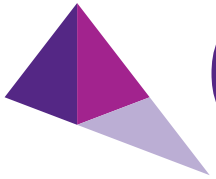

Objectivity/DB

Objectivity Release Notes

Release 11.0

Objectivity Release Notes

Part Number: 11.0-RN-0

Release 11.0, February 5, 2013

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 11.0. This chapter provides an overview of these changes. This chapter summarizes:

- New products and features
- Updated products
- Combined products
- Deprecated product
- Obsolete platforms
- New and updated books

For Additional Information About This Release

The Objectivity Developer Network has the latest detailed information about supported platforms and compilers, required operating-system patches and compiler patches, the open and fixed software problems, and documentation errata and corrections. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

For Information About Previous Releases

For information about previous releases, see the corresponding *Objectivity Release Notes* on the Objectivity Developer Network.

New Products and Features

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

Product or Feature	Description	See
Managed object-placement system	Model-based technique for automatically placing new persistent objects in a federated database.	page 15
File-storage locations	New mechanism for specifying storage locations for database and container files.	page 18
Model-based indexes	New mechanism for defining and creating indexes of persistent objects.	page 19
Administrative tools	New tools for supporting placement managed-federated databases.	page 22–page 23
Tool runner	New command-line mechanism for running Release 11.0 Objectivity/DB administrative tools.	page 21
Navigation query capability	(*Objectivity/C++*) Supports navigation of persistent objects as a graph.	page 27
Predicate query language (PQL)	Supports additional operators	page 19
Enhanced query server	Extends the parallel query capability to support distributed graph navigation.	page 25
Java interface to PQL expression trees	(*Objectivity for Java*) Support for the predicate expression tree interface, which provides support for external search agents and query builders and parsers.	page 39
Generated federated-database identifiers	Automatic assignment of unique identifiers to new federated database	page 25
Objectivity product installation	Full-featured installation program native to each supported platform.	page 20

Updated Products

Products With New, Changed, or Removed Features

Product Name	Product Description	See
Objectivity/DB	Distributed object database.	page 15, page 21
Objectivity/C++	C++ programming interface to Objectivity/DB.	page 27
Objectivity/C++ Data Definition Language (Objectivity/DDL)	Objectivity/C++ option for creating and maintaining a schema of persistence-capable class definitions.	page 35
Objectivity for Java	Java programming interface to Objectivity/DB.	page 37
Objectivity/.NET for C#	C# programming interface to Objectivity/DB.	page 47

Products Rebuilt for Compatibility

Product Name	Product Description
Objectivity/DB High Availability (Objectivity/HA)	Objectivity/DB option supporting autonomous partitions and data replication.
Objectivity/DB (Lock-Server Performance Monitor)	Programming interface for writing C++ programs that gather information about how database applications use a running Objectivity/DB lock server.
Objectivity/DB Open File System (Objectivity/OFS)	Customizable interface between Objectivity/DB and hierarchic storage systems.
Objectivity/DB Secure Framework (Objectivity/Secure)	Customizable interface between Objectivity/DB and standard security solutions.
Objectivity/Python	Python programming interface to Objectivity/DB.
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.

Combined Products

The following table lists Objectivity products or options that are no longer separately purchased or licensed, but are now combined with their base products:

Product	Description	Now Part Of
Objectivity/DB In-Process Lock Server	Objectivity/DB option that enables a database application to run a lock server within the same process to improve performance.	Objectivity/DB
Objectivity/DB Parallel Query Engine	Objectivity/DB option enabling applications to find persistent objects by searching separate portions of a federation in parallel.	Objectivity/DB
Objectivity/DB Active Schema for C++	C++ programming interface for reading and modifying an Objectivity/DB schema dynamically.	Objectivity/C++
Objectivity/DB Active Schema for Java	Java programming interface for reading and modifying an Objectivity/DB schema dynamically.	Objectivity for Java

Deprecated Product

Objectivity/Smalltalk for VisualWorks is deprecated in this release.

Obsolete Platforms

Support has been removed for the following Objectivity platforms:

Operating System or Compiler	Addressing Mode	Abbreviation
HP-UX 11iv3	32-bit	`hprisc`
Visual C++ 8.0 (Visual Studio 2005)	32-bit	Windows

New and Updated Books

During installation, the online books for Objectivity products are placed in the following subdirectory of your Release 11.0 Objectivity/DB installation directory *installDir*:

installDir/doc

NOTE Objectivity online books are also available on the Objectivity Developer Network.

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 11.0:
  - *Objectivity Release Notes, Release 11.0* (this document)
  - *Objectivity/DB Administration, Release 11.0*
  - *Getting Started With Managed Object Placement, Release 11.0—new book*
  - *Objectivity/DB High Availability, Release 11.0*
  - *Objectivity/C++ Programmer's Guide, Release 11.0*
  - *Objectivity/C++ Programmer's Reference, Release 11.0*
  - *Objectivity/C++ Data Definition Language, Release 11.0*
  - *Objectivity/DB Active Schema for C++, Release 11.0*
  - *Objectivity for Java Programmer's Guide, Release 11.0*
  - *Objectivity/DB Schema Development, Release 11.0*
  - *Objectivity/SQL++, Release 11.0*
  - *Monitoring Lock Server Performance, Release 11.0*

Note

The following books are no longer delivered as PDF files:

- *Installation and Platform Notes*
- *Objectivity/SQL++ ODBC Driver User's Guide*

This information is now available only on the Objectivity Developer Network.

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

where:

installDir Release 11.0 Objectivity/DB installation directory.

Books in HTML

The installation and configuration documentation is available on the Objectivity Developer Network.

The following books are available in HTML format in the Objectivity/DB installation directory:

- *Objectivity/DB Administration,* Release 11.0
- *Objectivity/DB High Availability*, Release 11.0
- *Objectivity for Java Programmer's Guide,* Release 11.0
- *Objectivity for Java Programmer's Reference,* Release 11.0
- *Objectivity/DB Active Schema for Java Programmer's Reference,* Release 11.0
- *Getting Started With Objectivity/Python,* Release 11.0
- *Objectivity/Python Programmer's Reference,* Release 11.0

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

installDir/doc/java/index.html
installDir/doc/objyHelp/index.html
installDir/doc/python/index.html

where:

installDir Release 11.0 Objectivity/DB installation directory.

Books in Compiled HTML Help

The *Objectivity/.NET for C# Programmer's Reference, Release 11.0* 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

Additional Topics

See the Objectivity Developer Network for the following:

- The PDF book *Objectivity/C++ Backward Compatibility,* which describes selected superseded Objectivity/C++ programming mechanisms.
- The advanced topics listed below:

Document Title	Describes
Page and Object Encryption	How to provide plugin code to encrypt and decrypt persistent objects and pages.
External Query Agents	How to provide plugin code to extend the query server so that Objectivity predicate scans can fetch information from sources other than Objectivity databases.
Expression Tree Interface	The predicate expression tree interface, which provides support for external search agents and query builders and parsers.
Custom Operator Support	How to provide custom operators that can be used by the Objectivity predicate query language.
Parallel Query Engine Customization	How to set up and use custom parallel queries.
Server Administration Security	How to provide plugin code that implements security restrictions on selected server operations, such as stopping the lock server.

Other advanced topics are added as they become available.

2

Base Products

This chapter provides an overview of Release 11.0 additions and changes to Objectivity/DB and other language-independent Objectivity products.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity/DB (Database Services)

This section summarizes the impact of Release 11.0 on the Objectivity/DB database services.

New Features

The following subsections describe new Objectivity/DB database services.

Managed Placement of Persistent Objects

Objectivity/DB now supports two techniques for placing new persistent objects in a federated database's storage hierarchy:

Technique	Description
Managed placement	Manages the placement of new persistent objects in containers and databases that are created automatically.
Explicit placement	Relies on the application to explicitly create containers and databases, and then place new persistent objects using a clustering strategy.

New applications should use managed placement. Explicit placement is supported for backward compatibility with existing, pre-Release 11.0 federated databases and applications.

Managed placement is a model-based technique for placing new persistent objects in a federated database. Whenever an application adds a new persistent object, it consults the federated database's *placement model*, which contains rules for choosing the location according to the type of object being added. The placement model also determines when and where to create new containers and databases to accommodate new persistent objects.

You get started with the new technique by using the `CreateFd` tool (page 22) to create a *placement-managed* federated database. The new federated database contains an initial placement model that is sufficient for simple demo applications or quick prototypes. When a more sophisticated placement design is required, you (or your team's database designer) perform the following steps:

1. Obtain an XML representation of the initial placement model by exporting it into a *placement-model document* (PMD).
2. Edit the PMD in an XML editor to add new rules and specifications.
3. Import the new version of the placement model from the PMD into the federated database.

You perform the export and import steps with new administrative tools; see "Administrative Support for Managed Placement" on page 23.

To learn about creating and installing a placement model, see *Getting Started With Managed Object Placement* (`installDir/doc/placementManagement.pdf`).

An application that accesses a placement-managed federated database:

- Places new data files in storage locations selected from a list of registered locations; see "Inventory of Available File-Storage Locations" on page 18.
- Uses any indexes that are defined in the placement model and created with administrative tools; see "Indexes in a Placement-Managed Federated Database" on page 19.

Managed placement has the following advantages over explicit placement:

Simplified code	Applications no longer need to include code for creating clustering strategies, storage objects, or indexes, allowing the development team to focus on the application's business logic.
Flexible	Changes to placement rules, to indexes, and to the inventory of available file-storage locations are made outside of any application, so you can easily make changes without revising code and recompiling.
Faster queries	As new objects are placed in the federated database, their locations are recorded in an internal placement map, which is implicitly consulted by query operations to narrow the search for objects of particular types.

Consistency	All applications and tools accessing a particular federated database use the same placement model and therefore place objects according to the same rules.
Self documenting	All placement rules for a federated database are recorded in a single, readable placement model document, instead of being located throughout the code of one or more applications.

Consequences for application development

Managed placement changes and simplifies Objectivity/DB application development in a number of ways. In particular, an application that accesses a placement-managed federated database no longer performs the following tasks:

- Create and manage databases and containers in a federated database, or use application-defined container classes. (An exception is thrown if an application tries to explicitly create a database or container.)
- Use a container or database as the scope for scanning, iterating, or scope naming.
- Use clustering strategies.
- Move, copy, or enable versioning for persistent objects.
- Define and create indexes over persistent objects; see "Indexes in a Placement-Managed Federated Database" on page 19.
- Use short object identifiers (short object references), short inline relationships (associations), or ordered collections with short object references to their elements.
- Purge schema history.
- Create and manage autonomous partitions or database images.

NOTE A placement-managed federated database is not compatible with Objectivity/DB High Availability or with Objectivity/SQL++. You can use Objectivity/Python to read from a placement-managed federated database, but not to add new persistent data.

The application can still specify an existing persistent object when creating a new persistent object. However, instead of being used by a clustering strategy, the specified object is passed to the federated database's placement model, which can be configured to use the existing object as a guide for placing the new object.

More generally, an application can create and use *placement conditions* to pass additional information to the placement model, such as the application's purpose for creating a new persistent object. The placement model can be configured to use that information to trigger specialized placement behavior.

NOTE An application must use managed placement when accessing a placement-managed federated database, and must use explicit placement when accessing a non-placement-managed federated database.

Summary of new and superseded items

- New tools (page 23)
- Superseded tools (page 26)
- New classes and methods in Objectivity/C++ (page 29), Objectivity for Java page 37, and Objectivity/.NET (page 48)
- Superseded classes and methods in Objectivity/C++ (page 32), Objectivity for Java (page 42), and Objectivity/.NET (page 53)

Inventory of Available File-Storage Locations

A placement-managed federated database (page 15) maintains an inventory of storage locations in which its database files or container files may be created. This inventory, called the *main storage group*, consists of a list of registered storage locations, where each storage location is a combination of a host name and a fully qualified directory path. Initially, a federated database's main storage group is empty, and all data files are placed on the same host and in the same directory as the system database file.

The main storage group is normally populated by a system administrator who is familiar with your network. Storage locations can be added or removed at any time using the new administrative tools `AddStorageLocation`, `RemoveStorageLocation`, and `ListStorage`. For more information about these tools, see "Administrative Support for Managed Placement" on page 23.

In general, the placement model controls how particular storage locations are selected from the main storage group. An application can influence the selection of storage locations for the data files created on its behalf by enabling *storage-location preferences*; see "Storage-Location Preferences in XML Configuration Files" on page 18.

Storage-Location Preferences in XML Configuration Files

You can specify *preferred* storage locations for the data files that are added to a placement-managed federated database (page 15). Storage-location preferences are a way of ranking storage locations (host-and-path combinations) so that when new data files are created, they are placed in the highest ranked locations first, until those locations become full; then the next ranked locations are used, and so on. The ranked storage-locations must also be listed in the federated database's *main storage group* (inventory of registered locations).

You specify preferences in one or more *configuration files*, which are XML files with the filename extension `.config`. Machine-wide preferences apply to all applications running on the same computer. You add machine-wide preferences to the following new file, which is provided with your Objectivity/DB installation:

installDir`/config/Machine.config`

You can create separate configuration files containing application-specific preferences; these are typically located in the application's current directory or in *installDir*`/config`, but can be located anywhere.

You can enable your application to read the machine-wide configuration file, any number of application-specific configuration files, or both. (If both are enabled, application-specific preferences take priority). For programming-interface details, see page 30 (Objectivity/C++); see page 38 (Objectivity for Java); see page 49 (Objectivity/.NET).

To learn about specifying storage-location preferences, see *Getting Started With Managed Object Placement* (*installDir*`/doc/placementManagement.pdf`).

Indexes in a Placement-Managed Federated Database

You define and add indexes over persistent objects in a placement-managed federated database (page 15) using either of the following techniques:

- Edit the federated database's placement model as described in *Getting Started With Managed Object Placement* (*installDir*`/doc/placementManagement.pdf`)
- Using the administrative tools `AddKeySpecification`, `AddIndexSpecification`, `AddIndex`, and `DropIndex`. For more information about these tools, see "Administrative Support for Managed Placement" on page 23.

New Predicate Query Language Operators

Several new operators are available in the predicate query language, as follows:

Operators	Description	Usage

String	Provide standard string operations such as working with substrings or converting ASCII characters to upper or lower case.	`CONTAINS(op1, op2)`
		`SUBSTRING(op1, op2, op3)`
		`SUBSTRING(op1, op2)`
		`UPPER(op1)`
		`LOWER(op1)`

Context	Returns the object reference for the current object being qualified.	`THIS()`

Changed Features

The following subsections describe changes to Objectivity/DB database services.

Installation Program

Objectivity products are now installed by full-featured programs that are native to each supported platform. The new installation programs:

- Automate more setup steps than did the programs and scripts used for previous releases.
- Enable you to selectively install individual Objectivity servers without having to perform a complete installation. For example, you can use the installer to install just AMS or the query server on a remote data-server host.
- Create a convenient `uninstall` program in your *`installDir`*.

The new installation programs introduce new options, such as:

- Copying your Objectivity license from a specified location into your Objectivity/DB installation directory (*`installDir`*).
- (Windows) Installing **Start** menu shortcuts.
- Setting environment variables.

The new installation programs introduce new default behavior, such as:

- Prompting for a different default *`installDir`* pathname, with the release number as a separate path component—for example:

  ```
  Objectivity\11.0
  ```

- (UNIX) Eliminating the architecture component *`arch`* of the *`installDir`* path.
- Setting up certain servers as manual services; see "Installation of Objectivity/DB Servers" on page 24.

NOTE You can now find installation and configuration documentation for each platform on the Objectivity Developer Network. The PDF books *Installation and Platform Notes* are no longer published.

New Locations for Sample Applications

Extended sample applications are no longer distributed with the Objectivity/DB installation. Instead, you can obtain samples by downloading them from the Objectivity Developer Network.

Objectivity/DB (Tools)

This section summarizes the impact of Release 11.0 on Objectivity/DB tools.

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 High Availability" on page 26.

New Tools

The following subsections describe new Objectivity/DB tools.

Tool Runner

You use the new command-line mechanism, or *tool runner*, to execute new tools introduced in Release 11.0.

The general syntax for running a tool using the tool runner is as follows:

`objy` *`toolName optionList`*

where *`optionList`* may contain any number of options of the form *`-option`* or *`-option argument`*

The main syntactic differences are that you enter the tool-runner name `objy` as part of the command, and *`toolName`* does not have the `oo` prefix that other Objectivity/DB tools have.

You can obtain a list of tool-runner based tools as follows:

- At a command prompt, enter:
  `objy`

You can obtain a list of options for a particular tool as follows:

- At a command prompt, enter:
  `objy` *`toolName`* `-help`

The tool runner depends on the presence of a file called `objectivity.crg` in the `etc` subdirectory of your Objectivity/DB installation directory. Do not delete, move, or edit this file.

The tool runner will eventually be the standard mechanism in a future Objectivity/DB release.

CreateFd

You can create a federated database using the new `CreateFd` tool—for example:

```
objy CreateFd -fdName fdSystemName
```

where *`fdSystemName`* is the system name of the new federated database.

You choose whether to use `createfd` or the older `oonewfd` tool based on whether you want to create a federated database that is compatible with the new managed object-placement system; see page 15:

`CreateFd`	Creates a *placement-managed federated database*, which has an installed placement model. When applications create new persistent objects in the federated database, all necessary placement actions will be performed automatically.
`oonewfd`	Creates a *non-placement-managed federated database*, which has no installed placement model. Applications accessing the federated database must explicitly create containers and databases, and place new persistent objects using clustering strategies.

The `CreateFd` tool differs from `oonewfd` in several other ways:

- You execute `CreateFd` using the new tool runner; see page 21.
- `CreateFd` has simpler tool options:
  - You can include just `-fdName` to specify the system name, which is used to generate the names of the boot file and the system-database file.
  - You can omit the `-lockserverhost` option to request the local host. For the complete list of options, see *Objectivity/DB Administration*.
- `CreateFd` has no tool options for specifying:
  - An identifier for the new federated database. Instead, a unique identifier is automatically assigned by Objectivity/DB.
  - A default storage-page size for the new federated database. Instead, the system database file's page size is always 8192, and the page sizes of individual databases are controlled by the placement model (8192 in the initial placement model).

Exporting and Importing Schema and Data

You use the following new tools to transfer schema between two placement-managed federated databases:

`ExportSchema`	Writes a placement-managed federated database's schema to an XML file.
`ImportSchema`	Imports schema from an XML file to a placement-managed federated database.

You can also use these tools to transfer from an existing non-placement-managed federated database to a new placement-managed federated database.

You execute the new tools using the new tool runner; see page 21.

The new tools no longer transfer catalog information, because placement in containers and databases is handled automatically by the destination federated database's placement model; see page 15.

You can still use the existing tools `ooexportcatalog`, `ooexportdata`, `ooexportfd`, `ooexportschema`, and `ooimport` to transfer catalog information, data, and schema between two non-placement-managed federated databases.

Administrative Support for Managed Placement

The following new administrative tools support managed placement. You execute these tools using the new tool runner; see page 21.

Management Activity	Tool Name	Description
Create	`CreateFd`	Create a placement-managed federated database; see page 22.

File Storage	`AddStorageLocation`
	`ListStorage`
	`RemoveStorageLocation`
		Manage the list of host and path combinations that can be used for placing the data files of a federated database.

Indexes	`AddKeySpecification`
	`AddIndexSpecification`
	`AddIndex`
	`ListIndexes`
	`DropIndex`
		Define, create, display, and delete indexes from the federated database.

Placement Model	`ExportPlacement` `ImportPlacement`	Transfer between a federated database's currently installed placement model and a placement model document (PMD).

Management Activity	Tool Name	Description
Provisioning	`CreateContainers`	Pre-create internal structures used by managed placement, to remove the performance impact from running applications.
Schema	`ExportSchema` `ImportSchema`	Transfer between an XML file and a placement-managed federated database's schema; see page 23.

Changed Tools

The following subsections describe changes to Objectivity/DB tools.

Installation of Objectivity/DB Servers

(*Windows, Macintosh, and Linux.*) For security reasons, AMS and the query server are now installed as services, but configured to require manual startup. If you use either or both of these services, you can use your operating system to configure the service for automatic startup. For details, see the installation and configuration documentation on Objectivity Developer Network.

In previous releases, these services were configured to start automatically whenever your computer restarted.

Performing and Restoring Backups

The `oobackupx` and `oorestorex` tools introduced with Release 10.1 are no longer provided as part of your Objectivity/DB installation in Release 11.0. You must use the `oobackup` and `oorestore` tools instead.

If you created a backup using `oobackupx`, and you want to restore data from that backup, please contact Objectivity Customer Support for guidance.

Creating Federated Databases With Objectivity/Assist

Using Objectivity/Assist to create a federated database is now equivalent to running the `createfd` command; see page 22. That is, Assist creates only placement-managed federated databases; see page 15.

Unique Identifiers for Federated Databases

Tools that assign an identifier to a federated database now check with the lock server to guarantee that the identifier is unique among the identifiers recognized by that lock server.

In particular, if you specify the `-fdnumber` option for the following tools, Objectivity/DB uses the specified identifier only if is found to be unique; otherwise Objectivity/DB selects a unique identifier to assign:

- `oonewfd`
- `oocopyfd`
- `ooinstallfd`
- `oochangedb`

Furthermore, if you to omit the `-fdnumber` option from the following tools, Objectivity/DB generates a unique identifier for a new federated database:

- `oonewfd`
- `oocopyfd`

If you run any of these tools with `-standalone`, Objectivity/DB cannot guarantee the uniqueness of the assigned identifier.

ooqueryserver

The query server now supports distributed navigation queries (see "Navigation Query Capability" on page 27), in addition to parallel queries. That is, if your application chooses to use distributed processing to perform a navigation query, you must start a query server process on each data-server host that contains a database to be searched.

oostartams

The maximum number of threads (specified with the `-numthreads` option) is now 64 on Windows platforms.

Superseded Tools

The following subsections describe superseded Objectivity/DB tools.

Superseded tools are still supported in this release, but should not be used with new federated databases.

Tools Supporting Explicit Placement Mechanism

The following tools can be used only with non-placement-managed federated databases. The actions performed by these tools are superseded by the managed object-placement system (see page 15), and so should not be performed on a placement-managed federated database.

- `ooattachdb`
- `oocopydb`
- `ooexportcatalog`
- `ooexportdata`
- `ooexportfd`
- `ooexportschema`
- `ooload`
- `oonewdb`
- `oonewfd`
- `ooupgrade`

Objectivity/DB High Availability

Objectivity/DB High Availability (Objectivity/HA) has no new features in this release.

NOTE The Objectivity/HA product is not compatible with the managed object-placement system (see page 15), and so should not be used in a placement-managed federated database.

Objectivity/HA is still supported for use with non-placement-managed federated databases.

3

C++ Products

This chapter provides an overview of Release 11.0 additions and changes to Objectivity/C++ and other C++-specific Objectivity products and extensions.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity/C++

This section summarizes the impact of Release 11.0 on Objectivity/C++.

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 Objectivity Developer Network.

New Features

The following subsections describe new Objectivity/C++ features.

Navigation Query Capability

Objectivity/C++ now lets you traverse a graph of related objects searching for paths to target objects that meets certain criteria. The objects in the graph can be related through multiple kinds of links, including references to objects, collections, variable-size arrays (VArrays), name maps, as well as by associations.

To perform a navigation query, you construct a *navigator* object that encapsulates information needed for the query. A navigator identifies the source object, specifies the criteria for qualifying target objects, and defines the mechanism for receiving and outputting found results. A navigator also includes filtering

capabilities and traversal algorithms. Navigation policies can be used to affect the behavior of the navigator as a whole, such as by specifying the maximum traversal depth (degrees of separation) and the maximum number of returned results.

The navigation query capability is provided through the following new classes, which are in the `objy::ra` namespaces.

Class	Description
`Navigator`	Constructs a navigator object that encapsulates the information and functionality needed to perform a navigation query.
`GraphView`	Provides a mechanism for filtering out part of a graph before performing a navigation query.
`PredicateQualifier`	Qualifies the target objects of a navigation query through the use of a predicate string in the predicate query language (PQL).
`ObjectIdQualifier`	Qualifies the target object of a navigation query through a handle or object identifier.
`CustomQualifier`	Qualifies the target object of a navigation query according to application-defined criteria. Can also qualify a path, for example, according to a particular sequence of steps.
`NavigationResultHandler`	Provides callbacks for navigators to enable them to output found results and clean up resources on completion of the navigation.
`Guide`	Specifies a high-level navigation approach for a navigator, such as depth-first or breadth-first.
`PathStep`	Encapsulates information about a link between two related objects in a graph
`Path`	Represents a series of steps that connect a source object to a target object during a navigation query.

New Namespaces

Selected Objectivity/C++ classes are now declared in one of the following new namespaces:

Namespace	Description	See
`objy::placement`	Contains placement-related types.	page 29
`objy::query`	Contains types pertaining to general queries.	page 32
`objy::ra`	Contains types and namespaces for navigation queries.	page 27

In general, only new Objectivity/C++ classes are declared in these namespaces; existing classes remain in the global namespace. The one exception is described in "ooObjectQualifier Class Renamed and Relocated" on page 32.

A class's namespace qualification, if any, is shown at the beginning of the description for the class in *Objectivity/C++ Programmer's Reference*.

Managed Placement in Objectivity/C++

Making Objects Persistent

Persistence-capable classes now inherit a new overload of `operator new` that accepts a placement-conditions object (page 29) as a parameter. You can use this overload to create a new persistent object in an application that accesses a placement-managed federated database (page 15).

Such applications can continue to use existing overloads with parameters for passing a *`nearObj`* value, a *`clusterStrategy`* value, or both. The *`nearObj`* value is interpreted as a placement guide, which the placement model may be configured to use. A *`clusterStrategy`* value, if any, is ignored.

Specifying Placement Conditions

You can use a *placement-conditions object* to encapsulate application-specific information to be used when placing a new persistent object in a placement-managed federated database (page 15). A placement-conditions object is an instance of the following new class:

- `objy::placement::Conditions`

When you use this class, you must add the following include directive:

```
#include <objy/placement/Conditions.h>
```

You pass the placement-conditions object as a parameter of a new overload of `operator new`; see "Making Objects Persistent" on page 29. The federated

database's placement model determines whether and how the information encapsulated in the placement-conditions object is used.

Enabling Location Preferences in XML Configuration Files

You can call the following new static method to enable your application to obtain storage-location preferences from one or more XML configuration files:

- `ooObjy::enableConfiguration`

Storage-location preferences are a way of specifying preferred storage locations for files added to a placement-managed federated database; see "Storage-Location Preferences in XML Configuration Files" on page 18.

Documentation

The following Objectivity/C++ books now describe managed-placement techniques *only*:

- *Objectivity/C++ Programmer's Guide*, *Release 11.0*
- *Objectivity/C++ Programmer's Reference*, *Release 11.0*
- *Objectivity/C++ Data Definition Language*, *Release 11.0*

NOTE If you are developing an Objectivity/C++ application and plan to continue using explicit placement, you must consult documentation from a previous release for information about the superseded items classes and methods listed on page 32.

Testing for a Session Pool

You can use the new `haveSessionPool` method of the `ooConnection` class to find out whether a session pool with a specified name already exists.

Refreshing a Container in an MROW Transaction

You can now test whether an MROW transaction's view of a container is out-of-date by calling the following new method on a handle to an object in the container:

- `isContainerUpdated` method of the *`ooRefHandle`*`(ooObj)` classes

The result is true if the container was opened by a concurrent update transaction that committed while your transaction had the MROW lock on the container.

The MROW transaction can refresh its view of an out-of-date container by calling the following new method on a handle to any object in the container:

- `refreshOpenContainer` method of the *`ooRefHandle`*`(ooObj)` classes

Changed Features

The following subsections describe changes to Objectivity/C++ features.

Replaced C++ Library on UNIX Solaris Architectures

Objectivity/DB libraries are now built with `-library=stlport4` instead of `-library=Cstd`.

Consequently, on the `solaris7`, `solaris86_64`, and `sparc64` architectures, you can no longer build Objectivity/DB applications with `libCstd`, and you can no longer link these applications to third-party libraries that are built with `libCstd`.

To accommodate this change, you may need to modify your build process and make minor code changes, in addition to eliminating or replacing any linked libraries that are built with `libCstd`.

Updated Names for Objectivity Shared Library

For consistency, some of the names of Objectivity shared libraries have changed. Check the shared library names in your installation hierarchy—-in particular:

- Linux shared library names now incorporate version numbers. For example, the Objectivity/DB kernel library is now named `liboo.11.0.so`.
- UNIX shared libraries that were previously missing the version numbers have been updated to include them.
- Windows DLLs that were previously missing the version numbers have been updated to include them.

Static Libraries Available Only On Request

(UNIX, Mac) Static runtime libraries such as `liboo.a` are no longer included with the Objectivity/DB distribution. If you need to link your application statically to Objectivity/DB, you must contact Objectivity Customer Support.

Documentation for Compilation Environment and Link Rules

You must now consult the installation and configuration documentation on the Objectivity Developer Network for information about compiling and linking Objectivity/C++ applications.

NOTE The *Installation and Platform Notes* are no longer distributed as PDF files.

Using Object Qualifiers

ooObjectQualifier Class Renamed and Relocated

The following names have changed:

- The `ooObjectQualifier` class has been renamed to `ObjectQualifier`.
- The renamed class is now declared in the new `objy::query` namespace.
- The `ooObjectQualifier.h` include file has been renamed to `ObjectQualifier.h`, and is now in the following subdirectory of the Objectivity/DB installation directory *`installDir`*:

  `installDir``/include/objy/query/ObjectQualifier.h`

Constructor Overloads That Accept Classname String

New overloads of the `ObjectQualifier` class constructor enable you to use a string (instead of the class's type number) to specify the name of the class of objects to be qualified.

Startup Restriction Lifted

You can now call the `ooObjy::startup` static method after a call to the `ooObjy::shutdown` static method. In previous releases, you could call `ooObjy::startup` only once.

Calling `ooObjy::startup` after a shutdown has the effect of re-initializing the interaction with Objectivity/DB, and resets application-wide options to their default values. Consequently, after an application restarts, it will need to repeat any calls to `ooObjy::setDefaultAutoRecover`, `ooObjy::setTuner`, or `ooSession` static methods that change default properties (for example, `ooSession::setDefaultMrowMode`).

Superseded Features

The following subsections describe superseded Objectivity/C++ features.

Superseded features are still supported in this release, but should not be used in new applications.

Explicit Placement Mechanism in Objectivity/C++

The classes and methods in the following table are used only for performing explicit placement of new persistent objects. These items are superseded by the managed object-placement system (see page 15), and so should not be used by an application that accesses a placement-managed federated database.

All of these items have been removed from the Objectivity/C++ documentation, although they are still supported for backward compatibility with applications that access existing non-placement-managed federated databases.

Category	Superseded Items
Clustering	`ooClusterStrategy` class `ooClusterPriorities` enumeration `ooSession::clusterStrategy` method `ooSession::setClusterStrategy` method `operator new` overloads with *`clusterStrategy`* parameter
Copying and Moving Persistent Objects	`ooRefHandle(ooObj)::copy` method `ooObj::ooCopyInit` method `ooRefHandle(ooObj)::move` method `ooObj::ooPostMoveInit` method `ooObj::ooPreMoveInit` method `ooCollectionIterator::moveCurrentTo` method
Versioning Persistent Objects	`ooVersMode` enumeration `ooGeneObj` class `ooObj` methods for accessing versioning associations `ooRefHandle(ooObj)` methods for managing versioning
Containers	`ooContObj`, `ooGCContObj`, `ooDefaultContObj` classes `ooRef(ooContObj)`, `ooHandle(ooContObj)` classes `ooItr(ooContObj)` class `ooNewConts` macro `ooRefHandle(ooObj)::containedIn` method
Databases and Database Images	`ooDBObj` class `ooRef(ooDBObj)`, `ooHandle(ooDBObj)` classes `ooItr(ooDBObj)` class `ooAvailability` type `ooRefHandle(ooFDObj)::contains` method `ooRefHandle(ooFDObj)::hasDb` method `ooSession::setAllowNonQuorumRead` method
Autonomous Partitions	`ooAPObj` class `ooRef(ooAPObj)`, `ooHandle(ooAPObj)` classes `ooItr(ooAPObj)` class `ooOfflineMode` type `ooPurgeAps` function `ooRefHandle(ooFDObj)::bootAP` method `ooRefHandle(ooFDObj)::contains` method `ooSession::getOfflineMode` method `ooSession::setOfflineMode` method

Category	Superseded Items
Short Object References and Short Inline Associations	`ooShortRef(ooObj)`, `ooShortRef(`*`appClass`*`)` classes `ooRefHandle(ooObj)::set_container` method `ooRefHandle(ooObj)::operator=` overload `ooRefHandle(ooObj)::operator==` overload `ooRefHandle(ooObj)::operator!=` overload *`useShortRefs`* parameter of ooTreeListX constructors *`useShortRefs`* parameter of ooTreeSetX constructors *`useShortRefs`* parameter of ooTreeMapX constructors `ooCompareShortRef` class
Indexes	`ooKeyDesc`, `ooKeyField` classes `ooLookupFieldBase` class and its subclasses `ooSession::defaultUseIndex` method `ooSession::setDefaultUseIndex` method `ooSession::setUseIndex` method `ooTuner::useIndex` method

Exceptions	`ooAttemptToAccessOfflinePartitionException`
	`ooNoQuorumException`
	`ooObjectPlacementFailedException`
	`ooOutOfContainerIDsException`

Other	`ooRefHandle(ooObj)::getPageFreeSpace` method
	`ooTwoMachineHandlePtr` function-pointer type
	`ooSession::regTwoMachineHandler` method
	`ooLookupKey` class
	`ooItr(ooObj)::scan` overload with *`lookupKey`* parameter
	`ooQuerySplitter` class
	
	`ooRefHandle(ooFDObj)::convertObjects` method with *`purge_schema`* set to `oocTrue`
	`ooRefHandle(ooFDObj)::upgradeObjects` method with *`purge_schema`* set to `oocTrue`

Objectivity/C++ Data Definition Language

This section summarizes the impact of Release 11.0 on Objectivity/C++ Data Definition Language (Objectivity/DDL).

For a complete description of new and changed features, see the updated online book *Objectivity/C++ Data Definition Language*.

New Features

The following subsections describe new Objectivity/DDL features.

Controlling the Location of ooddlx Output

You can use the new `-c++_directory` and `-header_directory` options of the `ooddlx` tool to specify where it should place the generated C++ implementation or header file, respectively.

You can request that the `ooddlx` tool place the output file in the same directory as the DDL file by specifying the new `-same_output_directory` option.

Objectivity/C++ (Active Schema)

This section summarizes the impact of Release 11.0 on the Objectivity/C++ interface to Active Schema.

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

Superseded Features

The following subsections describe superseded Active Schema features.

Superseded features are still supported in this release, but should not be used in new applications.

Short Object Identifiers and Short Object References

You should no longer use classes and members that create attributes and relationships that store short object identifiers or short object references. These items are still in the documentation, but they should not be used in applications that access a placement-managed federated database.

4

Java Product

This chapter describes an overview of Release 11.0 additions and changes to Objectivity for Java and its Active Schema extension.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity for Java

This section summarizes the impact of Release 11.0 on Objectivity for Java.

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

New Features

The following subsections describe new Objectivity for Java features.

Managed Placement in Objectivity for Java

Making Objects Persistent

You should use the following new methods to make instances of persistence-capable classes persistent in a placement-managed federated database (page 15):

- `com.objy.db.app.ooObj.persist()`
- `com.objy.db.app.ooObj.persist(PlacementConditions)`
- `com.objy.db.iapp.IooObj.persist()`
- `com.objy.db.iapp.IooObj.persist(PlacementConditions)`
- `com.objy.db.iapp.PooObj.persist()`
- `com.objy.db.iapp.PooObj.persist(PlacementConditions)`

You use these methods instead of using overloads of the following methods:

- `com.objy.db.app.ooObj.cluster`
- `com.objy.db.iapp.IooObj.cluster`
- `com.objy.db.iapp.PooObj.cluster`

Specifying Placement Conditions

You can use a *placement-conditions object* to encapsulate application-specific information to be used when placing a new persistent object in a placement-managed federated database (page 15). A placement-conditions object is an instance of the following new class:

- `com.objy.app.db.PlacementConditions`

You pass the placement-conditions object as a parameter of the new `persist` method; see "Making Objects Persistent" on page 37.

The federated database's placement model determines whether and how the information encapsulated in the placement-conditions object is used.

Enabling Location Preferences in XML Configuration Files

You can call the following new static methods to enable your application to obtain storage-location preferences from one or more XML configuration files:

- `com.objy.db.app.Connection.enableConfiguration()`
- `com.objy.db.app.Connection.enableConfiguration(bool, String)`
- `com.objy.db.app.Connection.enableConfiguration(bool, String, String)`

Storage-location preferences are a way of specifying preferred storage locations for files added to a placement-managed federated database; see "Storage-Location Preferences in XML Configuration Files" on page 18.

Documentation

The *Objectivity for Java Programmer's Reference* contains descriptions of new and reorganized items, but most descriptions apply to applications that access non-placement-managed federated databases.

The *Objectivity for Java Programmer's Guide* applies to applications that access non-placement-managed federated databases.

Testing for a Session Pool

You can use the new `haveSessionPool` method of the `Connection` class to find out whether a session pool with a specified name already exists.

Refreshing a Container in an MROW Transaction

You can now test whether an MROW transaction's view of a container is out-of-date by calling the following new method on an object in the container:

- `com.objy.db.app.ooObj.isContainerUpdated()`

The result is true if the container was opened by a concurrent update transaction that committed while your transaction had the MROW lock on the container.

The MROW transaction can refresh its view of an out-of-date container by calling the following new method on any object in the container:

- `com.objy.db.app.ooObj.refreshOpenContainer(int)`

Access to the Expression Tree Interface

The following new package provides the ability to manipulate predicate query language (PQL) expression trees:

- `com.objy.query.expression`

The classes in this package can be used to create an external query builder or parser for finding objects in Objectivity/DB.

If your application uses the Objectivity expression tree interface, you must redistribute the `ooexpr.jar file`.

Changed Features

The following subsections describe changes to Objectivity for Java features.

Reorganized Support for Explicit Placement

The following classes and interfaces are now declared in the new package `com.objy.db.app.storage`:

- `ClusterReason` interface
- `ClusterStrategy` class
- `DefaultClusterStrategy` class
- `ooAPObj` class
- `ooContObj` class
- `ooDBObj` class
- `ooDefaultContObj` class
- `ooGCContObj` class
- `ooGCRootsContObj` class

These classes are part of the superseded mechanism for explicitly placing persistent objects; see "Explicit Placement Mechanism in Objectivity for Java" on page 42.

You can use these classes by adding the following statement to your code:

```
import com.objy.db.app.storage.*
```

New Default Policy for Handling Transient Relationships

Sessions now use a *reachability policy* as the new default policy for determining whether to make transient objects persistent when they are linked through relationships. The reachability policy:

- Differs from the default policy that was in effect in previous releases, as described below.
- Overrides any calls to the deprecated methods listed in "Session Methods for Handling Transient Relationships" on page 45; all such calls are ignored.

Under the new reachability policy, a transient object in a relationship becomes persistent automatically only if it is *reachable* from a persistent object—that is, if one or more links can be followed from a persistent (source) object to the transient (destination) object. Otherwise, the transient object remains transient, enabling you to create graphs of transient objects that you can optionally make persistent at a later time. More specifically:

- The following actions cause a transient object to become persistent immediately:
  - Forming a unidirectional relationship from a persistent source object to a transient destination object.
  - Forming a bidirectional relationship between the transient object and a persistent object.
- The following actions allow a transient object to remain transient:
  - Forming a unidirectional relationship from a transient source object to any destination object.
  - Forming either a unidirectional or a bidirectional relationship between two transient objects.

The new reachability policy is equivalent to the pre-Release 11.0 *transient-relationships policy*, which was obtained by calling a session's `setFormTransientRelationships` method and passing the value `true`.

The new reachability policy differs from the *standard relationship-persistence policy*, which was the default prior to Release 11.0. Under the old default policy, *all* of the above-listed actions made transient objects persistent immediately, regardless of reachability. The pre-Release 11.0 standard relationship-persistence policy was

obtained by omitting any call to a session's `setFormTransientRelationships` method, or by calling that method with the value `false`.

In Release 11.0, the new reachability policy governs the behavior of all sessions unless you explicitly call the connection's `setPersistencePolicy` with the value `oo.PreRelease11Behavior`; see "Connection Method for Handling Transient Relationships" below.

If any session in your application depends on the pre-Release 11.0 default policy, you should change your code to accommodate the new default behavior; see "Adapting to the New Objectivity for Java Reachability Policy" on page 64.

Connection Method for Handling Transient Relationships

The following method now controls how transient objects are handled when they are linked through a relationship:

- `com.objy.db.app.Connection.setPersistencePolicy(int)`

If you never call this method, or call it with the value `oo.Reachability`, all sessions will use the new *reachability policy* as the new default policy for determining whether to make transient objects persistent when they are linked through relationships; see "New Default Policy for Handling Transient Relationships" above.

This method replaces the deprecated session methods listed on page 45. If you need to preserve the pre-Release 11.0 default policy in one or more sessions, or if you want to continue using any of the deprecated session methods, you must call the connection's `setPersistencePolicy` with the value `oo.PreRelease11Behavior`.

Using Object Qualifiers

ooObjectQualifier Class Renamed and Moved

The `ooObjectQualifier` class has been renamed to `ObjectQualifier`, and has been moved from the `com.objy.db.app` package to `com.objy.query.ObjectQualification`.

Constructor Overloads That Accept Classname String

The following new overloads of the `ObjectQualifier` class constructor enable you to use a string (instead of the class's type number) to specify the name of the class of objects to be qualified:

- `com.objy.query.ObjectQualifier(String, String)`
- `com.objy.query.ObjectQualifier(String, Expression)`

PQL Variables in Object Qualifiers

You can use a PQL variable inside a predicate string in an object qualifier. This lets you reuse the same object qualifier after substituting different literal values for the variable.

For applications that perform large numbers of queries where the predicate need only differ by the values of literals, using an object qualifier with a PQL variable is more efficient than repeated scan operations with different PQL expressions. See the *Objectivity for Java Programmer's Guide* for more information about PQL variables and the types of literal values they can represent.

scan and parallelScan Overloads That Accept Object Qualifiers

You can now use an object qualifier in a `scan` or `parallelScan` operation. The following new methods support this:

- In `com.objy.db.app`:
  - `ooFDObj.scan(String, ObjectQualifier)`
  - `ooFDObj.parallelScan(String, ObjectQualifier)`
  - `ooFDObj.parallelScan(`
`String, ObjectQualifier, QuerySplitter)`
  - `ToManyRelationship.scan(ObjectQualifier)`
- In `com.objy.db.app.storage`:
  - `ooContObj.scan(String, ObjectQualifier)`
  - `ooDBObj.scan(String, ObjectQualifier)`
  - `ooDBObj.parallelScan(String, ObjectQualifier)`
  - `ooDBObj.parallelScan(`
`String, ObjectQualifier, QuerySplitter)`

Superseded Features

The following subsections describe superseded Objectivity for Java features.

Superseded features are still supported in this release, but should not be used in new applications.

Explicit Placement Mechanism in Objectivity for Java

The classes and methods in the table below are used only for performing explicit placement of new persistent objects. These items are superseded by the managed object-placement system (see page 15), and so should not be used by an application that accesses a placement-managed federated database.

These items have been noted the *Objectivity for Java Programmer's Reference*, and are still supported for backward compatibility with applications that access existing federated databases.

In general, the superseded items are classes and interfaces in the `com.objy.db.app.storage` package, plus selected methods and constants in other packages.

Category	Superseded Items
Clustering	`com.objy.db.app.storage.ClusterStrategy` class `com.objy.db.app.storage.DefaultClusterStrategy` class `com.objy.db.app.storage.ClusterReason` interface `com.objy.db.app.ooObj.cluster` method overloads `com.objy.db.iapp.IooObj.cluster` method overloads `com.objy.db.iapp.PooObj.cluster` method overloads `com.objy.db.app.Session.getClusterStrategy` method `com.objy.db.app.Session.setClusterStrategy` method `com.objy.db.app.Session.requestCluster` method
Copying and Moving Persistent Objects	`com.objy.db.app.ooObj.copy` method `com.objy.db.iapp.IooObj.copy` method `com.objy.db.app.ooObj.move` method `com.objy.db.iapp.IooObj.move` method `com.objy.db.Relationship.COPY_COPY` constant `com.objy.db.Relationship.COPY_DELETE` constant `com.objy.db.Relationship.COPY_MOVE` constant
Versioning Persistent Objects	`com.objy.db.Relationship.VERSION_COPY` constant `com.objy.db.Relationship.VERSION_DELETE` constant `com.objy.db.Relationship.VERSION_MOVE` constant
Containers	`com.objy.db.app.storage.ooContObj` class `com.objy.db.app.storage.ooGCContObj` class `com.objy.db.app.storage.ooDefaultContObj` class `com.objy.db.app.storage.ooGCRootsContObj` class `com.objy.db.app.ooObj.getContainer` method `com.objy.db.iapp.ooObj.getContainer` method

Category	Superseded Items
Databases and Database Images	com.objy.db.app.storage.ooDBObj class com.objy.db.app.ooFDObj.lookupDB method overloads com.objy.db.app.ooFDObj.newDB method overloads com.objy.db.app.ooFDObj.containedDBs method com.objy.db.app.ooFDObj.hasDB method overloads com.objy.db.app.ooFDObj.getDefaultDB method com.objy.app.oo.UNAVAILABLE constant com.objy.app.oo.NON_QUORUM_READ_AVAILABLE constant com.objy.app.oo.QUORUM_AVAILABLE constant com.objy.app.oo.ALL_AVAILABLE constant com.objy.db.app.Session.setAllowNonQuorumRead method
Autonomous Partitions	com.objy.db.app.storage.ooAPObj class com.objy.db.app.ooFDObj.hasAP method com.objy.db.app.ooFDObj.lookupAP method com.objy.db.app.ooFDObj.newAP method overloads com.objy.db.app.ooFDObj.containedAPs method com.objy.db.app.ooFDObj.getBootAP method com.objy.app.oo.ENFORCE constant com.objy.app.oo.IGNORE constant com.objy.db.app.Session.getOfflineMode method com.objy.db.app.Session.setOfflineMode method
Indexes	com.objy.db.app.ooFDObj.addIndex method com.objy.db.app.ooFDObj.addUniqueIndex method com.objy.db.app.ooFDObj.dropIndex method com.objy.db.app.ooFDObj.hasIndex method com.objy.db.app.ooFDObj.indexConsistent method
Short Object Identifiers	com.objy.db.app.Relationship.INLINE_SHORT constant com.objy.db.util.ooTreeListX(...boolean *useShortRefs*...) com.objy.db.util.ooTreeSetX(...boolean *useShortRefs*...) com.objy.db.util.ooTreeMapX(...boolean *useShortRefs*...)
Other	com.objy.db.app.QuerySplitter interface com.objy.db.app.ooFDObj.parallelScan(...QuerySplitter *qs*...) com.objy.db.app.ooFDObj.convertObjects(boolean *purge_schema*) com.objy.db.app.ooFDObj.upgradeObjects(boolean *purge_schema*)

Deprecated Features (Removed in a Future Release)

The following subsections describe deprecated Objectivity for Java features.

Session Methods for Handling Transient Relationships

You should no longer use the following methods of `com.objy.db.app.Session` to control whether transient objects become persistent or remain transient when they are linked in relationships:

- `setFormTransientRelationships(boolean)`
- `getFormTransientRelationships()`
- `setAllowTransientRelationships(boolean)`
- `persistTransientRelationships()`
- `persistTransientRelationships(java.lang.Object, com.objy.db.app.ClusterReason)`
- `setPersistUnreachableTransientsWithRelationshipToPersistent(boolean)`
- `getPersistUnreachableTransientsWithRelationshipToPersistent()`

If any session in your application depends on any of these methods, you should change your code in one of the following ways:

- Add a call to the connection's `setPersistencePolicy` method, passing the value `oo.PreRelease11Behavior`. See "Connection Method for Handling Transient Relationships" on page 41.
- Remove the deprecated methods, and change your code so that the reachability policy produces the desired results; see "Adapting to the New Objectivity for Java Reachability Policy" on page 64.

Obsolete Features (Removed in This Release)

The following subsections describe obsolete Objectivity for Java features.

DefaultQuerySplitter

The `com.objy.db.app.DefaultQuerySplitter` class has been removed. Its functionality has been replaced by a built-in default query splitter. You should remove any code that references an instance of `DefaultQuerySplitter`.

Parallel Queries with filterName Parameter

The following overloads of `parallelScan` have been removed in this release:

- `com.objy.db.app.ooFDObj.parallelScan(`
  `String, String, QuerySplitter, String)`
- `com.objy.db.app.storage.ooDBObj.parallelScan(`
  `String, String, QuerySplitter, String)`

You should instead use the following new overloads, from which the obsolete final `String` parameter has been removed:

- `com.objy.db.app.ooFDObj.parallelScan(`
  `String, String, QuerySplitter)`
- `com.objy.db.app.storage.ooDBObj.parallelScan(`
  `String, String, QuerySplitter)`

Objectivity for Java (Active Schema)

This section describes the impact of Release 11.0 on the Objectivity for Java interface to Active Schema.

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

Superseded Features

The following subsections describe superseded Active Schema features.

Superseded features are still supported in this release, but should not be used in new applications.

Short Object Identifiers and Short Object References

You should no longer use classes and members that create attributes and relationships that store short object identifiers or short object references. These items are still in the documentation, but they should not be used in applications that access a placement-managed federated database.

5

.NET Product

This chapter describes an overview of Release 11.0 additions and changes to Objectivity/.NET for C#.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity/.NET Class Library

This section describes summarizes the impact of Release 11.0 on the Objectivity/.NET class library.

For a complete description of new and changed features in the class library, see the updated *Objectivity/.NET for C# Programmer's Reference*, 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

New Features

The following subsections describe new Objectivity/.NET features.

New Namespaces

Selected Objectivity/.NET classes are now declared in one of the following new namespaces, which are nested within the existing `Objectivity.Db` namespace:

Namespace Within Objectivity.Db	Description	See
`Collections.Specialized`	Contains classes that represent storable collections and iterators over their elements.	page 50
`Sessions`	Contains classes and extension methods that provide direct access to Objectivity/DB sessions.	page 51
`Storage`	Contains types that support explicit placement of new persistent objects.	page 52

Managed Placement in Objectivity/.NET

Making Objects Persistent

Persistence-capable classes now define one or more new constructor overloads that accept a placement-conditions object (page 49) as a parameter. You must use one of these overloads in an application that creates a new persistent object in a placement-managed federated database (page 15).

New overloads are defined for every persistence-capable class, including:

- The `Objectivity.Db.ReferenceableObject` class.
- Every generated application-defined class *`appclass`*.
- The classes in the `Objectivity.Db.Collections.Specialized` namespace: `HashSetX<T>`, `HashMapX<T>`, `Map<T>`, `TreeListX<T>`, `TreeSetX<T>`, `TreeMapX<T>`.
- The `Objectivity.Versioning.Genealogy` class.

For example, the `Objectivity.Db.ReferenceableObject` class has the following new constructor:

- `ReferenceableObject(PlacementConditions)`

NOTE Existing constructor overloads (with parameters of type `IStorable` and/or `ClusterStrategy`) should be used only for creating persistent objects in a non-placement-managed federated database.

Specifying Placement Conditions

You use *placement-conditions object* to encapsulate application-specific information to be used when placing a new persistent object in a placement-managed federated database (page 15). A placement-conditions object is an instance of the following new class:

- `Objectivity.Db.PlacementConditions`

You pass the placement-conditions object as a parameter to the new constructor overloads defined by persistence-capable classes; see "Making Objects Persistent" on page 48.

The federated database's placement model determines whether and how the information encapsulated in the placement-conditions object is used.

Enabling Location Preferences in XML Configuration Files

You can call the following new static method overloads to enable your application to obtain storage-location preferences from one or more XML configuration files:

- `Objectivity.Db.Objy.EnableConfiguration()`
- `Objectivity.Db.Objy.EnableConfiguration(bool, string)`
- `Objectivity.Db.Objy.EnableConfiguration(bool, string, string[])`

Storage-location preferences are a way of specifying preferred storage locations for files added to a placement-managed federated database; see "Storage-Location Preferences in XML Configuration Files" on page 18.

NOTE The configuration file(s) for specifying storage-location preferences are *adjunct configuration files* that supplement the application's configuration file, which is supported by the .NET configuration system.

Adjunct configuration files must be enabled explicitly by a call to one of the new method overloads, and cannot be accessed from within the application itself. In Release 11.0, these configuration files are limited to settings that specify location preferences.

In contrast, the application configuration file is enabled automatically when the application starts, and its `<objectivity>` section can be accessed by the application using classes in the `Objectivity.Configuration` namespace. Settings in the `<objectivity>` section enable you to control application-wide policies, specify the federation(s) to be accessed, and configure sessions.

Documentation

The *Objectivity/.NET for C#* reference documentation contains descriptions of new and reorganized items, but most descriptions apply to applications that access non-placement-managed federated databases.

Getting Started with Objectivity/.NET for C# shows a sample application that accesses a placement-managed federated database.

Refreshing a Container in an MROW Transaction

You can now test whether an MROW transaction's view of a container is out-of-date by accessing the following new property of an object in the container:

- `Objectivity.Db.IReferenceableObject.IsContainerUpdated`

The result is true if the container was opened by a concurrent update transaction that committed while your transaction had the MROW lock on the container.

The MROW transaction can refresh its view of an out-of-date container by calling the following new method on any object in the container:

- `Objectivity.Db.IReferenceableObject.RefreshOpenContainer( OpenMode, Boolean)`
- `Objectivity.Db.IReferenceableObject.RefreshOpenContainer( OpenMode, Boolean, Boolean)`

Testing Whether a Session is Disposed

You can now test whether a session is disposed by accessing the following new property:

- `Objectivity.Db.Sessions.Session.IsDisposed`

Changed Features

The following subsections describe changes to Objectivity/.NET features.

Reorganized Support for Storable Collections

The following classes are now declared in the new namespace `Objectivity.Db.Collections.Specialized`:

- `CollectionIterator<T>`
- `CollectionPairIterator<T>`
- `Comparator<T>`
- `HashBasedCollection<T>`
- `HashMapX<TKey, TValue>`

- HashSetX<T>
- Map<T>
- StorableCollection<T>
- TreeBasedCollection<T>
- TreeListX<T>
- TreeMapX<TKey, TValue>
- TreeSetX<T>

In previous releases, these classes were declared in the Objectivity.Db namespace.

To use any of the storable-collection classes, you:

➤ Add the following directive to your code:
```
using Objectivity.Db.Collections.Specialized;
```

Reorganized Support for Direct Session Access

Session Class Moved to New Namespace

The Session class is now declared in the new namespace Objectivity.Db.Sessions.

In previous releases, the Session class was declared in the Objectivity.Db namespace.

ObjyConnection.Session Replaced by Extension Method

The following property is now replaced by an extension method:

- Objectivity.Db.ObjyConnection.Session

To obtain a connection's session, you:

1. Add the following directive to your code:
```
using Objectivity.Db.Sessions;
```
2. Call the following method on the ObjyConnection object:
   - GetSession()

Reorganized Support for Explicit Placement

The Objectivity/.NET class library has reorganized many of the types and members that are used for performing explicit placement of new persistent objects in a non-placement-managed federated database. The reorganization distinguishes these items from the those that can be used with the managed object-placement system (see page 15).

The reorganized items are part of the superseded explicit-placement mechanism (see page 53).

Types Moved to Objectivity.Db.Storage Namespace

The following types are now declared in the new `Objectivity.Db.Storage` namespace:

- `ClusterPriorities`
- `ClusterStrategy`
- `Container`
- `Database`
- `KeyDescriptor`
- `KeyField`
- `ObjectPlacementFailedException`
- `OutOfContainerIDsException`

In previous releases, these types were declared in the `Objectivity.Db` namespace.

To continue using these types, you:

➤ Add the following directive to your code:

```
using Objectivity.Db.Storage;
```

Members Replaced by Extension Methods

The properties and methods in the following table are now replaced by extension methods declared in the new `Objectivity.Db.Storage` namespace:

Original Member	New Extension Method
`Objectivity.Db.IReferenceableObject` interface `Objectivity.Db.ReferenceableObject` class	
`ContainedIn` property	`ContainedIn()` method
`Copy(IStorable)` method	`Copy(IStorable)` method
`GetScopeName(Container)` method	`GetScopeName(Container)` method
`GetScopeName(Database)` method	`GetScopeName(Database)` method
`Move(Container)` method	`Move(Container)` method
`Objectivity.Db.Federation` class	
`Databases` property	`GetDatabases()` method

Original Member	New Extension Method
`HasDatabase(String)` method	`HasDatabase(String)` method
`LookUpDatabase(String)` method	`LookUpDatabase(String)` method
`Objectivity.Db.Sessions.Session` class	
`ClusterStrategy` property	`GetClusterStrategy()` method `SetClusterStrategy(ClusterStrategy)` method

To use a new extension method, you:

1. Add the following directive to your code:
   ```
   using Objectivity.Db.Storage;
   ```
2. Call the extension method on an instance of the appropriate class.

For example, the following code fragment iterates over the databases in a federated database `fd`:

```
using Objectivity.Db.Storage;
foreach ( Database d in fd.GetDatabases() )  { ... }
```

Renamed Method on Objectivity/.NET Classes

For consistency with .NET naming conventions, the `createProxyForId()` method of various Objectivity/.NET classes has been renamed:

- `CreateProxyForId()`

Superseded Features

The following subsections describe superseded Objectivity/.NET features.

Superseded features are still supported in this release, but should not be used in new applications.

Explicit Placement Mechanism in Objectivity/.NET

The types and members in the following table are used only for performing explicit placement of new persistent objects. These items are superseded by the managed object-placement system (see page 15), and so should not be used by an application that accesses a placement-managed federated database.

Many of these items are reorganized into different namespaces or have been replaced by extension methods (see page 51).

These items are described from the Objectivity/.NET documentation, although they are supported only for backward compatibility with applications that access existing federated databases.

Category	Superseded Items

Clustering	`Objectivity.Db.Storage` namespace
	`ClusterStrategy` class
	`ClusterPriorities` enumeration
	`GetClusterStrategy()` extension method on `Session`
	`SetClusterStrategy(ClusterStrategy)` extension method on `Session`
	
	Constructor overloads with *`nearObj`* and/or *`clusterStrategy`* parameters

Copying and Moving Persistent Objects	`Objectivity.Db.Storage` namespace
	`Copy()` extension method on `ReferenceableObject`
	`Move(Container)` extension method on `ReferenceableObject`

Versioning Persistent Objects	`Objectivity.Versioning` namespace `Genealogy` class `VersionMode` enumeration Extension methods on `ReferenceableObject`

Containers	`Objectivity.Db.Storage` namespace
	`Container` class
	`ContainedIn()` extension method on `ReferenceableObject`
	`GetScopeName(Container)` extension method on `ReferenceableObject`

Databases	`Objectivity.Db.Storage` namespace
	`Database` class
	`GetScopeName(Database)` extension method on `ReferenceableObject`
	`GetDatabases()` extension method on `Federation`
	`HasDatabase(String)` extension method on `Federation`
	`LookUpDatabase(string)` extension method on `Federation`

Autonomous Partitions and Database Images	`Objectivity.Db.HA` namespace
	`AttemptToAccessOfflinePartitionException`
	`Availability` type
	`DatabaseExtensionMethods` class
	`DatabaseImage` class
	`DatabaseImageIterator` class
	`FederationExtensionMethods` class
	`NoQuorumException`
	`OfflineMode` type
	`Partition` class
	`PartitionIterator` class
	`SessionExtensionMethods` class

Category	Superseded Items
Short Object Identifiers	In the `Objectivity.Db.Collections.Specialized` namespace: `TreeListX` constructor overloads with *`useShortRefs`* parameter `TreeSetX` constructor overloads with *`useShortRefs`* parameter `TreeMapX` constructor overloads with *`useShortRefs`* parameter
Indexes	`Objectivity.Db.Storage` namespace `KeyDescriptor` class `KeyField` class In the `Objectivity.Configuration` namespace `ISessionSettings.UseIndexes` property `SessionProfile.UseIndexes` property `SessionTemplate.UseIndexes` property

Exceptions	`Objectivity.Db.HA` namespace
	`AttemptToAccessOfflinePartitionException`
	`NoQuorumException`
	`Objectivity.Db.Storage` namespace
	`ObjectPlacementFailedException`
	`OutOfContainerIDsException`

Other	In the `Objectivity.Db` namespace: `Federation.UpgradeObjects` with *`purgeSchema`* set to true

Obsolete Features (Removed in This Release)

The following subsections describe deprecated Objectivity/.NET features.

Method for Moving a Collection Element

The `MoveCurrentTo(IStorable)` method of `Objectivity.Db.Collections.Specialized.CollectionIterator<T>` is no longer supported.

Objectivity/DB Persistence Designer

This section summarizes the impact of Release 11.0 on the Objectivity/DB Persistence Designer, which is delivered as a plug-in to Microsoft Visual Studio.

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`

Changed Features

The following subsections describe changes to Persistence Designer features.

Creating Federated Databases With the Persistence Designer

Using the Persistence Designer to create a federated database is now equivalent to running the `CreateFd` command; see page 22. That is, the Persistence Designer creates only placement-managed federated databases; see page 15.

Superseded Features

The following subsections describe superseded Persistence Designer features.

Superseded features are still supported in this release, but should not be used in new applications.

Short Reference Size

Short object references should be avoided as attribute values in objects that are stored in a placement-managed federated database; see page 15. Therefore, when you are using the Persistence Designer to design persistence-capable classes, you should make sure the **Reference size** property of the following attributes is set to **Long**:

- Reference attributes
- Inline relationships
- Ordered storable-collection attributes

Short object references are safe to use *only* with appropriate object placement, and should be considered only after obtaining technical consultation from Objectivity.

6

SQL Products

This chapter provides an overview of the Release 11.0 additions and changes to Objectivity/SQL++ and Objectivity/SQL++ ODBC Driver products.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity/SQL++

Objectivity/SQL++ has no new features in this release.

NOTE Objectivity/SQL++ provides access to data in *non-placement-managed* federated databases only.

Objectivity/SQL++ ODBC Driver

Objectivity/SQL++ ODBC Driver (Objectivity/ODBC) has no new features in this release.

NOTE Objectivity/ODBC provides access to data in *non-placement-managed* federated databases only.

7

Python Product

This chapter provides an overview of Release 11.0 additions and changes to Objectivity/Python.

See the Objectivity Developer Network for software or documentation problems that have been fixed in this release. Call Objectivity Customer Support to obtain access to this information, if you do not already have an account.

Objectivity/Python

Objectivity/Python has no new features in this release.

NOTE Objectivity/Python can be used to inspect data and perform queries in a placement-managed federated database (page 15). For information about using this product to create objects in placement-managed federated database, you can obtain technical consultation from Objectivity.

8

Release Compatibility

This chapter provides information about the impact, if any, of using Release 11.0 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 "Updating the Objectivity License" 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 62 if you plan to rebuild existing applications with the current release.
- See "Maintaining Earlier Objectivity/DB Applications" on page 69 if you need to continue using unrebuilt tools or applications from an earlier release.

NOTE Because Release 11.0 is a major release, you must upgrade any existing applications that are to access the data in a new Release 11.0 federated database.

Updating the Objectivity License

A Release 10.*x* federated database must have an Objectivity Release 11 license to authorize access by:

- Tools provided with Objectivity/DB Release 11.0
- New applications built with Objectivity/DB Release 11.0
- Existing applications that have been upgraded, recompiled, and relinked with Objectivity/DB Release 11.0 (see "Upgrading Existing Applications and Scripts" on page 62).

NOTE A Release 10.*x* federated database is non-placement-managed, and so may be accessed only with applications that use explicit placement (page 15).

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 11 license.

To update the Objectivity license for Release 10.x federated databases

1. Verify that you have set up a default license file containing your Objectivity Release 11 license.

   Note: If you did not set up a default Objectivity license file during product installation, see "Setting Up a License File" in Chapter 4 of *Objectivity/DB Administration*.

2. For each Release 10.*x* federated database to be accessed with Release 11.0 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.

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 against the latest headers and relink it with Release 11.0 libraries.

When planning whether to upgrade existing applications to Release 11.0, you should take into account any required code changes listed in the following subsections.

- Upgrading Tool Scripts
- Upgrading Objectivity/C++ Applications
- Upgrading C++ Active Schema Applications
- Upgrading Objectivity for Java Applications
- Upgrading Java Active Schema Applications
- Upgrading Objectivity/.NET for C# Applications

NOTE These subsections describe only the changes introduced in Release 11.0. For descriptions of code and script changes introduced in an earlier release, see the *Objectivity Release Notes* for that release on the Objectivity Developer Network.

Upgrading Tool Scripts

Use the following list to determine whether you must rewrite portions of existing scripts to accommodate Release 11.0 changes to Objectivity/DB tools.

- Adjust any backup and restore scripts so that they invoke the `oobackup` and `oorestore` tools, instead of `oobackupx` and `oorestorex`. See "Performing and Restoring Backups" on page 24.
- Review any tool script that you intend to use with placement-managed federated databases. You may need to revise the script to eliminate tools listed in "Tools Supporting Explicit Placement Mechanism" on page 26. Tool scripts for creating placement-managed federated databases should run the new `CreateFd` tool instead of `oonewfd`; see "CreateFd" on page 22.
- (Recommended) Review any tool script that assigns an identifier to a federated database, to see whether such assignment is affected by the changes described in "Unique Identifiers for Federated Databases" on page 25. In some cases, you may be able to simply remove the `-fdnumber` option.

Upgrading Your Objectivity/C++ Build Process

Use the following list to determine whether you must update your build process to accommodate Release 11.0 changes to libraries.

- (UNIX Solaris only) Adjust your build process to accommodate the changes in "Replaced C++ Library on UNIX Solaris Architectures" on page 31. In particular, you need to remove any dependencies on `libCstd`, which has been replaced by `libstlport4`.
- Adjust your build process as necessary to accommodate changes to the names of shared libraries; see "Updated Names for Objectivity Shared Library" on page 31.

Upgrading Objectivity/C++ Applications

Use the following list to determine whether you must rewrite portions of existing C++ applications to accommodate Release 11.0 changes in Objectivity/C++.

- Adjust any code that uses an object qualifier, to accommodate the changes in "ooObjectQualifier Class Renamed and Relocated" on page 32. In particular, you must:
  - Change the class name to `ObjectQualifier`, and you must either qualify the class name or add the following directive to your code:

    ```
    using namespace obj::query;
    ```

- Change:
  `#include <ooObjectQualifier.h>`
  to:
  `#include <objy/query/ObjectQualifier.h>`

- Decide whether any existing application is to access a placement-managed federated database. If so, change the application as described in "Changing From Explicit to Managed Placement" on page 67.

Upgrading C++ Active Schema Applications

Use the following list to determine whether you must rewrite portions of existing C++ applications to accommodate Release 11.0 changes in Active Schema.

- Decide whether any existing Active Schema application is to access a placement-managed federated database. If so, eliminate any operations that create attributes or relationships that store short object references.

Upgrading Objectivity for Java Applications

Use the following list to determine whether you must rewrite portions of existing Java applications to accommodate Release 11.0 changes in Objectivity for Java.

- Adjust any code that uses an object qualifier, to accommodate the changes in "ooObjectQualifier Class Renamed and Moved" on page 41. In particular, you must change the class name to `ObjectQualifier`, and you must import types from the new `com.objy.query` package.
- Read "Adapting to the New Objectivity for Java Reachability Policy" below to determine whether your Objectivity for Java application will be affected by the new default policy for handling transient relationships. If it is, you can use the temporary workaround or perform the recommended fix described in that section.
- Decide whether any existing application is to access a placement-managed federated database.
  - If so, change the application as described in "Changing From Explicit to Managed Placement" on page 67.
  - If not, enable the application to continue using explicit placement by accommodating the changes in "Reorganized Support for Explicit Placement" on page 39. In particular, you must import types from the new `com.objy.db.app.storage` package.
- Replace or revise any code for obsolete features listed on page 45.

Adapting to the New Objectivity for Java Reachability Policy

You may need to upgrade an existing Objectivity for Java application to accommodate the default reachability policy described in "New Default Policy

for Handling Transient Relationships" on page 40, or to remove calls to the deprecated methods listed on page 45.

No upgrade is necessary in either of the following cases:

- The application never forms any relationships with transient objects. You can optionally remove any calls to the `setFormTransientRelationships` or `setAllowTransientRelationships` methods.
- The application calls `setFormTransientRelationships(true)` in every session that forms relationships with transient objects. You can optionally remove those calls.

The following subsections describe cases that require an upgrade.

Upgrade for Code Relying on Auto-Persistence in Relationships

You must upgrade an application if it relies on the pre-Release 11.0 standard relationship-persistence policy to persist unreachable transient objects in relationships.

Specifically, you must change your code if all of the following are true for at least one session in the application:

- The session calls the deprecated method `setFormTransientRelationships(false)` *or* omits any call to that method.
- The session forms a unidirectional relationship from a transient source object *or* forms a bidirectional relationship between two transient objects.
- The application performs operations that assume the transient objects in these relationships were automatically made persistent.

Temporary Workaround. Add a call such as the following to the beginning of the application, and leave all calls to the deprecated method as is:

```
connection.setPersistencePolicy(oo.PreRelease11Behavior);
```

Recommended Upgrade. Remove any calls to the deprecated method, and arrange for the transient objects in question to be made persistent some other way. For example, cluster these objects explicitly after forming the relationships, or make them reachable from some other existing persistent object.

Upgrade for Code Persisting All Transient Objects on Demand

You must change your code if all of the following are true for at least one session in the application:

- The session calls `setFormTransientRelationships(true)` to allow transient objects to be linked through relationships.

- The session calls the deprecated `persistentTransientRelationships` method with a specific clustering object.

  Note: This affects any transient object in the session, not just those permitted by the transient-relationships policy.

Temporary Workaround. Add a call such as the following to the beginning of the application, and leave all calls to deprecated methods as is:

```
connection.setPersistencePolicy(oo.PreRelease11Behavior);
```

Recommended Upgrade. Remove any calls to deprecated methods, and arrange for the transient objects in question to be made persistent some other way. For example, cluster these objects explicitly after forming the relationships, or make them reachable from an existing persistent object.

Upgrading Java Active Schema Applications

Use the following list to determine whether you must rewrite portions of existing Java applications to accommodate Release 11.0 changes in Active Schema.

- Decide whether any existing Active Schema application is to access a placement-managed federated database. If so, eliminate any operations that create attributes or relationships that store short object references.

Upgrading Objectivity/.NET for C# Applications

Use the following list to determine whether you must rewrite portions of existing .NET/C# applications to accommodate Release 11.0 changes in Objectivity/.NET for C#.

- Adjust any code that works with storable collections, to accommodate the changes in "Reorganized Support for Storable Collections" on page 50. In particular, you must add a directive to use the types from the new `Objectivity.Db.Collections.Specialized` namespace.
- Adjust any code that works with sessions directly, to accommodate the changes in "Reorganized Support for Direct Session Access" on page 51. In particular, you must add a directive to use the types from the new `Objectivity.Db.Sessions` namespace, and replace a property with a new extension method.
- Adjust any code that calls `createProxyForId()`; see "Renamed Method on Objectivity/.NET Classes" on page 53.
- Decide whether any existing application is to access a placement-managed federated database.
  - If so, change the application as described in "Changing From Explicit to Managed Placement" on page 67.
  - If not, enable the application to continue using explicit placement by accommodating the changes in "Reorganized Support for Explicit

Placement" on page 51. In particular, you must add a directive to use the types from the new `Objectivity.Db.Storage` namespace, and replace certain properties with new extension methods.

- Replace or revise any code for obsolete features listed on page 55.

Changing From Explicit to Managed Placement

Managed placement of persistent objects (page 15) is recommended for all new Objectivity/DB applications. You can also consider changing existing pre-Release 11.0 applications so that they use managed placement instead of explicit placement mechanisms.

NOTE Applications that use managed placement can create and access data only in (new) placement-managed federated databases, and are incompatible with non-placement-managed federated databases.

The following subsections provide an overview of what's involved in changing an application from explicit to managed placement.

NOTE Although you can change an existing application to use managed placement, you cannot currently migrate existing data to a placement-managed federated database. (This capability is planned for a future release.) You must therefore run your converted application to re-create your data in a new placement-managed federated database.

Preparing a New Federated Database

An application that uses managed placement relies on the federated database's placement model.

1. Create a new, empty placement-managed federated database (page 22).
2. Design and install a placement model that is appropriate for your application.

   For tutorial information, see *Getting Started With Managed Object Placement* in *`installDir`*`/doc/placementManagement.pdf`
3. Use administrative tools to specify file-storage locations for database files (page 18) and to create any desired indexes (page 19).

Changing the Application

Changing an application to use managed placement generally involves simplifying it; see "Consequences for application development" on page 17. To change an application to use managed placement, perform these general steps:

- Remove any code that creates, finds, and manages:
  - Containers or databases
  - Clustering strategies
  - Indexes
  - Autonomous partitions and database images
- Change the scope of scan operations, iteration operations, parallel queries, and scope-naming operations to be the federated database.
- Adjust the code that creates each persistent object:
  - Make the creation code consistent with the placement model you created above. For example, consider passing the same `nearObj` if the applicable placement rule expects an existing persistent object to be specified.
  - Revise the creation code if it clusters with a container or a database, or explicitly specifies a clustering strategy.
  - In Objectivity for Java, use a `persist` method instead of a `cluster` method; see "Making Objects Persistent" on page 37.
  - In Objectivity/.NET for C#, use one of the new constructor overloads that accepts a placement-conditions object; see "Making Objects Persistent" on page 48.
- Eliminate the use of short identifiers or short object references.
  - If any ordered collections are created to use short object references (identifiers) to link to their elements, change the creation code so that standard object references (identifiers) are used instead.
  - Adjust the definitions of persistence-capable classes to change associations (relationships) from *short inline* to *inline*.
  - In Objectivity/C++, adjust the definitions of persistence-capable classes to change reference attributes from `ooShortRef` types to `ooRef` types.

For a list of specific types and methods to remove, see:

- "Explicit Placement Mechanism in Objectivity/C++" on page 32
- "Explicit Placement Mechanism in Objectivity for Java" on page 42
- "Explicit Placement Mechanism in Objectivity/.NET" on page 53

NOTE After you build the changed application, run it to populate the prepared placement-managed federated database.

Maintaining Earlier Objectivity/DB Applications

After installing Objectivity/DB Release 11.0 along with your chosen Objectivity programming interface, you normally:

- Develop new Release 11.0 applications.
- Upgrade existing applications and then recompile and relink them with Release 11.0; 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 11.0) Objectivity/DB release.

Maintaining Unrebuilt Release 10.*x* Applications

Unrebuilt Release 10.*x* applications are not guaranteed to work with either type of Release 11.0 federated database (placement-managed or non-placement-managed). It is strongly recommended that you upgrade, recompile, and relink existing applications that are to access new Release 11.0 federated databases.

An unrebuilt Release 10.*x* application can interoperate with new or upgraded Release 11.0 applications only if all applications are accessing existing pre-Release 11.0 federated databases. (The new or upgraded applications must all use the explicit-placement mechanism because existing federated databases are non-placement-managed.)

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