objectivity/DB High Availability

This section describes new, changed, deprecated, and obsolete features of Objectivity/DB High Availability (Objectivity/HA). See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed features, see the updated online book *Objectivity/DB High Availability*.

## Changed Features

### New Limits on Boot-File Names of Autonomous Partitions

The simple name of an autonomous-partition boot file can now contain from 1 to 127 characters, not including a filename extension. Prior to this release, the length could range from 2 to 31 characters.

# Objectivity/DB (Monitoring Lock-Server Performance)

This section describes new, changed, deprecated, and obsolete features of the C++ programming interface for monitoring Objectivity/DB lock servers. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed tools, see the updated online book *Monitoring Lock Server Performance, Release 10.0*.

## New Features

### Expressing the Absolute Time of a Lock-Server Event

The `oolsTimeval` global type is a struct that provides the absolute time of a lock-server event, expressed as seconds and milliseconds. This new type is used in message classes; see "Changed Timestamp Type in Message Classes" on page 27

## Changed Features

### Changed Timestamp Type in Message Classes

The type of the timestamp data member (`ts`) in message classes has been changed to the new global type `oolsTimeval`; see "Expressing the Absolute Time of a Lock-Server Event" on page 27. This change affects the following message classes:

- `oolsConnectMessage`
- `oolsDeadlockFirstMessage`
- `oolsDisconnectMessage`
- `oolsLockMessage`
- `oolsUnlockMessage`
- `oolsWaitFailedMessage`
- `oolsWaitSucceededMessage`

In each changed message class, the original data-member type was the now-obsolete `oolsTimeStamp`; see "Expressing the Relative Time of a Lock-Server Event" on page 28.

## Obsolete Features (Removed in This Release)

### Expressing the Relative Time of a Lock-Server Event

The `oolsTimestamp` global type has been removed from the programming interface. This type represented the relative time of a lock-server event, expressed in seconds since performance monitoring began. This type has been replaced by the new `oolsTimeval` struct; see "Changed Timestamp Type in Message Classes" on page 27.

# Objectivity/DB In-Process Lock Server

Objectivity/DB In-Process Lock Server (Objectivity/IPLS) has no new features in this release. See the Technical Support web site for software or documentation problems that have been fixed in this release.

# Objectivity/DB Open File System

Objectivity/DB Open File System (Objectivity/OFS) has no new features in this release.

A new shared library is available for linking to your applications. Contact your account representative to obtain Objectivity/OFS software and documentation.

# Objectivity/DB Parallel Query Engine

Objectivity/DB Parallel Query Engine (Objectivity/PQE) has no new or changed features in Release 10.0. See the Technical Support Web site for software or documentation problems that have been fixed in this release.

# Objectivity/DB Secure Framework

Objectivity/DB Secure Framework (Objectivity/Secure) has no new features in this release.

You may obtain Objectivity/Secure software and documentation by contacting your account representative.

# 3

# C++ Products

This chapter describes new and changed features of Objectivity/C++ and other C++-specific Objectivity products in Release 10.0.

## Objectivity/C++

This section describes new, changed, superseded, and obsolete features of Objectivity/C++. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of selected new and changed features, see the updated online books, *Objectivity/C++ Programmer's Guide* and *Objectivity/C++ Programmer's Reference*.

For descriptions of selected superseded and deprecated features, see the *Objectivity/C++ Backward Compatibility* book on the Technical Support web site.

# New Features

## Snapshots of Objectivity/C++ Applications

The new `snapShot` method of the `ooSession` class lets you capture a *snapshot* of an application in the current session, as described in "Capturing Snapshots of Objectivity/DB Applications" on page 17.

Three new enumerated types support the `snapShot` method:

| | |
|---|---|
| `ooInfoFormat` | Type for specifying the output format for the information captured in a session snapshot |
| `ooSnapShotInfo` | Type for specifying the kind of information to be captured in a session snapshot |
| `ooSortOrder` | Type for specifying the order in which to write the information captured in a session snapshot |

## Support for Container-Fill Policy in ooClusterStrategy Class

The `ooClusterStrategy` class now has the following new methods for getting and setting the container-fill policy described in "Container-Fill Policy in Clustering Strategies" on page 18:

- `getContainerPageLimit`
- `setContainerPageLimit`

You can set the container-fill policy to be one of the following new constants defined by the `ooClusterStrategy` class, or you can specify an integer:

| | |
|---|---|
| `reserveSpace` | Leave a small number of logical pages unused in a container, to accommodate any future expansion of VArrays, to-many associations, or persistent collections stored in the container. |
| `fastAccess` | Calculate the logical-page limit that will keep a container's page map small, and enforce that limit. |
| `maximum` | Allow clustering on the maximum possible number of logical pages in a container. |
| *`N`* | Limit clustering to *`N`* logical pages in a container, where *`N`* is a positive integer. |

## New Technique for Defining Custom Clustering Strategies

If you override the `newObj` method in a custom clustering-strategy class derived from `ooClusterStrategy`, you should now call the inherited `tryCreate` method in your `newObj` definition instead of calling the base class's `newObj` method. Although `tryCreate` is now recommended, existing implementations that call the base class's `newObj` method will still work.

A typical override selects a clustering directive based on some criteria and then calls `tryCreate` to create the new basic object in the first available location identified by the selected clustering directive in combination with the specified clustering priorities and container-fill policy.

The `tryCreate` method is now documented as a public method of the `ooClusteringStrategy` class. It has been reimplemented so that it tests for *all* clustering priorities. In previous releases, the method was included for internal use, and tested only for the priorities that apply within a single container (`oocSamePage`, `oocOtherPage`, `oocNewPage`), even if you specified `oocNewContainer` or `oocNewDatabase`. Furthermore, new overloads have been added to allow you to specify clustering-strategy properties other than those set for the instance.

## Calculating Fast-Access Container Size

The following new method calculates the maximum number of logical pages that a container should have for fast update access, as described in "Calculating the Optimal Container Size for Fast Access" on page 18:

- `maxPagesForSmallPageMap` method of the *`ooRefHandle`*`(ooContObj)` classes

## Support for Unique Comparison Arrays

The `ooTreeSetX` and `ooTreeMapX` constructors now accept an additional parameter that specifies whether a new sorted collection is to assume that all comparison arrays are unique; see "Unique Comparison Arrays in Sorted Collections" on page 23.

## Date and Time Classes

The following new non-persistence-capable classes can now be used to represent general-purpose date and time information:

- `ooDate`
- `ooTime`
- `ooInterval`
- `ooDateTime`

These date and time classes are defined in the `ooTime.h` header file. They provide an efficient alternative to the Objectivity/C++ language-specific compatibility classes, and are recommended over the superseded date and time classes; see ".NET/C# and ODMG-Compatible Date and Time Classes" on page 37.

## Unicode Character Iterators

You can now iterate over complete characters in a Unicode string using instances of the following *character iterator* classes:

- `ooUtf8CharacterIterator`
- `ooUtf16CharacterIterator`
- `ooUtf32CharacterIterator`

The character iterator classes are an alternative to the `ooUtf8Iterator`, `ooUtf16Iterator`, and `ooUtf32Iterator` classes, which let you iterate over individual Unicode code units.

## Converting UTF-8 Strings

The `ooUtf16String` class has a new `setFromUtf8` method to convert a UTF-8 string to a UTF-16 string. Similarly, the `ooUtf32String` class has a new `setFromUtf8` method to convert a UTF-8 string to a UTF-32 string.

## Exception Classes

The following new exception classes signal general error conditions:

- `ooArgumentOutofRangeException`
- `ooOverflowException`

The following new exception classes signal error conditions specific to the new date and time classes:

- `ooNullOperandException`
- `ooTimeKindMismatchException`
- `ooUnableToConvertDateTimeException`

For more information about the new date time classes, see "Date and Time Classes" on page 31.

## Specifying Line Length for Error Messages

The `ooErrorLineLength` global function sets the maximum line length (before wrapping) for warnings and fatal errors. This global function also applies to error message strings that are passed to the current message handler by the new

reportErrors method on ooException; see "Printing Error Messages" on page 33.

For backward compatibility, the ooErrorLineLength global function specifies the line length for error messages when exceptions are not enabled.

## Printing Error Messages

The reportErrors method of the ooException class prints the error message for an exception (by default, to stderr), and lets you provide an optional string to describe the context in which the error occurred. For kernel errors, reportErrors prints the primary error as well as any cascading errors that might be involved. Additional information, such as the error number and severity level, are also printed for kernel errors.

## New Method for Scalable-Collection Iterators

The ooCollectionIterator class has the following new method, which supports fast iteration from the end of the collection:

- goToLast()

## Growing Non-Sparse Database Files

The following new global functions let you specify whether to grow database files as sparse or non-sparse files, as described in "Optimized Process for Growing Database Files" on page 23:

- ooPermitSparseDbFiles
- ooGetPermitSparseDbFiles

Growing database files as nonsparse files can increase runtime speed of operations that access those files.

The default process for growing database files is optimized for each supported Objectivity/DB platform. An application can find out which optimization it is currently using by calling ooGetPermitSparseDbFiles.

## Requesting Scan-Order Optimization

The ooItr(ooObj) and ooItr(*appClass*) classes now have the following new methods for optimizing the order in which an object iterator scans a container or database for basic objects:

- optimizeScanOrder
- getOptimizeScanOrder

For more information, see "Optimizing Scan Order" on page 19.

### Requesting I/O Read-Ahead Thread

The following new global functions can be invoked before iterating over ordered collections or performing scan operations to enable or disable I/O thread creation; I/O thread creation is disabled by default.

- `ooPermitReadAhead`
- `ooGetPermitReadAhead`

For more information, see "I/O Read-Ahead Thread" on page 20.

## Changed Features

### Using Enhanced Predicate Query Language (PQL)

An Objectivity/C++ application now uses the enhanced PQL described in "Enhanced Predicate Query Language" on page 21.

An application can use the enhanced PQL in a scan or a parallel scan operation without including any extra header files and without any extra linking. However, the new libraries in Table 2-1 must be available for dynamic loading at runtime.

### Predicate Query Language Supports Wide Character Strings

An Objectivity/C++ application can now use predicate strings that reference fields of type `ooUtf16String` and `ooUtf32String` as string operands.

Accordingly, the following new method overloads accept wide character strings (`const wchar_t *`) for their *`predicate`* parameters:

- The `scan` method of the `ooItr(`*`appClass`*`)`, `ooItr(ooContObj)`, and `ooItr(ooObj)` classes
- The `parallelScan` method of the `ooItr(`*`appClass`*`)`, `ooItr(ooContObj)`, and `ooItr(ooObj)` classes

### Creating Custom Predicate-Query Operators

The new class `ooOperator` is the base class for all custom application-defined operators described in "Enhanced Support for Custom Operators" on page 21. To define a custom operator, an application derives a class from `ooOperator` and overrides a set of virtual functions. Next, the application registers an instance of the derived `ooOperator` class with the operator registry in one of the following ways:

- Directly in the application.
- As a plugin using the new Release 10.0 plugin support; see "Support for Plugins" on page 20.

The application or plugin library must explicitly include the new `ooObjectQualifier.h` header file, and must link to the new object qualification library listed in Table 2-1. The other shared libraries listed for your platform must be available at runtime.

---

***NOTE*** For detailed information about creating custom operators, see the Technical Support web site.

---

The previous mechanism for creating application-defined operators is obsolete; for a complete list of the obsolete items, see "Custom Operator Functions" on page 38.

## Qualifying Objects One at a Time

The new `ooObjectQualifier` class represents an *object qualifier*, which evaluates whether a persistent object matches a given predicate string.

The following new method overloads accept a parameter of type `ooObjectQualifier&` as an alternative to a predicate string:

- The `scan` method of the `ooItr(`*`appClass`*`)`, `ooItr(ooContObj)`, and `ooItr(ooObj)` classes
- The `parallelScan` method of the `ooItr(`*`appClass`*`)`, `ooItr(ooContObj)`, and `ooItr(ooObj)` classes

An object qualifier's predicate string can be a C++ wide character string (`const wchar_t *`).

An application that uses this class must explicitly include the new `ooObjectQualifier.h` header file, and must link to the new object qualification library listed in Table 2-1; the additional shared libraries listed for your platform must be available at runtime.

Object qualifiers are built on the expression-tree interface described in "Enhanced Object-Qualification Interface" on page 19.

This class supersedes the `ooQuery` class; see "ooQuery Class" on page 38.

## Name Maps Support Wide Character Strings For Key Names

Name maps (instances of the `ooMap` class) now support C++ wide character strings (`const wchar_t *`) as the names of keys.

## New Types Supported by Index Key Fields

An instance of the `ooKeyField` index key class can now created for attributes of the following string types, in addition to the previously supported string types (`char[]`, `ooVString`, `ooStringT<`*N*`>`, `ooUtf8String`):

| | | |
|---|---|---|
| `ooUtf16String` | `ooChar16[]` | `ooVArrayT<ooChar16>` |
| `ooUtf32String` | `ooChar32[]` | `ooVArrayT<ooChar32>` |

## Changed Return Types for ooMap-Related Methods

The following methods now return an object reference to a persistent object (an instance of `ooRef(ooObj)`) instead of an object identifier (an instance of `ooId`):

- `lookup` method of the `ooMap` class
- `oid` method of the `ooMapElem` class

You do not need to rewrite existing C++ application code to accommodate this change, because `ooRef(ooObj)` is a subclass of `ooId`.

## Enhanced releaseFiles Method

The `releaseFiles` method of the session class now deletes journal files and closes the connection to the lock server if called from outside a transaction.

## Changing Read-Only Databases to Read/Write

The `setReadOnly` method of the `ooRefHandle(ooDBObj)` classes has a new *`forceLockOverride`* parameter that lets you specify whether or not to allow the access status for a read-only database to be changed back to read/write while other read-only databases in the federated database are open. This override should be used only if the application verifies that there are no current readers of the database in question.

## Longer Boot File Names Allowed

In previous releases, the character limit for boot file names was between 1 and 32 characters long, inclusive. The new range is between 1 and 127 characters, inclusive. Methods that accept a boot file as a parameter, such as the `ooGetActiveTrans` method, can accommodate the longer boot file names.

### Change to Returned Storage-Page Count

The `nPage` method of the `ooRefHandle(ooContObj)` classes now returns a count that may, but need not, include any storage pages that were added to the container during the current transaction. If you want to ensure that added, but uncommitted, pages are included in the returned number, you should checkpoint the transaction before calling this method.

### Timing of Automatic Recovery

A recovery-enabled application now initiates automatic recovery on a federated database at the beginning of the first transaction to access that federated database. In previous releases, automatic recovery was performed when the first session was created.

### Connecting to Multiple Federated Databases

When working with multiple connected federated databases, an application can now connect with federated databases that have different schema definitions. (In previous releases, an application could connect with multiple federated databases only if they had identical schema definitions.)

When working with multiple connected federated databases with differentiae schemas, you use the normal mechanisms for accessing the schema of the federated database whose DDL files are compiled into your application. For any other connected federated database, you must use Objectivity/DB Active Schema to access its persistent objects.

## Superseded Features

### .NET/C# and ODMG-Compatible Date and Time Classes

The following date and time classes have been superseded and are now supported only for backward compatibility:

- `ooDotNetDateTime`
- `d_Date`
- `d_Time`
- `d_Interval`
- `d_Timestamp`

Although support for these classes will continue, it is recommended that you use the new general-purpose date and time classes in any new persistence-capable classes you create. See “Date and Time Classes” on page 31.

### ooQuery Class

The `ooQuery` class is superseded by the `ooObjectQualifier` class, as described in "Qualifying Objects One at a Time" on page 35. The `ooQuery` class is now supported only for backward compatibility.

In order to use the `ooQuery` class, you now need to include `oo.h` instead of `ooUserOper.h`.

## Obsolete Features (Removed in This Release)

### Custom Operator Functions

You no longer use the following items to write your own custom operators, because code containing these items will no longer compile:

- `ooOperatorSet` class
- `ooUserDefinedOperators` global variable
- `ooQueryOperatorType` function pointer type
- `ooDataType` global type

Instead, you use the new mechanism described in "Creating Custom Predicate-Query Operators" on page 34.

# Objectivity/C++ Data Definition Language

Objectivity/C++ Data Definition Language (Objectivity/DDL) has no new features in this release.

# Objectivity/DB Active Schema for C++

This section describes new, deprecated, and obsolete features of Objectivity/DB Active Schema (Active Schema) for C++. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed features, see the updated online book *Objectivity/DB Active Schema for C++*.

# New Features

## Unicode String Support

Objectivity/DB Active Schema now supports basic Unicode UTF-16 and UTF-32 strings.

The following new global types indicate the kind of string class of an attribute:

- `ooAsStringUTF16`
- `ooAsStringUTF32`

The following new methods of the `String_Value` class test whether a string value contains a basic UTF-16 or UTF-32 string:

- `is_utf16string`
- `is_utf32string`

The following new methods of the `String_Value` class return a pointer to the basic UTF-16 or UTF-32 string contained by a string value:

- `operator ooUtf16String *();`
- `operator ooUtf32String *();`

## Enhanced Proposed_Property Class

The following new methods of the `Proposed_Property` class get information about proposed properties:

- `default_value`
- `base_type`
- `element_base_type`
- `other_class`
- `type_of`
- `other_class_name`
- `other_class_namespace_name`

The following new methods of the `Proposed_Property` class test proposed properties:

- `has_default_value`
- `is_short`
- `element_is_short`

## Specifying String Attributes on Class Objects

You can use the new `set_string` method of `Class_Object` to set the value of a string attribute of an associated persistent object to the character string you provide.

### Casting a Class Object to a Void Pointer

The following new operator lets you cast a class object to a void pointer:

```
operator void *();
```

You can use this operator in code that casts a class object to a type supported in the Objectivity/C++ programming interface in order to use accessors available to instances of that type.

### Getting Information From Descriptors

The following new methods of `d_Meta_Object` provide information about the described named entities in the federated database schema.

- `is_proposed` tests whether the described entity is a proposed entity.
- `module_name` gets the name of the module that includes the described entity.
- `namespace_name` gets the name of the namespace of the described entity.

### Testing Whether an Attribute is a Relationship

The new `is_relationship` method of the `d_Attribute` class tests whether the described attribute is a relationship.

### Getting the Identifier for an Inverse Relationship

You can use the new `inverse_id` method of the `d_Relationship` class to get the integer identifier for the attribute specifying the inverse relationship of the described bidirectional relationship.

## Changed Features

### Return Types of Certain Proposed_Class Methods

In the `Proposed_Class` class, each method that adds a property to a proposed class now returns a descriptor for the proposed property (a value of type `Proposed_Property &`) instead of returning a value of type `ooStatus`.

The changed `Proposed_Class` methods are:

- `add_base_class`
- `add_basic_attribute`
- `add_bidirectional_relationship`
- `add_embedded_class_attribute`
- `add_property`

- add_ref_attribute
- add_unidirectional_relationship
- add_varray_attribute

## Deprecated Features (Removed in a Future Release)

### Proposing Classes Independently of Modules

You should no longer create a proposed class independently of any module, and then add it to a module later. That is, the following items are deprecated in Release 10.0:

- The constructor of the Proposed_Class class
- The propose_new_class(Proposed_Class *) method overload of the d_Module class

Instead, you should create a proposed class only by adding it immediately to a module—that is, by calling the module descriptor's remaining propose_new_class method overload and passing the class name parameter and, optionally, the type number and namespace parameters.

For example, statements such as the following are now deprecated:

```
Proposed_Class *machineP = new Proposed_Class("Machine", 0);
newMod.propose_new_class(machineP);
```

You should replace them with a statement such as this:

```
Proposed_Class &machine = newMod.propose_new_class("Machine");
```

## Obsolete Features (Removed in This Release)

### Sanitize Method

The following method was deprecated in Release 9.0 and has been removed from the product in Release 10.0:

- d_Module::sanitize

# Objectivity/C++ Standard Template Library

The Objectivity/C++ Standard Template Library (Objectivity/C++ STL) has no new features in this release.

You may obtain Objectivity/C++ STL software and documentation by contacting your account representative.

# 4

# Java Products

This chapter describes new and changed features of Objectivity for Java and other Java-specific Objectivity products in Release 10.0.

## Objectivity for Java

This section describes new, changed, deprecated, and obsolete features of Objectivity for Java. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed features, see the updated *Objectivity for Java Guide* and *Objectivity for Java Reference* (HTML).

### New Features

#### Snapshots of Objectivity for Java Applications

The new `com.objy.db.app.Session.snapShot` methods let you capture a *snapshot* of an application in the current session:

- `snapShot(int, int, String, boolean, int)`
- `snapShot(int, int, String, boolean, int, ooId)`

See "Capturing Snapshots of Objectivity/DB Applications" on page 17 for more information.

Three new categories of constants in `com.objy.db.app.oo` support the `snapShot` method:

| Category of Constants | Description |
|---|---|
| Snapshot info | Constants for specifying the output format for the information captured in a session snapshot |
| Snapshot info format | Constants for specifying the kind of information to be captured in a session snapshot |
| Snapshot sort order | Constants for specifying the order in which to write the information captured in a session snapshot |

## Support for Multiple Connections

An application typically works with a single federated database; however, an application can now access multiple federated databases that have the same schema. To do so, the application uses the `com.objy.db.app.Connection` class to make a logical connection to each federation so that there are multiple connection objects. An application maintains a single connection object for each connected federated database.

Each session can interact *only* with the federated database corresponding to the connection object that created the session.

The following methods of the `Connection` class have changed such that you can call them multiple times with different boot files:

- `open(String, int)`
- `open(String, int, int)`

New methods of the `Connection` class let you switch back and forth between connections:

- `join`
- `leave`

The following methods of the `Connection` class are now non-static and are applied per connection:

- `setPredicateScanAutoFlush(boolean)`
- `isPredicateScanAutoFlush()`
- `setUserClassLoader(ClassLoader)`
- `setErr(PrintStream)`
- `setOut(PrintStream)`

**NOTE** Because `setErr` is no longer static, it may be called only after a connection is open. Any errors in loading the `oojava` shared library are now sent to standard output.

## New ClusterStrategy Class Instead of Interface

A new class, `com.objy.db.app.ClusterStrategy`, now replaces the `ClusterStrategy` interface. The new class now provides default implementations for all of its methods. Consequently:

- A *standard clustering strategy* is now an object of the `ClusterStrategy` class, instead of the `DefaultClusteringStrategy` class.
- You can define a custom clustering-strategy class by extending the `ClusterStrategy` class directly, instead of implementing the interface or extending the `com.objy.db.app.DefaultClusterStrategy` class.

The `ClusterStrategy` class supersedes the `DefaultClusteringStrategy` class; see "DefaultClusterStrategy Class" on page 53.

## Support for Container-Fill Policy in Clustering Strategies

The `com.objy.db.app.ClusterStrategy` class now has the following new methods for getting and setting the container-fill policy described in "Container-Fill Policy in Clustering Strategies" on page 18:

- `containerPageLimit()`
- `setContainerPageLimit(int)`

You can set the container-fill policy to be one of the following new constants defined by the `ClusterStrategy` class, or you can specify an integer:

| | |
|---|---|
| `reserveSpace` | Leave a small number of logical pages unused in a container, to accommodate any future expansion of VArrays, to-many associations, or persistent collections stored in the container. |
| `fastAccess` | Calculate the logical-page limit that will keep a container's page map small, and enforce that limit. |
| `maximum` | Allow clustering on the maximum possible number of logical pages in a container. |
| *N* | Limit clustering to *N* logical pages in a container, where *N* is a positive integer. |

## Calculating Fast-Access Container Size

The following new method calculates the maximum number of logical pages that a container should have for fast update access, as described in "Calculating the Optimal Container Size for Fast Access" on page 18:

- `com.objy.db.app.ooContObj.getMaxPagesForSmallPageMap()`

## Customizing the Clustering of a Collection's Internal Objects

When Objectivity/DB adds new internal objects to a scalable persistent collection, it now consults the session's clustering strategy to place those objects. Such objects include the B-tree nodes of ordered collections and the hash buckets of unordered collections. Consequently, you can now define a custom clustering-strategy class to customize how the internal objects of scalable collections are clustered.

The com.objy.db.app.ooObj.ClusterStrategy class has the following new method that you can override in a custom clustering-strategy class to explicitly handle the internal objects of a scalable collection:

- requestClusterNative(long, ooId)

Your override must call the following new ClusterStrategy method:

- clusterNative(long, ooId)

Your override can optionally call the following new methods to test the type of internal objects being clustered:

- isTreeNode(long)
- isHashBucket(long)
- getSchemaName(long)

## Naming the Storage Objects Created by Clustering Strategies

You can now call the following new methods of the com.objy.db.app.ClusterStrategy class to access the system-name prefix for databases that are created by a clustering strategy's newDB method to accommodate new persistent objects:

- defaultDatabasePrefix()
- setDefaultDatabasePrefix(String)

Previously, you could change the default prefix "autoDB" only by overriding the newDB method in a custom clustering-strategy class. Now the default implementation of newDB uses the string you set.

Similarly, you can now call the following new ClusterStrategy methods to control whether a clustering strategy's newContainer method creates unnamed or named containers:

- defaultContainerName()
- setDefaultContainerName(String)

Previously, you could change the default behavior (creating unnamed containers) only by overriding the newContainer method in a custom clustering-strategy class. Now the default implementation of newContainer uses the string you set.

## Support for Unique Comparison Arrays

New overloads of the following constructors now accept a parameter that specifies whether a new sorted collection is to assume that all comparison arrays are unique; see "Unique Comparison Arrays in Sorted Collections" on page 23.

- `com.objy.db.util.ooTreeSetX(ooCompare, int, boolean, boolean)`
- `com.objy.db.util.ooTreeMapX(ooCompare, int, boolean, boolean)`

## Policy for Checking Memory Availability

You can now activate memory checking in a connection to cause an exception to be thrown when the application's total memory usage exceeds 95% of the Java virtual machine's memory. The memory checking occurs whenever a session commits or aborts a transaction, or whenever an object is either made persistent or retrieved in a session.

You use the following new methods on `com.objy.db.app.Connection` to manage the memory checking:

- `isMemoryCheckPolicy()`
- `setMemoryCheckPolicy(boolean)`

When memory checking is activated, the following new exception in `com.objy.db` is thrown as necessary:

- `ImpendingOutOfMemoryException`

## Growing Non-Sparse Database Files

The following new methods of `com.objy.db.app.Connection` let you specify whether to grow database files as sparse or non-sparse files, as described in "Optimized Process for Growing Database Files" on page 23:

- `setPermitSparseDbFiles(boolean)`
- `isPermitSparseDbFiles()`

Growing database files as nonsparse files can increase runtime speed of operations that access those files.

The default process for growing database files is optimized for each supported Objectivity/DB platform. An application can find out which optimization it is currently using by calling `isPermitSparseDbFiles()`.

## Requesting Scan-Order Optimization

The following new method optimizes the scan order of the next object iterator to be created in the same transaction, provided the object iterator is initialized to scan a container or database for basic objects:

- `com.objy.db.app.Session.setOptimizeScanOrder(boolean)`

The following new method tests whether a particular object iterator uses an optimized scan order:

- `com.objy.db.app.Iterator.isOptimizeScanOrder()`

For more information, see "Optimizing Scan Order" on page 19.

## Using Identifiers to Get Containers and Databases

You can now use the following new methods on `com.objy.db.app.ooDBObj` to get a container with a particular integer identifier:

- `hasContainer(int)`
- `lookupContainer(int)`
- `lookupContainer(int, int)`

You can now use the following new methods on `com.objy.db.app.ooFDObj` to get a database with a particular integer identifier:

- `hasDB(int)`
- `lookupDB(int)`

## New Methods for Scalable-Collection Iterators

The `com.objy.db.util.ooCollectionIterator` interface has the following new methods for working with scalable-collection iterators:

- `goToLast()`
- `collection()`
- `currentIndex()`

## Enforcing Immediate Persistence of Related Objects

The `com.objy.db.app.Session` class has the following new method for enforcing the standard relationship-persistence policy for the lifetime of a session:

- `setAllowTransientRelationships(boolean)`

Under the standard relationship-persistence policy, forming a relationship between transient objects of persistence-capable classes causes those objects to be

made persistent immediately, and adding an element to a transient name map causes an exception to be thrown.

Enforcing this policy can improve the runtime speed of certain operations, such as operations that create persistent objects. However, once this policy is enforced, you cannot call the session's `setFormTransientRelationships` method to switch to the transient-relationships policy if later operations need to delay the persistence of transient objects linked by relationships or add elements to transient name maps.

### New Methods for Supporting Objectivity/DB Active Schema

- `com.objy.db.app.ooObj.create_ooObj()`
- `com.objy.db.app.Session.getReturn_Class_Object()`
- `com.objy.db.app.Session.setReturn_Class_Object()`

### Unicode Strings as File and Directory Names

File or directory names provided as Unicode strings to methods such as `com.objy.db.app.Connection.open` correctly correspond to Unicode file or directory names on Windows file systems.

### Requesting I/O Read-Ahead Thread

The following new functions on `com.objy.db.app.Connection` can be invoked before iterating over ordered collections or performing scan operations to enable or disable I/O thread creation; I/O thread creation is disabled by default.

- `setPermitReadAhead(boolean)`
- `isPermitReadAhead`

For more information, see "I/O Read-Ahead Thread" on page 20.

## Changed Features

### Using Enhanced Predicate Query Language (PQL)

An Objectivity for Java application now uses the enhanced PQL described in "Enhanced Predicate Query Language" on page 21.

***NOTE*** Objectivity for Java does not support predicate strings that test embedded objects of application-defined classes.

## Qualifying Objects One at a Time

The new `ooObjectQualifier` class represents an *object qualifier*, which evaluates whether a persistent object matches a given predicate string.

Object qualifiers are built on the expression-tree interface described in "Enhanced Object-Qualification Interface" on page 19.

This class supersedes the `ooQuery` class; see "ooQuery Class" on page 53.

## Hot Mode Enabled By Default

Hot mode is now enabled by default in a newly created session or session pool.

## Certain Methods Now Sensitive to Transaction State

You should now call the methods of a persistent object only when the session that owns the object is in a transaction:

- `com.objy.db.app.ooObj.isModified()`
- `com.objy.db.app.ooObj.isFetchRequired()`

If you call `isModified` outside a transaction, it will always return false, regardless of the state of the object.

If you call `isFetchRequired` outside a transaction, it will always return true, regardless of the state of the object.

## Transfer of Persistent Data During a Fetch Operation

In previous releases, if you committed a transaction in which an object's data was fetched, the object would be marked as needing to have its data fetched before you could use it in a subsequent transaction in the same session. If you then called the object's `fetch` method in such a transaction, the fetch operation would transfer the persistent data, even if that data had not been changed.

As a performance optimization, committing a transaction in which you fetched an object's data no longer automatically marks the object as needing to have its data fetched. If you want to use the object in a subsequent transaction in the same session, you should call the object's `fetch` method, as in earlier releases; however, in this release, the fetch operation checks whether any updates have been made in the object's container since the last time the object's data was fetched. If so, the object's data is actually transferred. Otherwise, if the container's version number has not changed, the fetch operation performs no data transfer, although it will still lock the object as appropriate.

The `com.objy.db.app.ooObj.isFetchRequired` method now performs the same check as `fetch`, and returns false if it is called on an object that resides in a container that has not been updated since the last time the object's data was fetched.

As before, you can force an object's data to be transferred during a fetch operation by first calling the object's `com.objy.db.app.ooObj.setMarkFetchRequired` method.

## Change to Returned Storage-Page Count

The `com.objy.db.app.ooContObj.getPageCount` method now returns a count that may, but need not, include any storage pages that were added to the container during the current transaction. If you want to ensure that added, but uncommitted, pages are included in the returned number, you should checkpoint the transaction before calling this method.

## Longer Boot File Names Allowed

In previous releases, the character limit for boot file names was between 1 and 32 characters long, inclusive. The new range is between 1 and 127 characters, inclusive. The `com.objy.db.app.Connection.open` methods accommodate the longer boot file name.

## Change to Auto-Persistence in Unidirectional Relationships

When you form a unidirectional relationship link from a transient source object to a persistent destination object, the transient source object is now *always* made persistent immediately, regardless of the current relationship-persistence policy set by the session's `setFormTransientRelationships` method.

In previous releases, immediate persistence of a transient source object occurred only under the standard relationship-persistence policy. In contrast, under the transient-relationships policy, the transient source object of such a relationship would remain transient until some later event occurred.

## Enhanced releaseFiles Method

The `releaseFiles()` method of the session class now deletes journal files and closes the connection to the lock server if called from outside a transaction.

# Deprecated Features (Removed in the Next Release)

## DefaultClusterStrategy Class

The `DefaultClusterStrategy` class is superseded by the `ClusterStrategy` class, as described in "New ClusterStrategy Class Instead of Interface" on page 45. The `DefaultClusterStrategy` class now inherits the implementations of all its methods from `ClusterStrategy`, and is now supported only for backward compatibility.

## ooQuery Class

The `ooQuery` class is superseded by the `ooObjectQualifier` class, as described in "Qualifying Objects One at a Time" on page 51. The `ooQuery` class is now supported only for backward compatibility.

# Obsolete Features (Removed in This Release)

## Discontinued Support for JDK 1.5

Objectivity for Java no longer supports JDK 1.5, so all Objectivity for Java Release 10.0 applications must be built with JDK 1.6.

Upgrading to Sun JDK 1.6 may have consequences for the applications you are upgrading. See "Upgrading Objectivity for Java Applications" on page 77.

## Marking a Subset of Reference Attributes as Modified

Objectivity for Java no longer supports the following methods, which both mark an object as modified and perform a partial fetch of its reference attributes:

- `com.objy.db.app.ooObj.markModified(String)`
- `com.objy.db.app.ooObj.markModified(String, boolean)`
- `com.objy.db.iapp.IooObj.markModified(String)`
- `com.objy.db.iapp.IooObj.markModified(String, boolean)`
- `com.objy.db.iapp.PooObj.markModified(String)`
- `com.objy.db.iapp.PooObj.markModified(String, boolean)`

You can still use `com.objy.db.app.ooObj.markModified()` to mark an object modified and fetch all of its attributes.

# Objectivity/DB Active Schema for Java

This section describes new, changed, deprecated, and obsolete features of Objectivity/DB Active Schema (Active Schema) for Java. See the Technical Support web site for software or documentation problems that have been fixed in this release.

For a complete description of new and changed features, see the updated Active Schema packages of the *Objectivity for Java Reference* (HTML).

## New Features

### Activating Proposals in an Objectivity for Java Application

Active Schema operations are normally performed in separate, special-purpose applications. However, if such operations are performed in an Objectivity for Java application, it is now possible to activate proposed changes without dropping cached schema class descriptions. To do so, you call the following new method, setting the second parameter to true:

- `com.objy.as.app.d_Module.activate_proposals(boolean, boolean)`

Preserving cached class descriptions can be convenient if your Objectivity for Java application needs to continue unrelated work after proposals are activated. However, doing so is safe only if the set of cached class descriptions is disjoint from the set of class descriptions being changed by the activated proposals.

**Warning:** Data corruption will result if any proposal being activated will change a schema class description for which information is currently cached.

## Constructing Class Objects from Object Identifiers

You can now construct class objects from object identifiers using the following new methods on the `Class_Object` class:

- `com.objy.as.app.class_object_from_oid(String)`
- `com.objy.as.app.class_object_from_oid(long)`
- `com.objy.as.app.class_object_from_oid(ooId)`

## Deprecated Features (Removed in a Future Release)

### Proposing Classes Independently of Modules

You should no longer create a proposed class independently of any module, and then add it to a module later. That is, the following items are deprecated in Release 10.0:

- `com.objy.as.app.Proposed_Class(String)`
- `com.objy.as.app.Proposed_Class(String, long)`
- `com.objy.as.app.d_Module.propose_new_class(Proposed_Class)`

Instead, you should create a proposed class only by adding it immediately to a module—that is, by calling the module descriptor's remaining `propose_new_class` method overload and passing the class name parameter and, optionally, the type number and namespace parameters.

For example, statements such as the following are now deprecated:

```
Proposed_Class machine = new Proposed_Class("Machine", 0);
newMod.propose_new_class(machine);
```

You should replace them with a statement such as this:

```
Proposed_Class machine = newMod.propose_new_class("Machine");
```

## Obsolete Features (Removed in This Release)

### Sanitize Method

The following method was deprecated in Release 9.0 and has been removed from the product in Release 10.0:

- `com.objy.as.app.d_Module.sanitize()`

# 5

# .NET Product

This chapter describes Objectivity/.NET for C# for Release 10.0.

## Objectivity/.NET for C#

Objectivity/.NET for C# is a development kit for writing C# applications that access Objectivity/DB database management services. With Objectivity/.NET for C#, you can develop C# applications that store, find, and update persistent data in an Objectivity/DB federated database. Such C# applications are completely interoperable with applications that are written using other Objectivity/DB programming interfaces.

Objectivity/.NET for C# has two main components:

- The Objectivity/.NET class library
- The Objectivity/DB Persistence Designer

### Objectivity/.NET Class Library

The Objectivity/.NET class library provides types that enable your application to connect to the federated database, control transactions, manage the physical placement and logical organization of stored objects, find these objects through individual or group lookup, and monitor and tune database-related application performance.

The Objectivity/.NET class library adheres closely to .NET guidelines, and supports .NET features such as LINQ, configuration files, and distributed transactions.

For information about the class library, see the *Objectivity/.NET for C# Programmer's Reference, Release 10.0*, which is provided as a Microsoft Compiled HTML Help file on the Windows platform.

➤ To open the help file, double-click on it:
*installDir*\doc\ObjyNETcsharp.chm

## Objectivity/DB Persistence Designer

The Objectivity/DB Persistence Designer is a powerful and intuitive tool that lets you easily create schema for your object model, and then generate C# accessor code that is automatically added to your project.

The Persistence Designer is delivered as a plug-in to Microsoft Visual Studio 2008.

You use the Persistence Designer to view and specify the design of a particular federated database's schema. The Persistence Designer displays the schema design as a hierarchy of storable classes, and presents the various design choices as properties of individual classes and their contents.

With the Persistence Designer, you can create or change a schema design by adding or deleting classes, attributes, and relationships, or by changing their properties. By presenting the available set of properties for expressing each design choice to be made, the Persistence Designer serves as a guide for specifying a complete and well-formed schema.

When you finish developing the descriptions of your storable classes, you use the Persistence Designer to update the schema of your federated database, and to generate C# partial classes corresponding to the storable classes in the schema.

For a tutorial that uses the Persistence Designer to develop an Objectivity/DB schema, see the online book *Objectivity/DB Schema Development*.

- To open this book, double-click on the following file:
  *installDir*\doc\schemaDevelopment.pdf

# 6

# Smalltalk Product

This chapter describes new and changed features of Objectivity/Smalltalk for VisualWorks in Release 10.0.

## Objectivity/Smalltalk for VisualWorks

Objectivity/Smalltalk for VisualWorks has no new features in this release. See the Technical Support web site for software or documentation problems that have been fixed in this release.

# 7

# SQL Products

This chapter describes new and changed features of the Objectivity/SQL++ and Objectivity/SQL++ ODBC Driver products in Release 10.0.

## Objectivity/SQL++

Objectivity/SQL++ has no new features in this release. See the Technical Support web site for software or documentation problems that have been fixed in this release.

## Objectivity/SQL++ ODBC Driver

Objectivity/SQL++ ODBC Driver (Objectivity/ODBC) has no new features in this release. See the Technical Support web site for software or documentation problems that have been fixed in this release.

# 8

# Python Product

This chapter describes new and changed features of Objectivity/Python in Release 10.0.

## Objectivity/Python

Objectivity/Python has no new features in this release. See the Technical Support web site for software or documentation problems that have been fixed in this release.

# 9

# Release Compatibility

This chapter provides information about the impact, if any, of using Release 10.0 of Objectivity/DB with existing data, tools, or applications from an earlier release. You may need to perform an upgrade or be aware of limitations.

- See "Upgrading Existing Federated Databases" below if you plan to use tools or applications built with the current release to access data created with an earlier release.
- See "Upgrading Existing Applications and Scripts" on page 75" if you plan to rebuild existing applications with the current release.
- See "Maintaining Earlier Objectivity/DB Applications" on page 65 if you plan to continue using unrebuilt tools or applications from an earlier release.

## Upgrading Existing Federated Databases

Federated databases created with Release 8.*x* or Release 9.*x* of Objectivity/DB can be upgraded to match the new release.

At a minimum, an existing federated database requires an Objectivity Release 10 license before you can access the data in it with any of the following:

- Tools provided with Objectivity/DB Release 10.0
- New applications built with Objectivity/DB Release 10.0
- Existing applications that have been upgraded, recompiled, and relinked with Objectivity/DB Release 10.0 (see "Upgrading Existing Applications and Scripts" on page 75).

Other upgrades are optional, depending on the features and products to be used. See "Selecting the Upgrade Tasks to Perform" on page 52.

## Selecting the Upgrade Tasks to Perform

Use the following tables to determine which upgrade tasks to perform on a federated database that has not previously been upgraded.

***To upgrade a Release 9.3 or 9.4 federated database***

| Do This | If You Want To | See |
|---|---|---|
| Update the Objectivity license | Access the federated database with Release 10.0 | page 67 |

***To upgrade a Release 9.0, 9.1, or 9.2 federated database***

| Do This | If You Want To | See |
|---|---|---|
| Update the Objectivity license | Access the federated database with Release 10.0 | page 67 |
| Upgrade the federated database's schema | Use features introduced in Release 9.3 | page 74 |

***To upgrade a Release 8.x federated database***

| Do This | If You Want To | See |
|---|---|---|
| Update the Objectivity license | Access the federated database with Release 10.0 | page 67 |
| Upgrade the federated database's internal database format and schema | Use features enabled by the new internal database format introduced in Release 9.0, and the schema introduced in Release 9.3 | page 68 |

***NOTE*** You must update the Objectivity license before performing any other upgrades.

## Updating to an Objectivity Release 10 License

A Release 8.*x* or Release 9.*x* federated database must have an Objectivity Release 10 license to authorize access by Release 10.0 tools and applications. If you have not already done so, you must perform the following steps to replace the federated database's existing license with your Objectivity Release 10 license.

To update the Objectivity license for one or more Release 8.*x* or Release 9.*x* federated databases:

1. Verify that you have set up a default license file containing your Objectivity Release 10 license. See the Objectivity/DB installation chapter of *Installation and Platform Notes* for your platform.
2. For each Release 8.*x* or Release 9.*x* federated database to be accessed with Release 10.0 tools and applications, enter:

   `oolicense -fromdefault` *`bootFilePath`*

   where

   | | |
   |---|---|
   | `-fromdefault` | Obtains `oolicense.txt` in the Objectivity/DB installation directory *`installDir`*. |
   | *`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. |

## Upgrading to the Release 9.0 Internal Database Format

For backward compatibility, Release 10.0 tools and applications can access the data in a federated database created with Release 8.*x*, provided the license is updated. However, new tools and applications accessing a Release 8.*x* federated database cannot take advantage of many of the features introduced in Release 9.0 and later, unless you upgrade the federated database's internal database format. (For a list of the features enabled by the new internal database format, see *Objectivity Release Notes*, *Release 9.0*, on the Technical Support Web site.)

### About the Upgrade Procedure

The upgrade procedure uses the Release 10.0 tool `ooupgrade` to convert the internal database format of an existing Release 8.*x* federated database. This procedure additionally updates the federated database's schema to support features introduced in Release 9.3.

***NOTE*** You do not need to upgrade the internal database format of a Release 9.*x* federated database. If appropriate, you can upgrade the federated database's schema as described in "Upgrading the Schema to Release 9.3" on page 74.

You can upgrade an entire federated database in a single operation or one database at a time.

Each database is upgraded "in place." The original database is renamed with a tilde appended to the original filename. A database with the new format is created using the original filename, and is then populated with the information from the original database. The renamed original database is deleted, leaving the now-upgraded database with the original filename.The upgrade operation preserves the identifiers of all objects in an upgraded database.

***WARNING*** Once a federated database is upgraded, it cannot be accessed by pre-Release 9.0 tools and applications. If interoperating with such tools and applications is required, skip this upgrade procedure and see "Maintaining Earlier Objectivity/DB Applications" on page 81.

***WARNING*** The upgrade operation deletes any backup set entries and information about previous backups. The first backup after an upgrade should be a full backup.

## Preparing a Federated Database for Upgrade

Before upgrading all or part of a Release 8.*x* federated database, prepare it as follows:

1. If you have not already done so, update the Objectivity license in the existing federated database; see page 67.
2. Make a full backup. If you plan to upgrade one database at a time, the safest procedure is to make a full backup before you upgrade each database. See the Objectivity Technical Support Web site for more information on backup strategies.
3. Run the `oocheck` tool to verify the federated database is consistent. You will need to fix any inconsistencies before upgrading. For example, you could use the `oofix` tool or call Objectivity Customer Support.
4. If the existing federated database is partitioned, read "Clearing Autonomous Partitions Before Upgrading" on page 73, and clear autonomous partitions as necessary.
5. Ensure that the lock-server host for the federated database being upgraded is running a lock server from Objectivity/DB Release 7.1 or later. This is required for compatibility with the new internal database format.

## Preparing to Run ooupgrade

To prepare to run `ooupgrade` on a Release 8.*x* federated database:

1. If you have not already done so, make a full backup.
2. Be sure you have enough available disk space for the upgrade.

   Upgrading a database requires temporary access to free space equal to the database's original size. If multiple databases are being upgraded, the available free space must equal the size of the largest database. You need more free space if you choose an architecture in step 3 that causes conversion from a 32-bit disk format to a 64-bit disk format, or if you increase the storage-page size in step 4.
3. (Optional) Choose a different owning architecture for the storage pages of each database being upgraded. The architecture of a storage page determines the disk format used by the page.

   By default, the storage pages of an upgraded database retain their original owning architecture(s), except for the system database, which is converted to the disk format of the current host architecture. **Hint:** You can run `ooupgrade -help` to get a list of available architectures.
4. (Optional) Choose a larger storage-page size for each database being upgraded. For guidelines, see "Increasing the Storage-Page Size" on page 70.
5. Arrange to stop applications from accessing the federated database while the system database is being upgraded. After the system database is upgraded,

concurrent access can be resumed in databases other than the database currently being upgraded. (The `ooupgrade` tool obtains an exclusive lock on the database currently being upgraded.)

### Increasing the Storage-Page Size

By default, an upgrade operation preserves the original storage-page size used by each upgraded database. You can choose to increase the storage-page size of an upgraded database for reasons such as the following:

- If the database contains storage pages that are owned by a 32-bit architecture, and the upgrade operation will change the ownership to a 64-bit architecture.
- If the database contains storage pages that are currently full, and you want applications to be able to create additional persistent objects on those storage pages after the upgrade operation is complete.

---

**NOTE** Increasing the size of the storage pages in a database causes the overall size of the database file to increase. You may not decrease the storage-page size.

---

If you increase the storage-page size while upgrading multiple databases in a single operation, all of the upgraded databases will use the same storage-page size. If only a subset of your databases require a larger storage-page size, consider upgrading them separately, before upgrading the remaining databases as a group, preserving their original storage-page sizes.

If the new storage-page size is greater than the default storage-page size of the federated database, then the default storage-page size of the federated database (and the system database) is changed to the new value.

Increasing a database's storage-page size is recommended if the database contains storage pages owned by a 32-bit architecture, and you have chosen a 64-bit architecture in step 3 above. Such a change increases the sizes of basic objects on the converted storage pages. The upgrade operation automatically *redirects* objects to pages with more storage space as needed to accommodate the resized basic objects. Redirection can have a minor performance impact for operations that access the objects. You can avoid such redirection by specifying a larger storage-page size for the upgrade. The amount by which to increase a database's storage-page size depends on the data itself. Each reformatted basic object will require an additional 8 bytes of overhead, plus 4 bytes for each inline to-many association (if any), plus 4 bytes for each variable-sized array (VArray) attribute (if any). One conservative approach for avoiding redirection is to use one and one half times the current page size, if this is possible without exceeding the maximum possible page size.

If the database contains any basic object that was previously redirected as a result of schema evolution, such redirection is preserved, regardless of the page size.

## Running the ooupgrade Tool

***WARNING*** Proceed only if you have a valid backup of the federated database or database being upgraded.

Enter `ooupgrade` with the indicated options:

- To upgrade an entire federated database in a single operation, enter:
  `ooupgrade -all bootFilePath`
  where

  | | |
  |---|---|
  | `-all` | Upgrades the entire federated database, including the system database and every database. |
  | *`bootFilePath`* | Path to the boot file of the federated database. |

- To upgrade a federated database one database at a time, enter the following for each database:
  `ooupgrade -db dbSysName bootFilePath`
  where

  | | |
  |---|---|
  | `-db` *`dbSysName`* | Upgrades an individual database specified by the system name. If the system database has not yet been upgraded, it is automatically upgraded with the first database you specify. |
  | *`bootFilePath`* | Path to the boot file of the federated database. |

In either case, you may also need to include one or more of the following options:

| | |
|---|---|
| `-licensefile` *`fileName`* | File containing your current Objectivity license. The `ooupgrade` tool must be able to access this file to refresh the license in the upgraded system database. You can omit this option if your license file is in a default location. |
| `-format` *`newArch`* | New owning architecture for the storage pages in each database being upgraded (including the system database), if a different owning architecture is desired. |
| `-pagesize` *`size`* | Storage-page size, in bytes, for each database being upgraded, if a larger page size is required. See “Increasing the Storage-Page Size” on page 70 |

## After the Upgrade Operation Completes

After a federation database has been upgraded:

- No pre-Release 9.0 application or tool can access the upgraded federated database.
- The upgrade operation deletes any backup set entries and information about previous backups. Consequently, the first backup after an upgrade should be a full backup.
- The federation database's schema is up-to-date, and contains descriptions of all the system schema types that were added in Release 9.0 and Release 9.3.

  **Note:** The upgrade operation has no effect on the schema descriptions of application-specific classes. Consequently, if an application-specific class was added to the schema in Release 8.*x*, any persistent character or Boolean attributes in the class will continue to be represented using Objectivity/DB integer types.
- Any new databases subsequently added to the federated database will use the new internal database format.

## Optional Processing After Increasing the Storage-Page Size

If you performed an upgrade operation that increased the storage-page size in one or more databases of a federated database, you can optionally write and run a post-processing utility to ensure that every object takes advantage of the increased storage-page size. Doing so can improve the performance of subsequent application access to the federated database. Your utility can perform any of the following tasks as appropriate:

- Re-create existing persistent collections.

  The storage-page size of a database sets the maximum size of node-arrays in an ordered collection, and is a factor for determining the maximum capacity of hash buckets in an unordered collection. Increasing the storage-page size with `ooupgrade` does not affect these limits for existing persistent collections. If you want the node-arrays (or hash buckets) of an existing persistent collection to take advantage of the increased storage-page size, your utility must create a new, replacement persistent collection, and call the replacement collection's `addAll` method, passing the original collection as the parameter. Make sure the utility uses an appropriate clustering strategy for placing the nodes (or hash buckets) of the new collection.
- Drop and re-create indexes.

  The storage-page size of a database sets the maximum size of an index's internal data structure. Increasing the storage-page size with `ooupgrade` does not affect this limit for existing indexes. If you want an existing index to take advantage of the increased storage-page size, your utility must obtain the appropriate key description, and use it to drop and re-create the index.

- Open every basic object for update.

  The storage-page size of a database determines how many objects can fit on a single page. If a previous schema evolution has made existing basic objects too large to fit on their original pages, they are redirected to other pages that have more space. Increasing the storage-page size with `ooupgrade` does not eliminate any such redirection, even if the original pages now have sufficient space. If you want existing redirected objects to be moved back to their original pages wherever possible, your utility can iterate over each basic object and open it for update.

- Open every variable-size array (VArray) for update.

  The storage-page size of a database determines whether a VArray can fit on the same page as the basic object that owns it. If an existing VArray has too many elements to fit on the owning object's page, the VArray is redirected to another page that has more space. Increasing the storage-page size with `ooupgrade` does not eliminate any such redirection, even if there is sufficient space near the owning object. If you want existing redirected VArrays to be moved back with their owning objects wherever possible, your utility can iterate over each owning object, access its VArray attribute, and open the VArray for update.

## Clearing Autonomous Partitions Before Upgrading

Earlier releases of Objectivity/DB High Availability allowed you to transfer the control of a container from its home autonomous partition (the partition controlling the container's database) to a different partition. This feature is not supported in Release 9.0 (or later) federated databases.

Before you upgrade a partitioned federated database in which control of any container has been transferred, you must perform the following step:

➤ Run the `ooclearap` tool on every partition that controls a transferred container.

Clearing an autonomous partition physically returns each container controlled by the partition to its containing database. For more information about `ooclearap`, see *Objectivity/DB High Availability*.

## Upgrading the Schema to Release 9.3

For backward compatibility, Release 10.0 tools and applications can access the data in a federated database created with earlier releases provided the license is updated. However, new tools and applications accessing a pre-Release 9.3 federated database cannot take advantage of certain features introduced in Release 9.3. In particular, Release 9.3 introduced enhanced scalable persistent collections. (For information about these collections, see *Objectivity Release Notes, Release 9.3*, on the Technical Support Web site.)

To take advantage of the features introduced in Release 9.3, you can upgrade the federated database's schema as described below. You normally perform this procedure only on a Release 9.0, Release 9.1, or Release 9.2 federated database.

---

***NOTE*** You do not need to explicitly upgrade the schema of a Release 8.*x* federated database if you are upgrading it as described in "Upgrading Existing Federated Databases" on page 65.

---

To upgrade the schema of a pre-Release 9.3 federated database:

1. Make a backup of the federated database to be upgraded.
2. At a command prompt, enter:
   - On Windows:
     ```
     ooschemaupgrade installDir\etc\schema.9.3.dmp
        bootFilePath
     ```
   - On UNIX:
     ```
     ooschemaupgrade installDir/arch/etc/schema.9.3.dmp
        bootFilePath
     ```

   where

| | |
|---|---|
| *installDir* | Your Release 10.0 Objectivity/DB installation directory. |
| *arch* | Architecture name for your platform (UNIX only). |
| *bootFilePath* | Path to the boot file of the federated database to be upgraded. |

## Upgrading Existing Collections

Release 10.0 applications and tools can access an existing federated database that contains pre-Release 9.3 persistent collections, and can create new enhanced scalable persistent collections, provided you have upgraded the federated database's schema. Although both new and old persistent collections can coexist in the same federated database, they are accessible only by separate portions of your application code.

You can upgrade an existing pre-Release 9.3 persistent collection by writing a special-purpose application that creates a new, enhanced persistent collection as a replacement for the existing one. For guidelines for writing such an application, see *Objectivity Release Notes*, *Release 9.3*, on the Technical Support Web site.

# Upgrading Existing Applications and Scripts

You can upgrade an existing application to take advantage of this release's features and fixes. To upgrade an application, you must recompile it and relink it with Release 10.0 libraries.

When planning whether to upgrade existing applications to Release 10.0, you should take into account any required code changes listed in the following subsections.

---

***NOTE*** These subsections describe only the changes introduced in Release 10.0. For descriptions of code and script changes introduced in earlier releases, see the *Objectivity Release Notes* for Release 9.0, Release 9.1, Release 9.3, and Release 9.4 on the Technical Support Web site.

---

## Upgrading Tool Scripts

Use the following list to determine whether you must rewrite portions of existing scripts to accommodate Release 10.0 changes to Objectivity/DB tools.

- If an existing tool script invokes the `ooupgrade` tool solely for the purpose of converting between a 32-bit architecture and a 64-bit architecture, you can consider changing your script to use the `ooconvertformat` tool instead; see "Enhancements to ooconvertformat" on page 25. (`ooupgrade` may still be the better choice, because it enables you to increase the page size in the same operation, to accommodate objects that will grow because of the format conversion.)
- If an existing tool script invokes the `ooupgrade` tool without the `-format` option, it will no longer implicitly convert storage pages to the architecture of the host on which you are running the script; see "Enhancements to ooupgrade" on page 25. If you want the upgrade procedure to perform format conversion, you must change your script to include the `-format` option explicitly.

## Upgrading Objectivity/C++ Applications

Use the following list to determine whether you must rewrite portions of existing C++ applications to accommodate Release 10.0 changes in Objectivity/C++.

- If an existing Objectivity/C++ application deliberately uses a standard clustering strategy to fill containers to their maximum addressing space, you should change your code to accommodate the change described in "Support for Container-Fill Policy in ooClusterStrategy Class" on page 30. In particular, you should set the clustering strategy's new container-fill policy to enable the clustering strategy to add logical pages until the maximum logical-page identifier has been reached.
- If an existing Objectivity/C++ application deliberately uses a standard clustering strategy with default clustering priorities to keep new basic objects only within existing containers, you should change your code to accommodate the change described in "Default Clustering Priorities" on page 22. In particular, you should explicitly set the clustering priorities as follows:

  ```
  oocSamePage | oocOtherPage | oocNewPage
  ```

  This is highly recommended when adding objects to ordered persistent collections that use short object references.
- If an existing Objectivity/C++ application uses a standard clustering strategy while adding elements to an unordered persistent collection (a collection of class `ooHashSetX` or `ooHashMapX`), you may need to change your code to accommodate the change described in "Clustering of Hash Buckets in Unordered Collections" on page 22. In particular, if you want to store the collection's hash buckets in multiple containers or databases, you should either use the default clustering priorities or explicitly set the clustering priorities of the applicable clustering strategy to include the `oocNewContainer` and `oocNewDatabase` clustering rules.
- If an existing Objectivity/C++ application defines custom operators using the obsolete items listed in "Custom Operator Functions" on page 38, you must remove that code. You can re-create custom operators and register them using the new classes described in "Creating Custom Predicate-Query Operators" on page 34.
- If an existing Objectivity/C++ application relies on a precise storage-page count during a transaction, you should change your code to accommodate the change described in "Change to Returned Storage-Page Count" on page 37. In particular, you should checkpoint the transaction before calling this method.
- (Recommended) Replace any code for superseded items listed on page 37. If you continue using the superseded `ooQuery` class, you now need to include `oo.h` instead of `ooUserOper.h`.
- Replace any code for obsolete items listed on page 38.

## Upgrading Objectivity/DB Active Schema for C++ Applications

Use the following list to determine whether you must rewrite portions of existing applications to accommodate Release 10.0 changes in Active Schema for C++.

- If an existing Active Schema application calls a method that adds a property to a proposed class, and then tests the result by explicitly comparing it to the constant `oocSuccess`, you should change your code to accommodate the change described in "Return Types of Certain Proposed_Class Methods" on page 40.

  In particular, when a call to a property-addition method succeeds, it no longer returns a value that implicitly converts to the integer 1. Consequently, a test of the following form no longer works:

  ```
  // test no longer works in R10
  if (classProposal.add_basic_attribute(...) != oocSuccess)
     // Error handling
  ```

  You should change any such statement so that it tests the result of a property addition as follows:

  ```
  // test still works in R10
  if (!classProposal.add_basic_attribute(...))
     // Error handling
  ```

  If the call in this test fails, the null result implicitly converts to 0; if the call in this test succeeds, the non-null result is guaranteed to implicitly convert to a non-null integer.
- (Recommended) Replace any code for superseded items listed on page 41.
- Replace any code for obsolete items listed on page 41.

## Upgrading Objectivity for Java Applications

Use the following list to determine whether you must rewrite portions of existing Java applications to accommodate Release 10.0 changes in Objectivity for Java.

- Rebuild each existing Objectivity for Java application being upgraded with JDK 1.6, to accommodate the change described in "Discontinued Support for JDK 1.5" on page 53.
- If you are rebuilding an existing Objectivity for Java application with Sun JDK 1.6, and the application has persistence-capable classes that define a

`finalize()` method, you must define a default constructor for each such class. For example:

```
public class PCclass extends ooObj {
   // If this exists...
   protected void finalize() {
   }
   // ... you must add this...
   public PCclass() { // Construct a dataless object
   }
}
```

This is a workaround to a reported Sun JDK 1.6 defect that prevents finalizers from being called for Java objects created through JNI. For more details, see the Java application development appendix in the *Installation and Platform Notes* for your platform.

- If an existing Objectivity for Java application implements the `com.objy.db.app.ClusterStrategy` interface directly, you must change your code to accommodate the change described in "New ClusterStrategy Class Instead of Interface" on page 45. In particular, you should change your code either to extend the new `com.objy.db.app.ClusterStrategy` class. If your code already extends the `DefaultClusterStrategy` class, you should consider changing your code to extend the `ClusterStrategy` class.
- If an existing Objectivity for Java application deliberately uses a standard clustering strategy to fill containers to their maximum addressing space, you should change your code to accommodate the change described in "Support for Container-Fill Policy in Clustering Strategies" on page 46. In particular, you should set the clustering strategy's new container-fill policy to `maximum`, enable the clustering strategy to add logical pages until the maximum logical-page identifier has been reached.
- If an existing Objectivity for Java application deliberately uses a standard clustering strategy with default clustering priorities to keep new basic objects only within existing containers, you should change your code to accommodate the change described in "Default Clustering Priorities" on page 22. In particular, you should explicitly set the clustering priorities as follows:

  `SamePage | OtherPage | NewPage`

  This is highly recommended when adding objects to ordered persistent collections that use short object references.
- If an existing Objectivity for Java application uses a standard clustering strategy while adding elements to an unordered persistent collection (a collection of class `ooHashSetX` or `ooHashMapX`), you may need to change your code to accommodate the change described in "Clustering of Hash Buckets in Unordered Collections" on page 22. In particular, if you want to

store the collection's hash buckets in multiple containers or databases, you should either use the default clustering priorities or explicitly set the clustering priorities of the applicable clustering strategy to include the `NewContainer` and `NewDatabase` clustering rules.

- If an existing Objectivity for Java application includes calls to static methods of the `Connection` class, you may need to call those methods on a particular connection object to accommodate the change described in "Support for Multiple Connections" on page 44.
- If an existing Objectivity for Java application relies on the default hot mode setting for new sessions and session pools, you may need to change your code to accommodate the change described in "Hot Mode Enabled By Default" on page 51. You now need to explicitly turn hot mode off if you do not want it, and you may also wish remove redundant calls to `com.objy.db.app.Session.setHotMode(true)` from your code.
- If an existing Objectivity for Java application relies on a precise storage-page count during a transaction, you should change your code to accommodate the change described in "Change to Returned Storage-Page Count" on page 52. In particular, you should checkpoint the transaction before calling this method.
- If an existing Objectivity for Java application attempts to determine the state of an object outside a transaction, you should change your code to accommodate the change described in "Certain Methods Now Sensitive to Transaction State" on page 51. In particular, you should change your code so that the following methods are always called while the session is in an active transaction:
  - `com.objy.db.app.ooObj.isModified()`
  - `com.objy.db.app.ooObj.isFetchRequired()`
- If an existing Objectivity for Java application uses an obsolete method to both mark an object as modified and perform a partial fetch of its reference attributes, you must remove that method from your code. The obsolete methods are listed in "Marking a Subset of Reference Attributes as Modified" on page 53.
- If an existing Objectivity for Java application forms a unidirectional relationship link from a transient source object to a persistent destination object, you may need to change your code to accommodate the change described in "Change to Auto-Persistence in Unidirectional Relationships" on page 52. In particular, your code should not assume that calling `setFormTransientRelationships(true)` will allow the source object of the relationship to remain transient.
- (Recommended) Replace any code for deprecated items listed on page 53.
- Replace any code for obsolete items listed on page 53.

## Upgrading Objectivity/DB Active Schema for Java Applications

Use the following list to determine whether you must rewrite portions of existing Java applications to accommodate Release 10.0 changes in Active Schema for Java.

- (Recommended) Replace any code for deprecated items listed on page 55.
- Replace any code for obsolete items listed on page 55.

## Upgrading C++ Programs for Monitoring Lock-Server Performance

Use the following list to determine whether you must rewrite portions of existing C++ programs that monitor lock-server performance, to accommodate Release 10.0 changes in Objectivity/C++.

---

***NOTE*** Once you have upgraded a lock-server performance-monitoring program to work with a Release 10.0 lock server, your program will no longer be compatible with pre-Release 10.0 lock servers.

---

- If an existing program uses values of type `oolsTimestamp`, you must adjust your code to use values of type `oolsTimeval` instead, to accommodate changes in "Changed Timestamp Type in Message Classes" on page 27. For example, any code that assigns an `oolsTimestamp` value to a `long` variable should now assign the members of an `oolsTimeval` value into a pair of 32-bit unsigned integer variables (seconds and milliseconds). You may also need to adjust your code to reflect the new meaning of the value (absolute time of a lock-server event instead of time relative to the start of the monitoring program).

# Maintaining Earlier Objectivity/DB Applications

After installing Objectivity/DB Release 10.0 along with your chosen Objectivity programming interface, you normally:

- Develop new Release 10.0 applications.
- Upgrade existing applications and then recompile and relink them with Release 10.0; see "Upgrading Existing Applications and Scripts" on page 61.

In some situations, you may also need to maintain existing applications built with an earlier (pre-Release 10.0) Objectivity/DB release.

## Maintaining Unrebuilt Release 9.*x* Applications

If you are maintaining an unrebuilt Release 9.*x* application (that is, an application built with Objectivity/DB Release 9.0, Release 9.1, Release 9.2, Release 9.3, or Release 9.4), note that such earlier applications:

- Are completely compatible with Release 10.0 federated databases, with the following limitation:
  - An unrebuilt earlier application cannot access a Release 10.0 federated database or autonomous partition with a boot-file name that has only 1 character, or has 32 or more characters; see "Longer Boot File Names Allowed" on page 23 and "New Limits on Boot-File Names of Autonomous Partitions" on page 26.
- Can interoperate with new or upgraded applications, with the following limitation:
  - An unrebuilt pre-Release 9.3 application cannot access any enhanced collections created by a new or upgraded application.
- Cannot recognize the new client-host architectures introduced in Release 10.0; see "New Platforms" on page 13.

  If a new or upgraded application is compiled for one of the newly introduced architectures, any storage pages it creates cannot be read by an unrebuilt earlier application. You can prevent this problem by setting a disk-format property in the new or upgraded application so that any new pages it creates will use the disk format of an architecture that can be recognized by the earlier application.

***WARNING*** Do not operate on a pre-Release 10.0 federated database with an Objectivity/DB tool provided for one of the newly introduced architectures, if that tool could write new storage pages *and* if you plan to access the federated database with an unrebuilt pre-Release 10.0 application.

## Maintaining Unrebuilt pre-Release 9.0 Applications

If you are maintaining an unrebuilt pre-Release 9.0 application, you should consult the section on maintaining earlier Objectivity/DB applications in *Objectivity Release Notes*, *Release 9.0*, on the Technical Support Web site.

An unrebuilt pre-Release 9.0 application:

- Is not compatible with Release 10.0 federated databases.
- Cannot interoperate with Release 10.0 applications unless the newer applications follow certain guidelines.
- Does not recognize the client-host architectures introduced in Release 9.*x* or in Release 10.0.


OBJECTIVITY, INC.
640 West California Avenue, Suite 210
Sunnyvale, California 94086-3624
USA
+1 408.992.7100
+1 408.992.7171 Fax
www.objectivity.com
info@objectivity.com