OBJECTSTORE

C++ INTERFACE
RELEASE NOTES

RELEASE 5.1
FOR ALL PLATFORMS

March 1998

ObjectStore C++ Interface Release Notes

ObjectStore Release 5.1 for all platforms, March 1998

Object Design, Inc.
Twenty Five Mall Road
Burlington, MA 01803-4194

Part number: SW-OS-DOC-RNO-510

Contents

Preface

Purpose

The *ObjectStore C++ Interface Release Notes* describe the features and functions included in ObjectStore Release 5.1 that are new or have changed since the previous release.

Audience

This book is for administrators or developers responsible for the installation and maintenance of ObjectStore. It is assumed that you are familiar with the ObjectStore host platform and comfortable using the operating system.

Scope

In conjunction with *ObjectStore Installation and License for Solaris*, this document provides information for installing and running the ObjectStore Release 5.1 software.

How This Book Is Organized

The first chapter summarizes the platforms and compilers supported by ObjectStore Release 5.1. Chapter 1, New in Release 5.1, on page 1, briefly describes new features and provides cross-references to the ObjectStore documentation containing detailed information about each new feature or interface. Chapter 2, Changes and Additions to Existing Features, on page 23, highlights general release considerations that affect all platforms. Chapter 3, Platform-Specific Considerations, on page 35, describes platform-related considerations you should anticipate when using ObjectStore Release 5.1. Chapter 5, Compiler-Specific Considerations, on page 65, describes considerations peculiar to specific supported compilers.

Notation Conventions

This document uses the following conventions:

Convention	*Meaning*
Bold	Bold typeface indicates user input or code.
Sans serif	Sans serif typeface indicates system output.
Italic sans serif	Italic sans serif typeface indicates a variable for which you must supply a value. This most often appears in a syntax line or table.
Italic serif	In text, italic serif typeface indicates the first use of an important term.
[ ]	Brackets enclose optional arguments.
{ *a* | *b* | *c* }	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify *a* or *b* or *c*.
...	Three consecutive periods indicate that you can repeat the immediately previous item. In examples, they also indicate omissions.

ObjectStore Documentation

ObjectStore documentation is chiefly distributed on-line in web-browsable format. If you want to order printed books, contact your Object Design sales representative.

Internet Sources of More Information

World Wide Web

Object Design's support organization provides a number of information resources. These are available to you through a web browser such as Internet Explorer or Netscape. You can obtain information by accessing the Object Design home page with the URL **http://www.objectdesign.com**. Select **Technical Support**. Select **Support Communications** for detailed instructions about different methods of obtaining information from support.

Internet gateway

You can obtain information such as FAQs (answers to frequently asked questions) from Object Design's Internet gateway machine as well as from the web. This machine is called

ftp.objectdesign.com and its Internet address is 198.3.16.26. You can use **ftp** to retrieve the FAQs from there. Use the login name **odiftp** and the password obtained from **patch-info**. This password also changes monthly, but you can automatically receive the updated password by subscribing to **patch-info**. See the **README** file for guidelines for using this connection. The FAQs are in the subdirectory **./FAQ**. This directory contains a group of subdirectories organized by topic. The file **./FAQ/FAQ.tar.Z** is a compressed **tar** version of this hierarchy that you can download.

Automatic email notification

In addition to the previous methods of obtaining Object Design's latest patch updates (available on the **ftp** server as well as the Object Design Support home page) you can now automatically be notified of updates. To subscribe, send email to **patch-info-request@objectdesign.com** with the keyword **SUBSCRIBE patch-info** <*your siteid*> in the body of your email. This will subscribe you to Object Design's patch information server daemon that automatically provides site access information and notification of other changes to the on-line support services. Your site ID is listed on any shipment from Object Design, or you can contact your Object Design Sales Administrator for the site ID information.

Training

If you are in North America, for information about Object Design's educational offerings, or to order additional documents, call 781.674.5000, Monday through Friday from 8:30 AM to 5:30 PM Eastern Time. You can reach the Education Hotline at 781.674.5047.

If you are outside North America, call your Object Design sales representative.

Your Comments

Object Design welcomes your comments about ObjectStore documentation. Send your feedback to **support@objectdesign.com**. To expedite your message, begin the subject with **Doc:**. For example:

Subject: Doc: Incorrect message on page 76 of reference manual

You can also fax your comments to 781.674.5440.

Chapter 1
New in Release 5.1

The information in this and succeeding chapters is intended for use by sites upgrading from ObjectStore Release 5.*x* to Release 5.1. If you are upgrading from an earlier release than ObjectStore Release 5, read the *ObjectStore C++ Interface Release Notes* supporting that upgrade for pertinent information.

The new features included in ObjectStore Release 5.1 expand product capabilities in several important directions. This chapter summarizes the new features in the following order:

About This Release

ObjectStore is an *object-oriented database management system* suited for rapid application development and deployment in multitiered environments. It combines the data query and management capabilities of a traditional database with the flexibility and power of C++ and Java interfaces on all platforms. Additionally, ObjectStore for Windows offers support for the ActiveX interface.

This chapter provides general information about the release. Specific distributions of ObjectStore Release 5.1 for various platforms can be found in the **README.txt** file in the ObjectStore root directory. Solaris SPARC and Windows are supported for the release of ObjectStore Release 5.1.

Be sure to read all of the *ObjectStore C++ Interface Release Notes* before beginning the installation.

Product Modules

ObjectStore Release 5.1 comprises the C++ and Java interfaces, plus ActiveX support on Windows platforms.

Platforms and Compilers

The README file for this release itemizes the platforms on which the C++ interface to ObjectStore Release 5.1 is currently available, or where its availability is planned. You can also contact Object Design Technical Support for current information.

ANSI C++ Exceptions

The ObjectStore Release 5.1 C++ interface will support the use of ANSI C++ exceptions on the following platforms:

- Digital UNIX
- HP
- OS/2
- Windows
- and possibly SGI and AIX

It is Object Design's intention to support ANSI exceptions on all supported platforms in a future major release of ObjectStore. After support for ANSI exceptions is universal, the TIX exception mechanism will be obsolete and ultimately will not be supported.

Upcoming Changes

os_database::**open** and **close**	Previously and in the ObjectStore Release 5.1 the functions for creating, opening, and closing databases can be called either inside or outside a transaction. For example, you can do an **open update** followed by an **open read-only**. This behavior will not be supported in the next major release. In the future, consecutive **open**s or **close**s must be of the same type. For example, a **close read-only** can only be followed by another **close** if it is also a **close read-only**.
os_database::**close** or **destroy**	Also in the next major release, the function for destroying or closing databases must be called outside a transaction.
access hooks	The next major release of ObjectStore will introduce an upward incompatible change to the access hooks feature of ObjectStore C++. This change will make it necessary to modify the source and potentially the logic of ObjectStore Release 5.1 applications that use the access hook facility. The affected API is **os_database::set_access_hook**.
union discriminant functions	Union variants will be supported in a different manner in the next major release.

New Features

Component Server Framework

This new capability addresses a basic need to provide documentation, code examples, and general classes that enable application writers to develop a class of applications called application servers. An application server is basically the processing engine for thin client front-end applications. See the *ObjectStore Component Server Framework User Guide* for detailed information about this new feature.

Dump/Load

This new subsystem provides a facility to enable ObjectStore users to dump and load databases into and from a nondatabase format. For specific details see The Dump/Load Subsystem, in Chapter 1, Overview of Managing ObjectStore; Chapter 4, Utilities, in *ObjectStore Management*; and Chapter 8, Dump/Load Facility, in the *ObjectStore Advanced C++ API User Guide*.

Component Schema for DLL Use

This feature allows for incremental loading and unloading of schema at run time. This functionality enables building applications that consist of component DLLs and their associated schemas. Component schema functionality is described in the *ObjectStore Advanced C++ API User Guide* and *ObjectStore C++ API Reference*.

X/Open's XA Transaction Processing Standard Is Supported

In Release 5.1, ObjectStore clients and Servers can support X/Open's transaction processing standard (known as XA). The implementation of this interface is layered on top of the existing ObjectStore client library. An ObjectStore client can now act as a Resource Manager (RM). For more information, see Support for the XA Standard for Transaction Processing on page 19.

Address Space Reset

This feature allows applications to release address space that was assigned during the execution of a transaction. The existing ObjectStore entry point **objectstore::release_address_space()** is

extended so that it can now work within a transaction. There is also a mechanism for keeping certain address space assigned. This mechanism is built using **pvars** and a new facility implemented with the **os_retain_address** class.

For further discussion, see Controlling Address Space Usage During a Transaction on page 11.

Reduced Address Space Consumption in Queries

In earlier releases of ObjectStore, queries on very large collections could terminate because address space ran out. In this release, the ObjectStore query facility includes two new memory modes that automatically release address space and allow you to create queries that will never run out of address space.

For further discussion, see Address Space Usage with Queries on page 16.

Dynamic Extents

Dynamic extents are a mechanism for treating all objects of a particular type in a segment or database as a collection. The new class **os_dynamic_extent** is used for this.

For more information, see os_dynamic_extent Class on page 20.

String Conversion for Asian Language String Encodings

A new facility for converting Asian language string encodings is available in this release. See **os_str_conv** for details. See also Conversion Between Asian Language String Encodings on page 21.

Bit Vector-Assisted Relocation

An optimization to relocation in ObjectStore Release 5.1 is the use of a bit vector to assist in the relocation of pages (in or out) that have already been relocated in once, and are still in the cache.

osgc Utility Capabilities

The ObjectStore garbage collection utility, implemented to support the Java interface to ObjectStore, can now be used with ObjectStore C++ with some restrictions. The ObjectStore persistent garbage collector (GC) collects unreferenced objects

and ObjectStore collections in an ObjectStore database and frees space associated with these objects.

The **osgc** utility removes all data from the database that cannot be navigated to from a root or a protected reference. If your database is pointed to by cross-database pointers, cross-database references, or dumped or transient references, you cannot safely run **osgc** on it unless you ensure that each object referred to by any of the above is also the target of a root or protected reference in the database on which you intend to use the **osgc** utility.

osgc Release 4 databases

osgc is supported only on databases initiated in Release 5 or later. Databases upgraded from previous releases are not supported, and using **osgc** on them can corrupt them.

You can successfully use the **osgc** utility on databases from previous releases that have been dumped and reloaded to a Release 5 or later database using **osdump** and **osload**. See osdump: Dumping Databases and osload: Loading Databases in Chapter 4 of *ObjectStore Management*. See also Chapter 8, Dump/Load Facility, in the *ObjectStore Advanced C++ API User Guide*.

Databases containing schema information about template instantiations (including information about ObjectStore templated collection types) inadvertently contain unreferenced objects that **osgc** will remove. This is safe and will not affect correct operation.

It is safe to use **osgc** concurrently with other applications that modify the database. It is not safe to run more than one **osgc** on a database at the same time. See osgc: Garbage Collection Utility in *ObjectStore Management* for further information.

Documentation Enhancements

The ObjectStore Release 5.1 documentation is enhanced in content and form. The improvements are listed in the paragraphs that follow.

Clarification of Functionality

There is new documentation for previously existing ObjectStore features. The following paragraphs provide cross-references to the new information in other books in the ObjectStore Release 5.1 documentation set.

Collections documentation Improvements

Significant updates and clarification of the collections discussions in *ObjectStore Collections C++ API Reference* appear in the ObjectStore Release 5.1 documentation.

os_rDictionary

Added information about **os_rDictionary** is available in the *ObjectStore Collections C++ API Reference*. See **os_rDictionary** for details.

Checkpoint/refresh

There is new information about how transactions work with checkpoint / refresh in the *ObjectStore Advanced C++ API User Guide*. See Checkpoint: Committing and Continuing a Transaction for details.

Additions to the os_mop class

os_mop::current, **os_mop::find_name**, and **os_mop::reset** Previously undocumented, these are now included in the *ObjectStore C++ API Reference*.

Installing the On-Line Documentation

To install the ObjectStore full-text-searchable documentation, unpack the documentation distribution by doing one of the following:

For root installation

If ObjectStore has been installed as **root, $OS_ROOTDIR** is write protected. Therefore, you must complete the following steps.

```
# chmod +w $OS_ROOTDIR
# cd $OS_ROOTDIR
# uncompress -c /cdrom/packages/ostore/doc_sol2.tar.Z | tar xvf -
# chmod -w $OS_ROOTDIR
```

For non-root installation

If ObjectStore has been installed using the non-**root** option, the owner / installer has write permission in **$OS_ROOTDIR** (and all

subdirectories) so the **chmod** command is unnecessary. In this case, do the following steps.

```
# cd $OS_ROOTDIR
uncompress < /cdrom/packages/ostore/doc_sol2.tar.Z | tar xf -
```

When you run the **ossearch** command the first time, you will be asked whether to install it. After it installs, and on future invocations of **ossearch**, it will launch the configured browser on the root of the documentation tree.

Browser warnings

When you invoke the search application, you might see a stream of warnings before the browser actually appears. These complaints are associated with the release of X11 the application expects. If you are running X11 R6, no such warnings appear.

Viewing the On-Line Documentation

The documentation for ObjectStore Release 5.1 is distributed in machine-readable HTML format and PDF. The HTML format uses HTML frames, so JavaScript must be enabled. To view the documentation from a browser on UNIX, in the **$OS_ROOTDIR/ODI** directory, run the **ossearch** utility. This displays the catalog of ObjectStore documentation components.

On Windows platforms, you can invoke the searchable documentation from the ObjectStore Win32 group with the ObjectStore Documentation icon. In either case, your browser appears with a top index displayed. Select the documentation or bookshelf you want as usual.

You can search the entire ObjectStore Release 5.1 documentation set from the top-level bookshelf search button for each interface (for example, **5.1.0.0.0/ostore/doc/index.htm**). Once you have selected a book, you can search the rest of its documentation set by selecting the search button in the navigation bar above the book text frame.

Search by entering a word or series of words separated by commas in the query box and pressing the Return key. If you are uncertain about how to enter a query, you can refer to an on-line search query guide by clicking on the string **to learn additional query methods** that appears in the search form.

Changes and Additions to the C++ Interface

The following paragraphs summarize modifications to the C++ interface.

Changes to the API

The following functions are now callable within top-level transactions, as well as outside top-level transactions. In the previous release, they were callable only outside top-level transactions.

static void objectstore::retain_persistent_addresses();

static void objectstore::release_persistent_addresses();

static void objectstore::get_retain_persistent_addresses();

Additions to the API

The following functions have been added for this release. They make certain aspects of controlling address space behavior more explicit:

static void objectstore::set_retain_persistent_addresses(
os_boolean value);

static void objectstore::reset_persistent_addresses();

static void objectstore::release_persistent_addresses(
os_boolean force);

These new functions are introduced to make explicit the distinction between disabling the retain behavior (done by calling **objectstore::set_retain_persistent_addresses()** with the value argument false) and deassigning address space (done by calling **objectstore::reset_persistent_addresses()**).

Controlling Address Space Usage During a Transaction

ObjectStore Release 5.1 introduces the ability to release address space during a transaction. In earlier releases, all address space assignments made during a top-level transaction were retained until the transaction was completed.

As with top-level transaction boundaries, when **objectstore::retain_persistent_addresses** is not in use, releasing address space during a transaction requires that the application drop pointers to persistent memory locations that are released.

Restrictions

- This feature cannot be used within nested transactions.
- Some address space assignments cannot be released. These addresses correspond to the first range of segment 0 of any database in use, and the first range of any info segment in use. This is limited to one range per database and one range per segment.
- When address space is released, encached pages with pointers to that address space are evicted. This can have an impact on performance.
- When using multiple threads participating in a global transaction, those threads must synchronize at **objectstore::release_persistent_address** boundaries, just as for transaction commit.

Changes to the API

The following functions can now be called within top-level transactions, as well as outside top-level transactions. In the previous release, they were callable only outside top-level transactions.

static void objectstore::retain_persistent_addresses();

static void objectstore::release_persistent_addresses();

static void objectstore::get_retain_persistent_addresses();

Additions to the API

The following functions have been added for this release. They make certain aspects of controlling address space behavior more explicit:

static void objectstore::set_retain_persistent_addresses(os_boolean value);

static void objectstore::reset_persistent_addresses();

A new argument, **os_boolean force**, has been added to **objectstore::release_persistent_addresses()**

static void objectstore::release_persistent_addresses(os_boolean force);

These new functions are introduced to make explicit the distinction between disabling the retain behavior (done by calling **objectstore::set_retain_persistent_addresses()** with the value argument **false**) and deassigning address space (done by calling **objectstore::reset_persistent_addresses()**).

New Function	*Equivalent 5.0 Function*
objectstore::set_retain_persistent_addresses(true)	**objectstore::retain_persistent_addresses()**
objectstore::set_retain_persistent_addresses(false) and **objectstore::reset_persistent_addresses()**	**objectstore::retain_persistent_addresses()**
objectstore::reset_persistent_addresses()	No equivalent in release 5.0

Calling **objectstore::retain_persistent_addresses()** is equivalent to calling **objectstore::set_retain_persistent_addresses(true)** or to calling **objectstore::set_retain_persistent_addresses(false)** and **objectstore::reset_persistent_addresses()**.

Functionality associated with **objectstore::reset_persistent_addresses()** alone is new in Release 5.1.

Calling **objectstore::retain_persistent_addresses()** or **objectstore::set_retain_persistent_addresses(true)** within a transaction is no different from calling either of them before the transaction. All that these functions do is turn on a flag specifying that the client not perform a release automatically at the end of the top-level transaction. This flag can be turned on at any time.

Calling **objectstore::release_persistent_addresses()** with the **force** argument **true** is equivalent to calling release on all existing mechanisms that are retaining address space.

objectstore::get_address_space_generation_number()

Address Space Generation Number

os_unsigned_int32 get_address_space_generation_number()

This function returns an unsigned integer that is incremented by the client whenever it releases any address space. Its primary purpose is to support pointer caching, such as that used by ObjectStore collections in several circumstances. A transient cache of persistent pointers should be considered invalid whenever the value of **objectstore::get_address_space_generation_number()** increases. The **objectstore::get_address_space_generation_number()** function simply returns the value read from a variable, and so is fast enough to be called whenever a pointer cache is examined.

os_retain_address Class

The class **os_retain_address** allows an application to specify that certain address ranges be kept assigned across calls to **objectstore::release_persistent_addresses()** and top-level transactions.

See **os_retain_address** in *ObjectStore C++ API Reference* for further information.

Use of pvars with **os_retain_address**

Instances of **os_pvar** are treated specially by the address release operation when called within a transaction. Any such **os_pvars** that are *active* when address space is released act like instances of **os_retain_address** — the persistent address that they refer to continues to be assigned. However, unlike **os_retain_address**, active **os_pvars** do not hold address space across transaction boundaries when **objectstore::retain_persistent_addresses()** is not operating.

objectstore::set_retain_address() and objectstore::get_retain_counter()

The static functions **objectstore::set_retain_address()** and **objectstore::get_retain_counter()** can be used to retain and release individual address ranges. Each address range maintains a **retain_counter** that is initially 0. The function signatures are

```
static void objectstore::set_retain_address(
    void *address, os_boolean value = true);

static os_unsigned_int32 objectstore::get_retain_count(
    void *address);
```

Calls to **objectstore::set_retain_address()** with **value = true** on any address in the range will increment the counter. Calls to **objectstore::set_retain_address()** with **value = false** on any address in the range will decrement the counter (if it is greater than 0). Calling **objectstore::get_retain_count** on any address in a range returns the current value of the counter for that range.

Whenever a range has a retain count greater than zero, that range will not be released by any release operations (except a force release operation).

Incremental Release of Address Space: os_address_space_marker Class

The **objectstore::release_persistent_addresses()** call releases address space reserved since the beginning of a transaction, or since the last call to **objectstore::retain_persistent_addresses()**. Obviously, releasing all address space is something only the application can do directly, since the application must make sure that transient pointers to persistent objects get dropped.

However, there are certain address-space-consuming features that would benefit from having the ability to release address space in a manner that is transparent to the application. The primary example of such a feature is a collections query. During a query, address space might be consumed in large quantities. A new class, **os_address_space_marker**, provides the ability for a query to release the extra address space it consumed that is not required by the application outside the query. This allows queries that detect the address space full condition (err_address_space_full) and use this scheme to release the address space they have consumed and continue, to examine more objects than could fit into address space at any one time.

See **os_address_space_marker** in the *ObjectStore C++ API Reference* for a description of this class.

Interactions Between Different Address-Space Mechanisms

The different APIs for controlling address space can be ordered by the specificity (least to most) of the target address space, as follows:

- Default transaction boundary retain / release semantics
- Process-wide

  objectstore::retain_persistent_addresses()
  objectstore::release_persistent_addresses()
  objectstore::reset_persistent_addresses()
  objectstore::set_retain_persistent_addresses()
  os_address_space_marker class

- Specific range

  os_retain_address
  os_pvar
  objectstore::set_retain_address

For cases where several address space mechanisms are in place, the rule is that the more specific calls take precedence. The only exception is the force form of **objectstore::release_persistent_addresses()**, which causes all the mechanisms in effect at the time of the call to release. For calls at the same level of specificity, the retains take precedence over releases.

Example

Constructing an **os_retain_address** on a variable pointing to address A, followed by calling **objectstore::_reset_address** on A, will not result in A's being released. (This is the *same level of specificity* rule that says that retains take precedence over releases.)

Related Functions

Two functions related to controlling address space allocation are available in Release 5.1:

objectstore::get_unassigned_address_space()

static os_ptr_val objectstore::get_unassigned_address_space();

Returns the total amount of address space that is still available for assignment. The value returned is always a multiple of 64 KB.

objectstore::get_largest_contiguous_unassigned_address_space()

static os_ptr_val objectstore::
get_largest_contiguous_unassigned_address_space();

Returns the size of the largest contiguous region of address space that is still available for assignment. The returned value is always a multiple of 64 KB.

Address Space Usage with Queries

In earlier releases of ObjectStore, queries on very large collections could terminate because address space ran out. In this release, the ObjectStore query facility includes two new memory modes that allow you to create queries that will never run out of address space. You can control this behavior with the new function **os_collection::set_query_memory_mode()**. Specify the enumerator **os_query_memory_mode_none** if you want queries to use memory mode as they did in Release 5.0.

Customizing Address Space Usage in Collections

ObjectStore provides the following two classes for use in customizing address space usage with collections.

os_reference_cursor Class

Creates a transient reference-based list (**os_packed_rlist**) from any type of collection that can be iterated over using the member functions.

```
class os_reference_cursor {

public:
    // This is the public reference based API to this class
    os_reference * first();
    os_reference * last();
    os_reference * next();
    os_reference * previous();
    os_reference * retrieve();
    os_int32 more() const;
    os_int32 null() const { return !more(); };

    // Versions of functions that automatically check for
    // err_address_space_full
    void * retrieve(os_address_space_marker &);
    void * first(os_address_space_marker &);
    void * last(os_address_space_marker &);
    void * next(os_address_space_marker &);
    void * previous(os_address_space_marker &);

    // construction
    os_reference_cursor(os_collection *, os_unsigned_int32 flags =
0);

    // Destruction
```

```
    ~os_reference_cursor();
```

os_cursor_holder Class

Remembers the position of the cursor in the collections after a call to **os_collection::release_address_space** has been made.

```
class os_cursor_holder
{

public:
    // The public interface to os_cursor holder
    os_cursor_holder(os_cursor * cursor);
    os_cursor_holder(os_dictionary_cursor * cursor);
    os_cursor_holder();
    ~os_cursor_holder();
    void remember();
    void remember(os_cursor * cursor);
    void remember(os_dictionary_cursor * cursor);
    void restore();
    void init();
};
```

Using Component Schemas

ObjectStore Release 5.1 includes a new set of features that allow you to write applications that use DLLs (dynamically loaded and associated schemas).

A component schema, also referrred to here as a DLL schema, is a self-contained schema associated with a DLL. It plays the same role for the DLL as an application schema plays for an application. Like a DLL, and unlike an application schema, a DLL schema can be loaded and unloaded dynamically at run time. Unlike the application schema, multiple DLL schemas can be in effect at the same time in a single program. The file name extension **.adb** is used for both application schemas and DLL schemas. DLL schemas are generated by **ossg** just as application schemas are.

For further information, see

- In *ObjectStore C++ API User Guide*, see **OS_SCHEMA_DLL_ID**, **OS_SCHEMA_INFO_NAME**, and **OS_REPORT_DLL_LOAD_AND_UNLOAD**.
- In *ObjectStore Building C++ Interface Applications*, see Generating an Application or Component Schema in Chapter 3.

In *ObjectStore C++ API User Guide*, see Chapter 11, Component Schemas.

In the *ObjectStore C++ API Reference*, see the new classes **os_DLL_finder**, **os_DLL_schema_info**, **os_schema_handle**, **os_schema_info**, and additions to the classes **objectstore** and **os_database**.

Support for the XA Standard for Transaction Processing

ObjectStore supports X/Open's transaction processing standard (known as XA). For further information see Support for the XA Standard for Transaction Processing in the *ObjectStore C++ API User Guide*.

os_dynamic_extent Class

Derived from **os_Collection**, an instance of this class can be used to create an extended collection of all objects of a particular type, regardless of which segments the objects reside in. All objects are retrieved in an arbitrary order that is stable across traversals of the segments, as long as no objects are created or deleted from the segment, and no reorganization is performed (using schema evolution or compaction).

For further information see **os_dynamic_extent** in the *ObjectStore C++ API Reference*.

Conversion Between Asian Language String Encodings

There are many standards for encoding Asian characters. In Japan, for example, five encodings are in broad use: JIS, SJIS, EUC, Unicode, and UTF-8.

Usually an application uses one encoding for all strings to be stored inside a database. The encoding chosen is most often the one used in the operating system of the ObjectStore client.

However, if the application has heterogeneous clients using a variety of encodings, conversion from one encoding to another is necessary at some point. The clients could be traditional ObjectStore client processes or thin-client browsers that emit data in different encodings.

This release of ObjectStore provides conversion facilities for various Japanese language text encoding methods: EUC, JIS, SJIS, Unicode, and UTF8. For more information, see Using Asian Language String Encodings in the *ObjectStore C++ API User Guide* and **os_str_conv** in the *ObjectStore C++ API Reference*.

Chapter 2
Changes and Additions to Existing Features

In general, ObjectStore Release 5.1 is drop-in compatible with Release 4. It is, however, not compatible with applications that use Versions or other features eliminated from Release 5. If you are upgrading from Release 3, you must upgrade your databases with Release 4 before using Release 5.

This chapter includes information about changes to Release 5.1 ObjectStore C++ behavior, and specific conditions that apply independent of platform. The information is organized by specific compatibility with earlier releases. Topics covered include

Compilation Compatibility

ObjectStore programs built with Release 5.0.x can be compiled without source-level modifications and will continue to work with ObjectStore Release 5.1.

Link Compatibility

ObjectStore programs built using Release 5.0.x can be relinked using ObjectStore Release 5.1 libraries without the need to recompile source modules.

Drop-In Compatibility

ObjectStore programs built using Release 5.0.x can be pointed at ObjectStore Release 5.1 shared libraries without having to be rebuilt. In other words, ObjectStore Release 5.1 maintains drop-in compatibility with Release 5.0.x of ObjectStore. A result of this is that ObjectStore Release 5.1 is an acceptable replacement for the 5.0.x patch release stream.

Note: Applications compiled using ObjectStore Release 5.1 are not backward compatible with previous versions of ObjectStore.

Behavior Compatibility

With the following exceptions, ObjectStore programs from Release 5.0.x behave similarly:

- **OS_IMMEDIATE_THRESH** environment variable

  This environment variable has been renamed to **OS_INBOUND_RELOPT_THRESH**. **OS_OUTBOUND_RELOPT_THRESH** has been added also for symmetry and is synonymous with **OS_RELOPT_THRESH**.

- **OS_MAX_IMMEDIATE_RANGES** environment variable

  This environment variable is no longer being used.

Database Compatibility

There are two areas in which ObjectStore Release 5.1 is incompatible with Release 5.0.x.

PRM format

The first is that in order to use ObjectStore Release 5.1, a database must have been upgraded to use the enhanced PRM format. The earlier standard format PRMs are no longer supported. This also means that you cannot upgrade directly from Release 4.0 to ObjectStore Release 5.1 without first upgrading to ObjectStore Release 5.0.x enhanced PRM format. In fact, if you have ObjectStore Release 5.0.x databases that used the earlier standard prm format, you must upgrade them using the ObjectStore Release 5.0.x utility **osupgprm**.

Server transaction log change

In order to support XA, it was necessary to update slightly the format of the Server's transaction log. The result of this change is that existing Release 5.0.x ObjectStore Server logs cannot be propagated using an ObjectStore Release 5.1 Server.

Utility Compatibility: ossg

Changes to ossg Default

Object Design has changed the default behavior of **ossg** so that weak importing of vtables done on Solaris and other UNIX platforms is no longer supported by the default. The implication is that when you are building an application you might see unresolved symbols that are new.

Object Design recommends that the most portable method of dealing with this is to create a **force_vft()** function in your executable that will cause the **vfts** to be instantiated. See Symbols Missing When Linking ObjectStore Applications in *ObjectStore Building C++ Interface Applications* for more information.

The other (not recommended) way to deal with this is to use the **-weak_symbols** flag to **ossg** to revert **ossg** to ObjectStore Release 5.1 behavior. The **-no_weak_symbols** flag still exists in **ossg** but does emit a warning stating that **no_weak_symbols** is the default behavior.

-weak_symbols Option

In earlier releases, the schema info linked into an application schema used weak import references on some platforms to link to virtual function tables and union discriminant functions. This was the default behavior. It could be changed with the **-no_weak_symbols** option to the schema generation utility, **ossg**

In Release 5.1, default behavior has been changed so that weak import references are never used by default. This was done because using weak import references with DLL schema can cause unpredictable effects. The affected platforms are versions of UNIX. If the weak import feature is needed for some reason, you can restore it by using the **-weak_symbols** command-line argument to **ossg**.

ossg Limitations

Note the following limitations and their solutions for **ossg** in ObjectStore Release 5.1.

Explicit Template Specializations

ossg does not permit explicit template specializations. For example:

```
template <class A> class B { ... };
template <> class B<char> { ... };  // not accepted
```
Ossg: error message: "<file>":LINE <number>, syntax error on input ">"

Work around

Instead, use the following to specialize class template B:

```
class B<char> { ... };
```

Forward Declarations

Forward declarations, including friend declarations, that involve template instantiations can cause problems if the same instantiation appears later in the code. For example:

Example

```
template <class A> class B { ... };
class D {
    class B<int>; // or friend class B<int>;
};
class B<int> { ... };
```
Ossg: error message: "<file>":LINE <number> *** Defining a previously defined class <class>

Work around

The solution is to eliminate the forward reference. In the previous example, for instance, move **class B<int> { ... };** upward to a place before its use.

Class Declarations in Templates

Class declarations in templates can produce link errors if the class is derived from another class that defines virtual functions. For example:

Example

```
struct C { virtual void f(){} };
template< class T > struct A  {
    struct B  : public C { ... };
};
```
Ossg: ossg does not break but generates incorrect names

Work around

The work around for this is to move inner declarations to outside the template. In the previous example, for instance, use the following instead to specialize class template B:

```
class B<char> { ... };
```

Changes from the Previous Release

Deprecated Features and Interfaces

NETBIOS support	Support for NETBIOS is removed.
os_database::alloc() and **os_segment::alloc()**	The entrypoints **os_database::alloc()** and **os_segment::alloc()** will be removed from the next major release of ObjectStore.
C library interface	The C library interface to ObjectStore is deprecated in this release. Support for it will be removed in the next major release of ObjectStore.
Persistent relocation maps	The following functions related to persistent relocation maps are deprecated in this release and will be removed in the next major release of ObjectStore.

- **objectstore::set_new_dbs_standard_prm_format(osbool)**

  This signals an exception in ObjectStore Release 5.1 and is deprecated. It will be removed in the next major release of ObjectStore.

- **objectstore::get_new_dbs_with_standard_prm_format()**

  This now always return false and is deprecated. It will be removed in the next major release of ObjectStore.

- **os_database::get_prms_are_in_standard_prm_format()**

  This now always return false and is deprecated. It will be removed in the next major release of ObjectStore.

- **os_database::get_prms_are_in_standard_format()**

  This now always return false and is deprecated. It will be removed in the next major release of ObjectStore.

- **objectstore::read_counter()** and **objectstore::unassigned_address_space_counter()**

  Support for these functions has been removed. Use **objectstore::get_unassigned_address_space()** instead.

IP Addresses in UNC Pathnames

You can use IP addresses in UNC pathnames when opening a database. For example:

//198.316.17/top/dbs/db1

New Documentation for -O option to osrestore

The **osrestore** utility takes an option, **-O**, that restores the database image specified with the **-f** flag and then exits. There is no prompt for additional volumes.

Incompatible Changes to os_CString

To fix several reported problems, **os_CString** was substantially rewritten in Release 5.1. Applications that use **os_CString** must be recompiled.

The following changes were made:

1 The use of a common empty string was eliminated to avoid cross-segment pointers. The default constructor provides each **os_CString** object with its own empty string.

2 Like regular **CString**s, **os_CString**s share data when copied, by default. This can lead to undesirable cross-segment pointers. To avoid this, Object Design recommends that you call **os_CString::LockBuffer()** on persistent **os_CStrings**. Copying a transient **os_CString** to a persistent location, or copying a persistent **os_CString** from one database to another, will copy the data instead of sharing it.

3 Internal operations on **os_CString** use **_ODI_strlen** instead of **lstrlen**. If a persistent string is not currently mapped into memory, **_ODI_strlen** causes it to be mapped and returns the correct length, unlike **lstrlen**, which returns **0**.

Documentation Enhancements

Use of Change-Record Files with osbackup

For every set of databases you plan to back up, you need one change record file. Only one level 0 backup can be recorded in a change-record file. Subsequent level 0 backups refresh the change records, so you will lose information about the prior databases' backup status.

For example, the following will work and is the recommended usage:

```
osbackup -f ./test1.db.back0 -i test1.db.record -l 0 -a test1.db
osbackup -f ./test2.db.back0 -i test2.db.record -l 0 -a test2.db

osbackup -f ./test1.db.back4 -i test1.db.record -l 4 -a test1.db
osbackup -f ./test2.db.back4 -i test2.db.record -l 4 -a test2.db
```

However, using the default incremental record file for two backups like this will result in lost information about the backup level of **test1.db**:

```
osbackup -f ./test1.db.back0 -l 0 -a test1.db
osbackup -f ./test2.db.back0 -l 0 -a test2.db

osbackup -f ./test1.db.back4 -l 4 -a test1.db
osbackup -f ./test2.db.back4 -l 4 -a test2.db
```

In general, Object Design advises against using the default record file because it is easy to make this kind of mistake. You should always specify a unique record file for each set of databases backed up with the **-i** option.

Correction to Some Examples in the *ObjectStore C++ API User Guide*

Examples in the following section of the *ObjectStore C++ API User Guide* have some incorrect lines. They should appear as described in this section.

Using Nonparameterized References

If your compiler does not support class templates, you can use the nonparameterized reference class **os_reference**. You also should use **os_reference** if you need a reference to an instance of a built-in type like **int** or **char**; the referent type of an **os_Reference** must be a class.

os_reference is just like **os_Reference**, except the conversion constructor used is **os_reference(void*)** instead of **os_Reference(T*)**. In addition, the conversion operator used is operator **void*()** instead of operator **T*()**, which means that you should use a cast to pointer-to-referent type when dereferencing an **os_reference**.

Corrections to nonparameterized example 1

Nonparameterized example 1

Here are some examples:

```
#include <ostore/ostore.hh>
#include <stdio.h>
class employee {
```

The following line was omitted:

```
public:
    static os_typespec *get_os_typespec();
    . . .
```

The following line was omitted:

```
int emp_id;

    };
    class part {
```

The following line was omitted:

```
public:
    static os_typespec *get_os_typespec();
    . . .
    os_reference responsible_engineer;
    . . .
    };
    f() {
    objectstore::initialize();
    static os_database *db1 = os_database::open("/thx/parts");
    static os_database *db2 = os_database::open("/thx/parts");
    OS_BEGIN_TXN(tx1, 0, os_transaction::update)
        part *a_part = new(db1, part::get_os_typespec()) part;
        employee *an_emp =
            new(db2, employee::get_os_typespec()) employee;
        a_part->responsible_engineer = an_emp;
```

This line is incorrect:

```
        printf("%d\n",
        (employee*) (a_part->responsible_engineer)->emp_id);
```

It should be

```
    printf("%d\n",
        ((employee*) (void*)
        (a_part->responsible_engineer))->emp_id);

OS_END_TXN(tx1)
db1->close();
}
```

Corrections to nonparameterized example 2

Nonparameterized example 2

```
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "part.hh"
main() {
objectstore::initialize();
part *a_part;
os_database *db1 = os_database::open("/thx/parts");
os_reference part_set_ref;
OS_BEGIN_TXN(tx1, 0, os_transaction::update)
    part_set_ref = (os_set*) (
        db1->find_root("part_set")->get_value()
    ); /* retrieval */
    . . .
OS_END_TXN(tx1)
OS_BEGIN_TXN(tx2, 0, os_transaction::update)
```

This line is incorrect:

```
    a_part = (part*) (
        ((os_set*) (part_set_ref))->query_pick(
        "part", "part_number==123456", db1
        )
    ) ; /* OK */
```

It should be

```
a_part = (part*) (
    ((os_set*)(void*) (part_set_ref))->query_pick(
        "part", "part_number==123456", db1
    )
    ) ; /* OK */
    . . .
OS_END_TXN(tx2)
db1->close();
}
```

Chapter 3 Platform-Specific Considerations

This chapter describes platform-related considerations you should anticipate when using ObjectStore Release 5.1.

Note that some of the books contain platform-specific information about platforms for which ObjectStore Release 5.1 is not yet available. Such information refers to behavior using the previous version of ObjectStore. These release notes and the other documents in the documentation set will be updated as ObjectStore Release 5.1 becomes available on additional platforms.

The topics are organized in the following manner:

Windows

The following paragraphs describe platform-specific considerations for ObjectStore Release 5.1 on Windows NT.

Installing DEBUG.ZIP or DDEBUG.ZIP

To install **DEBUG.ZIP** or **DDEBUG.ZIP**, follow the steps described here.

1 Install ObjectStore Release 5 with the **SETUP** program. See ObjectStore Installation for Windows NT documentation for instructions.

2 Shut down the ObjectStore Server and Cache Manager by using the ObjectStore **SETUP** program. Answer **Yes** to the question about shutting down servers, then exit from **SETUP**.

3 Go to the **%OS_ROOTDIR%** directory.

4 Rename **bin** and **binsngl** directories (from the command prompt window or **Windows Explorer**) to **retail.bin** and **retail.binsngl**.

5 Unzip the file (**DEBUG.ZIP** or **DDEBUG.ZIP**) from the command prompt by typing the following command.The **-d** option creates and restores the directories included in the zip file.

```
pkunzip -d debug.zip
```

6 Run the ObjectStore **SETUP** program to start the Server. In the first setup dialog select the **Setup Server** option. In the menu **Choosing to start ObjectStore services automatically**, select **Yes**. Then a **Confirm Message** dialog asks if you want to start the services right now. Select **Yes**.

Now you can debug your application. For more information about required compilation options, see the Windows DEBUG and DDEBUG Builds of ObjectStore in the Windows section of Chapter 4, Compiling, Linking, and Debugging Programs in *ObjectStore Building C++ Interface Applications.*

Solaris 2

The following paragraphs describe platform-specific considerations for ObjectStore Release 5.1 on Solaris 2.

HP

HP aC++ source file naming clarification

A clarification has been added to the Chapter 4, Compiling, Linking, and Debugging Programs, of *ObjectStore Building C++ Interface Applications*. See HP aC++ Source Files for details.

16K Page Size and Heterogeneous Database Access

In releases prior to ObjectStore Release 5.1, there is a bug in the support of heterogeneous access to databases created on machines with a 16K page size, such as SGI platforms: databases created on 16K page big-endian platforms cannot be accessed from small-endian platforms. Although this has been fixed in Release 5.1, older databases created on 16K page size platforms must be upgraded to be accessible to small-endian platforms.

In Release 5.1 a tool is provided that can be used to check whether a database has such limitations. The tool can also be used to upgrade a database if such limitations exist. It can be used from a command line option to **osverifydb,** or with the API **os_dbutil::osverifydb()**.

Note

There is no need to use this tool if the database has been created on a platform that has a 4K or 8K page size.

New Option to osverfiydb

To check or upgrade a database for hetergeneous accessibility with the **osverifydb** command line utility, you can specify a newly added command line option, **-info_sector_tag_verify_opt** *option*. This verifies info segment sector tags in accordance with the option value you specify.

Valid *options* are

- **0** Skips verifying info segment sector tags (default).
- **1** Verifies info segment sector tags and reports whether the database can be used heterogeneously.
- **2** Upgrades the database for heterogeneous accessibility.
- **5** Causes **osverifydb** to report information for this option only. Other verifications usually performed by **osverifydb** are not made.
- **6** Performs an upgrade only. Other verifications usually performed by **osverifydb** are not made.

New Argument to osdbutil::osverifydb()

You can also use the upgrade tool by means of the **os_verifydb_options** argument to the API **os_dbutil::osverifydb()** with **os_verifydb_options::info_sector_tag_verify_opt** set to the desired value:

```
class os_verifydb_options
{ public:
    ...
    enum info_sector_tag_verify_opt_enum {
        verify_skip = 0,                /* do not verify info sector tag */
        verify_report_only = 1,         /* report only */
        verify_upgrade = 2,             /* upgrade info sector tag */
        verify_skip_others = 4,         /* skip other verifications */
    } info_sector_tag_verify_opt ;
    ...
}
```

Valid **os_verifydb_options::info_sector_tag_verify_opt** values are

```
verify_skip
verify_report_only
verify_upgrade
verify_report_only | verify_skip_others
verify_upgrade | verify_skip_others
```

Chapter 4
Sources of Technical Information

This chapter describes methods of obtaining technical assistance. Sources of technical information include Object Design's

- Local distributor or value-added reseller
- Training and Education
- Consulting
- Technical Support

The following paragraphs summarize each support alternative.

Local Distributor or VAR

If you obtained ObjectStore through a distributor or VAR, contact your representative for specific information.

Object Design Training and Education

Object Design provides a variety of courses that cover all aspects of Object Design products and object-oriented analysis, design, and programming. These courses are available in Public Education Centers around the world, or you can arrange to have them presented on site at your offices.

For immediate information about courses, send email inquiries to **ooclass@odi.com** or call the Object Design Education Hotline at 781.674.5047.

Object Design Consulting

Object Design Consulting Services is dedicated to helping you turn technology innovations into real business solutions. Through Object Design's versatile suite of services, consultants enable you to deliver projects that are timely, flexible, and cost-effective solutions. Object Design consultants are ready to assist you throughout the software development cycle, from design through deployment, whether you are creating a single application, or revamping your entire technology infrastructure.

With many years of combined experience in building distributed object computing solutions for intranet, Internet, and local solutions, a dedicated staff of consultants is equipped to help you deploy world-class systems.

For more information on how Object Design consultants can start helping you maximize your technology investment, check the World Wide Web at **http://www.odi.com/Services** or contact your local sales office.

Object Design Technical Support

Object Design Technical Support provides subscribers with technical assistance and software updates. The goal of Object Design Technical Support is your success when using Object Design's software products. The Object Design Technical Support team stands ready to provide the highest quality technical support and assistance with a range of services that include

- Worldwide availability — support centers in North America, Europe, and Asia
- Access to highly trained Support Engineers through email and by phone during normal business hours
- Around-the-clock support for critical applications
- Subscriber World Wide Web site that includes FAQ database, Product Documentation, Known Bugs Lists, coding examples, and miscellaneous additional information
- Subscriber FTP site containing the latest product updates and software patches
- Participation in restricted mail discussion groups

- Software update, patch, and support news by means of electronic mail

For more information, see the World Wide Web URL **http://support.odi.com** or send mail to **support@objectdesign.com**.

Index

L

N

O

Q

R

S

T

U

V

W

X