ObjectStore®

# ObjectStore Release Notes

**Release 6.3**

PROGRESS SOFTWARE | Real Time Division

# ObjectStore Release Notes

ObjectStore Release 6.3 for all platforms, October 2005



September 2005

# Contents

# Preface

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.

**Purpose**

This document describes changes to ObjectStore for Release 6.3.

**Audience**

This document 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.

**Installing this release**

For information about installing Release 6.3, see one of the following:

- *ObjectStore Installation for Windows*
- *ObjectStore Installation for UNIX*

## Notation Conventions

This document uses the following conventions:

| *Convention* | *Meaning* |
|---|---|
| `Courier` | `Courier font` indicates code, syntax, file names, API names, system output, and the like. |
| **`Bold Courier`** | **`Bold Courier font`** is used to emphasize particular code. |
| *`Italic Courier`* | *`Italic Courier font`* indicates the name of an argument or variable for which you must supply a value. |
| Sans serif | Sans serif typeface indicates the names of user interface elements such as dialog boxes, buttons, and fields. |
| *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. |

## Progress Software Real Time Division on the World Wide Web

The Progress Software Real Time Division Web site (www.progress.com/realtime) provides a variety of useful information about products, news and events, special programs, support, and training opportunities.

### Technical Support

To obtain information about purchasing technical support, contact your local sales office listed at www.progress.com/realtime/techsupport/contact, or in North America call 1-781-280-4833. When you purchase technical support, the following services are available to you:

- You can send questions to realtime-support@progress.com. Remember to include your serial number in the subject of the electronic mail message.
- You can call the Technical Support organization to get help resolving problems. If you are in North America, call 1-781-280-4005. If you are outside North America, refer to the Technical Support Web site at www.progress.com/realtime/techsupport/contact.
- You can file a report or question with Technical Support by going to www.progress.com/realtime/techsupport/techsupport_direct.
- You can access the Technical Support Web site, which includes
  - A template for submitting a support request. This helps you provide the necessary details, which speeds response time.
  - Solution Knowledge Base that you can browse and query.
  - Online documentation for all products.
  - White papers and short articles about using Real Time Division products.
  - Sample code and examples.
  - The latest versions of products, service packs, and publicly available patches that you can download.
  - Access to a support matrix that lists platform configurations supported by this release.
  - Support policies.
  - Local phone numbers and hours when support personnel can be reached.

### Education Services

To learn about standard course offerings and custom workshops, use the Real Time Division education services site (www.progress.com/realtime/services).

If you are in North America, you can call 1-800-477-6473 x4452 to register for classes. If you are outside North America, refer to the Technical Support Web site. For information on current course offerings or pricing, send e-mail to classes@progress.com.

### Searchable Documents

In addition to the online documentation that is included with your software distribution, the full set of product documentation is available on the Technical Support Web site at www.progress.com/realtime/techsupport/documentation. The site provides documentation for the most recent release and the previous supported release. Service Pack README files are also included to provide historical context for specific issues. Be sure to check this site for new information or documentation clarifications posted between releases.

## Your Comments

Real Time Division product development welcomes your comments about its documentation. Send any product feedback to `realtime-support@progress.com`. To expedite your documentation feedback, begin the subject with `Doc:`. For example:

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

# *Chapter 1*
# New and Changed Features in Release 6.3

ObjectStore Release 6.3 includes the following new and changed features:

- Enhanced Schema Evolution and Database Compaction Features
- New C++ Collections Features
- Changes to the Java Interface to ObjectStore
- Support for .NET Applications
- New OS_USE_MAP_FIXED Environment Variable
- Changed Default Propagation Buffer Size
- Additional Deadlock Information
- Changes to ossize Utility Output
- Modified Neutralization Option for Schema Generator
- New os_path_to_data Function

## Enhanced Schema Evolution and Database Compaction Features

The schema evolution and database compaction features have the following additions and changes:

- Databases Being Evolved or Compacted Can Contain Unions
- Schema Evolution and Database Compaction Allocation Option
- New Database Compaction Options and Functions
- Schema Evolution of Pointer-to-Member Types
- Reclassification of Instances is Again Supported
- New Schema Evolution Functions

This release fixes some schema evolution and compaction known problems and removes some restrictions. Specifically,

- Unions can now be in databases on which you are performing schema evolution or compaction.

- You can now perform schema evolution on a database with a remote schema.
- You can now evolve a class that contains `pointer-to-member` type members.
- You can now evolve a class that is the target of a `pointer-to-member` type. Note that this is not supported on AIX.
- The `-classes_to_be_removed` argument to the `ossevol` utility is supported.
- You can now evolve or compact a database that contains any pages that were last written by a client application that uses a byte or bit order that is different from the byte or bit order of the machine doing the evolution or compaction.

## Databases Being Evolved or Compacted Can Contain Unions

You can use the `oscompact` utility or the `os_compact` class to compact a database that contains unions.

You can use the `ossevol` utility or the `os_schema_evolution` class to evolve the schema of a database that contains union bystanders. Union bystanders are unions that are not directly involved in schema evolution. A detailed description of what this means is in *Advanced C++ API User Guide*, Evolving Schemas That Contain Union Bystanders.

## Schema Evolution and Database Compaction Allocation Option

New options and functions for the schema evolution and database compaction API allow you to specify that evolution or compaction operations should try to avoid placing small objects so they straddle page boundaries. (Small objects are smaller than page size, which is 4K.) In normal operation with small objects, if the object will not completely fit on a single page, ObjectStore stores part of the object on the current page and the remainder on an adjacent page. When you use the new feature, ObjectStore will try to move the entire object to another page, leaving some empty space on the current page. In this way, when ObjectStore fetches a small object, it needs to fetch only a single page rather than the two pages if the object straddled page boundaries.

This feature allows you to find the right balance between minimizing the size of your database and minimizing the number of pages ObjectStore must fetch and lock when your application works on objects in your database.

*Note*: In the context of this feature, all pages are server pages, which are always 4K in size. It does not matter what platform the server is running on.

The new options and functions are

- `os_schema_evolution::set_avoid_page_boundary()`
- `os_compact::set_avoid_page_boundary()`
- `ossevol -max_bytes_wasted_to_avoid_page_boundary` *`size`*
- `oscompact -max_bytes_wasted_to_avoid_page_boundary` *`size`*

## New Database Compaction Options and Functions

### Options

The `oscompact` utility accepts the following new options.

- `-address_space_release_interval` controls the frequency of address space releases during compaction.
- `-explanation_level` specifies the level of debugging information output by the oscompact utility.
- `-malloc_size` specifies the malloc size for the relocation map during compaction. The default is 1024 KB.
- `-maximum_cluster_size` specifies that clusters larger than a specified size will be split into multiple clusters. The default value is 1600 MB.
- `-memory` specifies the maximum memory size in MB to use for the pointer relocation map. The default is half the main memory or half the virtual memory, whichever is smaller.

Numeric values for the `oscompact` utility options can be in decimal or hexadecimal format. Hexadecimal values must be prefixed by the characters 0x

Functions

The `os_compact` class supports the followiing new functions:

- `set_address_space_release_interval()`
- `set_explanation_level()`
- `set_malloc_size()`
- `set_maplet_size()`
- `set_maximum_cluster_size()`
- `set_maximum_memory()`
- The `os_compact::augment_cluster_split_avoidance()` function specifies that during compaction any cluster containing objects of a specified class will not be split when the cluster size exceeds `set_maximum_cluster_size()`. This is useful when a cluster contains user defined data that should not be split across clusters. To help determine when objects should not be split, see the Clustering Techniques white paper at `http://www.progress.com/realtime/publications/index.ssp#odbms`.
- The `os_compact::augment_post_compact_transformers()` function adds the specified transformer binding to the set of transformer bindings to be used during subsequent compaction. This applies to compaction initiated in the current process. A transformer binding associates a class with a function so that the function is executed on each instance of the class after all objects are moved.

## Schema Evolution of Pointer-to-Member Types

You can evolve a schema that contains `pointer-to-member` types.

However, support for evolving schemas that contain `pointer-to-member-function` types is restricted. When one of the following conditions is met, the schema evolution facility copies `pointer-to-member-function` types to the evolved database:

- You do not evolve the owner class, and you do not evolve the `pointer-to-member-function` type so that it points to a member of a base class instead of a member of a derived class. Likewise, you do not evolve it so that it points to a member of a derived class instead of a member of a base class.

(The owner class is the class that contains the member that is the target of the pointer. )

- The value of the `pointer-to-member-function` type is null.

When you write the untranslatable pointer handler for a pointer-to-member type, you should not call the following functions:

- `os_untranslatable_pointer_handler::get_target_segment()`
- `os_untranslatable_pointer_handler::get_target_cluster()`
- `os_untranslatable_pointer_handler::get_target_offset()`

These functions either do not return valid values or they fail. This is because a pointer-to-member type does not point to an instance of the owner class. To detect if the handler was invoked for a pointer-to-member type, call the new `os_untranslatable_pointer_handler::is_PTOM()` function.

## Reclassification of Instances is Again Supported

ObjectStore 6.3 supports the reclassification of instances. Although releases prior to ObjectStore Release 6.2 supported reclassification of instances, ObjectStore 6.2 did not. The procedure and API for reclassifying instances in 6.3 is completely different from the procedure and API in 6.1 and older.

Reclassification of instances means that the schema evolution facility lets you migrate an instance to a subclass of its original class. You can do this only when the original class is not a virtual base class of the new class. Reclassifying an instance is particularly useful when you are adding a derived class to a schema and this derived class is a more appropriate class than the base class for existing instances.

This release introduces the `objectstore::change_type()` function to help reclassify instances. For details, see the *Advanced C++ API User Guide*, Advanced Schema Evolution, Instance Reclassification.

## New Schema Evolution Functions

The `os_schema_evolution::set_maximum_cluster_size()` specifies the largest size of a cluster in bytes. During schema evolution clusters larger than this are split into multiple clusters.

The `os_schema_evolution::augment_cluster_split_avoidance()` function specifies that during schema evolution any cluster containing objects of a specified class will not be split when the cluster size exceeds `set_maximum_cluster_size()`. This is useful when a cluster contains user defined data that should not be split across clusters.

Also, documentation has been added to the *ObjectStore C++ API Reference* for the following schema evolution methods, which were part of the previous release:

- `os_schema_evolution::augment_optional_classes()`
- `os_schema_evolution::get_enclosing_object()`
- `os_schema_evolution::get_evolved_schema()`
- `os_schema_evolution::get_evolved_schema_db_name()`

- os_schema_evolution::get_explanation_level()
- os_schema_evolution::get_unevolved_schema()
- os_schema_evolution::get_path_to_member()
- os_schema_evolution::set_disable_transformer_class_checks()

# New C++ Collections Features

This release includes the following new collections features:

- New Index Monitoring Tools
- New Address Space Marker Cursors
- New Order by DSCO Argument for Cursor Constructors
- New Constructor Overloadings for os_coll_range Class

## New Index Monitoring Tools

To achieve optimal query performance, you need to maintain the correct set of indexes. Consequently, it is important to know which indexes a query actually uses. This release introduces two ways to monitor index use during query processing:

- Set the OS_COLLECTION_TRACE_INDEX_USAGE environment variable. When this variable is set, ObjectStore writes index use information to stdout after execution of each query. The default is that this variable is not set.
- Call the trace_index_usage() function. The signature for this function is

  ```
  os_collection::trace_index_usage(os_boolean run_trace,
     const char * file_name = 0)
  ```

  Call this function with run_trace set to true to produce index usage information as each query statement is processed. Specify a value for file_name to direct the index usage information to a particular file. The default is to direct usage information to stdout.

For examples of using the index monitoring tools, see the *C++ Collections Guide and Reference*, Monitoring Index Use During Queries.

## New Address Space Marker Cursors

The ObjectStore collections facility allows you to program loops that process the elements of a collection one at a time. During this process, err_address_space_full exceptions can occur when retrieving element pointers from the collection. For managing address space while iterating over a large collection, the following set of cursor functions are now available:

```
void * os_cursor::first(os_address_space_marker &);
void * os_cursor::last(os_address_space_marker &);
void * os_cursor::next(os_address_space_marker &);
void * os_cursor::previous(os_address_space_marker &);
void * os_cursor::retrieve(os_address_space_marker &);
```

```
E os_Cursor<E>::first(os_address_space_marker &);
```

```
E os_Cursor<E>::last(os_address_space_marker &);
E os_Cursor<E>::next(os_address_space_marker &);
E os_Cursor<E>::previous(os_address_space_marker &);
E os_Cursor<E>::retrieve(os_address_space_marker &);
```

For details about how to use these cursors, see *C++ Collections Guide and Reference*, Using os_address_space_marker Cursors.

## New Order by DSCO Argument for Cursor Constructors

There is a new possible argument to `os_cursor` or `os_Cursor` constructors. You can specify `order_by_DSCO` instead of `os_cursor::order_by_address` to avoid `err_address_space_full` exceptions when creating the cursor. The ordering for `order_by_DSCO` is in address space order unless an `err_address_space_full` exception is hit at which time the sorting is done in DSCO order (database/segment/cluster/offset order).

An `order_by_DSCO` cursor is useful if you dereference each collection element as you retrieve it because the order can dramatically reduce paging overhead. When used for this purpose, both `order_by_DSCO` and `order_by_address` visit each object, although in different orders. An `order_by_DSCO` cursor is update insensitive. See Address Order Traversal on page 68 for additional information.

## New Constructor Overloadings for os_coll_range Class

The `os_coll_range` class has two new constructor overloadings:

```
os_coll_range(
  os_collection::restriction rel_op,
  os_coll_int64 value
);
```

```
os_coll_range(
  os_collection::restriction rel_op,
  os_coll_uint64 value
);
```

An `os_coll_int64` is `long long` on UNIX and `_int64` on Windows.

# Changes to the Java Interface to ObjectStore

Release 6.3 includes the following additions and changes to the Java interface to ObjectStore:

- Support for Remote Schemas
- OSJI Performance Improvements - Peer Objects and Peer Collections No Longer Supported
- Support for Long Immediate Strings
- JMTL No Longer Depends On Xerces Parser
- Soft References Are the Default
- Ant Requirement for Running JMTL Examples

- Fast Mode for Verifying OSJI Databases is Supported

## Support for Remote Schemas

OSJI now offers the ability to store schema information remotely. Typically, schema information for a database is stored locally within that database. ObjectStore 6.3 provides the ability for the schema information for a database to be stored in another database (the remote database). This is advantageous when many databases share the same schema information. (This has long been a feature of OSCI.)

## OSJI Performance Improvements - Peer Objects and Peer Collections No Longer Supported

To improve performance, this release changes the way OSJI represents object identity. A consequence of these changes is that peer objects and peer collections are no longer supported. Applications that use the `com.odi.coll` collections classes must change to use the `com.odi.util` collections classes.

Also, the `com.odi.jcpp` and `com.odi.odmg` classes have been removed in this release of ObjectStore. If your application uses these classes, you should contact Technical Support to resolve migration issues.

## Support for Long Immediate Strings

Long immediate strings are a new way to allocate persistent Java strings. ObjectStore uses long immediate strings only for strings that are referenced through other objects and made persistent as a result of reachability. This feature provides performance improvements during string allocation and contributes to smaller databases. Performance gains are achieved by providing a direct reference to the string data as opposed to a reference to the typical string header.

To enable a database to use long immediate strings, set the `com.odi.useLongImmediateStrings` property in the `Properties` object that you specify when you create a session. All databases that you create in that session will be able to use long immediate strings. The default is that `useLongImmediateStrings` is not set.

If a database is able to use long immediate strings, then it is incompatible with pre6.3 client applications. If you want an existing database to be able to use long immediate strings, you must run the `osupg630` utility on it or use the corresponding API.

When using long immediate strings

1. You cannot define external references to long intermediate strings. ObjectStore throws `com.odi.ObjectException` if you try to do this.
2. Explicitly migrating a string or using a nonpersistent string as a root value causes ObjectStore to write the header. The string is not stored in a long immediate format.
3. Using a nonpersistent string in a Btree (any of the `TreeMap` types) causes ObjectStore to write a header for that string. The string is not stored in a long immediate format.

4 Object cursors for the `String` type do not pick up long immediate strings.

These limitations are expected to be removed in upcoming OSJI releases.

If you want all newly created databases to have the capability to store long immediate strings, specify the `-D` option when you launch Java. This setting applies to all sessions. ObjectStore does not check whether `com.odi.useLongImmediateStrings` is set.

If you want some new databases to be able to store long immediate strings, but you want other new databases to not be able to store long immediate strings and so maintain compatibility with pre6.3 client applications, do not specify the `-D` option when you launch Java. Instead, set or do not set the `com.odi.useLongImmediateStrings` property in the properties object that you specify when you start the session in which you plan to create the database(s). All databases created in a given session, either have the ability to store long immediate strings, or do not have the ability to store long immediate strings.

## JMTL No Longer Depends On Xerces Parser

JMTL no longer depends on the Xerces XML parser. Instead, it relies on the XML implementation packaged with the JDK. You can use the Xerces parser, but it is no longer packaged with the ObjectStore installation.

## Soft References Are the Default

`SoftReferences` are now the default for OSJI rather than `WeakReferences`. The Java garbage collector reclaims `SoftReferences` less frequently than `WeakReferences`. The Java VM characteristics provide better caching performance when using `SoftReferences` and avoid unnecessary garbage collection.

## Ant Requirement for Running JMTL Examples

Build the JMTL examples with Ant 1.6.0, 1.6.1, or 1.6.2. Do not use Ant 1.6.3 or higher.

## Fast Mode for Verifying OSJI Databases is Supported

You can now specify the `-F` option (fast mode) to the `osverifydb` utility when verifying an OSJI database. The fast mode for `osverifydb` uses techniques optimized for performance purposes.

# Support for .NET Applications

ObjectStore 6.3 for Windows platforms include "ObjectStore .NET COM Interop" (OSNCI), which provides the ability for .NET applications to access ObjectStore databases.

# New OS_USE_MAP_FIXED Environment Variable

This environment variable is required on Linux platforms if the `OS_AS_START` environment variable has been set. It might also be necessary on Solaris platforms if `OS_AS_START` has been set and the operating system reports errors when assigning the PSR to the address range specified by `OS_AS_START`.

# Changed Default Propagation Buffer Size

The propagation buffer is used for holding data that are to be written to the database or to the log. If the buffer size is large enough, data written to the log can be kept in memory until propagation to the database, thus eliminating the need to read the log. The size of this buffer can be set with the `Propagation Buffer Size` server parameter. If the server parameter is not set, the default size of the buffer is calculated according to the machine hardware — the default value is 8% of main memory divided by the number of processors minus 64MB per processor, but not less than 4MB and not more than the following maximum values:

| | |
|---|---|
| AIX5 and HP32 | 64 MB |
| Other 32-bit platforms | 256 MB |
| All 64-bit platforms | 512 MB |

In the previous ObjectStore release, there was no maximum default value, which caused problems on some platforms with limited address space for the heap.

For more information on the `Propagation Buffer Size` server parameter, see Chapter 2: Server Parameters in *Managing ObjectStore*.

# Additional Deadlock Information

When ObjectStore detects a deadlock situation, the message ObjectStore generates provides new information that can help you to debug the exception. For each conflicting client, the `err_deadlock` exception now displays a description of the type of resource (for example, a write lock) that the client is waiting for. The `err_deadlock` exception also displays resuources that are held by a client that conflicts with the other deadlock clients. This is in addition to the information that the exception already generates.

# Changes to ossize Utility Output

The `ossize` utility runs faster in this release than in previous release. The output of the `ossize` utility and the `os_dbutil::ossize()` function has changed in the following ways:

- ObjectStore now counts free spaces when
  - You specify `-c` or `-C` when you execute the `ossize` utility.
  - The value of `flag_segments` is true for the `os_size_options` argument to `os_dbutil::ossize()`.
  - The value of `flag_total_database` is true for the `os_size_options` argument to `os_dbutil::ossize()`.
- For the `ossize` utility, ObjectStore displays the result of the `-o` option before it displays the results of the `-c` option. This is reversed in previous releases.

  Likewise, for `os_dbutil::ossize()`, when the values for `os_size_options::flag_every_object` and `os_size_options::flag_segments` are both true, the output displays information about every object before it displays information about segments. This too is reversed in previous releases.

# Modified Neutralization Option for Schema Generator

When you specify the `-portable_type_name` (or `-ptn`) neutralization option when you run the `ossg` utility, the schema generator no longer flattens portable type names that are used as template type actuals. This yields identical class names across platforms. For example, consider the following:

```
#ifdef WIN32
typedef __int64 some_8byte_integer;
#else
typedef long long some_8byte_integer;
#endif

template<class T> class foo {};

OS_MARK_SCHEMA_TYPE(foo<some_8byte_integer>);
```

In previous releases, the schema generator installed `foo<__int64>` or `foo<long long>` into the database schema, according to the platform. Only Windows platforms can access `foo<__int64>`, and only UNIX platforms can access `foo<long long>`. With Release 6.3, the schema generator installs `foo<some_8byte_integer>`, which all platforms can access.

# New os_path_to_data Function

## New os_path_to_data Function

The `os_path_to_data::outer_collocated_path()` function returns a path to the outermost base class or member located at exactly the same address as the object at the end of the current path. The function signature is

```
os_path_to_data* outer_collocated_path() const;
```

# *Chapter 2*
# Platform and Release Compatibility

This section discusses application, compiler, and database compatibility, and lists the platforms and compilers supported by this release of ObjectStore. The Support Matrix at (www.progress.com/realtime/techsupport/support_matrix/objectstore) (Technical Support web site) contains an up-to-date list of all supported and maintained platforms. Please refer to the Support Matrix if you are in any doubt whether your compiler or operating system are supported. If your compiler is not supported, you cannot use this release of ObjectStore. This section discusses the following topics:

- Application Compatibility
- Compiler Compatibility
- Client, Server and Database Compatibility
- Platform Configuration: Solaris 32-Bit
- Platform Configuration: Solaris 64-Bit
- Platform Configuration: Windows 32-Bit Visual C++ 6 and 7
- Platform Configuration: Windows Visual Studio .NET 2003
- Platform Configuration: Linux 32-Bit
- Platform Configuration: Linux 64-Bit
- Platform Configuration: HP-UX 32-Bit
- Platform Configuration: HP-UX 64-Bit
- Platform Configuration: AIX

## Application Compatibility

To upgrade a 6.0.*x*, 6.1.*x*, or 6.2 application to run under 6.3 (that is, you want it to use the 6.3 run-time libraries), you must re-compile and re-link the application.

ObjectStore public header files no longer contain `using` statements. If your pre-6.3 application depends on these statements, you must modify your application and recompile it with the 6.3 run-time libraries.

**Release 6.1 and 6.0**

You must modify the source code of an application built with ObjectStore 6.1.x or 6.0.*x* if it meets any one of the following conditions:

- Your application uses classic IO streams (for example, `iostream.h`). You must change your application to use standard streams (for example, `iostream`).
- Your application uses schema evolution APIs. You must use the new schema evolution APIs.

Otherwise, all code that previously compiled with ObjectStore 6.1 release families will continue to compile with 6.3.

**Release 6.0**

If the version of the compiler you used for 6.0.x differs from the version you are using for 6.3, you must also do the following:

- Regenerate and recompile the schema source files (using `ossg`'s `-assf` or `-asof` option), and regenerate the schema databases
- Run the `osscheq` utility with the `-layout` option to determine if the schema has changed. If it has, migrate your ObjectStore database. For information on migrating a database, see the *ObjectStore Migration Guide* at `www.progress.com/realtime/techsupport/documentation/objectstore`. The format for running the `osscheq` utility is as follows:

  ```
  osscheq -layout schema1 schema2
  ```

  Replace *`schema1`* with the filename of the old application schema or with the filename of a database that was built by using the old application schema. Replace *`schema2`* with the filename of the new application schema. When using component schema, you must compare each new component schema to the corresponding old component schema or to each old database.

Note that attempting to mix and match modules of application code compiled on different compiler versions will usually result in a compiler warning. On HP platforms, however, the compiler may not warn if you attempt to link a module compiled on aCC 3.63 with a module compiled on aCC 3.45. If the modules use virtual base classes, such mixing and matching can corrupt the database. For more information, see the *ObjectStore Migration Guide* at `www.progress.com/realtime/techsupport/documentation/objectstore`.

**IO Streams**

ObjectStore does not support code that is not supported by the compiler vendor. Specifically, you cannot intermingle

- ObjectStore code, which has been compiled and tested with standard IO streams
- Your own code that relies on classic IO streams

We recommend that you use only standard IO streams in your code.

**JDK 1.5**

The ObjectStore Java interface (OSJI) is developed with JDK 1.4. If you are going to develop OSJI applications with JDK 1.5, you must use the `-target 1.4` compiler switch. See Use of JDK 1.5 on page 40 for more information.

# Compiler Compatibility

If you are upgrading from Release 6.0, 6.1, or 6.2 and are using a supported compiler that is the same as the compiler you used for the previous release, the process of upgrading to Release 6.3 is straightforward.

If you are upgrading from a pre-6.0 release or your compiler has changed since the previous release, please refer to the *ObjectStore Migration Guide* (`www.progress.com/realtime/techsupport/documentation/objectstore`) that is available on the Technical Support web site. The *ObjectStore Migration Guide* will provide detailed instructions about upgrading to Release 6.3.

# Client, Server and Database Compatibility

ObjectStore 6.3 servers can fulfill requests from 6.1.*x*, 6.2.*x*, and 6.3 client applications. That is, the client application is running on a host that has the corresponding run-time library installed. For example, a 6.1.*x* client is running on a host on which the 6.1.*x* run-time library is installed.

ObjectStore 6.1.*x* and 6.2 servers cannot fulfill requests from 6.3 client applications.

ObjectStore 6.3 client applications can use 6.1.x and 6.2.x databases that are managed by 6.3 servers if the schema of the 6.3 client application is compatible with the schema of the code that generated the database. The schemas are compatible if at least one of the following is true:

- The client application and the code that generated the database were compiled by the same compiler version.
- When the code that generated the database was compiled, it was neutralized for the machine on which the client application was compiled.

After a 6.3 client uses a 6.1.*x* or 6.2.*x* database, that database is still usable by 6.1.*x* and 6.2.*x* clients only when those clients are running the most recent 6.1.*x* or 6.2.*x* service pack.

JDO

You can use a 6.1.*x* or 6.2.*x* database with the ObjectStore 6.3 JDO interface. However, if you do, 6.1.*x* and 6.2.*x* client applications can no longer access that database. Only Release 6.3 client applications can access that database. Consequently, you must run the `osjup62` utility before you use a database with JDO. This utility marks the database to ensure that 6.1.*x* and 6.2.*x* client applications cannot access it. The format for running the utility is

**`osjup62`** *`database_name`*

# Platform Configuration: Solaris 32-Bit

You can build and run 32-bit or 64-bit applications on 64-bit hardware, but you cannot build or run 64-bit applications on 32-bit hardware

| | |
|---|---|
| ***Supported Operating Systems*** | Solaris 8<br>Solaris 9 |
| ***Supported Clusters*** | Sun Cluster 3.0 5/02 for Solaris 8 |
| ***Supported C++ Compilers*** | Sun ONE Studio 8 (C++ 5.5) |
| ***Maintained C++ Compilers*** | Sun ONE Studio 7 (C++ 5.4) |
| ***Supported Java Compilers*** | Sun Java 2 SDK 1.4 |
| ***Recommended Patches*** | Sun Patch 108528-*xx* is recommended but not required for all Solaris 8 systems running ObjectStore for its resolution of BugID 449415. This patch is included in the current Solaris 8 recommended patch cluster. |

Note

The string *xx* refers to the latest available revision of the patch from Sun.

If you are using ObjectStore's built-in failover or failover as provided by the Sun Clusters 3.0 operating system, you may be able upgrade to Release 6.3 without having to take your system out of service by performing a rolling upgrade. For more information, see *ObjectStore Installation for UNIX*.

# Platform Configuration: Solaris 64-Bit

You can build and run 32-bit or 64-bit applications on 64-bit hardware, but you cannot build or run 64-bit applications on 32-bit hardware

| | |
|---|---|
| ***Supported Operating Systems*** | Solaris 8<br>Solaris 9 |
| ***Supported Clusters*** | Sun Cluster 3.0 5/02 for Solaris 8 |
| ***Supported C++ Compilers*** | Sun ONE Studio 8(C++ 5.5) |
| ***Maintained C++ Compiles*** | Sun ONE Studio 7(C++ 5.4) |
| ***Supported Java Compilers*** | Not supported |
| ***Required Patches*** | Solaris 8 systems require Sun Patches 108434-xx and 108435-xx for C++ compiler use.<br>Solaris 9 systems require Sun Patch 111711-xx and 111712-xx for Sun ONE Studio 7 compiler use. |
| ***Recommended Patches*** | Sun Patch 108528-*xx* is recommended but not required for all Solaris 8 systems running ObjectStore for its resolution of BugID 449415. This patch is included in the current Solaris 8 recommended patch cluster. |
| ***Unsupported Components*** | OSJI, JMTL, DDML |

Note

The string *xx* refers to the latest available revision of the patch from Sun.

If you are using ObjectStore's built-in failover or failover as provided by the Sun Clusters 3.0 operating system, you may be able upgrade to Release 6.3 without having to take your system out of service by performing a rolling upgrade. For more information, see *ObjectStore Installation for UNIX*.

# Platform Configuration: Windows 32-Bit Visual C++ 6 and 7

Visual C++ 6 and Visual C++ 7 are no longer supported. If you were using ObjectStore on Visual C++ 6 or 7, you must recompile with Visual Studio .NET 2003, also known as 32-Bit Visual C++ 7.1 when you migrate to ObjectStore Release 6.3. Then run the `osscheq` utility to determine if schema evolution is required. For information about running this utility, see page 24.

# Platform Configuration: Windows Visual Studio .NET 2003

| | |
|---|---|
| ***Supported Operating Systems*** | Windows 2000<br>Windows XP |
| ***Supported C++ Compilers*** | Microsoft Visual Studio .NET 2003, also known as 32-Bit Visual C++ 7.1<br>Hotfix required: KB888256 |
| ***Supported Java Compilers*** | Sun Java JDK 1.4 |
| ***Unsupported Components*** | OSJI, DDML, JMTL |

# Platform Configuration: Linux 32-Bit

| | |
|---|---|
| ***Supported Operating Systems*** | Red Hat Enterprise Linux 3.0 with kernel 2.4.21-4 |
| ***Supported C++ Compilers*** | gcc 3.2.3 |
| ***Supported Java Compilers*** | Sun JDK 1.4 |
| ***Unsupported Components*** | OSJI, DDML, JMTL |

If you are running your application on a Linux 32-bit platform, contact technical support for information about how to evolve your schema.

*Caution*: It is likely that you will corrupt your database if you try to evolve your schema without consulting with technical support.

Schema Evolution From 6.2.1

Padding instructions added by the ObjectStore 6.2.1 schema generator (`ossg`) are not required for ObjectStore 6.3. By default, the `ossg` utility leaves padding instructions in. However, if you want smaller class sizes, and the accompanying improved performance, remove padding instructions before generating 6.3 schemas.

Schema Evolution From 6.2 and Earlier Releases

If you are running ObjectStore 6.2 or an earlier ObjectStore release on a Linux 32-bit platform, contact technical support to obtain the 6.2 `ossg` patch. Install the patch and run the 6.2 `ossg` utility on your schema. The output of the utility indicates whether your schema contains tail padding that can be reused. If your schema contains tail padding that can be reused, contact technical support for additional instructions. If your schema does not contain such tail padding, install 6.3, regenerate your schema with 6.3, and evolve your schema as needed.

# Platform Configuration: Linux 64-Bit

| | |
|---|---|
| ***Supported Operating Systems*** | Red Hat Enterprise Linux 3.0 with kernel 2.4.21 |
| ***Supported C++ Compilers*** | gcc 3.2.3 |
| ***Supported Java Compilers*** | Not supported |
| ***Unsupported Components*** | OSJI, DDML, JMTL |

**Schema Evolution From 6.2.1**

Padding instructions added by the ObjectStore 6.2.1 schema generator (`ossg`) are not required for ObjectStore 6.3. By default, the `ossg` utility leaves padding instructions in. However, if you want smaller class sizes, and the accompanying improved performance, remove padding instructions before generating 6.3 schemas.

On Linux 64-bit platforms, regenerate your schema with 6.3 and evolve the schema if necessary. If you remove padding instructions before you regenerate the schema, you probably need to evolve the schema. The `ossg` utility indicates what is required.

**Schema Evolution From 6.2 and Earlier Installations**

When using ObjectStore 6.2 or earlier releases on Linux 64-bit platforms, schemas do not have tail padding that might be reused. You should install 6.3, regenerate your schema with 6.3, and evolve your schema as needed.

# Platform Configuration: HP-UX 32-Bit

| | |
|---|---|
| ***Supported Operating Systems*** | HP-UX 11i (11.11) 32-bit |
| ***Supported C++ Compilers*** | aCC 3.63 |
| ***Supported Java Compilers*** | Sun JDK 1.4 |

**Possible Database Incompatibility**

A bug in version 3.45 of HP's aCC compiler can cause a database incompatibility between ObjectStore 6.3 and previous releases of ObjectStore on HP-UX. For some applications that have schemas that contain virtual base classes, the database schema generated by previous releases of ObjectStore is incompatible with the database schema generated by ObjectStore 6.3.

A detailed explanation of the bug and its fix is on the HP Web site at `http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,7866,00.html#faq-compat)`

To determine whether your application will have this incompatibility, use ObjectStore 6.3's schema generator to regenerate your application's schema. Then use the `osscheq` utility to compare your old database schema with your 6.3 database schema. For information about running this utility, see page 24.

If the results of the `osscheq` utility indicate a schema incompatibility, use the `ossevol` utility to evolve your database schema. Information about the `ossevol` utility is also in *Managing ObjectStore*.

# Platform Configuration: HP-UX 64-Bit

| | |
|---|---|
| ***Supported Operating Systems*** | HP-UX 11i (11.11) 64-bit |
| ***Supported C++ Compilers*** | aCC 3.63 |
| ***Supported Java Compilers*** | Not supported |
| ***Unsupported Components*** | OSJI, JMTL, DDML |

# Platform Configuration: AIX

| | |
|---|---|
| ***Supported Operating Systems*** | AIX 5.2 |
| ***Supported C++ Compilers*** | Visual Age 6.0 |
| ***Supported Java Compilers*** | Not supported |
| ***Unsupported Components*** | OSJI, JMTL, DDML |

On AIX platforms, ObjectStore 6.3 does not support evolution of schemas that contain pointer-to-member types.

# *Chapter 3*
# Restrictions, Limitations, and Known Problems

The following sections describe restrictions, limitations, and known problems when using Release 6.3. Where possible, they also describe workarounds for the problems.

This section is organized into the following topics:

- Incompatibilities Between Visual C++ 6 and 7.1
- Red Hat Linux 8 Address Space Limitation
- Address-Space Release Facility
- Hostname Resolution and Performance
- Generating Schema for Empty Abstract Classes on Solaris
- Fixing Incorrect Vector Headers
- CMTL Restrictions
- DDML Restrictions
- Java Components of ObjectStore
- Linux Users Must Use Only the Current Linux Thread Libraries
- Following External Links with the Netscape 4.7 Browser

## Incompatibilities Between Visual C++ 6 and 7.1

Because of changes in Microsoft's Visual C++ compiler from version 6 (vc6) to version 7.1 (vc7.1), users who upgrade their applications from vc6 to vc7.1, or who wish to use both versions, may see schema incompatibilities that affect schema generation. The following sections discuss these incompatibilities and their workarounds.

### Differing Support for #pragma pack(pop,N)

Support for `#pragma pack(pop,N)`, where *N* is some alignment, differs between vc6 and vc7.1. In particular, the following set of pragmas results in different alignments between vc6 and vc7.1:

```
#pragma pack(push,l1,2)
#pragma pack(push,l2,4)
#pragma pack(pop,1)
```

On vc6, these pragmas result in a two-byte alignment; whereas on vc7.1 they result in a one-byte alignment. The workaround for this incompatibility is to replace the last pragma with either of two sets of pragmas. The first set ensures one-byte alignment that is compatible with vc7.1:

```
#pragma pack(pop)
#pragma pack(1)
```

The next set ensures two-byte alignment that is compatible with vc6:

```
#pragma pack(pop)
#pragma pack(2)
```

## Differing Support for Integral Extension Types

On vc6, the integral extension types are distinct types. On vc7.1, they are treated as typedefs for regular C++ types. Thus, the following code behaves differently on vc6 and vc7.1:

```
template<class T> class printer {
public:
  static void print(const char* source_name) {
    const type_info& ti = typeid(T);
    printf("in source %s, actual %s\n",ti.name());
  }
};
printer<__int16>::print("__int16");
printer<unsigned __int32>::print("unsigned __int32");
```

On vc6, the output is:

```
in source __int16, actual __int16
in source unsigned __int32, actual unsigned __int32
```

On vc7.1, the output is:

```
in source __int16, actual short
in source unsigned __int32, actual unsigned int
```

Furthermore, the symbol for `print` (or for the virtual function table, if there was one) is different. On vc6, the template argument for `__int16` would be `_F`; on vc7.1, it would be `F`.

If you wish to share code or databases between vc6 and vc7.1 applications, you cannot use integral extension types in any context that appears in the schema, including template instantiations and virtual function table pointers. If you are not sharing code between vc6 and vc7.1 applications, you can continue to use the integral types and the “mangling” will follow the rules for the compiler version you are using.

# Red Hat Linux 8 Address Space Limitation

On Red Hat Linux 8, the default setting for the `vm.max_map_count` kernel parameter (65536) limits the address space to 256 MB. This can create a potential address space limitation. You can change the value of this parameter with the following command line:

```
sysctl -w vm.max_map_count= value
```

To display all kernel parameters, use the following command line:

```
sysctl -a
```

You can also work around the address space limitation by setting the ObjectStore environment variable `OS_8K_PAGE` or `OS_16K_PAGE`. Setting either of these variables has the effect of increasing the page size read by ObjectStore to 8 KB or 16 KB. In some cases, increasing the page size might result in additional lock conflicts because ObjectStore creates locks on a per-page basis. A page size of 8 KB or 16 KB can have greater chances of lock conflicts than a page size of 4 KB.

# Address-Space Release Facility

The ObjectStore address-space release facility is not thread safe. As a result, there is the risk that one thread will release address space that is being used by another, leading to unpredictable behavior and potential database corruption. To protect against this risk, this release of CMTL disables the address-space release in software with which it is linked. It is possible that this protective measure can impose address-space limitations that might prevent a 32-bit CMTL application from being successfully deployed. A more general solution to the problem will be implemented in a future release of CMTL.

# Hostname Resolution and Performance

If ObjectStore cannot resolve a hostname (because, for example, the network was reconfigured or a database was moved to a new domain), application performance can suffer accordingly. If you think that hostname resolution is degrading the performance of your application, contact ObjectStore Technical Support.

# Generating Schema for Empty Abstract Classes on Solaris

The schema generator (`ossg`) cannot generate schema for code that contains empty abstract classes used as virtual base classes on Solaris platforms. As a workaround, `ossg` will emit instructions to add a padding member.

# Fixing Incorrect Vector Headers

In C++, vector headers contain data stored in front of a vector by the C++ run time. Vector headers are not normally visible to user applications. Only vectors allocated with `::operator new` have vector headers; vectors embedded in objects or allocated off the stack do not have vector headers. Also, only vectors of elements that have nondefault destructors have vector headers.

The vector header encodes information that the C++ run time uses when the vector is deleted to determine how to run the destructors on the elements. If the vector header is incorrect when the vector is deleted, the C++ run time is unlikely to run the correct number of destructors on its elements. Running an incorrect number of destructors on the vector could (depending on the behavior of the destructors) result in database corruption or unpredictable application behavior.

Incorrect vector headers can occur under the following conditions:

- The vector was created on an HP-UX 11.00 (64-bit) or Solaris/SPARC 2.8 (64-bit) platform without using the normal form of the ObjectStore overloading of `::operator new`. The following is an example of the normal form:

  ```
  void *p = ::operator new(
    sizeof(foo) * N, db, foo::get_os_typespec(), N);
  ```

- The vector was hetero-relocated from a different platform to an HP-UX 11.00 (64-bit), Solaris/SPARC 2.8, or Linux gcc platform; and the page containing the vector was committed.

Previous to Release 6.0 Service Pack 7, the ObjectStore client automatically fixed what were assumed to be incorrect vector headers during inbound relocation of a page only if the architecture that wrote to the page differed from the current architecture that is reading the page. Starting with Release 6.0 Service Pack 7, ObjectStore provides the following tools for checking if the vector headers are incorrect and for fixing incorrect vector headers:

- Environment variables: `OS_CHECK_VECTOR_HEADERS` and `OS_FIX_VECTOR_HEADERS`
- The following static functions:
  - `objectstore::set_check_vector_headers()`
  - `objectstore::get_check_vector_headers()`
  - `objectstore::set_fix_vector_headers()`
  - `objectstore::get_fix_vector_headers()`
- The `osfixvh` utility

The following sections describe how to use these tools.

## Using Environment Variables

The following Boolean-valued environment variables can be used to check and fix incorrect vector headers:

`OS_CHECK_VECTOR_HEADERS`

`OS_FIX_VECTOR_HEADERS`

To enable checking or fixing of vector headers, set the appropriate variable to `1` (`true`). To disable checking or fixing, set the variable to `0` (`false`). These variables are process-global.

By default, `OS_CHECK_VECTOR_HEADERS` is set to `1` and `OS_FIX_VECTOR_HEADERS` is set to `0`. In this case, the client checks all vector headers for correctness during inbound relocation and raises an exception if it finds an incorrect vector header. This behavior provides the highest degree of safety when dealing with vector headers.

If you set `OS_CHECK_VECTOR_HEADERS` to `0` and `OS_FIX_VECTOR_HEADERS` to `1`, the client silently fixes all vector headers during inbound relocation, regardless of whether they are correct or not. If you set both `OS_CHECK_VECTOR_HEADERS` and `OS_FIX_VECTOR_HEADERS` to `1`, the client checks all vector headers for correctness during inbound relocation, logs a message to `OS_DEBUG_OUT` (by default, `stderr`), fixes the vector headers, and continues.

There is a performance cost when either `OS_CHECK_VECTOR_HEADERS` or `OS_FIX_VECTOR_HEADERS`, or both, are set to `1`. In the worst case, an application can experience up to 30% loss in performance. However, this worst case is the highly unlikely case of a database in which every object is a small 1-element vector without any pointers or vtbls, and the application does nothing except fetch pages. The more likely impact on the performance of an actual ObjectStore application is much less than 30%, perhaps well below 1%.

In any case, you should set the variables at a level of safety that best meets the needs of your configuration. For example, you might want to set `OS_CHECK_VECTOR_HEADERS` to `1` and `OS_FIX_VECTOR_HEADERS` to `0`, with `OS_DEBUG_OUT` set to a file, so that your application stays up and running in the event of an incorrect vector header. In this case, the errors would be logged to a file so that you could determine whether your database was affected by an incorrect vector header. Alternatively, you might decide that you do not need to find out if incorrect vector headers occurred but want any such problems fixed without your intervention. In this case, you would set `OS_CHECK_VECTOR_HEADERS` to `0` and `OS_FIX_VECTOR_HEADERS` to `1`.

If your configuration of platforms and usage precludes the conditions in which incorrect vector headers could occur, or if you have fixed all vector headers that were previously incorrect and have taken steps to ensure that that such problems will not recur, you might decide to set both `OS_CHECK_VECTOR_HEADERS` and `OS_FIX_VECTOR_HEADERS` to `0`.

## Checking and Fixing Vector Headers Programmatically

You can check for and fix incorrect vector headers programmatically, using the following functions:

```
static void objectstore::set_check_vector_headers(
  os_boolean value);
```

```
static os_boolean objectstore::get_check_vector_headers();
```

```
static void objectstore::set_fix_vector_headers(
```

```
  os_boolean value);
static os_boolean objectstore::get_fix_vector_headers();
```

The `set...()` forms of these functions correspond to the `OS_CHECK_VECTOR_HEADERS` and `OS_FIX_VECTOR_HEADERS` environment variables, which are described in the previous section. Passing `true` (nonzero) as the *`value`* argument enables checking or fixing; `false` (`0`) disables checking or fixing.

You can use the `get...()` forms of the functions to determine whether vector-header checking or fixing is enabled.

The considerations that apply to the use of the environment variables also apply to the functions; see Using Environment Variables on page 36.

## Using osfixvh to Check and Fix Vector Headers

You can use the command-line utility `osfixvh` to check for and fix incorrect vector headers. To use this utility to detect and fix existing incorrect vector headers, perform the following steps:

1. Make sure that you have installed ObjectStore Release 6.3.
2. To check for incorrect vector headers without fixing them, execute the following command line:

   ```
   osfixvh -verify_only database_name
   ```

   This command line generates a report similar to the following example:

   ```
   % osfixvh -verify_only foo.db
     An incorrect vector header was found during inbound
     relocation at 0xe1804890. The source architecture is an HP
     with aCC. The address corresponds to offset 0x4890 within
     cluster #0, segment #0 of database "/bar/foo.db".
   %
   ```

3. To fix incorrect vector headers in a database, execute the following command line:

   ```
   osfixvh -fix database_name
   ```

   When it fixes an incorrect vector header, this command line generates a report similar to the following example:

   ```
   % osfixvh -fix foo.db
     An incorrect vector header was fixed during inbound
     relocation at 0xe1804890. The source architecture is an HP
     with aCC. The address corresponds to offset 0x4890 within
     cluster #0, segment #0 of database "/bar/foo.db".
   %
   ```

The `osfixvh` utility does not automatically check for and fix incorrect vector headers in schema databases and affiliated databases. You must check and fix such databases separately. If you need assistance, please contact ObjectStore Technical Support.

# CMTL Restrictions

The following sections describe restrictions when using the Release 6.3 version of CMTL.

## Configuring CMTL from XML on Linux Platforms

On Linux platforms, if your application configures CMTL from an XML-based configuration stream, you must specify the `ios::in` flag when creating the `ifstream` object that is passed as an argument to `os_cache_pool_manager_configuration::create from_xml_stream()`.

On Linux platforms, `ifstream` is not opened for reading by default. The workaround is to explicitly pass an argument to the `ifstream` constructor telling it to open the stream for reading, as in the following example:

```
ifstream xml_file(config_info, ios::in);
```

The `ios::in` flag tells the constructor to open the stream for reading. Note that adding this flag to the constructor is required only on Linux platforms.

## Setting the commit_if_idle attribute

The CMTL cache pool attribute `commit_if_idle` value default is `true` for read-only caches (`true` is also the value for update caches). Setting the `commit_if_idle` attribute value to `false` will result in a shutdown timing problem in the CMTL virtual transaction manager thread. If you have explicitly set the `commit_if_idle` value to false in your cache pool configuration, you need to modify it to avoid this timing issue.

# DDML Restrictions

The following restrictions apply to the Release 6.3 version of DDML:

- DDML is not supported on Solaris 64-bit (sol64) platforms, HP 64-bit platforms, Linux 64-bit platforms, or AIX.
- The following DDML warning message can be safely ignored:

```
Warning: com.odi.osdm.JosdmCPlusPlus.charPmap is a static field of
type com.odi.util.OSSmallMap which might refer to a persistent object.
If this field does refer to a persistent object it must be user
maintained.
```

# Java Components of ObjectStore

The following sections describe the restrictions concerning the Java components of ObjectStore — for example, OSJI and JMTL.

## Upgrading Pre-6.0 OSJI Databases

The `osjidump` utility packaged with this release of ObjectStore is not designed to dump pre-6.0 OSJI databases. If you want to upgrade a pre-6.0 OSJI database using the ObjectStore dump/load facility, contact Technical Support to obtain the most up-to-date version of the ObjectStore `osdump` utility.

Also, see the *ObjectStore Migration Guide* at `www.progress.com/realtime/techsupport/documentation/objectstore`.

## References to Objects in Destroyed Clusters, Segments, and Databases

If an object has a reference to another object in a cluster, segment, or database that has been destroyed, any attempt to access the object containing the reference , or any object on the same page in the database, will throw an exception. There may be cases where writing a page in the database that contains such references will also throw an exception. In previous releases, an exception was not thrown in this situation. If there is the possibility that your pre-Release 6.3 database contains references to objects in clusters, segments, or databases that have been destroyed, you should run the ObjectStore `osverifydb` utility on the database with the `-illegal_pointer_action` option before accessing the database with an Release 6.3 application. The result of running `osverifydb` will be to resolve references to objects in the destroyed placement as null values instead of throwing an exception.

Release 6.3 applications should use `Cluster.destroy()`, `Segment.destroy()` and `Database.destroy()` very carefully to prevent this situation from occurring.

## Not Supported on 64-Bit Platforms or AIX

The Java components of ObjectStore are not supported on 64-bit platforms or on AIX for this release.

## Running Java Browser on UNIX

Running the Java browser on UNIX platforms from Exceed clients have the following problems:

- The Message Box is Empty, because the Windows Manager does not resize the Message Box correctly. You have to resize the Message Box so the content will appear.
- Menu position is incorrect. You need to resize the main window so the menu position is correct.

## Use of JDK 1.5

The default options for the javac compiler in JDK 1.5 (5.0) target 1.5 and greater VMs. This means that classes compiled with this compiler with the default target are incompatible with OSJI. If you are going to develop OSJI applications with JDK 1.5, you must use the `-target 1.4` compiler switch.

In any case, it is possible to *run* OSJI applications developed with JDK 1.4 or lower JDKs with JDK 1.5. Complete JDK 1.5 support will be introduced in a future OSJI release.

## .Use of JDK 1.4 on Solaris

To ensure compatibility between ObjectStore and JDK 1.4 on Solaris platforms, you must set the `LD_PRELOAD` environment variable. C shell users can set this variable with the following command line:

```
setenv LD_PRELOAD libosopdel.so
```

Bourne or Korn shell users can use the following command line:

```
LD_PRELOAD=libosopdel.so ; export LD_PRELOAD
```

## Use of JDK 1.4 on HP-UX

To ensure compatibility between ObjectStore and JDK 1.4 on HP-UX 32-bit platforms, you must set the `LD_PRELOAD` environment variable. C shell users can set this variable with the following command line:

```
setenv LD_PRELOAD libos.sl:libosth.sl
```

Bourne or Korn shell users can use the following command line:

```
LD_PRELOAD=libos.sl:libosth.sl ; export LD_PRELOAD
```

## Level One Integration

Level one integration (JTA transaction integration) is supported for BEA WebLogic Server 8.1.

## Terminated Sessions Do Not Release All Resources

Currently, a terminated session does not release all of its resources until each thread in a session explicitly calls the `Session.leave()`, `Session.terminate()`, `Objectstore.shutdown()`, `Session.create()` or `Session.join()` method.

Terminating a session continues to make all threads leave the session for all other purposes, except for releasing the session's resources.

The `Session.leave()` method has been modified so it can be called by threads, even if they are not joined to a session. Previously, an exception would be thrown.

Applications must ensure that all threads explicitly leave sessions that have been terminated, so that session resources are freed. If this is not done, resources allocated to sessions are not released, resulting in a possible `com.odi.AddressSpaceFullException` being thrown when trying to create a new session.

## Threads Are Not Being Automatically Joined to Nonglobal Sessions

Threads that do not belong to a session are not automatically joined to a nonglobal session when they should be. Until this problem is resolved, applications cannot rely on session absorption to make a thread that accesses persistent objects join the

appropriate session. As a work around, applications can explicitly join each thread to a session by calling `Session.join()` or using a global session.

More specifically, API methods whose only session-implying arguments are persistent objects require that the calling thread already belong to the same session as the persistent objects. This applies to nonglobal sessions. This restriction will be lifted in a future release. The methods affected by this restriction include the following:

- `ObjectStore.deepFetch()`
- `ObjectStore.destroy()`
- `ObjectStore.dirty()`
- `ObjectStore.evict()`
- `ObjectStore.export()`
- `ObjectStore.fetch()`
- `ObjectStore.isDestroyed()`
- `ObjectStore.isExported()`
- `ObjectStore.migrate()`
- `Persistent.deepFetch()`
- `Persistent.destroy()`
- `Persistent.dirty()`
- `Persistent.evict()`
- `Persistent.fetch()`
- `Persistent.isDestroyed()`

## Schema Write-Lock Conflicts Might Occur Immediately Following Schema Installation

Users running ObjectStore applications concurrently against a database might encounter schema segment write-lock conflicts during a transaction that immediately follows a transaction in which the schema was installed.

To work around this problem, run a small dummy transaction immediately after the transaction that installed the schema. The dummy transaction must use the database, such as looking up a database root. Using the database validates the schema and prevents potential future write lock conflicts.

## Use of ossevol Utility

Do not use the `ossevol` utility on databases created with the Java interface to ObjectStore (OSJI). Also, do not use the C++ API for this utility on OSJI databases.

You can use the OSJI `Database.evolveSchema()` method to evolve the schema in an OSJI database.

## Use of osgc and oscompact Utilities

You should not run the `osgc` or `oscompact` utilities against databases that are currently opened with applications that retain references to non-exported objects. To prevent possible corruption you should close the databases before running either utility.

## Hosted Pathname Syntax Might Require Setting of Environment Variable

If the directory specified in `OS_LIBDIR` uses the hosted pathname syntax (host:/ dir) and the pathname syntax for that directory has the opposite style of slash from the one for local pathnames on the client (that is, Windows and UNIX), you must set the `OS_META_SCHEMA_DB` environment variable to the pathname of the metaschema database. That database is named `metaschm.db`, and its default location is in the lib subdirectory of `OS_ROOTDIR`.

## Transient Segmentation Violation Errors

On UNIX platforms there are some known interaction problems between the Java VM and OSJI. If you encounter any *transient segmentation violations* errors, choose a heap size and set both the initial and maximum heap size to that value on the command line. For example, if you choose a 64 MB heap size, specify both `-Xms64m` and `-Xmx64m` as arguments to the java command.

## Applets

ObjectStore is a Java application that uses C++ native methods. Consequently, you cannot use ObjectStore in an applet other than through the Sun JDK Appletviewer application.

# Accessing Multithreaded OSCI Libraries on Solaris

For information about accessing multithreaded ObjectStore C++ interface (OSCI) libraries on Solaris, see Compiling and Linking on Solaris 2 in Chapter 4 of *Building ObjectStore C++ Applications*. If your OSJI application is multithreaded and you want to access an OSCI library to call APIs that are not available in OSJI, contact ObjectStore Technical Support for recommendations.

# Linux Users Must Use Only the Current Linux Thread Libraries

On Linux platforms, ObjectStore 6.3 works with only the current version of the Linux thread libraries. You must ensure that you do not set the Linux environment

variable, `LD_ASSUME_KERNEL`, to a kernel version that does not use the current version of the Linux thread libraries. If you do, ObjectStore no longer functions properly.

You can set the `LD_ASSUME_KERNEL` variable to a different Linux kernel version only if the new kernel version uses the current version of the Linux thread libraries. Setting `LD_ASSUME_KERNEL` to anything less than `2.4.20` causes problems.

# Following External Links with the Netscape 4.7 Browser

If you are using the Netscape 4.7 browser to view the ObjectStore documentation and you click on a link to visit an external website, you will not be able to use the WebHelp facility to navigate back to the ObjectStore documentation. To work around this problem, right-click on the link and select Open in New Window from the menu.

# *Chapter 4*
# New and Changed Features in Release 6.2

ObjectStore Release 6.2 includes the following new and changed features:

- Support for Java Data Objects (JDO)
- Schema Evolution and Database Compaction
- JMTL Enhancements in 6.2
- Database Verification Optimization
- Stream Specification
- Solaris PSR Default Changes

## Support for Java Data Objects (JDO)

ObjectStore for Java provides a complete implementation of the Java Data Objects (JDO) 1.0.1 specification. Creating your application with JDO is an alternative to using the native ObjectStore interface for storing persistent data.

The ObjectStore storage engine in conjunction with the ObjectStore JDO interface lets you quickly read or modify portions of your persistent data. You are not required to read in all persistent data when you just want to look at a subset. This reduces start-up and transaction commit times and lets you run much larger Java applications without increasing the amount of memory or swap space on the system.

ObjectStore JDO provides most of the features available to the native ObjectStore Java interface. You need JDK 1.4 or later to use JDO with ObjectStore. For details, see the *ObjectStore Java API User Guide*.

## Schema Evolution and Database Compaction

Schema evolution and database compaction have been completely reimplemented in ObjectStore 6.2. They are much faster. ObjectStore 6.2 schema evolution has been measured to be 100 times faster than ObjectStore 6.0 on large databases. Schema evolution now takes advantage of modern hardware resources such as multiple

CPUs and large main memory, and is aware of the non-uniform memory access speed caused by multiple levels of hardware caching.

Schema evolution and database compaction use much less address space and are now very unlikely to run out of address space in the persistent storage region. Combined with the increased speed, it should now be practical to use schema evolution and database compaction on large databases, up to at least 10 GB.

The declustering effect of schema evolution has been eliminated. Schema evolution no longer changes the order of objects in persistent storage when it has to move objects because the size of some object has changed. The new implementation maintains the order of objects within a cluster. Where the page boundaries fall may change, but objects that used to be close together remain close together, on adjacent pages if not on the same page.

There have been extensive changes to the

- `ossevol` and `oscompact` command-line tools
- C++ `os_schema_evolution` and `os_compact` classes
- Java `Database.evolveSchema()` method

Read the updated documentation before using them.

Some forms of complex schema evolution are not implemented yet. Consult ObjectStore technical support before doing complex schema evolution. Some of these features will be added in a future release. The specific restrictions are:

- There is not yet a replacement for subtype selectors (also called *instance reclassification*).
- There is not yet automatic recognition of moving a data member between base and derived classes.
- Pointer-to-member where the target class evolves is not yet supported. The pointer's value will become garbage.
- Unions in databases being evolved or compacted continue not to be supported, as in previous versions of ObjectStore 6.x.
- There could be problems if there is a cluster whose size is close to the maximum (2 GB).
- If you think you need to write a transformer function that calls `os_schema_evolution::get_unevolved_object()`, you must adopt a different strategy. Contact ObjectStore technical support.

A machine performing schema evolution or compaction must have the same byte and bit order as the machine that wrote the pages in the database whose schema is being updated or whose data is being compacted. This applies when you are using

- `ossevol` and `oscompact` utility
- C++ `os_schema_evolution::evolve()` function and `os_compact::compact()` function
- Java `Database.evolveSchema()` function

For example, on a SPARC machine running Solaris, you cannot perform schema evolution on a database that was written on an Intel x86 machine running Linux.

# JMTL Enhancements in 6.2

With the JMTL proxy generator tool, ProxyCodeGenerator users can automatically create code that handles an application's JMTL transaction logic. This allows you to separate the application's business logic from the JMTL transaction logic. This tool can be used for JMTL applications that do not use an application server or that use session beans to perform JMTL transactions.

In addition, JMTL now provides full bean managed persistence (BMP) support for entity beans for all application servers (Integration By Proxy). This is accomplished with the JMTL proxy generator tool JMTLBmpAdapterGenerator. This tool generates integration code based on your entity bean, the bean descriptor, and the JMTL deployment descriptor. This level of integration eliminates the need for you to develop JMTL transaction logic. Complete information about using the proxy generator is in the *JMTL User Guide*.

# Database Verification Optimization

The new fast mode for the `osverifydb` utility uses techniques optimized for performance purposes. Run `osverifydb` with the `-F` option to specify fast mode. Alternatively, if you set the environment variable `OS_OSVERIFYDB_FAST` to true (non-zero), `osverifydb` will run in fast mode by default. You can use the environment variable `OS_OSVERIFYDB_BUFFER_SIZE` to specify the maximum size of the transient buffer used to optimize locality of reference by `osverifydb` in fast mode. The default size is 512 MB. The fast mode for `osverifydb` is not compatible with the following options:

- `-ignore_references`
- `-illegal_pointer_action`
- `-o`
- `-v`
- `-whohas`

# Stream Specification

ObjectStore now uses the standard specification for streams in place of the classic specification. Where your code uses the following:

```
#include <iostream.h>
#include <fstream.h>
```

You must change it to this:

```
#include <iostream>
#include <fstream>
```

# Solaris PSR Default Changes

The default size and address of the persistent storage region (PSR) on Solaris systems has changed. The new defaults are:

- PSR default size
  - 32-bit Solaris — `512` MB
  - 64-bit Solaris — `768` MB

  The previous default was `192` MB (`0xC000000`).
- PSR default address
  - 32-bit Solaris — Ends at the same address where it used to (`0xEE000000`).
  - 64-bit Solaris — Starts at 32 GB.

  The previous default PSR starting address was `0xE2000000`. The previous default PSR ending address was `0xEE000000`.

If you want to revert to the previous defaults, or specify some other size or starting address, use the `OS_AS_SIZE` and `OS_AS_START` environment variables. See *Managing ObjectStore*. To revert to the previous defaults, set the following environment variables to the values specified:

- `OS_AS_SIZE=0xC000000`
- `OS_AS_START=0xE2000000`

# *Chapter 5*
# New and Changed Features in Release 6.1 Service Pack 2

The following sections describes features of ObjectStore that were added or changed for Release 6.1 Service Pack 2.

## Programmatic Support for Backup and Restore Operations

The ObjectStore API has been extended for this release to include programmatic support for backup and restore operation. Using the following classes, you can write applications to perform operations similar to those you can perform with ObjectStore command-line utilities:

- `os_archiver`
- `os_archiver_options`
- `os_backup`
- `os_backup_options`
- `os_recover`
- `os_recover_options`
- `os_restore`
- `os_restore_options`

The classes and their member functions are described in the *ObjectStore C++ API Reference*.

## New Pathname Encoding Function

A new function, `objectstore::set_pathname_encoding()`, has been added to the C++ API for this release. The function enables you to specify a character set encoding to override the default encoding, as specified by the platform. The signature is:

```
void objectstore::set_pathname_encoding(const char* encoding);
```

where *encoding* can be one of the following:

| *Valid encoding values* | *Meaning* |
|---|---|
| ASCII | 7-bit ASCII |
| CP1252 | Microsoft Code Page 1252 (US English) |
| CP932 | Microsoft Code Page 932 (Japanese) |
| EUCJP | Extended UNIX Code (Japanese) |
| UTF8 | UCS Transformation Format 8 |
| NONE | No encoding translation; values 0x01 through 0xFF are passed through without modification. |

If you supply an invalid encoding value, calling this function will result in the err_invalid_pathname_encoding exception.

The objectstore::set_pathname_encoding() function must be called before calling objectstore::initialize() or objectstore::initialize_for_sessions().

The objectstore::set_pathname_encoding() function is similar to the OS_PATHNAME_ENCODING environment variable; for more information about the variable, see *Managing ObjectStore*, Chapter 3.

# Changes to the Schema Evolution API

The following changes have been to the schema evolution for this release:

- New functions:
  - os_class_type::get_dispatch_table_pointer_offset_other_compiler()
  - os_class_type::has_dispatch_table_other_compiler()
  - os_object_cursor::release_address_space()
  - os_schema_evolution::set_address_space_release_interval()
- The addition of the *layout_only* argument to the os_dbutil::compare_schemas() function
- New options to the following ObjectStore utilities:
  - osscheq -layout_only
  - ossevol -address_space_release_interval

For information about the new and changed functions, see the *ObjectStore C++ API Reference*. For information about the osscheq and ossevol utilities, see *Managing ObjectStore*.

# New Platform Support: HP-UX and AIX

This release includes support for the HP-UX and AIX platforms.

# Multi-process Dump and Load

This release supports the use of multi-process dump and load on all platforms, including Windows. For more information, see *Managing ObjectStore*

# Performance and Metadata Checking

ObjectStore performs certain checks whenever a page is fetched from the database and brought into the cache (inbound relocation) or is committed or evicted from the cache (outbound relocation). These checks involve testing the consistency between different pieces of metadata for the same page. Previously, relocation and the accompanying checking would be skipped whenever ObjectStore could determine that it was not needed for normal functional reasons.

Starting with this release, ObjectStore no longer skips relocation in the interest of additional metadata checking. For example, ObjectStore would previously skip outbound relocation for pages that do not contain pointers. This is no longer the case. Starting with this release, the default behavior is to perform relocation and checking. This default behavior allows ObjectStore to detect database corruption sooner than would be the case without the checking. When such checking occurs during inbound relocation, it can prevent an application from using a corrupt page. During outbound relocation, metadata checking can prevent database corruption from occurring in the first place.

However, the new default checking can add significant overhead in certain circumstances, especially during outbound relocation when pages that have never contained pointers are committed or evicted. As a result of the added checking, user applications may experience lowered performance in comparison with application performance in previous releases of ObjectStore.

To enable you to turn off the added checking and restore performance to its 6.1 levels, this release of ObjectStore provides the following new, Boolean-valued environment variables:

- `OS_ALLOW_OUTBOUND_RELOC_SKIPPING`
- `OS_ALLOW_INBOUND_RELOC_SKIPPING`
- `OS_SKIP_INBOUND_VERIFY_TAGS_INDEX`
- `OS_SKIP_FREE_SPACE_CONSISTENCY_CHECK`

To disable checking, set these environment variables to true (`1`). The following paragraphs describe each variable:

## OS_ALLOW_OUTBOUND_RELOC_SKIPPING

Default: false (`0`)

When this variable is set to true (1), modified pages that have never had pointers on them will skip outbound relocation and all associated checking -- that is, no metadata checking will occur for such pages during outbound relocation. Setting this variable to true enables users to get the performance advantage of skipping outbound relocation at the expense of losing the extra checking.

## OS_ALLOW_INBOUND_RELOC_SKIPPING

Default: false (`0`)

When this variable is set to `true` (`1`), pages that do not contain any data requiring inbound relocation will, under non-hetero conditions -- that is, the page architecture matches the client architecture -- skip inbound relocation. This skipping only occurs for pages that have no pointers, virtual base pointers, or virtual function table pointers. If inbound relocation is skipped, all metadata checks performed during inbound relocation will also be skipped. Setting this variable to true enables users to get the performance advantage of skipping inbound relocation at the expense of losing the extra checking.

A previous release of ObjectStore disabled inbound relocation skipping by default to allow checking for vector headers. If you want to skip inbound relocation and the associated metadata checking, you must set these vector-header variables in addition to `OS_ALLOW_INBOUND_RELOC_SKIPPING`, as follows:

```
setenv OS_ALLOW_INBOUND_RELOC_SKIPPING 1
setenv OS_CHECK_VECTOR_HEADERS 0
setenv OS_FIX_VECTOR_HEADERS 0
```

If you are using the `ksh`, the command lines would be:

```
export OS_ALLOW_INBOUND_RELOC_SKIPPING=1
export OS_CHECK_VECTOR_HEADERS=0
export OS_FIX_VECTOR_HEADERS=0
```

In Windows, you can either modify the environment settings in the control panel or use the following command lines:

```
set OS_ALLOW_INBOUND_RELOC_SKIPPING=1
set OS_CHECK_VECTOR_HEADERS=0
set OS_FIX_VECTOR_HEADERS=0
```

Note that, by default, `OS_CHECK_VECTOR_HEADERS` is set to `1` and `OS_FIX_VECTOR_HEADERS` is set to `0`.

## OS_SKIP_INBOUND_VERIFY_TAGS_INDEX

Default: false (`0`)

When this variable is set to true (`1`), inbound relocation on pages that are not yet relocated in the client cache will skip the new tags index check. Setting this variable to true enables users to get the performance advantage of not checking the tags index during inbound relocation at the expense of losing the extra checking. Note that

pages that skip inbound relocation will also skip the tags index check regardless of the setting of this variable. In other words, the effect of this variable is not orthogonal to the setting of the `OS_ALLOW_INBOUND_RELOC_SKIPPING` variable.

### OS_SKIP_FREE_SPACE_CONSISTENCY_CHECK

Default: false (`0`)

When this variable is set to true (`1`), the SPFS metadata is not tested for self-consistency when a page is prepared for relocation for the first time. (ObjectStore uses SPFS metadata to control object allocation.) Note that pages for which inbound relocation is skipped will still have this check done, since the check doesn't require any iteration over the tags. In other words, the effect of this variable is orthogonal to the setting of the `OS_ALLOW_INBOUND_RELOC_SKIPPING` variable.

## ossg's Default Front-End Parser

Previous to Release 6.1, the ObjectStore schema generation utility (`ossg`) required you to use the `-edgfe` option to enable an improved front-end parser that provides improved language support, including better support for nested classes and templates. Starting with Release 6.1, this parser is the default, and the `-edgfe` option is no longer recognized. If you believe you need to use the pre-6.1 Release front-end parser, contact ObjectStore technical support.

The new default front-end parser requires you to use the `OS_MARK_SCHEMA_TYPE` and `OS_MARK_SCHEMA_TYPESPEC` macros, as documented in the *ObjectStore C++ API Reference*, Chapter 4 ("System-Supplied Macros"), and in *Building ObjectStore C++ Applications*, Chapter 2 ("Working with Source Files"). For a detailed description of the `ossg` utility, see "`ossg`: Generating Schemas" in *Managing ObjectStore*, Chapter 4 ("Utilities").

Note

If your makefile invokes `ossg` with the `-edgfe` option, you must edit the makefile to remove this option.

## Improved Support for Failover

Release 6.1.2 provides improved support for failover, as follows:

- If you are using failover that is built into ObjectStore, you can configure failover so that the primary server and the secondary (or standby) server share the load during normal ObjectStore operations. In the event of failover, the secondary server assumes the full load.
- If you are using ObjectStore on the Sun Clusters 3.0 operating system (see Support for the Sun Clusters 3.0 on page 55), you can configure ObjectStore to use the support for failover provided by the operating system as an alternative to ObjectStore-managed failover.

For detailed information about failover, see *Managing ObjectStore*, Chapter 6 ("High Availability of Data").

ObjectStore's support for failover also allows you to perform rolling upgrades when installing a new release of ObjectStore. A rolling upgrade allows you to install a new release of ObjectStore without interrupting service to clients. For more information, see "`osconfig`: Configuring ObjectStore" in *Managing ObjectStore*, Chapter 4 ("Utilities").

# Replication API

Release 6.1.2 introduces the following classes for managing database replication within an application:

- `os_replicator`
- `os_replicator_options`
- `os_replicator_statistic_info`

For more information about these classes, see `os_replicator` in *C++ API Reference*, Chapter 2 ("Class Library").

# Defragmenting ObjectStore Databases

Release 6.1.2 provides support for defragmenting ObjectStore databases. This support includes:

- The `os_database::get_fragmentation()` function, which returns statistics on database fragmentation.
- The new `-f` option to the `ossize` utility. This option causes `ossize` to call the `os_database::get_fragmentation()` function.
- New server parameters that can be used to prevent fragmentation:
  - `Cluster Growth Policy`
  - `Database File Growth Policy`
  - `RAWFS Partition Growth Policy`
- New functions that can also be used to prevent fragmentation:
  - `os_cluster::set_size()`
  - `os_database::set_size()`
  - `os_database::set_size_in_sectors()`
- You can access these functions from the command-line with the `osdbcontrol` utility, using two new options: `-cluster_size` and `-size`

For information about defragmenting ObjectStore databases, see "Managing Database Fragmentation" in Chapter 1 ("Overview of Managing ObjectStore") of *Managing ObjectStore*.

# Support for the Sun Clusters 3.0

Release 6.1.2 supports Sun Clusters 3.0. This support enables ObjectStore to use the failover support that is built into the Sun Clusters operating system. For more information, see Improved Support for Failover on page 53.

# Architecture Sets for Release 6.1

Release 6.1 supports the following categories of architecture sets for use with the `ossg -arch` option when neutralizing schema:

- Standard architecture sets
- Versioned architecture sets
- User-defined architecture sets

In addition to the above architecture sets, Release 6.1 supports two other sets for use when neutralizing schema that includes virtual base classes; see Neutralizing Virtual Base Classes on page 56.

Note also that `ossg` has a new option, `-showsets`, which lists all architecture sets and their contents.

## Standard Architecture Sets

The standard architecture sets meet the needs of most applications that require neutralization. These sets are:

- `all32`
- `all64`

Note that the `all` set is no longer supported. Support for this set has been removed because `ossg` no longer supports neutralization across all 32-bit and 64-bit platforms for applications that use the collections facility. If you use the `all` set in a makefile or any other scripts, you must remove it and substitute either `all32` or `all64`.

## Versioned Architecture Sets

Starting with Release 6.1, you can use a versioned architecture set. A versioned architecture set contains only those platforms that are supported on a particular release of ObjectStore. For example, `all32_610` contains only those 32-bit platforms that are supported on Release 6.1. Unlike the contents of standard architecture sets, the contents of versioned architecture sets do not change from release to release.

For more information, see "Versioned Architecture Sets" in *Building ObjectStore C++ Applications*, Chapter 5 ("Building Applications for Multiple Platforms"). You can list all architecture sets and their contents by invoking `ossg` with the `-showsets` option, as described in "`ossg`: Generating Schemas" in *Managing ObjectStore*, Chapter 4 ("Utilities").

## User-Defined Architecture Sets

Release 6.1 allows you to define your own architecture sets, which can be specified as arguments to `ossg`'s `-arch` option. To define an architecture set, use the `OS_USER_ARCH_SET` environment variable, as described in *Managing ObjectStore*, Chapter 3 ("Environment Variables").

# Neutralizing Virtual Base Classes

If you are neutralizing schema against all 32-bit platforms, and the schema contains virtual base classes that use other virtual base classes, the `ossg` schema generator for Release 6.1 will prompt you to replace the virtual base classes with forced-order base classes. Replacing virtual base classes with forced-order classes ensures that the allocation order for the virtual base classes is the same across all 32-bit platforms.

Forced-order base classes are needed because of differences in the way different platforms allocate the virtual base classes. On linux3 (gcc3) platforms, virtual base classes are allocated in inheritance order. Currently, all other 32-bit platforms use post-traversal order. This difference results in a different layout order that requires neutralization.

If you are neutralizing against all 32-bit platforms *except* linux3, and your schema contains virtual base classes that use virtual base classes, you can specify the `all32vbtrav` architecture set to prevent the need for forced-order classes during neutralization. (The use of forced-order classes can increase code size.) For more information, see "Neutralizing the Allocation Order of Virtual Base Classes" in *Building ObjectStore C++ Applications*, Chapter 5 ("Building Applications for Multiple Platforms"). For more information about architecture sets, see Architecture Sets for Release 6.1 on page 55.

# Changing the Windows Registration Location

Release 6.1 enables you to change the Windows registry location that is used by ObjectStore. Changing the registry location is especially useful when you have embedded ObjectStore in an application. By changing the registry location, you can prevent another application that also uses ObjectStore from overwriting information in the registry location that your application uses.

You can use the following to change the registry location:

- `os_authentication` class — For more information about this new class, see the description of the class in Chapter 2 ("Class Library") of the *C++ API Reference*.
- `osserver -r` — For more information about the new `-r` option to `osserver`, see "osserver: Starting the Server" in Chapter 4 ("Utilities") of *Managing ObjectStore*. Note that the `oscmgr6` utility for starting the cache manager now has the same new `-r` option.
- `OS_REMOTE_AUTH_REGISTRY_LOCATION` — For more information about this new environment variable, see "osserver: Starting the Server" in Chapter 3 ("Environment Variables") of *Managing ObjectStore*.

For information about changing the registry location on NT machines, see "Setting the Registry Location for ObjectStore (Windows Only)" in *Managing ObjectStore*, Chapter 8.

# Preventing Excessive Page Faulting

Release 6.1 provides the following functions of the `objectstore` class that enable you to prevent excessive page faults by disabling address markers:

- `objectstore::get_asmarkers_useless()`
- `objectstore::set_asmarkers_useless()`

Note, however, that if you disable address-space markers in an application for which address space consumption is critical, the application runs the risk of running out of address space.

You can also use a new environment variable, `OS_ASMARKERS_USELESS`, to disable address markers. For information about the environment variable, see the description in Chapter 3 ("Environment Variables") of *Managing ObjectStore*. The functions are described in Chapter 2 ("Class Library") of the *C++ API Reference*.

# Changes to Dump and Load

Release 6.1 includes enhanced versions of the ObjectStore `osdump` and `osload` utilities. Changes to these utilities include:

- Support for multiple processors in both `osdump` and `osload`. The option to specify the use of multiprocessors is `-pr`.
- Support for the ability to restart the `osload` process.
- The mechanism for generating the source code for loader applications is moved from `osdump` to `osload`. This option to specify this operation remains `-emit`.
- Elimination of the `-ds` option to dump the database schema. The database schema is always dumped in Release 6.1.

For more information, see `osdump` and `osload` in Chapter 4 ("Utilities") of *Managing ObjectStore*.

## Using Dump and Load to Migrate Databases

If you plan to use `osdump` and `osload` to migrate databases from Release 5.1 or release 6.0, you should always check the ObjectStore Technical Support Web site (`www.objectstore.net/support`) for the most recent version of each of these utilities.

The dump and load subsystem has undergone considerable revision since the last release. Data files dumped with prior releases of `osdump` are not compatible with current and future releases of `osload`. The databases from which those files were generated will need to be re-dumped with an updated version of `osdump`.

Updated versions of `osdump` for ObjectStore 5.1 and ObjectStore 6.0 can be obtained by contacting ObjectStore Technical Support. If you need to run `osdump` or `osload` with more than one process under any version of Microsoft Windows, you need to obtain updated versions from support as well. If you need to upgrade an OSJI database on any platform you must obtain updated libraries as well.

Applications built or linked with `libosdump` or `libosload` will at the very least need to be recompiled and relinked to take advantage of recent bug fixes. The following APIs have changed as well and may require code changes for some applications:

- `os_Database_table` has changed to allocate at the cluster level rather than the segment so any function calls which used to take an `os_Segment` as an argument will now take an `os_Cluster`.
- `os_Database_table::insert()` has been replaced with explicitly named function calls. This eliminates the confusion of 12 different overloadings of the `insert()` function.
- `os_Loader::load()` has been changed to `os_Loader::start_load()`.

## Resumption of osload in the Event of Failure

In the event of failure during load, do not remove any temporary databases and simply re-start `osload`. The `osload` application will look for a current work database and attempt to resume based on the state of the found work database. In general this is only useful if the failure is due to hardware or user failures, such as out of disk space, network failure, or user interruptions.

## Easier Schema Generation for osload

The updated process to generate a schema specific `osload`:

1. Generate a 6.1 application schema; for example, `my_new_schema.adb`
2. Emit the `osload` specializations required for that schema; for example:

   ```
   osload -emit my_new_schema.adb
   ```

3. Compile the new `osload` with the files generated in step 2. You will need to update the makefile with your application specific libraries and headers.

### Simplified osload Usage

The usage of `osload` has changed and it no longer requires you to input the dumped file names. By default `osload` will look for a file named `db_table.dmp` in the current directory and, using that master file, it builds the work file list. If the files are located in a directory other than the current directory the `-dir` switch must be used.

## Performance Improvements

New features of `osdump` and `osload` in ObjectStore 6.1 and post-ObjectStore 5.1.5 include performance and scalability improvements. General performance improvements require no additional steps, while scalability improvements are achieved by running `osdump` or `osload` with more than one process. To do so use the `-pr` switch and provide the number of process to do the work (for example. `-pr 2`).

## Note on Multi-process osdump and osload

There is a fair amount of overhead necessary to manage the child processes in the 6.1 release so it may at times be slower than running `osload` with a single process. (It is expected that this overhead will be reduced in the future). The dump and load work is partitioned among child processes by the segments in a supplied database. In general, if the segments to be dumped or loaded are large (i.e., greater than 100 MB) and you have more than one in a given database, you should use multi-process `osdump` or `osload`. On the other hand, if you have a database with 2000 segments of approximately 5 MB each or a database with one segment approximately 2GB in size, you should use osdump or osload with a single process.

## New Macro for Functions Used in Queries

Release 6.1 provides a new macro, `OS_MARK_QUERY_FUNCTION_WITH_NAMESPACE()`, for use in schema source files to include a query function in the schema. Use this macro instead of the `OS_MARK_QUERY_FUNCTION()` macro when the function is a member of a class that is declared in a namespace. For more information about the `OS_MARK_QUERY_FUNCTION_WITH_NAMESPACE()` macro, see the *C++ Collections Guide and Reference*, Chapter 9.

## Optimizing Collections

Previous to Release 6.1, you could optimize the creation of transient sets, lists, and cursors by manually adding an optimization flag to the constructor. The flag would instruct ObjectStore to use hard pointers instead of soft pointers for the constructed type.

Release 6.1 introduces the following changes for optimizing transient sets, lists, and cursors:

- The `os_cursor::optimized` flag is no longer needed to construct an optimized transient cursor. By default, all transient cursors use hard pointers.
- Two new defines have been added for the release: `_OS_COLL_LIST_OPTIMIZE` and `_OS_COLL_SET_OPTIMIZE`. By setting these defines in your application or on the compile line, users can globally change their applications to use transient lists and sets that are based on hard pointers, without having to specify the optimization flags in the constructors.

For more information about these optimizations, see "Compiling for Collections Optimization" in *C++ Collections Guide and Reference*, Chapter 2.

# Compiling Single-Threaded Applications on Solaris

When using the Solaris compiler to compile single-threaded applications, it is no longer necessary to use the `-mt` option. You must use this option when compiling a Release 6.1 application only if your application uses multiple threads. Refer to the Solaris documentation for more detailed information about the `-mt` option.

Note that, if you want a Release 6.1 application to use a Release 6.0 cache manager, you must compile the application with the `-mt` option, even if it is single-threaded. In this situation, however, the performance of the client/cache interface will be the same as for a 6.0 application that uses a 6.0 cache manager — not as fast as for a 6.1 application that uses a new 6.1 cache manager. For best performance and to avoid the overhead of linking with the ObjectStore thread-safe library (`libosth`), 6.1 single-threaded applications should use the 6.1 cache manager.

# objectstore::export() Renamed

The `objectstore::export()` function has been renamed `objectstore::export_object()` to reflect its purpose more clearly — to export persistent, top-level objects. For more information about the function, see its description in Chapter 2 ("Class Library") of the *C++ API Reference*. If you wish to continue using the old name, you must define `OS_EXPORT_API` at the top of your source file, before the ObjectStore header files are included.

# Changes to the ObjectStore Java Interface

# (OSJI)

Release 6.1 of the ObjectStore Java Interface (OSJI) now supports both JDK 1.3 and 1.4. Support for JDK 1.2 and earlier is no longer maintained.

# Changes to JMTL

In Release 6.1, the Java Middle Tier Library (JMTL) has been incorporated into the ObjectStore product and uses the same release numbering.

## com.odi.jmtl.env.JVMEnviroment Class Renamed

The `com.odi.jmtl.env.JVMEnviroment` class has been renamed to `com.odi.env.JVMEnviroment`. Java applications that use the `JVMEnvironment` class need to be modified and recompiled.

## com.odi.jmtl.env.JVMEnviroment.initialize() Method Renamed

The `com.odi.jmtl.env.JVMEnviroment.initialize()` method is replaced by `com.odi.env.JVMEnviroment.deploy()`. Any Java classes using this method need to be modified and recompiled.

## New JMTL Deployment Descriptor Format

With Release 6.1, JTML has a new deployment descriptor format. The new deployment descriptor format requires the following XML modifications:

| *Release 1.2 Element* | *Release 6.1 Element* |
|---|---|
| `CachePoolManager` | `cache-pool-manager` |
| `CachePools` | `cache-pools` |
| `CachePool` | `cache-pool` |
| `CacheStorage` | `cache-storage` |
| `Databases` | `databases` |
| `DatabaseDescriptor` | `database-descriptor` |
| `Roots` | `roots` |
| `RootDescriptor` | `root-descriptor` |
| `RootObjectDescriptor` | `root-object-descriptor` |
| `Component` | `component-descriptor` |
| `Method` | `method` |
| `MethodDescriptor` | `method-descriptor` |
| `ExtentDescriptor` | unsupported |
| `FinderDescriptor` | unsupported |

The new deployment descriptor format is defined by the DTD file `jmtl-dd.dtd` located in `jmtl.jar`. To use the new format, change the `DOCTYPE` entry from:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/xml/EnvConfig.dtd">
```

to:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/jmtl-dd.dtd">
```

Release 6.1 includes a DTD file `jmtl-dd-compat.dtd` that you can use to make existing JMTL 1.2 XML deployment descriptor files compatible with Release 6.1. To use this file, change the `DOCTYPE` entry from:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/xml/EnvConfig.dtd">
```

to:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/jmtl-dd-compat.dtd">
```

## New Cache Pool Attributes

Release 6.1 includes new cache pool attributes that influence cache performance. The new attributes are:

- `CommitIfIdle`
- `CommitIfIdleMvcc`
- `GroupOpenInterval`
- `GroupOpenIntervalMvcc`
- `LockTimeout`
- `MaxConcurrentTransactions`

For more information on these attributes, see “Declarative Configuration” in the *JMTL User Guide*.

## Integration with WebLogic Server 7.0

Release 6.1 includes a new application server specific jar file that provides level one integration with BEA WebLogic Server 7.0.

## Updated Examples Using Ant Build Files

The JMTL examples have been updated to use Apache Ant to simplify configuration, building, and deployment. For additional information on using Ant, see the `README.html` file in the examples directory.

# Changes to the Documentation

The following changes have been made to the documentation for Release 6.1:

- The ObjectStore online documentation has been reorganized into two bookshelves:
  - C++ bookshelf
  - Java bookshelf
- The online documentation is delivered through WebHelp, which provides full-text search capability.

- All of the documentation has been consolidated into the `doc` directory. This directory is directly under the installation directory for ObjectStore and contains the documentation for all installable components of this release, regardless of whether or not you decide to install them all.
- The following features of ObjectStore were previously released but never documented until this release:
  - `os_schema_evolution::set_explanation_level()`: See the description in Chapter 2 ("Class Library") of the *C++ API Reference*.
  - `os_schema_evolution::set_resolve_ambiguous_void_pointers()`: See the description in Chapter 2 ("Class Library") of the *C++ API Reference*.
  - `osconfig`: See the description in Chapter 4 ("Utilities") of *Managing ObjectStore*.
- The *JMTL User Guide* contains a new chapter titled "Chapter 7: Using the JMTL Console"

# Summary of Changes in 6.1

The following sections summarize all user-visible changes to ObjectStore for Release 6.1, including changes that have been more fully described in the preceding sections. The information in these sections is provided as a migration aid for users who might want to know at a glance how Release 6.1 will impact their applications and scripts. Each section also includes references to the relevant parts of the ObjectStore documentation for more detailed information about the changes.

## New C++ Classes and Functions in 6.1

The following classes and functions have been added to the ObjectStore API for Release 6.1:

- `objectstore::get_asmarkers_useless()`
- `objectstore::set_asmarkers_useless()`
- `os_authentication`
- `os_cluster::set_size()`
- `os_database::get_fragmentation()`
- `os_database::set_size()`
- `os_database::set_size_in_sectors()`
- `os_dbutil::install_backrest_control_c_handler()`
- `os_dbutil::start_backrest_logging()`
- `os_dbutil::stop_backrest_logging()`
- `os_dbutil::svr_machine()`
- `os_replicator`
- `os_replicator_options`

- os_replicator_statistic_info
- os_schema_evolution::set_explanation_level()
- os_schema_evolution::set_resolve_ambiguous_void_pointers()

Some of these functions are described in other sections of this document. For detailed information about all of the new classes and functions, see the descriptions in Chapter 2 ("Class Library") of the *C++ API Reference*.

## New Environment Variables in 6.1

The following environment variables are new for Release 6.1:

- OS_16K_PAGE
- OS_32K_PAGE
- OS_64K_PAGE
- OS_8K_PAGE
- OS_ASMARKERS_USELESS
- OS_CORE_DIR
- OS_NETWORK_SERVICE
- OS_PREALLOCATE_CACHE_FILES
- OS_REMOTE_AUTH_REGISTRY_LOCATION
- OS_USER_ARCH_SET

Some of these environment variables are described in other sections of this document. For detailed information about all of the new environment variables, see their descriptions in Chapter 3 ("Environment Variables") of *Managing ObjectStore*.

## New Server Parameters in 6.1

The following server parameters are new for Release 6.1:

- Cluster Growth Policy
- Database File Growth Policy
- Failover Heartbeat File
- Failover Script
- Identical Pathnames on Failover Server
- RAWFS Partition Growth Policy
- RPC Timeout

Some of these server parameters are described in other sections of this document. For detailed information about all of the new server parameters, see their descriptions in Chapter 2 ("Server Parameters") of *Managing ObjectStore*.

## New Options for ObjectStore Utilities in 6.1

New options have been added to the following ObjectStore utilities:

- osarchiv -P -T

- `osbackup -P -r -T`
- `oscopy rawfs_dir`
- `osdbcontrol -cluster_size -size`
- `osdump -dir -pr -v`
- `osload -dir -pr -v`
- `osrecovr -s -T`
- `osreplic -c -C -P -R -s -T`
- `osrestore -s -T`
- `osserver -con -hostname -M -server_name -upgradeRAWFS`
- `ossg -auditor -showsets -ignore_vbo`

  Note that the `-auditor` option is provided as a migration tool and is not described in *Managing ObjectStore*. For information about this option, see ossg's Default Front-End Parser on page 53.

- `ossize -f`
- `ossvrstat -databases`
- `osverifydb -all -tag`

For information about the new options refer to the descriptions of the utilities in Chapter 4 ("Utilities") of *Managing ObjectStore*.

## Unsupported C++ Functions in 6.1

The `objectstore::export()` function is not supported and has been replaced by the `objectstore::export_object()` function. For more information, see objectstore::export() Renamed on page 60.

## Unsupported Java Methods in 6.1

The `JVMEnvironment.initialize()` method is not supported and has been replaced by the `JVMEnvironment.deploy()` method. For more information, see com.odi.jmtl.env.JVMEnviroment.initialize() Method Renamed on page 61.

## Unsupported Options for ObjectStore Utilities in 6.1

The following options to the ossg utility are no longer supported:

- The `-edgfe` option; for more information, see ossg's Default Front-End Parser on page 53.
- The `-store_function_parameters` (`-sfp`) option. Specifying the `-store_member_functions` (`-smf`) option provides the same functionality; see the description of the `ossg` utility in Chapter 4 ("Utilities") of *Managing ObjectStore*.

# Index

## E



## F



## G



## H



## I



## J

## L



## M



## N



## O

## P

## Q



## R



## S

## T



## U



## V



## W