Oracle® Coherence

Release Notes for Oracle Coherence

Release 3.4.2

E14298-01

February 2009

Oracle Coherence Release Notes for Oracle Coherence, Release 3.4.2

E14298-01

Primary Author: Thomas Pfaeffle

Contributing Author: Noah Arliss, Mark Falco, Alex Gleyzer, Gene Gleyzer, Jason Howes, James Kirsch, Adam Leftik, Rob Misek, Patrick Peralta

Contents

Preface v

Audience v
Documentation Accessibility v
Related Documents vi
Conventions vi

1 Technical Changes and Enhancements

Oracle Coherence for Java 3.4.2 1-1
Oracle Coherence for C++ 3.4.2 1-2
Oracle Coherence for .NET 3.4.2 1-2

2 Documentation Errata

Corrections to "Overview for Implementors" 2-1
Corrections to "Coherence Features by Edition" 2-1
Corrections to "Understanding the Coherence C++ Object Model" 2-2
Corrections to "Building Integration Objects for C++ Clients" 2-4
Corrections to "How to Manage Coherence Using JMX" 2-7
Corrections to "Production Checklist" 2-7

Index

List of Tables

2–1 Changes Made to the Querying the Cache Section 2-1
2–2 Changes Made to the Coherence Features by Edition Appendix 2-2
2–3 Text Changes Made to the Thread Safety Section 2-2
2–4 Changes Made to the Thread Safe Handles Section 2-3
2–5 Changes Made to the Memory Leak Detection Section 2-4
2–6 Changes Made to the Building Integration Objects for C++ Clients Overview 2-5
2–7 Changes Made to the Need for Java Classes Section 2-6
2–8 Changes Made to the Configuring the Coherence Management Framework Section 2-7
2–9 Changes Made to the JVM Section 2-7
2–10 Changes Made to the Coherence Cache Configuration Section 2-7

Preface

This document describes changes and enhancements that have been made to the Oracle Coherence product since the 3.4.1 release.

Audience

This document is intended for users of Oracle Coherence.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at `http://www.consumer.att.com/relay/tty/standard2.html`. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

Related Documents

For more information, see the following documents in the Oracle Coherence documentation set:

- *Getting Started with Oracle Coherence*
- *User's Guide for Oracle Coherence*
- *Developer's Guide for Oracle Coherence*
- *Tutorial for Oracle Coherence*
- *User's Guide for Oracle Coherence*Web*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Technical Changes and Enhancements

This chapter describes the changes and enhancements made to the Oracle Coherence product since the 3.4.1 release. This document is accurate at the time of publication. Oracle updates the release notes periodically after the software release.

Oracle Coherence for Java 3.4.2

The following is a list of new features, improvements, and bug fixes in Oracle Coherence for Java 3.4.2:

Coherence*Web Enhancements and Fixes:

- Added native integration with WebLogic Server 10.3 and WebLogic Portal 10.3 that eliminates the need for instrumenting applications and Web container libraries. See *Coherence*Web and WebLogic Server 10.3* and *Coherence*Web and WebLogic Portal* in the *User's Guide for Oracle Coherence*Web* .
- Added support for running Coherence*Web on top of Coherence*Extend.
- Added a pluggable interface and default implementations for defining session attribute sharing policies. See *Session and Session Attribute Scoping* in the *User's Guide for Oracle Coherence*Web*.
- Significantly improved the performance of the split session model.
- Detect and prevent inconsistent sticky session mode configuration due to startup failure of the invocation service.
- Detect and prevent concurrent modifications of session models when the non-locking access mode is enabled.
- Fixed an issue that caused the Coherence logger thread to be leaked upon application redeployment.
- Fixed the WebInstaller to preserve the encoding XML attribute in the generated `web.xml` file allowing for proper handling of internationalized applications.

Other Enhancements and Fixes

- Added support for delivery of synthetic events from a `ReadWriteBackingMap`.
- Made the replicated cache service fully POF compatible.
- Hardened the JMX management shutdown protocol.
- Fixed a regression in the distributed cache service that caused missed invalidation events for near caches on Coherence*Extend clients.
- Fixed a performance regression in the distributed cache backup protocol.

- Fixed a regression in the replicated cache service that caused values to be intermittently returned prior to deserialization.
- Fixed POF serialization of nested empty arrays.

Oracle Coherence for C++ 3.4.2

The following is a list of new features, improvements, and bug fixes in Oracle Coherence for C++ 3.4.2:

- Improved `LinkedList` to avoid recursion during destruction.
- Improved the effectiveness of `Map::Entry::hashCode()`.
- Fixed race condition in `SafeHashMap` that could lead to dropped updates.
- Fixed an issue in `Array::copy()` related to overlapping arrays.
- Fixed POF serialization of nested empty arrays.
- Added dedicated thread for handling signals.

Oracle Coherence for .NET 3.4.2

The following is a list of new features, improvements, and bug fixes in Oracle Coherence for .NET 3.4.2:

- Enhanced `DefaultConfigurableCacheFactory.Shutdown()` to release local caches.
- Fixed POF serialization of nested empty arrays.

2 Documentation Errata

This chapter describes changes, enhancements, and corrections made to the existing Oracle Coherence library (*Getting Started*, *Developer's Guide*, and *User's Guide*). The current Oracle Coherence documentation library can be found at the following URL:

```
http://download.oracle.com/docs/cd/E13924_01/index.htm
```

Corrections to "Overview for Implementors"

This section describes the changes made to the *Overview for Implementors* chapter in *Getting Started with Oracle Coherence*.

Table 2–1 illustrates the code changes made to Examples 8-1 and 8-2 in the *Querying the Cache* section. The changes are illustrated in *italic* and *`codeItalic`*.

Table 2–1 Changes Made to the Querying the Cache Section

Old Text	New Text

Example 8–1 creates an index:
Example 8-1 Sample Code to Create an Index
`NamedCache cache =`
`CacheFactory.getCache("MyCache");`
`ValueExtractor extractor = new`
`ReflectionExtractor("getAttribute");`
`cache.addIndex(extractor, true, null);`
	Example 8–1 creates an index:
	Example 8-1 Sample Code to Create an Index
	`NamedCache cache =`
	`CacheFactory.getCache("MyCache");`
	`ValueExtractor extractor = new`
	`ReflectionExtractor("`*`getQuantity`*`");`
	`cache.addIndex(extractor, true, null);`

The code in Example 8–2 queries a `NamedCache` and returns the keys corresponding to all of the value objects with an `Attribute` greater than `5`:
Example 8-2 Sample Code to Query a NamedCache
`NamedCache cache =`
`CacheFactory.getCache("MyCache");`
`Filter filter = new`
`GreaterFilter("getAttribute", 5);`
`Set keySet = cache.keySet(filter);`
	Note: *the named attribute refers to a public zero-argument method on the data object.*
	The code in Example 8–2 queries a `NamedCache` and returns the keys corresponding to all of the value objects with an *`Quantity`* greater than `5`:
	Example 8-2 Sample Code to Query a NamedCache
	`NamedCache cache =`
	`CacheFactory.getCache("MyCache");`
	`Filter filter = new GreaterFilter("`*`getQuantity`*`",`
	`5);`
	`Set keySet = cache.keySet(filter);`

Corrections to "Coherence Features by Edition"

In the *Coherence Features by Edition* appendix of *Getting Started with Oracle Coherence*, a change has been made to the text of Note 6.

Table 2–2 Changes Made to the Coherence Features by Edition Appendix

Old Text	New Text
6. InvocationService requests from Data Client and Real Time Client are client-only, and there are limitations in Coherence 3.2.	6. Data Client and Real Time Client invocations are executed by the Extend Proxy Server they are connected to

Corrections to "Understanding the Coherence C++ Object Model"

This section describes the changes made to the *Understanding the Coherence C++ Object Model* chapter in the *Users Guide for Oracle Coherence*. Corrected text appears in *italics*.

Table 2–3 illustrates the changes made to the *Thread Safety* section

Table 2–3 Text Changes Made to the Thread Safety Section

Old Text	New Text
Although the object model includes a thread-safe reference count, this does not provide automatic thread safety for the state of derived classes. As is typical it is up to each individual class implementation to choose to provide for higher level thread-safety. Regardless of the presence or lack of higher level thread-safety, the reference count remains thread-safe.	Although the *base Object class is thread-safe*, this *cannot* provide automatic thread safety for the state of derived classes. As is typical it is up to each individual *derived* class implementation to provide for higher level thread-safety. *The object model provides several facilities to aid in writing thread-safe code.*

Table 2–4 illustrates the changes made to the *Thread Safe Handles* section.

Table 2–4 Changes Made to the Thread Safe Handles Section

Old Text	New Text
The Handle, View, and Holder nested types defined on managed classes are *not* thread-safe. That is it is not safe to have multiple threads use the same handle if any of them may change the handle to reference another object. There is an important distinction here, we are discussing the thread-safety of the handle, not the object referenced by the handle. Note it is safe to have multiple distinct handles reference the same object from different threads without additional synchronization.	The Handle, View, and Holder nested types defined on managed classes are *intentionally not* thread-safe. That is it is not safe to have multiple threads *share a single handle*. There is an important distinction here, we are discussing the thread-safety of the handle, not the object referenced by the handle. *It* is safe to have multiple distinct handles reference the same object from different threads without additional synchronization.
This lack of thread-safety for these handle types offers a significant performance optimization by assuming that the vast majority of handles are stack allocated. So long as references to these stack allocated handles are not shared across threads, there is no thread-safety issue to be concerned with. Typically precautions are needed when sharing a handle in one of the following conditions. ■ Managed class implementations. It should be assumed that any instance of a managed class may be shared by multiple threads. Though this may not be strictly true, if the object is passed to code outside of your control (for instance put into a cache), there is no guarantee that the object will not made be visible to other threads. ■ Non-managed multi-threaded application code.	This lack of thread-safety for these handle types offers a significant performance optimization *as* the vast majority of handles are stack allocated. So long as references to these stack allocated handles are not shared across threads, there is no thread-safety issue to be concerned with. Thread-safe handles are needed any time a single handle may be referenced by multiple threads. Typical cases include: ■ Global handles - using the standard handle types as global or static variable is not safe. ■ Non-managed multi-threaded application code - Use of standard handles within data structures which may be shared across threads is unsafe. ■ Managed classes with handles as data members - It should be assumed that any instance of a managed class may be shared by multiple threads, and thus using standard handles as data members is unsafe. Note that while it may not be strictly true that all managed classes may be shared across threads, if an instance is passed to code outside of your explicit control (for instance put into a cache), there is no guarantee that the object will not made be visible to other threads.
There are optimizations in place for the first case, namely the special thread-safe handle types: ■ `coherence::lang::MemberHandle<T>` - thread-safe version of `T::Handle` ■ `coherence::lang::MemberView<T>` - thread-safe version of `T::View` ■ `coherence::lang::MemberHolder<T>` - thread-safe version of `T::Holder` ■ `coherence::lang::WeakHandle<T>` - thread-safe weak handle to `T` ■ `coherence::lang::WeakView<T>` - thread-safe weak view to `T`	The use of standard handles should be replaced with thread-safe handles in such cases. The object model includes the following set of thread-safe handles. ■ `coherence::lang::MemberHandle<T>` - thread-safe version of `T::Handle` ■ `coherence::lang::MemberView<T>` - thread-safe version of `T::View` ■ `coherence::lang::MemberHolder<T>` - thread-safe version of `T::Holder` ■ `coherence::lang::WeakHandle<T>` - thread-safe weak handle to `T` ■ `coherence::lang::WeakView<T>` - thread-safe weak view to `T`

Table 2–4 (Cont.) Changes Made to the Thread Safe Handles Section

Old Text	New Text
These handle types may be read and written from multiple thread without the need for additional synchronization. They are primarily intended for use as the data-members of other managed classes, and they make use of their parent's internal atomic state to provide thread-safety. When using these handle types it is recommended that they be read into a normal stack based handle if they will be accessed more than once within a code block. This assignment to a normal stack based handle *is* thread-safe, and when completed allows for essentially free dereferencing of the stack based handle. Note that when initializing thread-safe handles a reference to a guardian Object must be supplied as the first parameter, this reference can be obtained by calling `self()` on the enclosing object.	These handle types may be read and written from multiple thread without the need for additional synchronization. They are primarily intended for use as the data-members of other managed classes, *each instance is provided with a reference to a guardian managed Object. The guardian's* internal *thread-safe* atomic state *is used* to provide *thread-safety to the handle*. When using these handle types it is recommended that they be read into a normal stack based handle if they will be accessed more than once within a code block. This assignment to a *standard* stack based handle is thread-safe, and when completed allows for essentially free dereferencing of the stack based handle. Note that when initializing thread-safe handles a reference to a guardian Object must be supplied as the first parameter, this reference can be obtained by calling `self()` on the enclosing object.
The same basic technique can be applied to non-managed classes as well. Since non-managed classes do not extend `coherence::lang::Object` they cannot be used as the guardian of thread-safe handles. It is possible however to use another Object as the guardian. When taking this approach it is crucial to ensure that the guardian Object outlives the guarded thread-safe handle. To facilitate this Coherence starting with Coherence 3.4.1 you can obtain an immortal guardian from `coherence::lang::System` by calling the `System::common()`.	The same basic technique can be applied to non-managed classes as well. Since non-managed classes do not extend `coherence::lang::Object` they cannot be used as the guardian of thread-safe handles. It is possible however to use another Object as the guardian. When taking this approach it is crucial to ensure that the guardian Object outlives the guarded thread-safe handle. To facilitate this Coherence starting with Coherence 3.4.1 you can obtain *a random* immortal guardian from `coherence::lang::System` *through a call* to `System::common()`. Note that when writing managed classes it is preferable to obtain a guardian through a call to `self()` then to `System::common()`.
NA	Add the following note before Example 2-18: **Note:** *In the rare case that one of these handles is declared by using the* `mutable` *keyword, it must be informed of this fact by setting* `fMutable` *to* `true` *during construction.*

Table 2–5 illustrates the changes made to the *Memory Leak Detection* section:

Table 2–5 Changes Made to the Memory Leak Detection Section

Old Text	New Text
■ `object` - The `coherence::lang::ObjectCountHeapAnalyzer` will be used. It provides simple heap analysis based solely on the count of the number of live objects in the system. This is the default analyzer.	■ `object` - The `coherence::lang::ObjectCountHeapAnalyzer` will be used. It provides simple heap analysis based solely on the count of the number of live objects in the system.

Corrections to "Building Integration Objects for C++ Clients"

This section describes the changes made to the *Building Integration Objects for C++ Clients* chapter in the *User's Guide for Oracle Coherence*.

Table 2–6 illustrates the changes made to the chapter overview. Corrected text appears in *italics*.

Table 2–6 Changes Made to the Building Integration Objects for C++ Clients Overview

Old Text	New Text
Enabling C++ clients to successfully store C++ based objects within a Coherence cluster relies on a platform-independent serialization format known as POF (Portable Object Format). POF allows value objects to be encoded into a binary stream in such a way that the platform and language origin of the object is irrelevant.	Enabling C+\+ clients to successfully store C+\+ based objects within a Coherence cluster relies on a platform-independent serialization format known as POF (Portable Object Format). POF allows value objects to be encoded into a binary stream in such a way that the platform and language origin of the object is irrelevant. *The stream can then be deserialized in an alternate language using a similar POF-based class definition.*

A new section, *POF Intrinsics*, has been added to the *Building Integration Objects for C++ Clients* chapter.

POF Intrinsics

The following types are internally supported by POF, and do not require special handling by the user:

- String
- Integer16, Integer64
- Float32, Float64
- Array<> of primitives
- ObjectArray
- Boolean
- Octet
- Character16

Additionally, automatic POF serialization is provided for classes implementing these common interfaces:

- Map
- Collection
- Exception

Corrections were made to Example 3-6 in the *PofSerializer (External Serialization)* section. The corrected code appears in ***bold italic***.

```
#include "coherence/lang.ns"
using namespace coherence::lang;
class Address
  : public cloneable_spec<Address> // extends<Object> is implied
  {
  friend class factory<Address>;
  protected: // constructors
    Address(String::View vsCity, String::View vsState, int32_t nZip)
       : m_vsCity(self(), vsCity), m_vsState(self(), vsState), m_nZip(nZip) {}
    Address(const Address& that)
       : super(that), m_vsCity(self(), that.m_vsCity), m_sState(self(), that.m_
vsState), m_nZip(that.m_nZip) {}
  public: // Address interface
    virtual String::View  getCity()  const {return m_vsCity;}
    virtual String::View  getState() const {return m_vsState;}
    virtual int32_t       getZip()   const {return m_nZip;}
  public: // Object interface
    virtual bool equals(Object::View that) const
      {
      if (instanceof<Address::View>(that))
        {
        Address::View vThat = cast<Address::View>(that);
        return getZip() == vThat->getZip() &&
               Object::equals(getState(), vThat->getState()) &&
               Object::equals(getCity(), vThat->getCity());
        }
      return false;
      }
    virtual size32_t hashCode() const
      {
      return (size32_t) m_nZip;
      }
    virtual void toStream(std::ostream& out) const
      {
      out << getCity() << ", " << getState() << "  " << getZip();
      }
  private:
    const MemberView<String> m_vsCity;
    const MemberView<String> m_vsState;
    const int32_t            m_nZip;
  };
```

Table 2–7 illustrates the changes made to the *Need for Java Classes* section. Corrected text appears in *italic*.

Table 2–7 Changes Made to the Need for Java Classes Section

Old Text	New Text
After completing any of the above approaches your data object will be ready to be stored within the Coherence cluster. This will allow you to perform `get`- and `put`-based operations with your objects. If, however, you want to make use of more advanced features of Coherence, such as queries, or entry processors you will need to write some Java code. For these advanced features to work the Coherence Java-based cache servers need to be able to interact with your data object, rather then simply holding onto a serialized representation of it. To interact with it, and access its properties, a Java version must be made available to the cache servers. The approach to making the Java version serializable over POF is quite similar to the above examples, see `com.tangosol.io.pof.PortableObject`, and `com.tangosol.io.pof.PofSerializer` for details, either of which is compatible with all three of the C++ based approaches.	After completing any of the above approaches your data object will be ready to be stored within the Coherence cluster. *This alone* will allow you to perform `get`- and `put`-based operations with your objects. If, however, you want to make use of more advanced features of Coherence, such as queries, or entry processors you will need to write some Java code. *To use these advanced features from a C++ client the* Java-based cache servers need to be able to interact with your data objects, rather then simply holding onto a serialized representation. To interact with it, and access its properties, *Java versions of your data-object classes must be made available to the cache servers.* The approach to making the Java version serializable over POF is quite similar to the above examples, see `com.tangosol.io.pof.PortableObject`, and `com.tangosol.io.pof.PofSerializer` for details, either of which is compatible with all three of the C++ based approaches. *Once the cache server has theses corresponding Java data-object classes you can invoke queries, entry processors, and other advanced Coherence features directly from C++ clients.*

Corrections to "How to Manage Coherence Using JMX"

This section describes the changes made to the *How to Manage Coherence Using JMX* chapter of the *Developer's Guide for Oracle Coherence*.

Table 2–8 illustrates the changes made to the *Configuring the Coherence Management Framework* section. Changed text is in *italic*.

Table 2–8 Changes Made to the Configuring the Coherence Management Framework Section

Old Text	New Text
The use of dedicated JMX cluster members is a common pattern. This approach avoids loading JMX software into every single cluster member, while still providing fault-tolerance should a single JMX member run into issues.	The use of dedicated JMX cluster members is a common pattern. This approach avoids loading JMX software into every single cluster member, while still providing fault-tolerance should a single JMX member run into issues. *Setting* `tangosol.coherence.management` *to* `all` *on nodes that are not acting as MBeanServer hosts, will consume capacity (~3-5%) when the node starts and increase the time required for the node to join the cluster. Therefore, it is recommended to set the value to* `none` or `local-only` *for remotely managed nodes.*

Corrections to "Production Checklist"

This section describes the changes made to the *Production Checklist* appendix of the *Developer's Guide for Oracle Coherence*.

Table 2–9 illustrates changes made to the *JVM* section. New text is in *italic*.

Table 2–9 Changes Made to the JVM Section

Old Text	New Text
Oracle recommends testing and deploying using the latest supported Sun JVM based on your platform and Coherence version.	Oracle recommends testing and deploying using the latest supported Sun JVM based on your platform and Coherence version. *It has been observed that running Coherence on 1.5 and later JVMs exhibits significant performance improvements as compared to running on older JVMs.*

Table 2–10 illustrates the changes made to the *Coherence Cache Configuration* section. Corrected text is in *italic*.

Table 2–10 Changes Made to the Coherence Cache Configuration Section

Old Text	New Text
Unless explicitly specified all cluster nodes will be storage enabled, i.e. will act as cache servers. It is important to control which nodes in your production environment will be storage enabled and storage disabled. The `tangosol.coherence.distributed.localstorage` system property may be utilized to control this, setting it to either `true` or `false`. Generally only dedicated cache servers, all other cluster nodes should be configured as storage disabled. This is especially important for short lived processes which may join the cluster perform some work, and exit the cluster, having these nodes as storage disable will introduce unneeded repartitioning.	Unless explicitly specified all cluster nodes will be storage enabled, i.e. will act as cache servers. It is important to control which nodes in your production environment will be storage enabled and storage disabled. The `tangosol.coherence.distributed.localstorage` system property may be utilized to control this, setting it to either `true` or `false`. Generally only dedicated cache servers, all other cluster nodes should be configured as storage disabled. This is especially important for short lived processes which may join the cluster perform some work, and exit the cluster, having these nodes as storage *enabled* will introduce unneeded repartitioning.

Index

C

C++ clients, building integration objects, 2-4
Coherence*Extend, 1-1

D

distributed cache service, 1-1

I

InvocationService, 2-2

J

Java classes, 2-6
JMX management, 1-1, 2-7
JVMs, Sun JVMs, 2-7

L

logger thread, 1-1

N

NamedCache, 2-1

P

POF compatibility, 1-1
POF intrinsics, 2-5
POF serialization, 1-2
PofSerializer, 2-5

R

real time client, 2-2
replicated cache service, 1-1, 1-2

S

session attribute sharing, 1-1
session models, 1-1
split session model, 1-1
sticky session mode, 1-1
synthetic events, 1-1

T

thread safe handles, 2-2
thread safety, 2-2

W

WebInstaller, 1-1
WebLogic Portal 10.3, 1-1
WebLogic Server 10.3, 1-1