Oracle® Coherence

User's Guide for Oracle Coherence

Release 3.4

E12192-01

November 2008

ORACLE®

Oracle Coherence User's Guide for Oracle Coherence, Release 3.4

E12192-01

Primary Author: Thomas Pfaeffle

Contributing Author: Noah Arliss, Jason Howes, Mark Falco, Alex Gleyzer, Gene Gleyzer, David Leibs, Andy Nguyen, Brian Oliver, Patrick Peralta, Cameron Purdy, Jonathan Purdy, Everet Williams, Tom Beerbower, John Speidel

Contents

26 Configuring Coherence for TopLink Essentials

27 Configuring Coherence for JPA

Part V Integration with Hibernate

28 Using Coherence as the Hibernate L2 Cache

29 Using Hibernate as a CacheStore for Coherence

A Sample C++ Applications

Preface

Oracle Coherence is a JCache-compliant in-memory caching and data management solution for clustered J2EE applications and application servers. Coherence makes sharing and managing data in a cluster as simple as on a single server. It accomplishes this by coordinating updates to the data using clusterwide concurrency control, replicating and distributing data modifications across the cluster using the highest performing clustered protocol available, and delivering notifications of data modifications to any servers that request them. Developers can easily take advantage of Coherence features using the standard Java collections API to access and modify data, and use the standard JavaBean event model to receive data change notifications. Functionality such as HTTP Session Management is available out-of-the-box for applications deployed to WebLogic, WebSphere, Tomcat, Jetty and other Servlet 2.2, 2.3 and 2.3 compliant application servers.

Audience

This document is targeted at software developers and architects. It provides detailed technical information for writing and deploying C++ and .NET applications that interact with the Coherence cache. It also provides information on integrating Coherence with the Web Logic Server (WLS), TopLink Essentials, and Hibernate.

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*
- *Developer's Guide for Oracle Coherence*

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.

Part I

Coherence for C++

Coherence for C++ allows C++ applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses of Coherence for C++ include desktop and web applications that require access to Coherence caches.

Coherence for C++ consists of a native C++ library that connects to a Coherence*Extend clustered service instance running within the Coherence cluster using a high performance TCP/IP-based communication layer. This library sends all client requests to the Coherence*Extend clustered service which, in turn, responds to client requests by delegating to an actual Coherence clustered service (for example, a partitioned or replicated cache service).

A `NamedCache` instance is retrieved by using the `CacheFactory::getCache(...)` API call. Once it is obtained, a client accesses the `NamedCache` in the same way as it would if it were part of the Coherence cluster. The fact that `NamedCache` operations are being sent to a remote cluster node (over TCP/IP) is completely transparent to the client application.

> **Note:** The C++ client follows the interface and concepts of the Java client, and users familiar with Coherence for Java should find migrating to Coherence for C++ straight forward.

Coherence for C++ contains the following chapters:

1

Requirements, Installation, and Deployment for Coherence for C++

After installing Coherence for C++ and setting up the environment, you can try running the sample applications.

Package Requirements

The following are required to use Coherence for C++:

- *Coherence Data Grid Edition 3.4 (or later)*
- C++ development environment

Supported Environments

The current release of Coherence for C++ is supported on the platforms and operating systems listed in Table 1–1:

Table 1–1 Platform and Operating System Support for Coherence for C++

Operating System	Compiler	Architecture
Microsoft Windows 2000+ (see note 1)	MSVC 2005 SP1+ (see note 3)	x86
Microsoft Windows Server 2003+ (see note 2)	MSVC 2005 SP1+ (see note 3)	x64
Linux	GCC 3.4+ (see Note 4)	x86
Linux	GCC 3.4+ (see Note 4)	x64
Apple OS X 10.4+	GCC 3.4+ (see Note 4)	x86

Notes:

1. Including Windows 32b XP, Vista, 2000, 2003, and 2008.
2. Including Windows 64b XP, Vista, 2003, and 2008.
3. Specifically MSVC 2005 SP1 (14.00.5+), and MSVC 2008 and express versions are supported.
4. Specifically GCC 3.4.6-4 and above, and GCC 4.x versions are supported.

Note: When deploying on Microsoft Windows, just as with any MSVC-based application, the Visual Studio 2005 SP1 C++ redistributable runtime libraries are required.

Installing Coherence for C++

1. Download the Coherence for C++ package for your target environment.
2. Extract the archive.
3. Download the most recent patch for your target environment.
4. Extract the patch archive, and copy contents over the installation directory

Building Coherence-Based Applications

Compiler Settings

When integrating Coherence for C++ into your application's build process, it is important that certain compiler and linker settings be enabled. Some settings are optional, but still highly recommended.

MSVC (Visual Studio)

Table 1–2 Compiler Settings for MSVC (Visual Studio)

Setting	Build Type	Required?	Description
/EHsc	All	Yes	Enables C++ exception support
/GR	All	Yes	Enables C++ RTTI
/O2	Release	No	Enables speed optimizations
/MD	Release	Yes	Link against multi-threaded DLLs
/MDd	Debug	Yes	Link against multi-threaded debug DLLs

g++

Table 1–3 Compiler Settings for g++

Setting	Build Type	Required	Description
-O3	Release	No	Enables speed optimizations

Coherence Header Files

Coherence ships with a set of header files which your application will need to compile code which uses the Coherence API. The header files are available under the

installation's `include` directory. This `include` directory must be part of your compiler's include search path.

Linking

Coherence for C++ ships with both a debug and release version of the Coherence library. It is recommended that when compiling your application in debug mode that you also link against the Coherence debug library, by linking against `coherence-debug` rather then `coherence`. These libraries are located in the installation's `lib` directory. During linking this directory will need to be part of your linker's library path.

Table 1–4 Names of Linking Libraries for Release and Debug Versions

Operating System	Release Library	Debug Library
Windows	coherence.lib	coherence-debug.lib
Linux	libcoherence.so	libcoherence-debug.so
Apple OS X	libcoherence.dylib	libcoherence-debug.dylib

Runtime Library and Search Path

During execution of a Coherence enabled application the Coherence for C++ shared library must be available from your application's library search path. This is achieved by adding the directory which contains the shared library to an operating system dependent environment variable. The installation includes libraries in its lib subdirectory.

Table 1–5 Name of the Coherence for C++ Library and Environment Variables

Operating System	Environment Variable
Windows	PATH
Linux	LD_LIBRARY_PATH
Apple (Mac) OS X	DYLD_LIBRARY_PATH

For example, to set the PATH environment variable on Windows execute:

```
c:\coherence\coherence-cpp\examples> set
PATH=%PATH%;c:\coherence\coherence-cpp\lib
```

As with the Java version of Coherence, the C++ version supports a concept of System Properties to override configuration defaults. System Properties in C++ are set by using standard OS environment variables, and use the same names as their Java counterparts. The tangosol.coherence.cacheconfig system property can be used to specify the location of the cache configuration file. You may also set the configuration location programatically (`CacheFactory::configure()`) from application code, the examples however do not do this.

Table 1–6 Cache Configuration System Property Value for Various Operating Systems

Operating System	System Property
Windows	tangosol.coherence.cacheconfig
Linux	TangosolCoherenceCacheConfig
Solaris	TangosolCoherenceCacheConfig

Table 1–6 (Cont.) Cache Configuration System Property Value for Various Operating

Operating System	System Property
Apple (Mac) OS X	TangosolCoherenceCacheConfig

Note: Some OS shells, such as the UNIX bash shell, do not support environment variables which include the '.' character. In this case, you may specify the name in camel case, where the first letter, and every letter following a '.' is capitalized. That is, "`tangosol.coherence.cacheconfig`" becomes "`TangosolCoherenceCacheConfig`".

For example, to set the configuration location on Windows execute:

```
c:\coherence\coherence-cpp\examples> set
tangosol.coherence.cacheconfig=config\extend-cache-config.xml
```

Deploying Coherence for C++

Coherence for C++ requires no specialized deployment configuration. Simply link your application with the Coherence library and follow the configuration instructions. See the sample applications for examples of build scripts and configuration.

Note: When deploying to Microsoft Windows the *Visual Studio 2005 SP1* C++ runtime libraries are required. To build the samples a version of *Visual Studio 2005 SP1* or higher is required.

2

Understanding the Coherence C++ Object Model

The Coherence Extend C++ API contains a C++ object model. You should become familiar with this object model if you want to implement the Coherence API. This section contains the following information:

Using the Object Model

The following section contains general information for writing code which uses the object model.

Coherence Namespaces

This **coherence** namespace contains the following general purpose namespaces:

- `coherence::lang`—the essential classes that make up the object model
- `coherence::util`—utility code, including collections
- `coherence::net`—network and cache
- `coherence::stl`—C++ Standard Template Library integration
- `coherence::io`—serialization

Although each class is defined within its own header file, you can use namespace-wide header files to facilitate the inclusion of related classes. We recommend including, at a minimum, `coherence/lang.ns` in code that uses this object model.

Understanding the Base Object

The `coherence::lang::Object` class is the root of the class hierarchy. This class provides the common interface for abstractly working with Coherence class instances. Object is an instantiable class that provides default implementations for the following functions.

- `equals`
- `hashCode`
- `clone` (optional)
- `toStream` (that is, writing an `Object` to an `std::ostream`)

See `coherence::lang::Object` in the C++ API for more information.

Automatically Managed Memory

In addition to its public interface, the Object class provides several features used internally. Of these features, the *reference counter* is perhaps the most important. It provides automatic memory management for the object. This automatic management eliminates many of the problems associated with object reference validity and object deletion responsibility. This management reduces the potential of programming errors which may lead to memory leaks or corruption. This results in a stable platform for building complex systems.

The reference count, and other object "life-cycle" information, operates in an efficient and thread-safe manner by using lock-free atomic compare-and-set operations. This allows objects to be safely shared between threads without the risk of corrupting the count or of the object being unexpectedly deleted due to the action of another thread.

Referencing Managed Objects

To track the number of references to a specific object, there must be a level of cooperation between pointer assignments and a memory manager (in this case the object). Essentially the memory manager must be informed each time a pointer is set to reference a managed object. Using regular C++ pointers, the task of informing the memory manager would be left up to the programmer as part of each pointer assignment. In addition to being quite burdensome, the effects of forgetting to inform the memory manager would lead to memory leaks or corruption. For this reason the task of informing the memory manager is removed from the application developer, and placed on the object model, though the use of *smart pointers*. Smart pointers offer a syntax similar to normal C++ pointers, but they do the bookkeeping automatically.

The Coherence C++ object model contains a variety of smart pointer types, the most prominent being:

- `View`—A smart pointer that can call only `const` methods on the referenced object
- `Handle`—A smart pointer that can call both `const` and non-`const` methods on the referenced object.
- `Holder`—A special type of handle that enables you to reference an object as either `const` or non-`const`. The holder remembers how the object was initially assigned, and returns only a compatible form.

Other specialized smart pointers are described later in this section, but the View, Handle, and Holder smart pointers will be used most commonly.

Note: In this documentation, the term handle (with a lowercase "h") refers to the various object model smart pointers. The term Handle (with an uppercase "H") refers to the specific Handle smart pointer.

Using handles

By convention each managed class will have these nested-types corresponding to these handles. For instance the managed `coherence::lang::String` class defines `String::Handle`, `String::View`, `String::Holder`.

Assignment of handles Assignment of handles follows normal inheritance assignment rules. That is, a Handle may be assigned to a View, but a View may not be assigned to a Handle, just like a `const` pointer cannot be assigned to a non-`const` pointer.

Dereferencing handles When dereferencing a handle that references NULL, the system will throw a `coherence::lang::NullPointerException` instead of triggering a traditional segmentation fault.

For example, this code would throw a `NullPointerException` if `hs == NULL`:

```
String::Handle hs = getStringFromElsewhere();
cout << "length is " << hs->length() << end1;
```

Managed Object Instantiation

All managed objects are heap allocated. The reference count—not the stack—determines when an object can be deleted. To prevent against accidental stack-based allocations, all constructors are marked protected, and public factory methods are used to instantiate objects.

The factory method is named create and there is one create method for each constructor. The create method returns a Handle rather than a raw pointer. For example, the following code will create a new instance of a string:

```
String::Handle hs = String::create("hello world");
```

By comparison, these examples are incorrect and will not compile:

```
String str("hello world);
String* ps = new String("hello world);
```

Managed Strings

All objects within the model, including strings, are managed and extend from `Object`. Instead of using `char*` or `std::string`, the object model uses its own managed `coherence::lang::String` class. The `String` class supports ASCII and the full Unicode BML character set.

String Instantiation

String objects can easily be constructed from `char*` or `std::string` strings, as shown in these examples:

Example 2–1 Examples of Constructing String Objects

```
const char*    pcstr = "hello world";
std:string     stdstr(pcstr);
String::Handle hs   = String::create(pcstr);
String::Handle hs2  = String::create(stdstr);
```

The managed string is a copy of the supplied string and contains no references or pointers to the original. You can convert back, from a managed String to any other string type, by using `getCString()` method. This returns a pointer to the original `const char*`. Strings can also be created using the standard C++ `<<` operator, when coupled with the `COH_TO_STRING` macro.

Example 2–2 Constructing String Objects with the "<<" Operator

```
String::Handle hs = COH_TO_STRING("hello " << getName() << " it is currently " <<
getTime());
```

Auto-Boxed Strings

To facilitate the use of quoted string literals, the `String::Handle` and `String::View` support auto-boxing from `const char*`, and `const std::string`. This enables you to write the code shown in the prior samples as:

Example 2–3 Autoboxing Examples

```
String::Handle hs  = "hello world";
String::Handle hs2 = stdstr;
```

Auto-boxing is also available for other types. See `coherence::lang::BoxHandle` for details.

Type Safe Casting

Handles are *type safe*, in the following example, the compiler will not allow you to assign an `Object::Handle` to a `String::Handle`, because not all Objects are Strings.

```
Object::Handle ho = getObjectFromSomewhere();
String::Handel hs = ho; // will not compile
```

However, this example *will* compile, as all Strings are Objects.

Example 2–4 Type Safe Casting Examples

```
String::Handle hs = String::create("hello world");
Object::Handle ho = hs; // will compile
```

Down Casting

For situations in which you want to down-cast to a derived Object type, you must perform a *dynamic cast* using the C++ RTTI (runtime type information) check and ensure that the cast is valid. The Object model provides helper functions to ease the syntax.

- `cast<H>(o)`—attempt to transform the supplied handle `o` to type `H`, throwing an `ClassCastException` on failure
- `instanceof<H>(o)`—test if a cast of `o` to `H` is allowable, returning `true` for success, or `false` for failure

These functions are similar to the standard C++ `dynamic_cast<T>`, but do not require access to the raw pointer.

The following example shows how to down cast a `Object::Handle` to a `String::Handle`:

Example 2–5 Down Casting Examples

```
Object::Handle ho = getObjectFromSomewhere();
String::Handle hs = cast<String::Handle>(ho);
```

The `cast<H>` function will throw a `coherence::lang::ClassCastException` if the supplied object was not of the expected type. The `instanceof<H>` function can be used to test if an Object is of a particular type without risking an exception being thrown. Such checks or generally only needed for places where the actual type is in doubt.

Example 2–6 Object Type Checking with the instanceof<H> Function

```
Object::Handle ho = getObjectFromSomewhere();

if (instanceof<String::Handle>(ho))
  {
  String::Handle hs = cast<String::Handle>(ho);
  }
else if (instanceof<Integer32::Handle>(ho))
  {
  Integer32::Handle hn = cast<Integer32::Handle>(ho);
  }
else
  {
  ...
  }
```

Managed Arrays

Managed arrays are provided by using the `coherence::lang::Array<T>` template class. In addition to being managed and adding safe and automatic memory management, this class includes the overall length of the array, and bounds checked indexing.

You can index an array by using its Handle's subscript operator, as shown in this example:

Example 2–7 Indexing an Array

```
Array<int32_t>::Handle harr = Array<int32_t>::create(10);

int32_t nTotal = 0;
for (size32_t i = 0, c = harr->length; i < c; ++i)
    {
    nTotal += harr[i];
    }
```

The object model supports arrays of C++ primitives and managed `Objects`. Arrays of derived `Object` types are not supported, only arrays of `Object`, casting must be employed to retrieve the derived handle type. Arrays of Objects are technically `Array<MemberHolder<Object> >`, and `typedef`'d to `ObjectArray` for easier readability.

Collection Classes

The `coherence::util*` namespace includes several collection classes and interfaces that may be useful in your application. These include:

- `coherence::util::Collection` —interface
- `coherence::util::List`—interface
- `coherence::util::Set`—interface
- `coherence::util::Queue`—interface
- `coherence::util::Map`—interface
- `coherence::util::LinkedList`—implementation
- `coherence::util::HashSet`—implementation
- `coherence::util::DualQueue`—implementation

- `coherence::util::HashSet`—implementation
- `coherence::util::SafeHashMap`—implementation
- `coherence::util::WeakHashMap`—implementation
- `coherence::util::IdentityHashMap`—implementation

These classes also appear as part of the Coherence Extend API.

Similar to `ObjectArray`, `Collections` contain `Object::Holders`, allowing them to store any managed object instance type.

Example 2–8 Storing Managed Object Instances

```
Map::Handle  hMap = HashSet::create();
String::View vKey = "hello world";

hMap->put(vKey, Integer32::create(123));

Integer32::Handle hValue = cast<Integer32::Handle>(hMap->get(vKey));
```

Managed Exceptions

In the object model, exceptions are also managed objects. This enables you to hold onto caught exceptions as a local variable or data member without the risk of *object slicing*.

All Coherence exceptions are defined by using a `throwable_spec` and derive from the `coherence::lang::Exception` class, which derives from `Object`. Managed exceptions are not explicitly thrown by using the standard C++ `throw` statement, but rather by using a `COH_THROW` macro. This macro will set stack information, and then call the exception's `raise` method, which ultimately calls throw. The resulting thrown object may be caught an the corresponding exceptions `View` type, or an inherited `View` type. Additionally these managed exceptions may be caught as standard `const std::exception` classes. The following example shows a try/catch block with managed exceptions:

Example 2–9 A Try/Catch Block with Managed Exceptions

```
try
    {
    Object::Handle h = NULL;
    h->hashCode(); // trigger an exception
    }
catch (NullPointerException::View e)
    {
    cerr << "caught" << e <<endl;
    COH_THROW(e); // rethrow
    }
```

Note: This exception could also have been caught as `Exception::View` or `const std::exception&`.

Object Immutability

In C++ the information of *how* an object was declared (such as const) is not available from a pointer or reference to an object. For instance a pointer of type `const Foo*`, only indicates that the user of that pointer cannot change the objects state. It does not

indicate if the referenced object was actually declared const, and is guaranteed not to change. The object model adds a runtime immutability feature to allow the identification of objects which can no longer change state.

The `Object` class maintains two reference counters: one for Handles and one for Views. If an object is referenced only from Views, then it is by definition **immutable**, as Views cannot change the state, and Handles cannot be obtained from Views. The `isImmutable()` method (included in the `Object` class) can test for this condition. The method is virtual, allowing subclasses to alter the definition of immutable. For example, String contains no non-const methods, and therefore has an isImmutable() method that always returns true.

Note that once immutable, an object cannot revert to being mutable. You cannot cast away `const`-ness to turn a Handle into a View as this would violate the proved immutability.

Immutability is important with respect to caching. The Coherence `NearCache` and `ContinuouQueryCache` can take advantage of the immutability to determine if a direct reference of an object can be stored in the cache, or if a copy must be created. Additionally, knowing that an object cannot change allows safe multi-threaded interaction without synchronization.

Integrating Existing Classes into the Object Model

Frequently there will be the need to integrate existing classes into the object model. A typical example would be the need to store a data-object into a Coherence cache, which only supports storage of managed objects. As it would not be reasonable to require that pre-existing classes be modified to extend from `coherence::lang::Object`, the object model provides an adapter which will automatically convert a non-managed plain old C++ class instance into a managed class instance at runtime.

This is accomplished by using the `coherence::lang::Managed<T>` template class. This template class extends from Object and from the supplied template parameter type `T`, effectively producing a new class which is both an Object and a `T`. The new class can be initialized from a `T`, and converted back to a `T`. The result is an easy to use, yet very powerful bridge between managed and non-managed code.

See the API doc for `coherence::lang::Managed` for details and examples.

Writing New Managed Classes

The following section provides information necessary to write new managed classes, that is, classes which extend from `Object`. The creation of new managed classes is required when you are creating new `EventListeners`, `EntryProcessors`, or `Filter` types. They are not required when you are working with existing C++ data objects or making use of the Coherence C++ API. See the previous section for details on integration non-managed classes into the object model.

Specification-Based Managed Class Definition

Specification-based definitions, or "specs" enables you to quickly define managed classes in C++.

Specification-based definitions are helpful when you are writing your own implementation of managed objects.

There are various forms of specs used to create different class types:

- `class_spec`—standard instantiatable class definitions
- `cloneable_spec`—cloneable class definitions
- `abstract_spec`—non-instantiatable class definitions, with zero or more pure virtual methods
- `interface_spec`—for defining interfaces (pure virtual, multiply inheritable classes)
- `throwable_spec`—managed classes capable of being thrown as exceptions

Specs automatically define these features on the class being spec'd:

- Handles, Views, Holders
- static `create()` methods which delegate to protected constructors
- virtual `clone()` method delegating to the copy constructor
- virtual `sizeOf()` method based on `::sizeof()`
- super `typedef` for referencing the class from which the defined class derives
- inheritance from `coherence::lang::Object`, when no parent class is specified by using `extends<>`

To define a class using specs, the class publicly inherits from one of the above specs. Each of these specs are parametrized templates. The parameters are as follows:

- The name of the class being defined.
- The class to publicly inherit from, specified by using an `extends<>` statement, defaults to `extends<Object>`
  - This element is not supplied in interface_spec
  - Except in the case of `extends<Object>`, the parent class is not derived from virtually
- A list of interfaces implemented by the class, specified by using an `implements<>` statement
  - All interfaces are derived from using public virtual inheritance

Note that the `extends<>` parameter is note used in defining interfaces.

Example 2–10 illustrates using `interface_spec` to define a `Comparable` interface:

Example 2–10 An Interface Defined by interface_spec

```
class Comparable
    : public interface_spec<Comparable>
    {
    public:
        virtual int32_t compareTo(Object::View v) const = 0;
    };
```

Example 2–11 illustrates using `interface_spec` to define a derived interface `Number`:

Example 2–11 A Derived Interface Defined by interface_spec

```
class Number
    : public interface_spec<Number,
        implements<Comparable> >
    {
```

```
    public:
        virtual int32_t getValue() const = 0;
    };
```

Next a `cloneable_spec` is used to produce an implementation. This is illustrated in in Example 2–12.

> **Note:** To support the auto-generated create methods, instantiatable classes must declare the `coherence::lang::factory<>` template as a friend. By convention this is the first statement within the class body.

Example 2–12 An Implementation Defined by cloneable_spec

```
class Integer
    : public cloneable_spec<Integer,
        extends<Object>,
        implements<Number> >
    {
    friend class factory<Integer>;

    protected:
        Integer(int32_t n)
            : super(), m_n(n)
            {
            }

        Integer(const Integer& that)
            : super(that), m_n(that.m_n)
            {
            }

    public:
        virtual int32_t getValue() const
            {
            return m_n;
            }

        virtual int32_t compareTo(Object::View v) const
            {
            return getValue() - cast<Integer::View>(v)->getValue();
            }

        virtual void toStream(std::ostream& out) const
            {
            out << getValue();
            }

    private:
        int32_t m_n;
    };
```

The class definition in Example 2–12 is the equivalent the non-spec based definitions in Example 2–13.

Example 2–13 Defining a Class Without the use of specs

```
class Integer
    : public virtual Object, public virtual Number
```

```
{
public:
    typedef TypedHandle<const Integer> View;   // was auto-generated
    typedef TypedHandle<Integer>       Handle; // was auto-generated
    typedef TypedHolder<Integer>       Holder; // was auto-generated
    typedef super                      Object; // was auto-generated

    // was auto-generated
    static Integer::Handle create(const int32_t& n)
        {
        return new Integer(n);
        }

protected:
    Integer(int32_t n)
        : super(), m_n(n)
        {
        }

    Integer(const Integer& that)
        : super(that), m_n(that.n)
        {
        }

public:
    virtual int32_t getValue() const
        {
        return m_n;
        }

    virtual int32_t compareTo(Object::View v) const
        {
        return getValue() - cast<Integer::View>(v)->getValue();
        }

    virtual void toStream(std::ostream& out) const
        {
        out << getValue();
        }

    // was auto-generated
    virtual Object::Handle clone() const
        {
        return new Integer(*this);
        }

    // was auto-generated
    virtual size32_t sizeOf() const
        {
        return ::sizeof(Integer);
        }

private:
    int32_t m_n;
};
```

Example 2–14 illustrates using the spec'd class:

Example 2–14 Using specs to Define a Class

```
Integer::Handle hNum1 = Integer::create(123);
Integer::Handle hNum2 = Integer::create(456);

if (hNum1->compareTo(hNum2) > 0)
    {
    std::cout << hNum1 << " is greater then " << hNum2 << std::endl;
    }
```

Equality, Hashing, Cloning, Immutability, and Serialization

What do all these concepts have in common? They all identify the state of an object, and as such will generally have similar implementation concerns. Simply put all data members referenced in one of these methods, will likely need to be referenced in all of the methods. Conversely any data members which are not referenced by one, should likely not be referenced by any of these methods. Consider the simple case of a `HashSet::Entry`, which contains the well known key and value data members. Certainly these are to be considered in the equals method, and would likely be tested for equality by using a call to their own equals method, rather than through reference equality. Now what if this Entry also contains as part of the implementation of the `HashSet` a handle to the next Entry within the `HashSet`'s bucket, and perhaps also contains a handle back to the `HashSet` itself. Should these be considered in equals as well? Likely not, it would seem reasonable that comparing two entries consisting of equal keys and values, from two maps should be considered equal. Following this line of thought the `hashCode` method on Entry would completely ignore data members except for key and value, and the Entry's `hashCode` would be computed using the results of its key and value hashCode, rather then using their identity `hashCode`. that is, a deep equality check in equals implies a deep hash in `hashCode`. Moving onto clone it can be seen that in cloning an Entry, we would not want to clone all its data member, but only the key and value. Obviously cloning the parent `Map` as part of clone the Entry would make no sense, and a similar argument can be made for cloning the handle to the next Entry. This line of thinking can be extended to the isImmutable method, and to serialization as well. While it is certainly not hard and fast rule it is worth considering this approach when implementing any of these methods.

Threading

The object model includes managed threads, which allows for easy creation of platform independent, multi-threaded, applications. The threading abstraction includes support for creating, interrupting, and joining threads. Thread local storage is available from the `coherence::lang::ThreadLocal` class. Thread dumps are also available for diagnostic and troubleshooting purposes. The managed threads are ultimately wrappers around the system's native thread type, such as POSIX or Windows Threads. This threading abstraction is used internally by Coherence, but is available for the application, if necessary.

Example 2–15 illustrates how to create a new `Runnable` instance and spawn a thread:

Example 2–15 Creating a Runnable Instance and Spawning a Thread

```
class HelloRunner
      : public class_spec<HelloRunner,
          extends<Object>,
          implements<Runnable> >
     {
     friend class factory<HelloRunner>;
```

```
    protected:
         HelloRunner(int cReps)
             : super(), m_cReps(cReps)
             {
             }

    public:
         virtual void run()
             {
             for (int i = 0; i < m_Reps; ++i)
                 {
                 Thread::sleep(1000);
                 std::cout << "hello world" << std::endl;
                 }
             }

     protected:
         int m_cReps;
     };

...

Thread::Handle hThread = Thread::create(HelloRunner::create(10));
hThread->start();
hThread->join();
```

Refer to `coherence::lang::Thread` and `coherence::lang::Runnable` for more information.

Weak References

The primary functional limitation of a reference counting scheme is automatic cleanup of cyclical object graphs. Consider the simple bi-directional relationship illustrated in Figure 2–1.

Figure 2–1 A Bi-Directional Relationship

This figure is described in the text.

****

In this picture, both A and B have a reference count of one, which keeps them active. What they don't realize is that they are the only things keeping each other active, and that no external references to them exist. Reference counting alone in unable to handle these self sustaining graphs, and memory would be leaked.

The provided mechanism for dealing with graphs is weak references. A weak reference is one which will reference an object, but not prevent it from being deleted. As illustrated in Figure 2–2, the A->B->A issue could be resolved by changing it to the following.

Figure 2–2 Establishing a Weak Reference

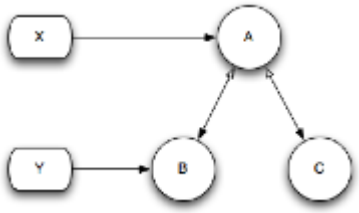
This figure is described in the text.

****

Where A now has a weak reference to B. If B were to reach a point where it was only referenced weakly, it would clear all weak references to itself and then be deleted. In this simple example that would also trigger the deletion of A, as B had held the only reference to A.

Weak references allow for construction of more complicated structures then this. But it becomes necessary to adopt a convention for which references are weak and which are strong. Consider a tree illustrated in Figure 2–3. The tree consists of nodes A, B, C; and two external references to the tree X, and Y.

Figure 2–3 Weak and Strong References to a Tree

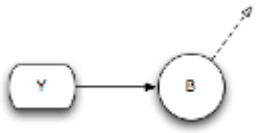


This figure is described in the text.

****

In this tree parent (A) use strong references to children (B, C), and children use weak references to their parent. With the picture as it is, reference Y could navigate the entire tree, starting at child B, and moving up to A, and then down to C. But what if reference X were to be reset to NULL? This would leave A only being weakly referenced and it would clear all weak references to itself, and be deleted. In deleting itself there would no longer be any references to C, which would also be deleted. At this point reference Y, without having taken any action would now refer to the situation illustrated in Figure 2–4.

Figure 2–4 Artifacts after Deleting the Weak References



This figure is described in the text.

****

This is not necessarily a problem, just a possibility which must be considered when using weak references. To work around this issue, the holder of Y would also likely need to maintain a reference to A to ensure the tree did not dissolve away unexpectedly.

See the Javadoc for `coherence::lang::WeakReference`, `WeakHandle`, and `WeakView` for usage details.

Virtual Constructors

As is typical in C++, referencing an object under construction can be dangerous. Specifically references to this are to be avoided within a constructor, as the object initialization has not yet completed. For managed objects, creating a handle to this from the constructor will in most cases cause the object to be destructed before it ever finishes being created. To address this, the object model includes support for virtual constructors. The virtual constructor `onInit` is defined by `Object` and can be overridden on derived classes. This method is called automatically by the object model just after construction completes, and just before the new object is returned from its static create method. Within the `onInit` method it is safe to reference this, to call virtual functions, and to hand out references to the new object to other class instances. Any derived implementation of onInit must include a call to `super::onInit()` to allow the parent class to also initialize itself.

Advanced Handle Types

In addition to the Handle and View smart pointers (discussed previously), the object model contains several other specialized variants that can be used. For the most part use of these specialized smart pointers is limited to writing new managed classes, and they will not show up in normal application code.

Table 2–1 Advanced Handle Types Supported by Coherence for C++

Type	Thread-safe?	View	Notes
coherence:lang:TypedHandle<T>	No	Conditional on T	The implementation of Handle and View
coherence:lang:BoxHandle<T>	No	Conditional on T	Allows automatic creating of managed objects from primitive types.
coherence:lang:TypedHolder<T>	No	May	May act as a Handle or a View. Basic types stored in collections
coherence:lang:Immutable<T>	No	Yes	Ensures `const`-ness of referring object.
coherence:lang:WeakHandle<T>	Yes	No	Does not prevent destruction of referring object.
coherence:lang:WeakView<T>	Yes	Yes	Does not prevent destruction of referring object.
coherence:lang:MemberHandle<T>	Yes	No	Transfers `const`-ness of enclosing object.
coherence:lang:MemberView<T>	Yes	Yes	Thread-safe View.
coherence:lang:MemberHolder<T>	Yes	May	May act a thread-safe Handle or View.

Thread Safety

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 of thread safety. Regardless of the presence or lack of higher level thread-safety, the reference count remains thread-safe.

Synchronization and Notification

Every `Object` in the object model can be a point of synchronization and notification. To synchronize an object and acquire its internal monitor, use a `COH_SYNCHRONIZED` macro code block, as shown in Example 2–16:

Example 2–16 A Sample COH_SYNCHRONIZED Macro Code Block

```
SomeClass::Handle h = getObjectFromSomewhere();

COH_SYNCHRONIZED (h)
    {
    // monitor of Object referenced by h has been acquired

    if (h->checkSomeState())
        {
        h->actOnThatState();
        }
    } // monitor is automatically released
```

The `COH_SYNCHRONIZED` block performs the monitor acquisition and release. You can safely exit the block with `return`, `throw`, `COH_THROW`, `break`, `continue`, and `goto` statements.

The `Object` class includes `wait()`, `wait(timed)`, `notify()`, and `notifyAll()` methods for notification purposes. To call these methods, the caller must have acquired the objects's monitor. Refer to `coherence::lang::Object` for details.

Read-write locks are also provided see `coherence::util::ThreadGate` for details.

Thread Safe Handles

The Handle, View, and Holder inner types defined on managed classes are *not* thread-safe. That is it is not safe without some form of synchronization 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.

This lack of thread-safety is an performance optimization and assumes that the vast majority of handles are stack allocated. Stack allocated handles, with very few provisos are by their very nature thread-safe. The provisos are that it is not a static or global variable, and that references to the handle are not shared with other threads.

There are then two types of code which must account for thread-safety of handles.

- 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 a handle to the object is supplied 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.

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`

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 then once within a code block. This assignment to a normal stack-based handle *is* thread-safe, and once completed allows for essentially free dereferencing of the stack-based handle. Note that when initializing thread-safe handles a reference to the parent class must be supplied as the first parameter, this reference can be obtained by calling `self()` on the parent object.

Example 2–17 illustrates a trivial example of such a usage:

Example 2–17 Thread-safe Handle

```
class Employee
    : public class_spec<Employee>
    {
    friend class factory<Employee>;

    protected:
        Employee(String::View vsName, int32_t nId)
            : super(), m_vsName(self(), vsName), m_nId(nId)
            {
            }

    public:
        String::View getName() const
            {
            return m_vsName; // read is automatically thread-safe
            }

        void setName(String::View vsName)
            {
            m_vsName = vsName; // write is automatically thread-safe
            }

        int32_t getId() const
            {
            return m_nId;
            }

    private:
        MemberView<String>    m_vsName;
        const int32_t         m_nId;
    };
```

The same basic technique can be applied to non-managed classes as well. This is made possible, by using the defined synchronization point as the "parent" for the thread-safe handles. With this approach you must ensure that the "parent" `m_hMutex` Object outlives the thread-safe handle `m_vsName`. This is easily accomplished by declaring them as data-members of the same class, and declaring the "parent" before the handle. This is illustrated in Example 2–18:

Example 2–18 Thread-safe Handle as a Non-Managed Class

```
class Employee
    {
```

```
    public:
        Employee(String::View vsName, int32_t nId)
            : m_hMutex(Object::create()), m_vsName(*m_hMutex, vsName), m_nId(nId)
            {
            }

    public:
        String::View getName() const
            {
            return m_vsName;
            }

        void setName(String::View vsName)
            {
            m_vsName = vsName;
            }

        int32_t getId() const
            {
            return m_nId;
            }

    private:
        const Object::Handle m_hMutex;
        MemberView<String>   m_vsName;
        const int32_t        m_nId;
    };
```

Diagnostics and Troubleshooting

This section provides information which can aid in diagnosing issues in applications which make use of the object mode.

Thread Dumps

Thread dumps are available for diagnostic and troubleshooting purposes. These thread dumps also include the stack trace. You can generate a thread dump by performing a `CTRL+BREAK` (Windows) or a `CTRL+BACKSLASH` (UNIX). Example 2–19 illustrates a sample thread dump:

Example 2–19 Sample Thread Dump

```
Thread dump Oracle Coherence for C++ v3.4b397 (Pre-release) (Apple Mac OS X x86
debug) pid=0xf853; spanning 190ms

"main" tid=0x101790 runnable: <native>
    at coherence::lang::Object::wait(long long) const
    at coherence::lang::Thread::dumpStacks(std::ostream&, long long)
    at main
    at start

"coherence::util::logging::Logger" tid=0x127eb0 runnable: Daemon{State=DAEMON_
RUNNING, Notification=false,
StartTimeStamp=1216390067197, WaitTime=0,
ThreadName=coherence::util::logging::Logger}
    at coherence::lang::Object::wait(long long) const
    at coherence::component::util::Daemon::onWait()
    at coherence::component::util::Daemon::run()
    at coherence::lang::Thread::run()
```

Memory Leak Detection

While the managed object model reference counting helps to prevent memory leaks they are still possible. The most common way in which they are triggered is through cyclical object graphs. The object model includes heap analysis support to help identify if leaks are occurring, by tracking the number of live objects in the system. Comparing this value over time provides a simple means of detecting if the object count is consistently increasing, and thereby likely leaking. Once a probable leak has been detected, the heap analyzer can help track it down as well, by provided statistics on what types of objects appeared to have leaked.

Coherence provides a pluggable `coherence::lang::HeapAnalyzer` interface. The `HeapAnalyzer` implementation can be specified by using the `tangosol.coherence.heap.analyzer system` property. The property can be set to one of the following values:

- `none`—No heap analysis will be performed.
- `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.
- `class`—The `coherence::lang::ClassBasedHeapAnalyzer` will be used. It provides heap analysis at the class level, that is it tracks the number of live instances of each class, and the associated byte level usage.
- `custom`—Lets you define your own analysis routines. You specify the name of a class registered with the `SystemClassLoader`.

Heap information is returned when you perform a `CTRL+BREAK` (Windows) or `CTRL+BACKSLASH` (UNIX). The following is an example of heap analysis information is returned by the class based analyzer:

Example 2–20 Data Returned by a Heap Analyzer

```
Space           Count           Class
44 B            1               coherence::lang::Integer32
70 B            1               coherence::lang::String
132 B           1               coherence::util::SafeHashMap::Entry

Total: 246 B, 3 objects, 3 classes
```

The above example was the heap analysis delta resulting from the insertion of a new entry into a `Map`.

Memory Corruption Detection

For all that the object model does to prevent memory corruption, it will typically be used along side non-managed code which could cause corruption. To combat this, the object model includes memory corruption detection support. When enabled, the object model's memory allocator will pad the beginning and end of each object allocation by a configurable number of pad bytes. This padding is encoded with a pattern which can later be validated to ensure that the pad has not been touched. If memory corruption occurs, and hits one of the pads, subsequent validations will detect the corruption. Validation is performed when the object is destroyed.

The debug version of the Coherence C++ API has padding enabled by default, using a pad size of 2*(word size), on each side of an object allocation. In a 32-bit build, this adds 16 bytes per object. Increasing the size of the padding will increase the chances of corruption hitting a pad, and thus the chance of detecting corruption.

The size of the pad can be configured by using the tangosol.coherence.heap.padding system property, which can be set to the number of bytes for the pre/post pad. Setting this system property to a non-zero value will enable the feature, and is available even in release builds.

Example 2–21 illustrates the results from an instance of memory corruption detection:

Example 2–21 Results from a Memory Corruption Run

```
Error during ~MemberHolder: coherence::lang::IllegalStateException: memory
corruption detected in 5B post-padding at offset 4 of memory allocated at 0x132095
```

3

Building Integration Objects for C++ Clients

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.

While the Coherence C++ API includes several POF serializable classes, custom data types require serialization support.

> **Note:** This document assumes familiarity with the Coherence C++ Object Model, including advanced concepts such as specification-based class definitions. For more information on these topics, see Chapter 2, "Understanding the Coherence C++ Object Model."

Serialization Options

While the Coherence C++ API offers a single serialization format (POF), it offers a variety of APIs for making a class serializable. Ultimately whichever approach is used, the same binary POF format is produced. The following approaches are available for making a class serializable:

- Use the `Managed<T>` adapter template, and add external free-function serializers. See "Managed<T> (Free-Function Serialization)" on page 3-2 for more information.
- Modify the data object to extend `Object`, and implement the `PortableObject` interface, to allow for object to self-serialize. See "PortableObject (Self-Serialization)" on page 3-4 for more information.
- Modify the data object to extend `Object`, and produce a `PofSerializer` class to perform external serialization. See "PofSerializer (External Serialization)" on page 3-6 for more information.

Table 3–1 lists some of the requirements and limitations of each approach.

Table 3–1 Requirements and Limitations of Serialization Options

Approach	Requires derivation from Object	Supports const data-members	External serialization routine	Requires zero-arg constructor
Managed<T>	No	Yes	Yes	Yes
PortableObject	Yes	No	No	Yes
PofSerializer	Yes	Yes	Yes	No

All of these approaches share certain similarities:

- you must implement serialization routines that will allow the data items to be encoded to POF
- the data object's fields are identified by using numeric indices
- the data object class and serialization mechanism must be registered with Coherence
- data objects used as cache keys, must support equality comparisons, and hashing

Managed<T> (Free-Function Serialization)

For most pre-existing data object classes, the use of `Managed<T>` offers the easiest means of integrating with Coherence for C++.

For a non-managed class to be compatible with `Managed<T>` it must have the following characteristics:

- zero parameter constructor (public or protected): *CustomType::CustomType*()
- copy constructor (public or protected): *CustomType::CustomType*(const *CustomType*&)
- equality comparison operator: bool operator==(const *CustomType*&, const *CustomType*&)
- `std::ostream` output function: `std::ostream& operator<<(std::ostream&, const` *CustomType*&)
- hash function: `size_t hash_value`(const *CustomType*&)

The following example presents a simple `Address` class, which has no direct knowledge of Coherence, but is suitable for use with the `Managed<T>` template.

> **Note:** In the interest of brevity, example class definitions are in-lined within the declaration.

Example 3–1 A Non-Managed Class

```
class Address
  {
  public:
    Address(const std::string& sCity, const std::String& sState, int nZip)
       : m_sCity(sCity), m_sState(sState), m_nZip(nZip) {}

    Address(const Address& that) // required by Managed<T>
       : m_sCity(that.m_sCity), m_sState(that.m_sState), m_nZip(that.m_nZip) {}

  protected:
    Address() // required by Managed<T>
      : m_nZip(0) {}

  public:
    std::string  getCity()  const {return m_sCity;}
    std::string  getState() const {return m_sState;}
    int          getZip()   const {return m_nZip;}

  private:
    const std::string m_sCity;
    const std::string m_sState;
```

```
    const int         m_nZip;
  };

bool operator==(const Address& addra, const Address& addrb) // required by
Managed<T>
  {
  return addra.getZip()   == addrb.getZip() &&
         addra.getState() == addrb.getState() &&
         addra.getCity()  == addrb.getCity();
  }

std::ostream& operator<<(std::ostream& out, const Address& addr) // required by
Managed<T>
  {
  out << addr.getCity() << ", " << addr.getState() << "  " << addr.getZip();
  return out;
  }

size_t hash_value(const Address& addr) // required by Managed<T>
  {
  return (size_t) addr.getZip();
  }
```

When combined with `Managed<T>`, this simple class definition becomes a true "managed object", and is usable by the Coherence C++ API. This definition has yet to address serialization. Serialization support is added Example 3–2:

Example 3–2 Managed Class using Serialization

```
#include "coherence/io/pof/SystemPofContext.hpp"

#include "Address.hpp"

using namespace coherence::io::pof;

COH_REGISTER_MANAGED_CLASS(1234, Address); // type ID registration—this must
                                           // appear in the .cpp not the .hpp

template<> void serialize<Address>(PofWriter::Handle hOut, const Address& addr)
  {
  hOut->writeString(0, addr.getCity());
  hOut->writeString(1, addr.getState());
  hOut->writeInt32 (2, addr.getZip());
  }

template<> Address deserialize<Address>(PofReader::Handle hIn)
  {
  std::string sCity  = hIn->readString(0);
  std::string sState = hIn->readString(1);
  int         nZip   = hIn->readInt32 (2);
  return Address(sCity, sState, nZip);
  }
```

Note: The serialization routines must have knowledge of Coherence. They do not, however, need to be part of the class definition file. They can be placed in an independent source file, and if they are linked into the final application, they will take effect.

With the above pieces in place, Example 3–3 illustrates instances of the `Address` class wrapped by using `Managed<T>` as `Managed<Address>`, and supplied to the Coherence APIs:

Example 3–3 Instances of a Class Wrapped with Managed<T>

```
// construct the non-managed version as usual
Address office("Redwood Shores", "CA", 94065);

// the managed version can be initialized from the non-managed version
// the result is a new object, which does not reference the original
Managed<Address>::View vOffice = Managed<Address>::create(office);
String::View           vKey    = "Oracle";

// the managed version is suitable for use with caches
hCache->put(vKey, vAddr);
vOffice = cast<Managed<Address>::View>(hCache->get(vKey));

// the non-managed class's public methods/fields remain accessible
assert(vOffice->getCity()  == office.getCity());
assert(vOffice->getState() == office.getState());
assert(vOffice->getZip()   == office.getZip());

// conversion back to the non-managed type may be performed using the
// non-managed class's copy constructor.
Address officeOut = *vOffice;
```

PortableObject (Self-Serialization)

The `PortableObject` interface is supported by the Java, .NET, and C++ versions of Coherence. It is an interface similar in concept to `java.io.Externalizable`, which allows an object to control how it is serialized. Any class which extends from `coherence::lang::Object` is free to implement the `coherence::io::pof::PortableObject` interface to add serialization support. Note that the class **must** extend from `Object`, which then dictates its life cycle.

In Example 3–4, we can re-write the above `Address` example as a managed class, and implement the `PortableObject` interface. In doing so, we are choosing to fully embrace the Coherence object model as part of the definition of the class, for instance using `coherence::lang::String` rather then `std::string` for data members.

Example 3–4 A Managed Class that Implements PortableObject

```
#include "coherence/lang.ns"

#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PortableObject.hpp"

#include "coherence/io/pof/SystemPofContext.hpp"

using namespace coherence::lang;

using coherence::io::pof::PofReader;
using coherence::io::pof::PofWriter;

class Address
  : public cloneable_spec<Address,
      extends<Object>,
      implements<PortableObject> >
```

```
  {
  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) {}

    Address() // required by PortableObject
      : m_nZip(0) {}

  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: // PortableObject interface    virtual void
writeExternal(PofWriter::Handle hOut) const
      {
      hOut->writeString(0, getCity());
      hOut->writeString(1, getState());
      hOut->writeInt32 (2, getZip());
      }

    virtual void readExternal(PofReader::Handle hIn)
      {
      m_vsCity  = hIn->readString(0);
      m_vsState = hIn->readString(1);
      m_nZip    = hIn->readInt32 (2);
      }

  public: // Objectinterface    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:
    MemberView<String> m_vsCity;
```

```
    MemberView<String> m_vsState;
    int32_t            m_nZip;
  };
COH_REGISTER_PORTABLE_CLASS(1234, Address); // type ID registration—this must
                                            // appear in the .cpp not the .hpp
```

Example 3–5 illustrates a managed variant of the `Address` that does not require the use of the `Managed<T>` adapter and can be used directly with the Coherence API:

Example 3–5 A Managed Class without Managed<T>

```
Address::View vAddr = Address::create("Redwood Shores", "CA", 94065);
String::View  vKey  = "Oracle";

hCache->put(vKey, vAddr);
Address::View vOffice = cast<Address::View>(hCache->get(vKey));
```

Serialization by using `PortableObject` is a good choice when the application has already decided to make use of the Coherence object model for representing its data objects. One drawback to PortableObject is that it does not easily support const data members, as the `readExternal` method is called after construction, and must assign these values.

PofSerializer (External Serialization)

The third serialization option is also the lowest level one. `PofSerializers` are classes that provide the serialization logic for other classes. For example, we will write an example `AddressSerializer` which can serialize a non-`PortableObject` version of the above managed `Address` class. Under the covers the prior two approaches were delegating through `PofSerializers`, they were just being created automatically rather then explicitly. In most cases, it will not be necessary to use this approach, as either the `Managed<T>` or `PortableObject` approaches will suffice. This approach is primarily of interest when you have a managed object with const data members. Consider Example 3–6, a non-`PortableObject` version of a managed `Address`.

Example 3–6 A non-PortableObject Version of a Managed Class

```
#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(vsCity), m_vsState(vsState), m_nZip(nZip) {}


    Address(const Address& that)
       : super(that), m_vsCity(that.m_vsCity), m_sState(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: // Objectinterface    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 String::View m_vsCity;
    const String::View m_vsState;
    const int32_t      m_nZip;
  };
```

Note that this version uses `const` data members, which makes it not well-suited for `PortableObject`. Example 3–7 illustrates an external class, `AddressSerializer`, which will be registered as being responsible for serialization of `Address` instances.

Example 3–7 An External Class Responsible for Serialization

```
#include "coherence/lang.ns"

#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PortableObject.hpp"
#include "coherence/io/pof/SystemPofContext.hpp"

#include "Address.hpp"

using namespace coherence::lang;

using coherence::io::pof::PofReader;
using coherence::io::pof::PofWriter;

class AddressSerializer
  : public class_spec<AddressSerializer,
      extends<Object>,
      implements<PofSerializer> >
  {
  friend class factory<AddressSerializer>;

  protected:
```

```
    AddressSerializer();

  public: // PofSerializer interface    virtual void serialize(PofWriter::Handle
hOut, Object::View v) const
        {
        Address::View vAddr = cast<Address::View>(v);
        hOut->writeString(0, vAddr->getCity());
        hOut->writeString(1, vAddr->getState());
        hOut->writeInt32 (2, vAddr->getZip());
        hOut->writeRemainder(NULL);
        }

    virtual Object::Holder deserialize(PofReader::Handle hIn) const
      {
      String::View vsCity  = hIn->readString(0);
      String::View vsState = hIn->readString(1);
      int32_t      nZip    = hIn->readInt32 (2);
      hIn->readRemainder();

      return Address::create(vsCity, vsState, nZip);
      }
  };
COH_REGISTER_POF_SERIALIZER(1234, TypedBarrenClass<Address>::create(),
AddressSerializer::create()); // This must appear in the .cpp not the .hpp
```

Usage of the `Address` remains unchanged:

```
Address::View vAddr = Address::create("Redwood Shores", "CA", 94065);
String::View  vKey  = "Oracle";

hCache->put(vKey, vAddr);
Address::View vOffice = cast<Address::View>(hCache->get(vKey));
```

POF Registration

In addition to being made serializable, each class must also be associated with numeric type IDs. These IDs are well-known across the cluster. Within the cluster, the ID-to-class mapping is configured by using POF user type configuration elements; within C++, the mapping is embedded within the class definition in the form of an ID registration, which is placed within the class's `.cpp` source file.

The registration technique differs slightly with each serialization approach:

- `COH_REGISTER_MANAGED_CLASS(ID, TYPE)`—for use with `Managed<T>`
- `COH_REGISTER_PORTABLE_CLASS(ID, TYPE)`—for use with `PortableObject`
- `COH_REGISTER_POF_SERIALIZER(ID, CLASS, SERIALIZER)`—for use with `PofSerializer`

Examples of these registrations can be found in above examples.

> **Note:** Registrations must appear only in the implementation (`.cpp`) files.

Need for Java Classes

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.

Performance

Both `Managed<T>` and `PortableObject` behind the scenes use a `PofSerializer` to perform serialization. Each of these approaches also adds some of its own overhead, for instance the `Managed<T>` approach involves the creation of a temporary version of non-managed form of the data object during deserialization. In the case of `PortableObject` the lack of support for const data members can have a cost as it avoids optimizations which would have been allowed for const data members. Overall the performance differences may be negligible, but if seeking to achieve the maximum possible performance, direct utilization of `PofSerializer` may be worth consideration.

4

Configuration and Usage for C++ Clients

This section provides general instructions for setting up Coherence for C++, integrating Coherence*Extend, and configuring the logger.

General Instructions

Configuring and using Coherence for C++ requires five basic steps:

1. Implement the C++ Application using the Coherence for C++ API. See Chapter 5, "Understanding the Coherence for C++ API." for more information on the API.
2. Compile and Link the application.
3. Configure paths.
4. Configure Coherence*Extend on both the client and on one or more JVMs within the cluster.
5. Configure a POF context on the client and on all of the JVMs within the cluster that run the Coherence*Extend clustered service.
6. Make sure the Coherence cluster is up and running.
7. Launch the C++ client application.

The following sections describe each of these steps in detail.

Implementing the C++ Application

Coherence for C++ provides an API that allows C++ applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster.

Coherence for C++ API consists of:

- a set of C++ public header files
- version of static libraries build by all supported C++ compilers
- several samples

The library allows C++ applications to connect to a Coherence*Extend clustered service instance running within the Coherence cluster using a high performance TCP/IP-based communication layer. The library sends all client requests to the Coherence*Extend clustered service which, in turn, responds to client requests by delegating to an actual Coherence clustered service (for example, a Partitioned or Replicated cache service).

Chapter 5, "Understanding the Coherence for C++ API", provides an overview of the key classes in the API. For a detailed description of the classes, see the API itself which is included in the `doc` directory of the Coherence for C++ distribution.

Compiling and Linking the Application

The platforms on which you can compile applications that employ Coherence for C++ are listed in the Supported Platforms and Operating Systems topic.

For example, the following `build.cmd` file for the Windows 32-bit platform builds, compiles, and links the files for the Coherence for C++ demo. The variables in the file have the following meanings:

- `OPT` and `LOPT` point to compiler options
- `INC` points to the Coherence for C++ API files in the include directory
- `SRC` points to the C++ header and code files in the common directory
- `OUT` points to the file that the compiler/linker should generate when it is finished compiling the code
- `LIBPATH` points to the library directory
- `LIBS` points to the Coherence for C++ shared library file

After setting these environment variables, the file compiles the C++ code and header files, the API files and the OPT files, links the `LOPT`, the Coherence for C++ shared library, the generated object files, and the `OUT` files. It finishes by deleting the object files. A sample run of the `build.cmd` file is illustrated in Example 4–1.

Example 4–1 Sample Run of the build.cmd File

```
@echo off
setlocal

set EXAMPLE=%1%

if "%EXAMPLE%"=="" (
   echo You must supply the name of an example to build.
   goto exit
   )

set OPT=/c /nologo /EHsc /Zi /RTC1 /MD /GR /DWIN32
set LOPT=/NOLOGO /SUBSYSTEM:CONSOLE /INCREMENTAL:NO
set INC=/I%EXAMPLE% /Icommon /I..\include
set SRC=%EXAMPLE%\*.cpp common\*.cpp
set OUT=%EXAMPLE%\%EXAMPLE%.exe
set LIBPATH=..\lib
set LIBS=%LIBPATH%\coherence.lib

echo building %OUT% ...
cl %OPT% %INC% %SRC%
link %LOPT% %LIBS% *.obj /OUT:%OUT%

del *.obj

echo To run this example execute 'run %EXAMPLE%'

:exit
```

Configure Paths

Set up the configuration path to the Coherence for C++ library. This involves setting an environment variable to point to the library. The name of the environment variable and the file name of the library will be different depending on your platform environment. For a list of the environment variables and library names for each platform, see "Setting Environment Variables for Compiling and Linking".

Configure Coherence*Extend

To configure Coherence*Extend, add the appropriate configuration elements to both the cluster and client-side cache configuration descriptors. The cluster-side cache configuration elements instruct a Coherence `DefaultCacheServer` to start a Coherence*Extend clustered service that will listen for incoming TCP/IP requests from Coherence*Extend clients. The client-side cache configuration elements are used by the client library connect to the cluster. The configuration specifies the IP address and port of one or more servers in the cluster that run the Coherence*Extend clustered service so that it can connect to the cluster. It also contains various connection-related parameters, such as connection and request timeouts.

Configure Coherence*Extend in the Cluster

For a Coherence*Extend client to connect to a Coherence cluster, one or more DefaultCacheServer JVMs within the cluster must run a TCP/IP Coherence*Extend clustered service. To configure a DefaultCacheServer to run this service, a proxy-scheme element with a child tcp-acceptor element must be added to the cache configuration descriptor used by the `DefaultCacheServer`.

For example, the cache configuration descriptor in Example 4–2 defines two clustered services, one that allows remote Coherence*Extend clients to connect to the Coherence cluster over TCP/IP and a standard Partitioned cache service. Since this descriptor is used by a `DefaultCacheServer`, it is important that the autostart configuration element for each service is set to true so that clustered services are automatically restarted upon termination. The `proxy-scheme` element has a `tcp-acceptor` child element which includes all TCP/IP-specific information needed to accept client connection requests over TCP/IP. The `acceptor-config` has also been configured to use a `ConfigurablePofContext` for its serializer. The C++ Extend client requires the use of POF for serialization.

See Chapter 3, "Building Integration Objects for C++ Clients" for more information on serialization and PIF/POF.

The Coherence*Extend clustered service configured below will listen for incoming requests on the `localhost` address and `port 9099`. When, for example, a client attempts to connect to a Coherence cache called `dist-extend`, the Coherence*Extend clustered service will proxy subsequent requests to the `NamedCache` with the same name which, in this example, will be a Partitioned cache.

Example 4–2 Cache Configuration for Two Clustered Services

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>dist-default</scheme-name>
```

```
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>
      <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <thread-count>5</thread-count>
      <acceptor-config>
        <tcp-acceptor>
          <local-address>
            <address>localhost</address>
            <port>9099</port>
          </local-address>
        </tcp-acceptor>
        <serializer>
          <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
        </serializer>
      </acceptor-config>
      <autostart>true</autostart>
    </proxy-scheme>
  </caching-schemes>
</cache-config>
```

Configuring Coherence*Extend on the Client

The key element within the Coherence*Extend client configuration is cache-config. This element contains the path to a cache configuration descriptor which contains the cache configuration. This cache configuration descriptor is used by the DefaultConfigurableCacheFactory.

A Coherence*Extend client uses the information within an `initiator-config` cache configuration descriptor element to connect to and communicate with a Coherence*Extend clustered service running within a Coherence cluster.

For example, the cache configuration descriptor in Example 4–3 defines a caching scheme that connects to a remote Coherence cluster. The `remote-cache-scheme` element has a `tcp-initiator` child element which includes all TCP/IP-specific information needed to connect the client with the Coherence*Extend clustered service running within the remote Coherence cluster.

When the client application retrieves a named cache with `CacheFactory` using, for example, the name `dist-extend`, the Coherence*Extend client will connect to the Coherence cluster by using TCP/IP (using the address `localhost` and `port 9099`) and return a `NamedCache` implementation that routes requests to the `NamedCache` with the same name running within the remote cluster. Note that the `remote-addresses` configuration element can contain multiple `socket-address` child elements. The Coherence*Extend client will attempt to connect to the addresses in a random order, until either the list is exhausted or a TCP/IP connection is established.

Example 4–3 A Caching Scheme that Connects to a Remote Coherence Cluster

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>local-*</cache-name>
      <scheme-name>local-example</scheme-name>
    </cache-mapping>

    <cache-mapping>
      <cache-name>dist-extend</cache-name>
      <scheme-name>extend-dist</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <local-scheme>
      <scheme-name>local-example</scheme-name>
    </local-scheme>

    <remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <initiator-config>
        <tcp-initiator>
          <remote-addresses>
            <socket-address>
              <address
system-property="tangosol.coherence.proxy.address">localhost</address>
              <port system-property="tangosol.coherence.proxy.port">9099</port>
            </socket-address>
          </remote-addresses>
          <connect-timeout>10s</connect-timeout>
        </tcp-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

Connection Error Detection and Failover

When a Coherence*Extend client service detects that the connection between the client and cluster has been severed (for example, due to a network, software, or hardware failure), the Coherence*Extend client service implementation (that is, `CacheService` or `InvocationService`) will raise a `MemberEventType.Left` event (by using the `MemberEventHandler delegate`) and the service will be stopped. If the client application attempts to subsequently use the service, the service will automatically restart itself and attempt to reconnect to the cluster. If the connection is successful, the service will raise a MemberEventType.Joined event; otherwise, a fatal exception will be thrown to the client application.

A Coherence*Extend service has several mechanisms for detecting dropped connections. Some mechanisms are inherit to the underlying protocol (such as TCP/IP in Extend-TCP), whereas others are implemented by the service itself. The latter

mechanisms are configured by using the `outgoing-message-handler` configuration element.

The primary configurable mechanism used by a Coherence*Extend client service to detect dropped connections is a request timeout. When the service sends a request to the remote cluster and does not receive a response within the request timeout interval (see `<request-timeout>`), the service assumes that the connection has been dropped. The Coherence*Extend client and clustered services can also be configured to send a periodic heartbeat over the connection (see `<heartbeat-interval>` and `<heartbeat-timeout>`). If the service does not receive a response within the configured heartbeat timeout interval, the service assumes that the connection has been dropped.

Configuring and Using the Coherence for C++ Client Library

To use the Coherence for C++ library in your C++ applications, you must link Coherence for C++ library with your application and provide a Coherence for C++ cache configuration and its location.

The location of the cache configuration file can be set by an environment variable specified in the sample application section or programmatically.

Setting the Configuration File Location with an Environment Variable

As described in "Runtime Library and Search Path" on page 1-3, the `tangosol.coherence.cacheconfig` system property can be used to specify the location of the cache configuration file. To set the configuration location on Windows execute:

```
c:\coherence_cpp\examples> set
tangosol.coherence.cacheconfig=config\extend-cache-config.xml
```

Setting the Configuration File Location Programmatically

You can set the location programmatically by using either `DefaultConfigurableCacheFactory::create` or `CacheFactory::configure` (using the `CacheFactory::loadXmlFile` helper method, if needed).

Example 4–4 Setting the Configuration File Location

```
static Handle coherence::net::DefaultConfigurableCacheFactory::create
(String::View vsFile = String::NULL_STRING)
```

The `create` method of the `DefaultConfigurableCacheFactory` class creates a new `Coherence cache` factory. The `vsFile` parameter specifies the name and location of the Coherence configuration file to load.

Example 4–5 Creating a Coherence Cache Factory

```
static void coherence::net::CacheFactory::configure (XmlElement::View vXmlCache,
XmlElement::View  vXmlCoherence = NULL)
```

The `configure` method configures the `CacheFactory` and local member. The `vXmlCache` parameter specifies an XML element corresponding to a `cache-config.dtd` and `vXmlCoherence` specifies an XML element corresponding to `coherence.dtd`.

Example 4–6 Configuring a CacheFactory and a Local Member

```
static XmlElement::Handle coherence::net::CacheFactory::loadXmlFile (String::View
vsFile)
```

The `loadXmlFile` method reads an `XmlElement` from the named file. This method does not configure the `CacheFactory`, but it can be used to obtain a configuration which can be supplied to the `configure` method. The parameter `vsFile` specifies the name of the file to read from.

The C++ code in Example 4–7 uses the `CacheFactory::configure` method to set the location of the cache configuration files for the server/cluster (`coherence-extend-config.xml`) and for the C++ client (`tangosol-operation-config.xml`).

Example 4–7 Setting the Cache Configuration File Location for the Server/Cluster

```
...
// Configure the cache
CacheFactory::configure(CacheFactory::loadXmlFile(String::create("C:\coherence-ext
end-config.xml")),

CacheFactory::loadXmlFile(String::create("C:\tangosol-operation-config.xml")));
...
```

Operational Configuration File (tangosol-coherence-override.xml)

The operational configuration override file (called `tangosol-coherence-override.xml` by default), controls the operational and runtime settings used by Oracle Coherence to create, configure and maintain its clustering, communication, and data management services. As with the Java client use of this file is optional for the C++ client.

In the case of a C++ client, the file can be used to specify or override general operations settings for a Coherence application that are not specifically related to caching. For a C++ client, the key elements are for logging, the Coherence product edition, and the location and role assignment of particular cluster members.

The operational configuration can be configured either programmatically or in the `tangosol-coherence-override.xml` file. To configure the operational configuration programmatically, specify an XML file that follows the `coherence.dtd` and contains at least one of the following elements in the `vXmlCoherence` parameter of the `CacheFactory::configure` method (`coherence::net::CacheFactory::configure (View vXmlCache, View vXmlCoherence)`).

- `license-config`—The `license-config` element contains subelements that allow you to configure the edition and operational mode for Coherence. The edition-name subelement specifies the product edition (such as Grid Edition, Enterprise Edition, Real Time Client, and so on) that the member will use. This allows multiple product editions to be used within the same cluster, with each member specifying the edition that it will be using. Only the RTC (real time client) and DC (data client) values are recognized for the Coherence for C++ client. The `license-config` is an optional subelement of the `coherence` element, and defaults to RTC.
- `logging-config`— The `logging-config` element contains subelements that allow you to configure how messages will be logged for your system. This element enables you to specify destination of the log messages, the severity level for logged

messages, and the log message format. The `logging-config` is a required subelement of the `coherence` element. For more information on logging, see "Configuring a Logger" on page 4-8.

- `member-identity`—The `member-identity` element specifies detailed identity information that is useful for defining the location and role of the cluster member. You can use this element to specify the name of the cluster, rack, site, machine, role, and so on, to which the member belongs. The `member-identity` is an optional subelement of the `cluster-config` element. Example 4–8 illustrates the contents of a sample `tangosol-coherence.xml` file.

Example 4–8 Sample Operational Configuration

```
<?xml version='1.0'?>

<coherence>
  <cluster-config>
    <member-identity>
      <site-name>extend site</site-name>
      <rack-name>rack 1</rack-name>
      <machine-name>machine 1</machine-name>
    </member-identity>
  </cluster-config>

  <logging-config>
    <destination>stderr</destination>
    <severity-level>5</severity-level>
    <message-format>(thread={thread}): {text}</message-format>
    <character-limit>8192</character-limit>
  </logging-config>

  <license-config>
    <edition-name>RTC</edition-name>
    <license-mode>production</license-mode>
  </license-config>
</coherence>
```

Operational Configuration Elements provides more detailed information on the operational configuration file and the elements that it can define.

Configuring a Logger

The Logger is configured using the `logging-config` element in the operational configuration file. The element provides the following attributes that can record detailed information about logged errors.

- `destination`—determines the type of `LogOutput` used by the Logger. Valid values are:
  - `stderr` for `Console.Error`
  - `stdout` for `Console.Out`
  - file path if messages should be directed to a file
- `severity-level`—determines the log level that a message must meet or exceed to be logged.
- `message-format`—determines the log message format.

- `character-limit`—determines the maximum number of characters that the logger daemon will process from the message queue before discarding all remaining messages in the queue. Example 4–9 illustrates an operational configuration that contains a logging configuration. For more information on operational configuration, see "Operational Configuration File (tangosol-coherence-override.xml)" on page 4-7.

Example 4–9 Operational Configuration File that Includes a Logger

```
<coherence>
  <logging-config>
    <destination>stderr</destination>
    <severity-level>5</severity-level>
    <message-format>(thread={thread}): {text}</message-format>
    <character-limit>8192</character-limit>
  </logging-config>
</coherence>
```

Launching a Coherence DefaultCacheServer Proxy

To start a `DefaultCacheServer` that uses the cluster-side Coherence cache configuration described earlier to allow Coherence for C++ clients to connect to the Coherence cluster by using TCP/IP, you need to do the following:

1. Change the current directory to the Oracle Coherence library directory (`%COHERENCE_HOME%\lib` on Windows and `$COHERENCE_HOME/lib` on UNIX).
2. Make sure that the paths are configured so that the Java command will run.
3. Start the `DefaultCacheServer` using the command line below:

Example 4–10 Sample Command to Start the DefaultCacheServer

```
java -cp coherence.jar -Dtangosol.coherence.cacheconfig=file://<path to the
server-side cache configuration descriptor>
     com.tangosol.net.DefaultCacheServer
```

5

Understanding the Coherence for C++ API

The Coherence for C++ API allows C++ applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster.

Documentation of the Coherence for C++ API is available in two locations. The online API documentation and also in the `doc` directory of the Coherence for C++ distribution.

CacheFactory

CacheFactory provides several static methods for retrieving and releasing NamedCache instances:

- `NamedCache::Handle getCache(String::View vsName)`—retrieves a `NamedCache` implementation that corresponds to the `NamedCache` with the specified name running within the remote Coherence cluster.
- `void releaseCache(NamedCache::Handle hCache)`—releases all local resources associated with the specified instance of the cache. After a cache is released, it can no longer be used. The content of the cache, however, is not affected.
- `void destroyCache(NamedCache::Handle hCache)`—destroys the specified cache across the Coherence cluster.

NamedCache

A `NamedCache` is a map of resources shared among members of a cluster. The `NamedCache` provides several methods used to retrieve the name of the cache and the service, and to release or destroy the cache:

- `String::View getCacheName()`—returns the name of the cache as a `String`.
- `CacheService::Handle getCacheService()`—returns a handle to the `CacheService` that this `NamedCache` is a part of.
- `bool isActive()`—specifies whether this `NamedCache` is active.
- `void release()`—releases the local resources associated with this instance of the `NamedCache`. The cache is no longer usable, but the cache contents are not affected.
- `void destroy()`—releases and destroys this instance of the `NamedCache`.

`NamedCache` interface also extends the following interfaces: `QueryMap`, `InvocableMap`, `ConcurrentMap`, `CacheMap` and `ObservableMap`.

QueryMap

A `QueryMap` can be thought of as an extension of the `Map` class with additional query features. These features allow the ability to query a cache using various filters. Filters are described in "Filter" on page 5-3.

- `Set::View` **keySet**`(Filter::View vFilter)`—returns a set of the keys contained in this map for entries that satisfy the criteria expressed by the filter.
- `Set::View entrySet(Filter::View vFilter)`—returns a set of the entries contained in this map that satisfy the criteria expressed by the filter. Each element in the returned set is a `Map::Entry` object.
- `Set::View entrySet(Filter::View vFilter, Comparator::View vComparator)`—returns a set of the entries contained in this map that satisfy the criteria expressed by the filter. Each element in the returned set is a `Map::Entry` object. This version of `entrySet` further guarantees that its iterator will traverse the set in ascending order based on the entry values which are sorted by the specified `Comparator` or according to the natural ordering.

Additionally, the `QueryMap` class includes the ability to add and remove indexes. Indexes are used to correlate values stored in the cache to their corresponding keys and can dramatically increase the performance of the `keySet` and `entrySet` methods.

- `void addIndex(ValueExtractor::View vExtractor, boolean_t fOrdered, Comparator::View vComparator)`—adds an index to this `QueryMap`. This enables you to correlate values stored in this indexed `Map` (or attributes of those values) to the corresponding keys in the indexed `Map` and increase the performance of `keySet` and `entrySet` methods.
- `void removeIndex(ValueExtractor::View vExtractor)`—removes an index from this `QueryMap`.

See "Query the Cache for C++ Clients" for a more in depth look at queries. See also the C++ examples in "Simple Queries" on page 10-1

ObservableMap

An `ObservableMap` provides an application with the ability to listen for cache changes. Applications that implement `ObservableMap` can add key and filter listeners to receive events from any cache, regardless of whether that cache is local, partitioned, near, replicated, using read-through, write-through, write-behind, overflow, disk storage, and so on. `ObservableMap` also provides methods to remove these listeners.

- `void addKeyListener(MapListener::Handle hListener, Object::View vKey, bool fLite)`—adds a map listener for a specific key.
- `void removeKeyListener(MapListener::Handle hListener, Object::View vKey)`—removes a map listener that previously signed up for events about a specific key.
- `void addFilterListener(MapListener::Handle hListener, Filter::View vFilter = NULL, bool fLite = false)`—adds a map listener that receives events based on a filter evaluation.
- `void removeFilterListener(MapListener::Handle hListener, Filter::View vFilter = NULL)`—removes a map listener that previously signed up for events based on a filter evaluation.

See the C++ examples in "Signing Up for all Events" on page 12-5.

InvocableMap

An InvocableMap is a cache against which both entry-targeted processing and aggregating operations can be invoked. The operations against the cache contents are executed by (and thus within the localized context of) a cache. This is particularly efficient in a distributed environment because it localizes processing: the processing of the cache contents are moved to the location at which the entries-to-be-processed are being managed. For more information abuot processors and aggregators, see "Entry Processors" on page 5-5 and "Entry Aggregators" on page 5-5.

- `Object::Holder invoke(Object::View vKey, EntryProcessor::Handle hAgent)`—invokes the passed processor (`EntryProcessor`) against the entry (`Entry`) specified by the passed key, returning the result of the invocation.
- `Map::View invokeAll(Collection::View vCollKeys, EntryProcessor::Handle hAgent)`—invokes the passed processor (`EntryProcessor`) against the entries (`Entry` objects) specified by the passed keys, returning the result of the invocation for each.
- `Map::View invokeAll(Filter::View vFilter, EntryProcessor::Handle hAgent)`—invokes the passed processor (`EntryProcessor`) against the entries (`Entry` objects) that are selected by the given filter, returning the result of the invocation for each.
- `Object::Holder aggregate(Collection::View vCollKeys, EntryAggregator::Handle hAgent)`—performs an aggregating operation against the entries specified by the passed keys.
- `Object::Holder aggregate(Filter::View vFilter, EntryAggregator::Handle hAgent)`—performs an aggregating operation against the entries that are selected by the given filter.

See the C++ examples in "InvocableMap Aggregation" on page A-17 and "InvocableMap Invoke All" on page A-17.

Filter

Filter provides the ability to filter results and only return objects that meet a given set of criteria. All filters must implement Filter. Filters are commonly used with the `QueryMap` API to query the cache for entries that meet a given criteria. See also "QueryMap" on page 5-2.

- `bool evaluate(Object::View v)`—applies a test to the specified object and returns true if the test passes, false otherwise.

Coherence for C++ includes many concrete `Filter` implementations in the `coherence::util::filter` namespace. Below are several commonly used filters:

- `EqualsFilter` is used to test for equality. To create an `EqualsFilter` to test that an object equals 5:

Example 5–1 Using the EqualsFilter Method

```
EqualsFilter::View vEqualsFilter =
EqualsFilter::create(IdentityExtractor::getInstance(), Integer32::valueOf(5));
```

- `GreaterEqualsFilter` is used to test a "Greater or Equals" condition. To create a `GreaterEqualsFilter` that tests that an objects value is `>= 55`:

Example 5–2 Using the GreaterEqualsFilter Method

```
GreaterEqualsFilter::View vGreaterEqualsFilter =
GreaterEqualsFilter::create(IdentityExtractor::getInstance(),
Integer32::valueOf(55));
```

- `LikeFilter` is used for pattern matching. To create a `LikeFilter` that tests that the string representation of an object begins with "`Belg`":

Example 5–3 Using the LikeFilter Method

```
LikeFilter::View vLikeFilter =
LikeFilter::create(IdentityExtractor::getInstance(), "Belg%");
```

Some filters can be used to combine two filters to create a compound condition.

- `AndFilter` is used to combine two filters to create an "`AND`" condition. To create an `AndFilter` that tests that an objects value is greater than 10 and less than 20:

Example 5–4 Using the AndFilter Method

```
AndFilter::View vAndFilter = AndFilter::create(
        GreaterFilter::create(IdentityExtractor::getInstance(),
Integer32::valueOf(10)),
        LessFilter::create(IdentityExtractor::getInstance(),
Integer32::valueOf(20)));
```

- `OrFilter` is used to combine two filters to create an "`OR`" condition. To create an `OrFilter` that tests that an object's value is less than `10` or greater than `20`:

Example 5–5 Using the OrFilter Method

```
OrFilter::View vOrFilter = OrFilter::create(
        LessFilter::create(IdentityExtractor::getInstance(),
Integer32::valueOf(10)),
        GreaterFilter::create(IdentityExtractor::getInstance(),
Integer32::valueOf(20)));
```

Value Extractors

An Extractor is used to extract values from an object and to provide an identity for the extraction. All extractors must implement `ValueExtractor`.

> **Note:** All concrete extractor implementations must also explicitly implement the `hashCode` and `equals` functions in a way that is based solely on the object's serializable state.

- `Object::Holder extract(Object::Holder ohTarget)`—extracts the value from the passed object.
- `bool equals(Object::View v)`—compares the `ValueExtractor` with another object to determine equality. Two `ValueExtractor` objects, `ve1` and `ve2` are considered equal if and only if `ve1->extract(v)` equals `ve2->extract(v)` for all values of `v`.

- `size32_t hashCode()`—determine a hash value for the `ValueExtractor` object according to the general `Object#hashCode()` contract.

Coherence for C++ includes the following extractors:

- `ChainedExtractor`—is a composite `ValueExtractor` implementation based on an array of extractors. The extractors in the array are applied sequentially left-to-right, so a result of a previous extractor serves as a target object for a next one.
- `ComparisonValueExtractor`—returns a result of comparison between two values extracted from the same target.
- `IdentityExtractor`—is a trivial implementation that does not actually extract anything from the passed value, but returns the value itself.
- `KeyExtractor`—is a special purpose implementation that serves as an indicator that a query should be run against the key objects rather than the values.
- `MultiExtractor`—is a composite ValueExtractor implementation based on an array of extractors. All extractors in the array are applied to the same target object and the result of the extraction is a List of extracted values.
- `ReflectionExtractor`—extracts a value from a specified object property.

See the C++ examples in "Query Concepts" on page 10-3.

Entry Processors

An `EntryProcessor` is an invokable agent that operates against the entry objects within a cache. All entry processors must implement `EntryProcessor`.

- `Object::Holder process(InvocableMap::Entry::Handle hEntry)`—process the specified entry.
- `Map::View processAll(Set::View vSetEntries)`—process a collection of entries.

Coherence for C++ includes several `EntryProcessor` implementations in the `coherence::util::processor` namespace.

See the C++ example in "Sample Code for the hellogrid Example" on page A-15.

Entry Aggregators

An `EntryAggregator` represents processing that can be directed to occur against some subset of the entries in an `InvocableMap`, resulting in an aggregated result. Common examples of aggregation include functions such as minimum, maximum, sum, and average. However, the concept of aggregation applies to any process that must evaluate a group of entries to come up with a single answer. Aggregation is explicitly capable of being run in parallel, for example in a distributed environment.

All aggregators must implement the `EntryAggregator` interface:

- `Object::Holder aggregate(Collection::View vCollKeys)`—processes a collection of entries to produce an aggregate result.

Coherence for C++ includes several `EntryAggregator` implementations in the `coherence::util::aggregator` namespace.

Note: Like cached value objects, all custom `Filter`, `ValueExtractor`, `EntryProcessor`, and `EntryAggregator` implementation classes must be correctly registered in the POF context of the C++ application and cluster-side node to which the client is connected. As such, corresponding Java implementations of the custom C++ types must be created, compiled, and deployed on the cluster-side node. Note that the actual execution of the these custom types is performed by the Java implementation and not the C++ implementation. See Chapter 3, "Building Integration Objects for C++ Clients," for additional details.

6

Sample Applications for C++ Clients

The instructions and command line examples assume that you have extracted the Java Coherence 3.4 archive and the C++ Coherence 3.4 archive onto your file system:

- the Java Coherence 3.4 archive was extracted into the top-level of your file system. For example, it would appear as `C:\coherence` on Windows.
- the C++ Coherence 3.4 archive was extracted into the Java Coherence 3.4 root directory. The root directory for the C++ version is `coherence-cpp`. Thus, on Windows it would appear in the file system as `C:\coherence\coherence-cpp`.

See Chapter 1, "Requirements, Installation, and Deployment for Coherence for C++" for more information on installing Coherence for C++.

> **Note:** Coherence C++ does not have any local dependencies on the Java installation. While this section assumes that you have installed both the Java and C++ versions of Coherence on the machine that will be used to run the examples, installation of the Java version is optional. If the Java version is not installed, the Cache Server will need to be running on a remote machine and the Java console example will not be available.

Coherence for C++ provides the following sample applications in the `coherence-cpp/examples` directory of the installed product:

- `console`—A command line application that enables you to interact with the cache using simple commands.
- `hellogrid`—An example of basic cache access.
- `contacts`—An example of how to store pre-existing (that is, non-Coherence) C++ classes in the grid.

Prerequisites for Building and Running the Sample Applications

The requirements for running a sample include:

- The Coherence C++ shared library, found under the platform specific `coherence-cpp/lib` directory of the installation.
- A Coherence extend cache configuration file, found under the `coherence-cpp/examples/config` directory.
- A running Coherence 3.4 Proxy Service and Cache Server; these are Java components.

Starting a Coherence Proxy Service and Cache Server

A sample command to start the proxy service and cache server is listed below. You must be sure to point the proxy at the server cache configuration file, such as `extend-server-config.xml` provided in the `config` directory. For example, on Windows execute:

Example 6–1 Sample Command to Start the Proxy Service and the Cache Server

```
c:\coherence\lib> java
-Dtangosol.coherence.cacheconfig=c:\coherence\coherence-cpp\examples\config\extend
-server-config.xml -cp coherence.jar "com.tangosol.net.DefaultCacheServer"
```

Note: For the contacts example you will also need to use the additional POF configuration and custom classes included in the `examples/java/ContactCache` directory.

Building the Sample Applications

The Coherence for C++ distribution includes platform specific build scripts. Each script takes a single command line parameter, which is the name of the sample to build. For example, to build the console example on Windows, open a new command prompt window and execute:

```
c:\coherence\coherence-cpp\examples> build console
```

The sample executable will be created within the particular `examples` subdirectory, that is:

```
c:\coherence\coherence-cpp\examples\console\console.exe
```

Note: On UNIX platforms you may need to mark the build script as executable first `chmod +x build` from the `examples` directory.

Starting a Sample Application

Now that configuration has been specified and the proxy/cache server has been started, you can start the client. For example, to run the console example on Windows, run the following command from the examples directory:

```
c:\coherence\coherence-cpp\examples> run console
```

The Coherence logging for the application will be directed to console.log in the examples directory.

Running the console Example

The console example enables you to enter data into the cache through a C++ console, then read it out through a Java console. Once you start the console example (by running `run console`), you will be provided with the familiar `Map(?):` prompt from the console. The C++ console supports a subset of the commands available from Java, enter `help` to get the list. The caches you can interact with are defined within the `extend-cache-config.xml` configuration file, but basically all you need to worry about is that `local-*` caches will be local only, and `dist-*` caches will be remote

and use PIF/POF, and `near-*` will pull remote data into an in-process coherent near cache.

1. Enter `cache dist-hello` to connect to the cache. Enter the commands illustrated in the following example to enter data into the cache and display it.

```
Map(?): cache dist-hello

Map(dist-hello): put hello world
NULL

Map(dist-hello): get hello
world

Map(dist-hello): size
1

Map(dist-hello): put from C++
NULL

Map(dist-hello): list
from = C++
hello = world

Map(dist-hello):
```

2. Launch a Java console to interact with the C++ console. Note that in the startup command, the Java client application must point to the same cache configuration as the C++ client. For example, on Windows, open a new command prompt window and execute the following command. (Note, the command is broken into two lines for formatting purposes).

```
c:\coherence\lib> java -Dtangosol.coherence.cacheconfig=
  c:\coherence\coherence-cpp\examples\config\extend-cache-config.xml -jar
coherence.jar
```

3. Use the same console syntax that you used in the C++ console to access the cache. For example, on Windows, open a new command prompt window and execute the commands illustrated in the following figure:

```
Map(?): cache dist-hello
2008-04-25 09:01:02.207 Oracle Coherence GE 3.4/396 Alpha <D5>
(thread=DistributedCache, member=3): Service
DistributedCache joined the cluster with senior service member 1
2008-04-25 09:01:02.239 Oracle Coherence GE 3.4/396 Alpha <D5>
(thread=DistributedCache, member=3): Service
DistributedCache: received ServiceConfigSync containing 259 entries
<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <local-scheme//>
  </backing-map-scheme>
  <autostart>true
```

4. Now that you've run the example, you are encouraged to have a look at the code. See "Sample Code for the console Example" on page A-1. Each sample has a corresponding directory under `examples` which contains its sample specific

source. There is also a `common` directory which contains source used in all samples.

Running the hellogrid Example

The `hellogrid` example exercises the cache by entering various types of data into the cache and reading them out, printing cache contents, querying the cache, and so on. Follow these steps to build and run the `hellogrid` example:

1. Stop any instance of the proxy and cache server that may be currently running.
2. Start a new instance of the proxy and cache server.
3. Build the `hellogrid` example. For example, on Windows, open a new command prompt window and invoke the build file.

```
C:\coherence\coherence-cpp\examples>build hellogrid
building hellogrid\hellogrid.exe ...
hellogrid.cpp
ContactInfo.cpp
PortableContactInfo.cpp
StreamParser.cpp
Generating Code...
C:\coherence\coherence-cpp\examples>
```

4. Run the `hellogrid` example. The window will display output similar to the following:

```
C:\coherence\coherence-cpp\examples>run hellogrid
retrieved cache "dist-hello" containing 0 entries
        put: hello = grid
        get: hello = grid
        get: dummy = NULL
entire cache contents:
        34567 = 8.9
        23456 = 7.8
        12345 = 6.7
        hello = grid
updated cache contents:
        34567 = 8.9
        23456 = 7.8
        12345 = 6.7
        45678 = 9.1
filtered cache contents by coherence::util::filter::GreaterFilter:
(IdentityExtr
actor, 7)
        34567 = 8.9
        23456 = 7.8
        45678 = 9.1
minimum: 6.7
increment results by 6.7
        34567 = 15.6
        23456 = 14.5
        12345 = 13.4
        45678 = 15.8

C:\coherence\coherence-cpp\examples>
```

5. Now that you've run the example, you are encouraged to have a look at the code. See "Sample Code for the hellogrid Example" on page A-15. Each sample has a corresponding directory under `examples` which contains its sample specific

source. There is also a `common` directory which contains source used in all samples.

Running the contacts Example

The contact example enables you to enter names and addresses into the cache, then query to display the entries. The following commands can be run from the example:

- `help`—returns a list of commands that the example can run
- `bye`—stops the example and returns you to the command prompt
- `create`—responds with prompts for a person's contact information: name, street address, city, state, zip code
- `find`—prompts you for a name. The example will return the contact information associated with the name.

Follow these steps to build and run the `contacts` example:

1. Stop any instance of the proxy and cache server that may be currently running.
2. Start a new instance of the proxy and cache server.
3. Build the `contacts` example. For example, on Windows, open a new command prompt window and invoke the build file.

```
C:\coherence\coherence-cpp\examples>build contacts
building contacts\contacts.exe ...
contacts.cpp
ContactInfo.cpp
PortableContactInfo.cpp
StreamParser.cpp
Generating Code...
C:\coherence\coherence-cpp\examples>
```

4. Run the `contacts` example. The window will display output similar to the following:

```
C:\coherence\coherence-cpp\examples>run contacts
contacts> help
commands are:
bye
create
find <street | city | state | zip | all>
contacts>
```

5. Exercise the example by entering the commands `help`, `create`, `find`, and `bye`.

```
contacts> help
commands are:
bye
create
find <street | city | state | zip | all>

contacts> create
Name: Tom
Street: Oracle Parkway
City: Redwood Shores
State: California
Zip: 94065
storing: ContactInfo(Name=Tom, Street=Oracle Parkway, City=Redwood Shores,
State
```

```
=California, Zip=94065)

contacts> find
Name: Tom
ContactInfo(Name=Tom, Street=Oracle Parkway, City=Redwood Shores,
State=California, Zip=94065)

contacts> bye

C:\coherence\coherence-cpp\examples>
```

6. Now that you've run the example, you are encouraged to have a look at the code. See "Sample Code for the contacts Example" on page A-6. Each sample has a corresponding directory under `examples` which contains its sample specific source. There is also a `common` directory which contains source used in all samples.

7

Configuring a Local Cache for C++ Clients

A **Local Cache** is a cache that is local to (completely contained within) a particular C++ application. There are several attributes of the Local Cache that are particularly interesting:

- The local cache implements the same interfaces that the remote caches implement, meaning that there is no programming difference between using a local and a remote cache.
- The Local Cache can be size-limited. This means that the Local Cache can restrict the number of entries that it caches, and automatically evict entries when the cache becomes full. Furthermore, both the sizing of entries and the eviction policies are customizable, for example allowing the cache to be size-limited based on the memory used by the cached entries. The default eviction policy uses a combination of Most Frequently Used (MFU) and Most Recently Used (MRU) information, scaled on a logarithmic curve, to determine what cache items to evict. This algorithm is the best general-purpose eviction algorithm because it works well for short duration and long duration caches, and it balances frequency versus recentness to avoid cache thrashing. The pure LRU and pure LFU algorithms are also supported, and the ability to plug in custom eviction policies.
- The Local Cache supports automatic expiration of cached entries, meaning that each cache entry can be assigned a time-to-live value in the cache. Furthermore, the entire cache can be configured to flush itself on a periodic basis or at a preset time.
- The Local Cache is thread safe and highly concurrent.
- The Local Cache provides cache "get" statistics. It maintains hit and miss statistics. These runtime statistics can be used to accurately project the effectiveness of the cache, and adjust its size-limiting and auto-expiring settings accordingly while the cache is running.

For additional information, see *"Local Cache"* in *"Getting Started with Oracle Coherence"*.

Configuring the Local Cache

The key element for configuring the Local Cache is `<local-scheme>`. Local caches are generally nested within other cache schemes, for instance as the front-tier of a near-scheme. Thus, this element can appear as a subelement of any of these elements in the `coherence-cache-config` file: `<caching-schemes>`, `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<near-scheme>`, `<versioned-near-scheme>`, `<overflow-scheme>`, `<read-write-backing-map-scheme>`, and `<versioned-backing-map-scheme>`.

The <local-scheme> provides several optional subelements that let you define the characteristics of the cache. For example, the <low-units> and <high-units> subelements allow you to limit the cache in terms of size. Once the cache reaches its maximum allowable size it prunes itself back to a specified smaller size, choosing which entries to evict according to a specified eviction-policy (<eviction-policy>). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (<unit-calculator>).

You can also limit the cache in terms of time. The <expiry-delay> subelement specifies the amount of time from last update that entries will be kept by the cache before being marked as expired. Any attempt to read an expired entry will result in a reloading of the entry from the configured cache store (<cachestore-scheme>). Expired values are periodically discarded from the cache based on the flush-delay.

If a <cache-store-scheme> is not specified, then the cached data will only reside in memory, and only reflect operations performed on the cache itself. See <local-scheme> for a complete description of all of the available subelements.

The XML code in Example 7–1 illustrates the configuration of a Local Cache. See *"Sample Cache Configurations"* for additional examples.

Example 7–1 Local Cache Configuration

```
<?xml version="1.0"?>

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example-local-cache</cache-name>
      <scheme-name>example-local</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <local-scheme>
      <scheme-name>example-local</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>32000</high-units>
      <low-units>10</low-units>
      <unit-calculator>FIXED</unit-calculator>
      <expiry-delay>10ms</expiry-delay>
      <flush-delay>1000ms</flush-delay>
      <cachestore-scheme>
        <class-scheme>
          <class-name>ExampleCacheStore</class-name>
        </class-scheme>
      </cachestore-scheme>
      <pre-load>true</pre-load>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

Obtaining a Local Cache Reference for C++ Clients

A reference to a configured Local Cache can be obtained by name by using the CacheFactory class:

```
NamedCache::Handle hCache = CacheFactory::GetCache("example-local-cache");
```

Cleaning Up Resources Associated with a LocalCache

Instances of all `NamedCache` implementations, including `LocalCache`, should be explicitly released by calling the `NamedCache::Release()` method when they are no longer needed, to free up any resources they might hold.

If the particular `NamedCache` is used for the duration of the application, then the resources will be cleaned up when the application is shut down or otherwise stops. However, if it is only used for a period, the application should call its Release() method when it has finished using it.

8

Configuring a Near Cache for C++ Clients

This section describes the Near Cache as it pertains to Coherence for C++ clients. For a complete discussion of the concepts behind a Near Cache, its configuration, and ways to keep it synchronized with the back tier see *"Near Cache"* in *"Getting Started with Oracle Coherence"*.

In Coherence for C++, the Near Cache is a `coherence::net::NamedCache` implementation that wraps the front cache and the back cache using a read-through/write-through approach. If the back cache implements the `ObservableCache` interface, then the Near Cache can use either the listen `None`, `Present`, `All`, or `Auto` strategy to invalidate any front cache entries that might have been changed in the back cache.

A typical Near Cache is configured to use a local cache (thread safe, highly concurrent, size-limited and/or auto-expiring local cache) as the front cache and a remote cache as a back cache. A Near Cache is configured by using the near-scheme which has two child elements: a front-scheme for configuring a local (front) cache and a back-scheme for defining a remote (back) cache.

Configuring the Near Cache

A Near Cache is configured by using the `<near-scheme>` element in the `coherence-cache-config` file. This element has two required subelements: `front-scheme` for configuring a local (front-tier) cache and a `back-scheme` for defining a remote (back-tier) cache. While a local cache (`<local-scheme>`) is a typical choice for the front-tier, you can also use non-JVM heap based caches, (`<external-scheme>` or `<paged-external-scheme>`) or schemes based on Java objects (`<class-scheme>`).

The remote or back-tier cache is described by the `<back-scheme>` element. A back-tier cache can be either a distributed cache (`<distributed-scheme>`) or a remote cache (`<remote-cache-scheme>`). The `<remote-cache-scheme>` element enables you to use a clustered cache from outside the current cluster.

Optional subelements of `<near-scheme>` include `<invalidation-strategy>` for specifying how the front-tier and back-tier objects will be kept synchronized and `<listener>` for specifying a listener which will be notified of events occurring on the cache.

For an example configuration, see *"Sample Near Cache Configuration"*. The elements in the file are described in the `<near-scheme>` topic.

Obtaining a Near Cache Reference with C++

A reference to a configured Near Cache can be obtained by name by using the `coherence::net::CacheFactory` class:

Example 8–1 Reference to a Configured Near Cache

```
NamedCache::Handle hCache = CacheFactory::getCache("example-near-cache");
```

Cleaning up Resources Associated with a Near Cache

Instances of all `NamedCache` implementations, including `NearCache`, should be explicitly released by calling the `NamedCache::release()` method when they are no longer needed, to free up any resources they might hold.

If the particular `NamedCache` is used for the duration of the application, then the resources will be cleaned up when the application is shut down or otherwise stops. However, if it is only used for a period, the application should call its `release()` method when finished using it.

9

Perform Continuous Query for C++ Clients

While Coherence provides the ability to obtain a point in time query result from a Coherence cache and the ability to receive events that would change the result of that query, it also provides a feature that combines a query result with a continuous stream of related events to maintain an up-to-date query result in a real-time fashion. This capability is called *Continuous Query* because it has the same effect as if the desired query had zero latency *and* the query were being executed several times every millisecond!

A continuous query cache is similar to a materialized view in the Oracle database. A materialized view copies data queried from the database tables into the view. If there are any changes to the data in the database, then the data in the view is automatically updated. This enables you to see changes to the result set. In continuous query, a local copy of the cache is created on the client. Filters allow you to limit the size and content of the cache. Combined with an event listener, the cache can be updated in real time.

For example, assume that you want to monitor, in real time, all sales orders for several customers. To do this, you can create a continuous query cache and set up an event listener that will listen for any events pertaining to the customers. Coherence will query for all of the data objects on the grid that pertain to a particular customer and copy them to a local cache. The event listener on the query will listen for any inserts, updates, or deletes that take place on the grid for the customer. When an event occurs, the local copy of the customer data is updated.

Uses of Continuous Query Caching

There are several different general use categories for Continuous Query Caching:

- It is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.
- It is ideal for situations in which an application repeats a particular query and would benefit from always having instant access to the up-to-date result of that query.
- A Continuous Query Cache is analogous to a *materialized view* and is useful for accessing and manipulating the results of a query using the standard `NamedCache` API, and receiving an ongoing stream of events related to that query.
- A Continuous Query Cache can be used in a manner similar to a Near Cache because it maintains an up-to-date set of data locally *where it is being used*, for example, on a particular server node or on a client. Note that while a Near Cache is invalidation-based, a Continuous Query Cache actually maintains its data in an up-to-date manner.

By combining the Coherence*Extend functionality with Continuous Query Caching, an application can support literally tens of thousands of concurrent users.

> **Note:** Continuous Query Caches are useful in almost every type of application, including both client-based and server-based applications, because they provide the ability to very easily and efficiently maintain an up-to-date local copy of a specified sub-set of a much larger and potentially distributed cached data set.

The Coherence Continuous Query Cache

The Coherence implementation of Continuous Query is found in the ContinuousQueryCache class. This class, like all Coherence caches, implements the standard NamedCache interface, which includes the following capabilities:

- Cache access and manipulation using the Map interface: `NamedCache` extends the `Map` interface, which is based on the `Map` interface from the Java Collections Framework.
- Events for all object modifications that occur within the cache: `NamedCache` extends the `ObservableMap` interface.
- Identity-based clusterwide locking of objects in the cache: `NamedCache` extends the `ConcurrentMap` interface.
- Querying the objects in the cache: `NamedCache` extends the `QueryMap` interface.
- Distributed Parallel Processing and Aggregation of objects in the cache: `NamedCache` extends the `InvocableMap` interface.

Since the `ContinuousQueryCache` implements the `NamedCache` interface, which is the same API provided by all Coherence caches, it is extremely simple to use, and it can be easily substituted for another cache when its functionality is called for.

Defining a Continuous Query Cache

There are two features that define a Continuous Query Cache:

- The underlying cache that the Continuous Query is based on.
- A query of the underlying cache that produces the sub-set that the Continuous Query Cache will cache.

The underlying cache can be any Coherence cache, including another Continuous Query Cache. The most straight-forward way of obtaining a cache is by using the CacheFactory class. This class enables you to create a cache simply by specifying its name. It will be created automatically and its configuration will be based on the application's cache configuration elements. For example, the following line of code creates a cache named orders:

```
NamedCache::Handle hCache = CacheFactory::getCache("orders");
```

The query is the same type of query that would be used to query any other cache. Example 9–1 illustrates how you can use code filters to find a given trader with a given order status:

Example 9–1 Using Filters for Querying

```
ValueExtractor::Handle hTraderExtractor =
ReflectionExtractor::create("getTrader");
```

```
ValueExtractor::Handle hStatusExtractor =
ReflectionExtractor::create("getStatus");

Filter::Handle hFilter = AndFilter::create(EqualsFilter::create(hTraderExtractor,
vTraderId),
                                EqualsFilter::create(hStatusExtractor, vStatus));
```

Normally, to query a cache, you could use one of the methods from the `QueryMap` class. For example, to obtain a snap-shot of all open trades for this trader:

```
Set::View vSetOpenTrades = hCache->entrySet(hFilter);
```

In contrast, the Continuous Query Cache is constructed from the `ContinuousQueryCache::create` method, passing the cache and the filter:

```
ContinuousQueryCache::Handle hCacheOpenTrades  =
ContinuousQueryCache::create(hCache, hFilter);
```

Cleaning up Resources Associated with a Continuous Query Cache

A Continuous Query Cache places one or more event listeners on its underlying cache. If the Continuous Query Cache is used for the duration of the application, then the resources will be cleaned up when the node is shut down or otherwise stops. However, if the Continuous Query Cache is only used for a period, then the application must call the release() method on the Continuous Query Cache when it is done using it.

Caching Only Keys, or Caching Both Keys and Values

When constructing a Continuous Query Cache, you can specify that the cache should only keep track of the keys that result from the query and obtain the values from the underlying cache only when they are asked for. This feature may be useful for creating a Continuous Query Cache that represents a very large query result set or if the values are never or rarely requested. To specify that only the keys should be cached, pass false when creating the `ContinuousQueryCache`; for example:

```
ContinuousQueryCache::Handle hCacheOpenTrades  =
        ContinuousQueryCache::create(hCache, hFilter, false);
```

If necessary, the `CacheValues` property can be modified after the cache has been instantiated; for example:

```
hCacheOpenTrades->setCacheValues(true);
```

CacheValues Property and Event Listeners

If the Continuous Query Cache has any standard (non-lite) event listeners, or if any of the event listeners are filtered, then the CacheValues property will automatically be set to true. This is because the Continuous Query Cache uses the locally cached values to filter events and to supply the old and new values for the events that it raises.

Using ReflectionExtractor with Continuous Query Caches

When the Continuous Query Cache is configured to cache values, the use of the `ReflectionExtractor` is not supported. This is because the `ReflectionExtractor` does not support reflection in C++. In this case, you must provide a custom extractor. When the Continuous Query Cache is not caching values locally, the `ReflectionExtractor` can be used since it does not perform the

extraction on the client but instead passes the necessary extraction information to the cluster to perform the query.

Listening to the Continuous Query Cache

Since the Continuous Query Cache is itself observable, it is possible for the client to place one or more event listeners onto it. For example:

Example 9–2 Placing a Listener into a Continuous Query Cache

```
ContinuousQueryCache::Handle hCacheOpenTrades  =
ContinuousQueryCache::create(hCache, hFilter);
hCacheOpenTrades->addFilterListener(hListener);
```

If your application has to perform some processing against every item that is already in the cache **and** every item added to the cache, then provide the listener during construction. The resulting cache will receive one event for each item that is in the Continuous Query Cache, whether it was there to begin with (because it was in the query) or if it got added during or after the construction of the cache. One form of the factory create method of `ContinuousQueryCache` enables you to specify a cache, a filter, and a listener:

Example 9–3 Creating a Continuous Query Cache with a Filter and a Listener

```
ContinuousQueryCache::Handle hCacheOpenTrades  = ContinuousQueryCache::create(
        hRemoteCache, hFilter, true, hListener);
```

Avoiding Unexpected Results

There are two alternate approaches to processing the items in the Continuous Query Cache, both of which could yield unexpected and unwanted results. First, if you perform the processing and then add the listener to handle any later additions, then events that occur in the split second after the iteration and before the listener is added will be missed! This is illustrated in Example 9–4:

Example 9–4 Processing the Data, then Adding the Listener

```
ContinuousQueryCache::Handle hCacheOpenTrades  =
ContinuousQueryCache::create(hCache, hFilter);

for (Iterator::Handle hIter = hCacheOpenTrades->entrySet()->iterator();
hIter->hasNext(); )
    {
    Map::Entry::View vEntry = cast<Map::Entry::View>(hIter->next());
    // .. process the cache entry
    }
hCacheOpenTrades->addFilterListener(hListener);
```

The second approach is to add a listener first, so that no events are missed, and then do the processing. In this case, it is possible that the same entry will show up in both an event and in the Iterator. The events can be asynchronous, so the sequence of operations cannot be guaranteed.

Example 9–5 Adding the Listener, then Processing the Data

```
ContinuousQueryCache::Handle hCacheOpenTrades  =
        ContinuousQueryCache::create(hRemoteCache, hFilter);
```

```
hCacheOpenTrades->addFilterListener(hListener);
for (Iterator::Handle hIter = hCacheOpenTrades->entrySet()->iterator();
hIter->hasNext(); )
    {
    Map::Entry::View vEntry = cast<Map::Entry::View>(hIter->next());
    // .. process the cache entry
    }
```

Achieving a Stable Materialized View

The Continuous Query Cache implementation faced the same challenge: How to assemble an exact point-in-time snapshot of an underlying cache *while receiving a stream of modification events from that same cache*. The solution has several parts. First, Coherence supports an option for synchronous events, which provides a set of ordering guarantees. Secondly, the Continuous Query Cache has a two-phase implementation of its initial population that allows it to first query the underlying cache and then subsequently resolve all of the events that came in during the first phase. Since achieving these guarantees of data visibility without any missing or repeated events is fairly complex, the `ContinuousQueryCache` allows a developer to pass a listener during construction, thus avoiding exposing these same complexities to the application developer.

Support for Synchronous and Asynchronous Listeners

By default, listeners to the Continuous Query Cache will have their events delivered asynchronously. However, the `ContinuousQueryCache` implementation does respect the option for synchronous events as provided by the `SynchronousListener` interface.

Making the Continuous Query Cache Read-Only

The Continuous Query Cache can be made into a read-only cache by using the boolean `setReadOnly` method on the `ContinuousQueryCache` class; for example:

```
hCacheOpenTrades->setReadOnly(true);
```

A read-only Continuous Query Cache will not allow objects to be added to, changed in, removed from or locked in the cache.

Once a Continuous Query Cache has been set to read-only, it cannot be changed back to read/write.

10

Query the Cache for C++ Clients

Coherence can perform queries and indexes against currently cached data that meets a given set of criteria. Queries and indexes can be simple, employing filters packaged with Coherence, or they can be run against multi-value attributes such as collections and arrays.

Query Functionality

Coherence provides the ability to search for cache entries that meet a given set of criteria. The result set may be sorted if desired. Queries are evaluated with Read Committed isolation.

It should be noted that queries apply only to currently cached data (and will not use the `CacheLoader` interface to retrieve additional data that may satisfy the query). Thus, the dataset should be loaded entirely into cache before queries are performed. In cases where the dataset is too large to fit into available memory, it may be possible to restrict the cache contents along a specific dimension (for example, "date") and manually switch between cache queries and database queries based on the structure of the query. For maintainability, this is usually best implemented inside a cache-aware data access object (DAO).

Indexing requires the ability to extract attributes on each Partitioned cache node; in the case of dedicated `CacheServer` instances, this implies (usually) that application classes must be installed in the `CacheServer` classpath.

For Local and Replicated caches, queries are evaluated locally against unindexed data. For Partitioned caches, queries are performed in parallel across the cluster, using indexes if available. Coherence includes a Cost-Based Optimizer (CBO). Access to unindexed attributes requires object deserialization (though indexing on other attributes can reduce the number of objects that must be evaluated).

Simple Queries

Querying cache content is very simple, as Example 10–1 illustrates:

Example 10–1 Querying Cache Content

```
ValueExtractor::Handle hExtractor = ReflectionExtractor::create("getAge");
Filter::View vFilter = GreaterEqualsFilter::create(hExtractor,
Integer32::valueOf(18));

for (Iterator::Handle hIter = hCache->entrySet(vFilter)->iterator();
hIter->hasNext(); )
    {
    Map::Entry::Handle hEntry  = cast<Map::Entry::Handle>(hIter->next());
```

```
    Integer32::View    vKey    = cast<Integer32::View>(hEntry->getKey());
    Person::Handle     hPerson = cast<Person::Handle>(hEntry->getValue());
    std::cout << "key=" << vKey << " person=" << hPerson;
    }
```

Coherence provides a wide range of filters in the `coherence::util::Filter` package. A `LimitFilter` may be used to limit the amount of data sent to the client, and also to provide "paging" for users:

Example 10–2 Using the LimitFilter Method

```
int32_t                nPageSize  = 25;
ValueExtractor::Handle hExtractor = ReflectionExtractor::create("getAge");
Filter::View           vFilter    = GreaterEqualsFilter::create(hExtractor,
Integer32::valueOf(18));

// get entries 1-25
LimitFilter::Handle    hLimitFilter = LimitFilter::create(vFilter, nPageSize);
Set::View              vEntries     = hCache->entrySet(hLimitFilter);

// get entries 26-50
hLimitFilter->nextPage();
vEntries = hCache->entrySet(hLimitFilter);
```

Any queryable attribute may be indexed with the `addIndex` method of the `QueryMap` class:

Example 10–3 Indexing a Queryable Attribute

```
// addIndex(ValueExtractor::View vExtractor, boolean_t fOrdered, Comparator::View
vComparator)
hCache->addIndex(hExtractor, true, NULL);
```

The `fOrdered` argument specifies whether the index structure is sorted. Sorted indexes are useful for range queries, including "select all entries that fall between two dates" and "select all employees whose family name begins with 'S'". For "equality" queries, an unordered index may be used, which may have better efficiency in terms of space and time.

The comparator argument can be used to provide a `custom java.util.Comparator` for ordering the index.

Note: This method is only intended as a hint to the cache implementation, and as such it may be ignored by the cache if indexes are not supported or if the desired index (or a similar index) already exists. It is expected that an application will call this method to suggest an index even if the index may already exist, just so that the application is certain that index has been suggested. For example, in a distributed environment each server will likely suggest the same set of indexes when it starts, and there is no downside to the application blindly requesting those indexes regardless of whether another server has already requested the same indexes.

Indexes are a feature of Coherence Enterprise Edition or higher. This method will have no effect when using Coherence Standard Edition.

Note that queries can be combined by Coherence if necessary, and also that Coherence includes a cost-based optimizer (CBO) to prioritize the usage of indexes. To take

advantage of an index, queries must use extractors that are equal (`(Object->equals())`) to the one used in the query.

Querying Partitioned Caches

When using the Coherence Enterprise Edition or Grid Edition, the Partitioned Cache implements the QueryMap interface using the Parallel Query feature. When using Coherence Standard Edition, the Parallel Query feature is not available, resulting in lower performance for most queries, particularly when querying large data sets.

Querying Near Caches

Although queries can be executed through a near cache, the query will not use the front portion of a near cache. If using a near cache with queries, the best approach is to use the following sequence:

```
Set::View vSetKeys   = hCache->keySet(vFilter);
Map::View vMapResult = hCache->getAll(vSetKeys);
```

Query Concepts

This section goes into more detail on the design of the query interface, building up from the core components.

The concept of querying is based on the ValueExtractor interface. A value extractor is used to extract an attribute from a given object for querying (and similarly, indexing). Most developers will need only the ReflectionExtractor implementation of this interface. The ReflectionExtractor uses reflection to extract an attribute from a value object by referring to a method name, typically a "getter" method like getName().

```
ReflectionExtractor::Handle hExtractor = ReflectionExtractor::create("getName");
```

Any "void argument" method can be used, including `Object` methods like `toString()` (useful for prototyping/debugging). Indexes may be either traditional "field indexes" (indexing fields of objects) or "functional indexes" (indexing "virtual" object attributes). For example, if a class has field accessors `getFirstName` and `getLastName`, the class may define a function `getFullName` which concatenates those names, and this function may be indexed.

To query a cache that contains objects with `getName` attributes, a `Filter` must be used. A filter has a single method which determines whether a given object meets a criterion.

```
Filter::Handle hEqualsFilter = EqualsFilter::create(hExtractor,
String::create("Bob Smith"));
```

To select the entries of a cache that satisfy a particular filter:

Example 10–4 Selecting Entries of a Cache that Satisfy a Particular Filter

```
for (Iterator::Handle hIter = hCache->entrySet(hEqualsFilter)->iterator();
hIter->hasNext(); )
    {
    Map::Entry::Handle hEntry  = cast<Map::Entry::Handle>(hIter->next());
    Integer32::View    vKey    = cast<Integer32::View>(hEntry->getKey());
    Person::Handle     hPerson = cast<Person::Handle>(hEntry->getValue());
    std::cout << "key=" << vKey << " person=" << hPerson;
    }
```

To select and also sort the entries:

Example 10–5 Selecting and Sorting Entries

```
// entrySet(Filter::View vFilter, Comparator::View vComparator)
Iterator::Handle hIter = hCache->entrySet(hEqualsFilter, NULL)->iterator();
```

The additional NULL argument specifies that the result set should be sorted using the "natural ordering" of Comparable objects within the cache. The client may explicitly specify the ordering of the result set by providing an implementation of Comparator. Note that sorting places significant restrictions on the optimizations that Coherence can apply, as sorting requires that the entire result set be available before sorting.

Using the `keySet` form of the queries—combined with `getAll()`—may provide more control over memory usage:

Example 10–6 Using the keySet Form of a Query

```
// keySet(Filter::View vFilter)
Set::View   vSetKeys     = hCache->keySet(vFilter);
Set::Handle hSetPageKeys = HashSet::create();
int32_t     PAGE_SIZE    = 100;
for (Iterator::Handle hIter = vSetKeys->iterator(); hIter->hasNext();)
    {
    hSetPageKeys->add(hIter->next());
    if (hSetPageKeys->size() == PAGE_SIZE || !hIter->hasNext())
        {
        // get a block of values
        Map::View vMapResult = hCache->getAll(hSetPageKeys);

        // process the block
        // ...

        hSetPageKeys->clear();
        }
    }
```

Queries Involving Multi-Value Attributes

Coherence supports indexing and querying of multi-value attributes including collections and arrays. When an object is indexed, Coherence will check to see if it is a multi-value type, and will then index it as a collection rather than a singleton. The `ContainsAllFilter`, `ContainsAnyFilter`, and `ContainsFilter` are used to query against these collections.

Example 10–7 Indexing and Querying Multi-Value Attributes

```
Set::Handle hSearchTerms = HashSet::create();
hSearchTerms->add(String::create("java"));
hSearchTerms->add(String::create("clustering"));
hSearchTerms->add(String::create("books"));

// The cache contains instances of a class "Document" which has a method
// "getWords" which returns a Collection<String> containing the set of
// words that appear in the document.
ValueExtractor::Handle hExtractor = ReflectionExtractor::create("getWords");
Filter::View           vFilter    = ContainsAllFilter::create(hExtractor,
hSearchTerms);
```

```
Set::View vEntrySet = hCache->entrySet(vFilter);

// iterate through the search results
// ...
```

ChainedExtractor

The `ChainedExtractor` implementation allows chained invocation of zero-argument (accessor) methods. In Example 10–8, the extractor will first use reflection to call `getName()` on each cached `Person` object, and then use reflection to call `length()` on the returned `String`. This extractor could be passed into a query, allowing queries (for example) to select all people with names not exceeding 10 letters.

Example 10–8 Using a ChainedExtractor Implementation

```
ChainedExtractor::Handle hExtractor =

ChainedExtractor::create(ChainedExtractor::createExtractors("getName.length"));
```

Method invocations may be chained indefinitely, for example: `getName.trim.length`.

11

Remote Invocation Service for C++ Clients

An `Invocable` can execute any arbitrary action and can use any cluster-side services (cache services, grid services, and so on) necessary to perform their work. The Invocable operations can also be stateful, which means that their state is serialized and transmitted to the grid nodes on which the Invocable is run.

Coherence for C++ provides a **Remote Invocation Service** which allows the execution of `Invocables` within the cluster-side JVM to which the client is connected. In Java, Invocables are simply runnable application classes that implement the `com.tangosol.net.Invocable` interface. To employ an `Invocable` in Coherence for C++, you must deploy a compiled Java implementation of the `Invocable` task on the cluster-side node, in addition to providing a C++ implementation of `Invocable`: `coherence::net::Invocable`. Since execution is server-side (that is, Java), the C++ invocable only must be concerned with state; the methods themselves can be no-ops.

Configuring and Using the Remote Invocation Service

A Remote Invocation Service is configured using the `remote-invocation-scheme` element in the cache configuration descriptor. Example 11–1 illustrates a sample remote invocation scheme configuration.

Example 11–1 Sample Remote Invocation Scheme Configuration

```
<remote-invocation-scheme>
    <scheme-name>example-invocation</scheme-name>
    <service-name>ExtendTcpInvocationService</service-name>
    <initiator-config>
      <tcp-initiator>
        <remote-addresses>
          <socket-address>
            <address>localhost</address>
            <port>9099</port>
          </socket-address>
        </remote-addresses>
      </tcp-initiator>

      <outgoing-message-handler>
        <request-timeout>30s</request-timeout>
      </outgoing-message-handler>
    </initiator-config>
</remote-invocation-scheme>
```

A reference to a configured Remote Invocation Service can then be obtained by name by using the `coherence::net::CacheFactory` class:

Example 11–2 Reference to a Remote Invocation Service

```
InvocationService::Handle hService =
hService::getService("ExtendTcpInvocationService");
```

To execute an agent on the grid node to which the client is connected requires **only one line of code**:

```
Map::View hResult = hService->query(myTask::create(), NULL);
```

The `Map` returned from query is keyed by the member on which the query is run. For Extend clients, there is no concept of membership, so the result is keyed by the local member which can be retrieved by calling `CacheFactory::getConfigurableCacheFactory()::GetLocalMember()`

Registering Invocable Implementation Classes

Like cached value objects, all `Invocable` implementation classes must be correctly registered in the POF context of the C++ application (see the `PortableObject` description in *"Building Integration Objects for C++ Clients"*) and cluster-side node to which the client is connected. As such, a Java implementation of the `Invocable` task (a `com.tangosol.net.Invocable` implementation) must be created, compiled, and deployed on the cluster-side node.

See *"POF Registration"* for additional details.

12

Deliver Events for Changes as they Occur (C++)

Coherence provides cache events. It is extremely simple to receive the events that you need, where you need them, regardless of where the changes are actually occurring in the cluster.

Listener Interface and Event Object

In the event model, there is an EventListener interface that all listeners must extend. Coherence provides a `MapListener` interface, which allows application logic to receive events when data in a Coherence cache is added, modified or removed. Example 12–1 illustrates a segment of the `MapListener` API.

Example 12–1 Excerpt from the coherence::util::MapListener Class File

```
class MapListener
    : public interface_spec<MapListener,
        implements<EventListener> >
    {
    // ----- handle definitions ---------------------------------------------

    public:
        /**
        * Handle definition.
        */
        typedef TypedHandle<MapListener> Handle;

        /**
        * View definition.
        */
        typedef TypedHandle<const MapListener> View;

        /**
        * MapEvent View definition.
        */
        typedef TypedHandle<const MapEvent> MapEventView;


    // ----- MapListener interface ------------------------------------------

    public:
        /**
        * Invoked when a map entry has been inserted.
        *
```

```
        * @param vEvent  the MapEvent carrying the insert information
        */
        virtual void entryInserted(MapEventView vEvent) = 0;

        /**
        * Invoked when a map entry has been updated.
        *
        * @param vEvent  the MapEvent carrying the update information
        */
        virtual void entryUpdated(MapEventView vEvent) = 0;

        /**
        * Invoked when a map entry has been removed.
        *
        * @param vEvent  the MapEvent carrying the delete information
        */
        virtual void entryDeleted(MapEventView vEvent) = 0;
    };
```

An application object that implements the `MapListener` interface can sign up for events from any Coherence cache or class that implements the `ObservableMap` interface, simply by passing an instance of the application's `MapListener` implementation to one of the `addMapListener()` methods.

The `MapEvent` object that is passed to the `MapListener` carries all of the necessary information about the event that has occurred, including the *source* (`ObservableMap`) that raised the event, the *identity* (key) that the event is related to, what the *action* was against that identity (insert, update or delete), what the old value was and what the new value is. Example 12–2 illustrates a segment of the `MapEvent` API.

Example 12–2 Excerpt from coherence::util::MapEvent

```
class MapEvent
     : public class_spec<MapEvent,
         extends<EventObject> >
     {
     friend class factory<MapEvent>;


     // ----- MapEvent interface -------------------------------------------

     public:
         /**
         * Return an ObservableMap object on which this event has actually
         * occurred.
         *
         * @return an ObservableMap object
         */
         virtual ObservableMap::Handle getMap() const;

         /**
         * Return this event's id. The event id is one of the ENTRY_*
         * enumerated constants.
         *
         * @return an id
         */
         virtual int32_t getId() const;

         /**
         * Return a key associated with this event.
```

```
    *
    * @return a key
    */
    virtual Object::View getKey() const;

    /**
    * Return an old value associated with this event.
    * <p>
    * The old value represents a value deleted from or updated in a map.
    * It is always NULL for "insert" notifications.
    *
    * @return an old value
    */
    virtual Object::View getOldValue() const;

    /**
    * Return a new value associated with this event.
    * <p>
    * The new value represents a new value inserted into or updated in
    * a map. It is always NULL for "delete" notifications.
    *
    * @return a new value
    */
    virtual Object::View getNewValue() const;


// ----- Objectinterface ---------------------------------------------

public:
    /**
    * {@inheritDoc}
    */
    virtual void toStream(std::ostream& out) const;


// ----- helper methods ----------------------------------------------

public:
    /**
    * Dispatch this event to the specified listeners collection.
    * <p>
    * This call is equivalent to
    * <pre>
    *   dispatch(listeners, true);
    * </pre>
    *
    * @param vListeners the listeners collection
    *
    * @throws ClassCastException if any of the targets is not
    *         an instance of MapListener interface
    */
    virtual void dispatch(Listeners::View vListeners) const;

    /**
    * Dispatch this event to the specified listeners collection.
    *
    * @param vListeners the listeners collection
    * @param fStrict    if true then any RuntimeException thrown by event
    *                   handlers stops all further event processing and
    *                   the exception is re-thrown; if false then all
```

```
    *                  exceptions are logged and the process continues
    *
    * @throws ClassCastException if any of the targets is not
    *         an instance of MapListener interface
    */
    virtual void dispatch(Listeners::View vListeners,
            bool fStrict) const;

    /**
    * Dispatch this event to the specified MapListener.
    *
    * @param hListener  the listener
    */
    virtual void dispatch(MapListener::Handle hListener) const;

    /**
    * Get the event's description.
    *
    * @return this event's description
    */
    virtual String::View getDescription() const;

    /**
    * Convert an event ID into a human-readable string.
    *
    * @param nId  an event ID, one of the ENTRY_* enumerated values
    *
    * @return a corresponding human-readable string, for example
    *         "inserted"
    */
    static String::View getDescription(int32_t nId);


// ----- constants ----------------------------------------------------

public:
    /**
    * This event indicates that an entry has been added to the map.
    */
    static const int32_t ENTRY_INSERTED = 1;

    /**
    * This event indicates that an entry has been updated in the map.
    */
    static const int32_t ENTRY_UPDATED  = 2;

    /**
    * This event indicates that an entry has been removed from the map.
    */
    static const int32_t ENTRY_DELETED  = 3;
};
```

Caches and Classes that Support Events

All Coherence caches implement `ObservableMap`; in fact, the NamedCache interface that is implemented by all Coherence caches extends the `ObservableMap` interface. That means that an application can sign up to receive events from any cache, regardless of whether that cache is local, partitioned, near, replicated, using read-through, write-through, write-behind, overflow, disk storage, and so on.

Note: Regardless of the cache topology and the number of servers, and even if the modifications are being made by other servers, the events will be delivered to the application's listeners.

In addition to the Coherence caches (those objects obtained through a Coherence cache factory), several other supporting classes in Coherence also implement the `ObservableMap` interface:

- `ObservableHashMap`
- `LocalCache`
- `OverflowMap`
- `NearCache`
- `ReadWriteBackingMap`
- `AbstractSerializationCache`, `SerializationCache`, and `SerializationPagedCache`
- `WrapperObservableMap`, `WrapperConcurrentMap`, and `WrapperNamedCache`

For a full list of published implementing classes, see the Coherence API for ObservableMap.

Signing Up for all Events

To sign up for events, simply pass an object that implements the `MapListener` interface to one of the `addMapListener` methods on `ObservableMap`:

Example 12–3 ObservableMap methods

```
virtual void addKeyListener(MapListener::Handle hListener, Object::View vKey, bool
fLite) = 0;
virtual void removeKeyListener(MapListener::Handle hListener, Object::View vKey) =
0;
virtual void addFilterListener(MapListener::Handle hListener, Filter::View vFilter
= NULL, bool fLite = false) = 0;
virtual void removeFilterListener(MapListener::Handle hListener, Filter::View
vFilter = NULL) = 0;
```

Let's create an example `MapListener` implementation:

Example 12–4 Example MapListener implementation

```
#include "coherence/util/MapEvent.hpp"
#include "coherence/util/MapListener.hpp"

#include <iostream>

using coherence::util::MapEvent;
using coherence::util::MapListener;
using namespace std;

/**
* A MapListener implementation that prints each event as it receives
* them.
*/
```

```
class EventPrinter
    : public class_spec<EventPrinter,
        extends<Object>,
        implements<MapListener> >
    {
    friend class factory<EventPrinter>;

    public:
        virtual void entryInserted(MapEventView vEvent)
            {
            cout << vEvent << endl;
            }

        virtual void entryUpdated(MapEventView vEvent)
            {
            cout << vEvent << endl;
            }

        virtual void entryDeleted(MapEventView vEvent)
            {
            cout << vEvent << endl;
            }
    };
```

Using this implementation, it is extremely simple to print out all events from any given cache (since all caches implement the `ObservableMap` interface):

Example 12–5 Printing Events

```
NamedCache::Handle hCache;
...
hCache->addFilterListener(EventPrinter::create());
```

Of course, to be able to later remove the listener, it is necessary to hold on to a reference to the listener:

Example 12–6 Holding a Reference to a Listener

```
MapListener::Handle hListener = EventPrinter::create();
hCache->addFilterListener(hListener);
m_hListener = hListener; // store the listener in a member field
```

Later, to remove the listener:

Example 12–7 Removing a Reference to a Listener

```
MapListener::Handle hListener = m_hListener;
if (hListener != NULL)
    {
    hCache->removeFilterListener(hListener);
    m_hListener = NULL; // clean up the listener field
    }
```

Each `add*Listener` method on the `ObservableMap` interface has a corresponding `remove*Listener` method. To remove a listener, use the `remove*Listener` method that corresponds to the `add*Listener` method that was used to add the listener.

MultiplexingMapListener

Another helpful base class for creating a `MapListener` is the `MultiplexingMapListener`, which routes all events to a single method for handling. Example 12–8 illustrates a simplified version of the `EventPrinter` example:

Example 12–8 Using MultiplexingMapListener to Route Events

```
#include "coherence/util/MultiplexingMapListener.hpp"

#include <iostream>

using coherence::util::MultiplexingMapListener;


class EventPrinter
    : public class_spec<EventPrinter,
        extends<MultiplexingMapListener> >
    {
    public:
        virtual void onMapEvent(MapEventView vEvent)
            {
            std::cout << vEvent << std::endl;
            }
    };
```

Configuring a MapListener for a Cache

If the listener should always be on a particular cache, then place it into the cache configuration using the `<listener>` element and Coherence will automatically add the listener when it configures the cache.

Signing Up for Events on Specific Identities

Signing up for events that occur against specific identities (keys) is just as simple. The C++ code in Example 12–9 prints all events that occur against the `Integer` key `5`:

Example 12–9 Printing Events that Occur Against a Specified Integer Key

```
hCache->addKeyListener(EventPrinter::create(), Integer32::create(5), false);
```

The code in Example 12–10 would only trigger an event when the `Integer` key `5` is inserted or updated:

Example 12–10 Triggering an Event for a Specified Integer Key Value

```
for (int32_t i = 0; i < 10; ++i)
    {
    Integer32::View vKey   = Integer32::create(i);
    Integer32::View vValue = vKey;
    hCache->put(vKey, vValue);
    }
```

Filtering Events

Similar to listening to a particular key, it is possible to listen to particular events. In Example 12–11, a listener is added to the cache with a filter that allows the listener to only receive delete events.

Example 12–11 Adding a Listener with a Filter that Allows only Deleted Events

```
// Filters used with partitioned caches must implement
coherence::io::pof::PortableObject

#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PortableObject.hpp"
#include "coherence/util/Filter.hpp"
#include "coherence/util/MapEvent.hpp"

using coherence::io::pof::PofReader;
using coherence::io::pof::PofWriter;
using coherence::io::pof::PortableObject;
using coherence::util::Filter;
using coherence::util::MapEvent;

class DeletedFilter
    : public class_spec<DeletedFilter,
        extends<Object>,
        implements<Filter, PortableObject> >
    {
    public:
        // Filter interface        virtual bool evaluate(Object::View v) const
            {
            MapEvent::View vEvt = cast<MapEvent::View>(v);
            return MapEvent::ENTRY_DELETED == vEvt->getId();
            }

        // PortableObject interface        virtual void
readExternal(PofReader::Handle hIn)
            {
            }

        virtual void writeExternal(PofWriter::Handle hOut) const
            {
            }
    };

hCache->addFilterListener(EventPrinter::create(), DeletedFilter::create(), false);
```

For example, if the following sequence of calls were made:

Example 12–12 Inserting and Removing Data from the Cache

```
cache::put(String::create("hello"), String::create("world"));
cache::put(String::create("hello"), String::create("again"));
cache::remove(String::create("hello"));
```

The result would be:

```
CacheEvent{LocalCache deleted: key=hello, value=again}
```

For more information, see "Advanced: Listening to Queries" on page 12-10.

Filtering Events Versus Filtering Cached Data

When building a `Filter` for querying, the object that will be passed to the evaluate method of the `Filter` will be a value from the cache, or, if the `Filter` implements the `EntryFilter` interface, the entire `Map::Entry` from the cache. When building a `Filter` for filtering events for a `MapListener`, the object that will be passed to the `evaluate` method of the `Filter` will always be of type `MapEvent`.

For more information on how to use a query filter to listen to cache events, see *Advanced: Listening to Queries*.

"Lite" Events

By default, Coherence provides both the old and the new value as part of an event. Consider the following example:

Example 12–13 Inserting, Updating, and Removing a Value

```
MapListener::Handle hListener = EventPrinter::create();
// add listener with the default"lite" value of
falsehCache->addFilterListener(hListener);

// insert a 1KB value
String::View vKey = String::create("test");
hCache->put(vKey, Array<octet_t>::create(1024));

// update with a 2KB value
hCache->put(vKey, Array<octet_t>::create(2048));

// remove the value
hCache->remove(vKey);
```

When the above code is run, the insert event carries the new 1KB value, the update event carries both the old 1KB value and the new 2KB value and the remove event carries the removed 2KB value.

When an application does not require the old and the new value to be included in the event, it can indicate that by requesting only "lite" events. When adding a listener, you can request lite events by using either the `addFilterListener` or the `addKeyListener` method that takes an additional boolean `fLite` parameter. In the above example, the only change would be:

Example 12–14 Requesting Only "Lite" Events

```
cache->addFilterListener(hListener, (Filter::View) NULL, true);
```

Note: Obviously, a lite event's old value and new value may be `NULL`. However, even if you request lite events, the old and the new value *may* be included if there is no additional cost to generate and deliver the event. In other words, requesting that a `MapListener` receive lite events is simply a hint to the system that the `MapListener` does not need to know the old and new values for the event.

Advanced: Listening to Queries

All Coherence caches support querying by any criteria. When an application queries for data from a cache, the result is a point-in-time snapshot, either as a set of identities (`keySet`) or a set of identity/value pairs (`entrySet`). The mechanism for determining the contents of the resulting set is referred to as *filtering*, and it allows an application developer to construct queries of arbitrary complexity using a rich set of out-of-the-box filters (for example, equals, less-than, like, between, and so on), or to provide their own custom filters (for example, XPath).

The same filters that are used to query a cache can be used to listen to events from a cache. For example, in a trading system it is possible to query for all open `Order` objects for a particular trader.

Note: Executing Queries in the Cluster: Example 12–15 uses the `coherence::util::extractor::ReflectionExtractor` class. While the C++ client doesn't support reflection, the `ReflectionExtractor` can be used for queries which are executed in the cluster. In this case, the `ReflectionExtractor` simply passes the necessary extraction information to the cluster to perform the query. In cases where the `ReflectionExtractor` would extract the data on the client, such as the `ContinuousQueryCache` when caching values locally, the use of the `ReflectionExtractor` is not supported. For these cases, you must provide a custom extractor.

Example 12–15 Filtering for Cache Events

```
NamedCache::Handle hMapTrades = ...
Filter::Handle hFilter = AndFilter::create(
        EqualsFilter::create(ReflectionExtractor::create("getTrader"), vTraderId),
        EqualsFilter::create(ReflectionExtractor::create("getStatus"),
Status::OPEN));
Set::View vSetOpenTrades = hMapTrades->entrySet(hFilter);
```

To receive notifications of new trades being opened for that trader, closed by that trader or reassigned to or from another trader, the application can use the same filter:

Example 12–16 Filtering for Specialized Events

```
// receive events for all trade IDs that this trader is interested in
hMapTrades->addFilterListener(hListener, MapEventFilter::create(hFilter), true);
```

The `MapEventFilter` converts a query filter into an event filter.

Note: Filtering events versus filtering cached data: When building a `Filter` for querying, the object that will be passed to the `evaluate` method of the Filter will be a value from the cache, or, if the `Filter` implements the `EntryFilter` interface, the entire `Map::Entry` from the cache. When building a `Filter` for filtering events for a `MapListener`, the object that will be passed to the `evaluate` method of the `Filter` will always be of type `MapEvent`.

The `MapEventFilter` converts a `Filter` that is used to do a query into a `Filter` that is used to filter events for a `MapListener`. In other words, the `MapEventFilter` is constructed from a `Filter` that queries a cache, and the resulting `MapEventFilter` is a filter that evaluates `MapEvent` objects by converting them into the objects that a query `Filter` would expect.

The `MapEventFilter` has several very powerful options, allowing an application listener to receive only the events that it is specifically interested in. More importantly for scalability and performance, only the desired events have to be communicated over the network, and they are communicated only to the servers and clients that have expressed interest in those specific events. For example:

Example 12–17 Communicating Only Specialized Events over the Network

```
// receive all events for all trades that this trader is interested in
int32_t nMask = MapEventFilter::E_ALL;
hMapTrades->addFilterListener(hListener, MapEventFilter::create(nMask, hFilter),
true);

// receive events for all this trader's trades that are closed or
// re-assigned to a different trader
nMask = MapEventFilter::E_UPDATED_LEFT | MapEventFilter::E_DELETED;
hMapTrades->addFilterListener(hListener, MapEventFilter::create(nMask, hFilter),
true);

// receive events for all trades as they are assigned to this trader
nMask = MapEventFilter::E_INSERTED | MapEventFilter::E_UPDATED_ENTERED;
hMapTrades->addFilterListener(hListener, MapEventFilter::create(nMask, hFilter),
true);

// receive events only for new trades assigned to this trader
nMask = MapEventFilter::E_INSERTED;
hMapTrades->addFilterListener(hListener, MapEventFilter::create(nMask, hFilter),
true);
```

For more information on the various options supported, see the API documentation for `MapEventFilter`.

Advanced: Synthetic Events

Events usually reflect the changes being made to a cache. For example, one server is modifying one entry in a cache while another server is adding several items to a cache while a third server is removing an item from the same cache, all while fifty threads on each and every server in the cluster is accessing data from the same cache! All the modifying actions will produce events that any server within the cluster can choose to receive. We refer to these actions as *client actions*, and the events as being *dispatched to clients*, even though the "clients" in this case are actually servers. This is a natural

concept in a true peer-to-peer architecture, such as a Coherence cluster: Each and every peer is both a client and a server, both consuming services from its peers and providing services to its peers. In a typical Java Enterprise application, a "peer" is an application server instance that is acting as a container for the application, and the "client" is that part of the application that is directly accessing and modifying the caches and listening to events from the caches.

Some events originate from within a cache itself. There are many examples, but the most common cases are:

- When entries automatically expire from a cache;
- When entries are evicted from a cache because the maximum size of the cache has been reached;
- When entries are transparently added to a cache as the result of a Read-Through operation;
- When entries in a cache are transparently updated as the result of a Read-Ahead or Refresh-Ahead operation.

Each of these represents a modification, but the modifications represent natural (and typically automatic) operations from within a cache. These events are referred to as *synthetic* events.

When necessary, an application can differentiate between client-induced and synthetic events simply by asking the event if it is synthetic. This information is carried on a sub-class of the `MapEvent`, called `CacheEvent`. Using the previous `EventPrinter` example, it is possible to print only the synthetic events:

Example 12–18 Differentiating Between Client-Induced and Synthetic Events

```
class EventPrinter
    : public class_spec<EventPrinter,
        extends<MultiplexingMapListener> >
    {
    friend class factory<EventPrinter>;

    public:
        void onMapEvent(MapEvent::View vEvt)
            {
            if (instanceof<CacheEvent::View>(vEvt) &&
                (cast<CacheEvent::View>(vEvt)->isSynthetic()))
                {
                std::cout << vEvt;
                }
            }
    };
```

For more information on this feature, see the API documentation for `CacheEvent`.

Advanced: Backing Map Events

While it is possible to listen to events from Coherence caches, each of which presents a local view of distributed, partitioned, replicated, near-cached, continuously-queried, read-through/write-through and/or write-behind data, it is also possible to peek behind the curtains, so to speak.

For some advanced use cases, it may be necessary to peek behind the curtain—or more correctly, to "listen to" the "map" behind the "service". Replication, partitioning and other approaches to managing data in a distributed environment are all distribution

services. The service still has to have something in which to actually *manage* the data, and that *something* is called a "backing map".

Backing maps are configurable. If all the data for a particular cache should be kept in object form on the heap, then use an unlimited and non-expiring `LocalCache` (or a `SafeHashMap` if statistics are not required). If only a small number of items should be kept in memory, use a `LocalCache`. If data are to be read on demand from a database, then use a `ReadWriteBackingMap` (which knows how to read and write through an application's DAO implementation), and in turn give the `ReadWriteBackingMap` a backing map such as a `SafeHashMap` or a `LocalCache` to store its data in.

Some backing maps are observable. The events coming from these backing maps are not usually of direct interest to the application. Instead, Coherence translates them into actions that must be taken (by Coherence) to keep data synchronized and properly backed up, and it also translates them when appropriate into clustered events that are delivered throughout the cluster as requested by application listeners. For example, if a partitioned cache has a `LocalCache` as its backing map, and the local cache expires an entry, that event causes Coherence to expire all of the backup copies of that entry. Furthermore, if any listeners have been registered on the partitioned cache, and if the event matches their event filter(s), then that event will be delivered to those listeners on the servers where those listeners were registered.

In some advanced use cases, an application must process events on the server where the data are being maintained, and it must do so on the structure (backing map) that is actually managing the data. In these cases, if the backing map is an observable map, a listener can be configured on the backing map or one can be programmatically added to the backing map. (If the backing map is not observable, it can be made observable by wrapping it in an `WrapperObservableMap`.)

For more information on this feature, see the API documentation for `BackingMapManager`.

Advanced: Synchronous Event Listeners

Some events are delivered asynchronously, so that application listeners do not disrupt the cache services that are generating the events. In some rare scenarios, asynchronous delivery can cause ambiguity of the ordering of events compared to the results of ongoing operations. To guarantee that the cache API operations and the events are ordered as if the local view of the clustered system were single-threaded, a `MapListener` must implement the `SynchronousListener` marker interface.

One example in Coherence itself that uses synchronous listeners is the Near Cache, which can use events to invalidate locally cached data ("Seppuku").

For more information on this feature, see the API documentation for `SynchronousListener`.

Summary

Coherence provides an extremely rich event model for caches, providing the means for an application to request the specific events it requires, and the means to have those events delivered only to those parts of the application that require them.

Part II

Coherence for .NET

Coherence for .NET allows .NET applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses of Coherence for .NET include desktop and web applications that require access to Coherence caches.

Coherence for .NET consists of a lightweight .NET library that connects to a Coherence*Extend clustered service instance running within the Coherence cluster using a high performance TCP/IP-based communication layer. This library sends all client requests to the Coherence*Extend clustered service which, in turn, responds to client requests by delegating to an actual Coherence clustered service (for example, a Partitioned or Replicated cache service).

An `INamedCache` instance is retrieved by using the `CacheFactory.GetCache(...)` API call. Once it is obtained, a client accesses the `INamedCache` in the same way as it would if it were part of the Coherence cluster. The fact that `INamedCache` operations are being sent to a remote cluster node (over TCP/IP) is completely transparent to the client application.

Coherence for .NET contains the following chapters:

13

Requirements, Installation and Deployment for Coherence for .NET

This chapter describes the requirements for installing and deploying Coherence on the .NET platform.

Package Requirements

The following are required to use Coherence for .NET:

- Microsoft.NET 1.1, 2.0, 3.0, or 3.5 Runtime
- Microsoft.NET 1.1, 2.0, 3.0, or 3.5 SDK
- Java Standard Edition 1.4.x SDK (or later)
- Coherence Data Grid Edition 3.4
- Supported Microsoft Windows operating system (see the system requirements for the appropriate .NET Runtime above)

In addition to the software listed above, the following is required to build and run the examples included with Coherence for .NET:

- Microsoft Visual Studio 2005

Installation

1. Download the Coherence for .NET Windows installer (typically named `coherence-net.msi`).
2. Run the installer by double clicking on the installer file.

Deployment

Coherence for .NET requires no specialized deployment configuration. Simply add a reference to the appropriate `Coherence.dll` to your Microsoft.NET application. If you are using .NET 1.1, use the library found in the `bin\net\1.1` folder. If you are using .NET 2.0 or higher, use the library found in the `bin\net\2.0` folder.

14

Configuration and Usage for .NET Clients

This chapter describes how to configure .NET clients for Coherence*Extend, the POF context, and the .NET client library.

General Instructions

Configuring and using Coherence for .NET requires five basic steps:

1. Configure Coherence*Extend on both the client and on one or more JVMs within the cluster.
2. Configure a POF context on the client and on all of the JVMs within the cluster that run the Coherence*Extend clustered service.
3. Implement the .NET client application using the Coherence for .NET API.
4. Make sure the Coherence cluster is up and running.
5. Launch the .NET client application.

The following sections describe each of these steps in detail.

Configuring Coherence*Extend

To configure Coherence*Extend, you need to add the appropriate configuration elements to both the cluster and client-side cache configuration descriptors. The cluster-side cache configuration elements instruct a Coherence `DefaultCacheServer` to start a Coherence*Extend clustered service that will listen for incoming TCP/IP requests from Coherence*Extend clients. The client-side cache configuration elements are used by the client library to determine the IP address and port of one or more servers in the cluster that run the Coherence*Extend clustered service so that it can connect to the cluster. It also contains various connection-related parameters, such as connection and request timeouts.

Configuring Coherence*Extend in the Cluster

In order for a Coherence*Extend client to connect to a Coherence cluster, one or more `DefaultCacheServer` JVMs within the cluster must run a TCP/IP Coherence*Extend clustered service. To configure a `DefaultCacheServer` to run this service, a `proxy-scheme` element with a child `tcp-acceptor` element must be added to the cache configuration descriptor used by the `DefaultCacheServer`. this is illustrated in Example 14–1.

Example 14–1 Configuration of a Default Cache Server for Coherence*Extend

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>dist-default</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>
      <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <thread-count>5</thread-count>
      <acceptor-config>
        <tcp-acceptor>
          <local-address>
            <address>localhost</address>
            <port>9099</port>
          </local-address>
        </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
    </proxy-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines two clustered services, one that allows remote Coherence*Extend clients to connect to the Coherence cluster over TCP/IP and a standard Partitioned cache service. Since this descriptor is used by a `DefaultCacheServer`, it is important that the `autostart` configuration element for each service is set to `true` so that clustered services are automatically restarted upon termination. The `proxy-scheme` element has a `tcp-acceptor` child element which includes all TCP/IP-specific information needed to accept client connection requests over TCP/IP.

The Coherence*Extend clustered service configured above will listen for incoming requests on the `localhost` address and `port 9099`. When, for example, a client attempts to connect to a Coherence cache called `dist-extend`, the Coherence*Extend clustered service will proxy subsequent requests to the `NamedCache` with the same name which, in this example, will be a Partitioned cache.

Configuring Coherence*Extend on the Client

A Coherence*Extend client uses the information within an `initiator-config` cache configuration descriptor element to connect to and communicate with a Coherence*Extend clustered service running within a Coherence cluster. This is illustrated in Example 14–2.

Example 14–2 Configuration to Connect to a Remote Coherence Cluster

```
<?xml version="1.0"?>

<cache-config xmlns="http://schemas.tangosol.com/cache">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-extend</cache-name>
      <scheme-name>extend-dist</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <initiator-config>
        <tcp-initiator>
          <remote-addresses>
            <socket-address>
              <address>localhost</address>
              <port>9099</port>
            </socket-address>
          </remote-addresses>
        </tcp-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines a caching scheme that connects to a remote Coherence cluster. The `remote-cache-scheme` element has a `tcp-initiator` child element which includes all TCP/IP-specific information needed to connect the client with the Coherence*Extend clustered service running within the remote Coherence cluster.

When the client application retrieves a named cache with `CacheFactory` using, for example, the name `dist-extend`, the Coherence*Extend client will connect to the Coherence cluster by using TCP/IP (using the address `localhost` and `port 9099`) and return a `INamedCache` implementation that routes requests to the `NamedCache` with the same name running within the remote cluster. Note that the `remote-addresses` configuration element can contain multiple `socket-address` child elements. The Coherence*Extend client will attempt to connect to the addresses in a random order, until either the list is exhausted or a TCP/IP connection is established.

Connection Error Detection and Failover

When a Coherence*Extend client service detects that the connection between the client and cluster has been severed (for example, due to a network, software, or hardware failure), the Coherence*Extend client service implementation (that is, `ICacheService` or `IInvocationService`) will raise a `MemberEventType.Left` event (by using the `MemberEventHandler` delegate) and the service will be stopped. If the client application attempts to subsequently use the service, the service will automatically restart itself and attempt to reconnect to the cluster. If the connection is successful, the

service will raise a `MemberEventType.Joined` event; otherwise, a fatal exception will be thrown to the client application.

A Coherence*Extend service has several mechanisms for detecting dropped connections. Some mechanisms are inherent to the underlying protocol (such as TCP/IP in Extend-TCP), whereas others are implemented by the service itself. The latter mechanisms are configured by using the `outgoing-message-handler` configuration element.

The primary configurable mechanism used by a Coherence*Extend client service to detect dropped connections is a request timeout. When the service sends a request to the remote cluster and does not receive a response within the request timeout interval (see `<request-timeout>`), the service assumes that the connection has been dropped. The Coherence*Extend client and clustered services can also be configured to send a periodic heartbeat over the connection (see `<heartbeat-interval>` and `<heartbeat-timeout>`). If the service does not receive a response within the configured heartbeat timeout interval, the service assumes that the connection has been dropped.

15

Building Integratable Objects for .NET Clients

Coherence caches are used to cache value objects. Enabling .NET clients to successfully communicate with a Coherence JVM requires a platform-independent serialization format that allows both .NET clients and Coherence JVMs (including Coherence*Extend Java clients) to properly serialize and deserialize value objects stored in Coherence caches. The Coherence for .NET client library and Coherence*Extend clustered service use a serialization format known as Portable Object Format (POF). 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.

Configuring a POF Context

POF supports all common .NET and Java types out-of-the-box. Any custom .NET and Java class can also be serialized to a POF stream; however, there are additional steps required to do so:

1. Create a .NET class that implements the `IPortableObject` interface. (See "Creating an IPortableObject Implementation (.NET)")
2. Create a matching Java class that implements the `PortableObject` interface in the same way. (See "Creating a PortableObject Implementation (Java)")
3. Register your custom .NET class on the client. (See "Registering Custom Types on the .NET Client")
4. Register your custom Java class on each of the servers running the Coherence*Extend clustered service. (See "Registering Custom Types in the Cluster")

Once these steps are complete, you can cache your custom .NET classes in a Coherence cache in the same way as a built-in data type. Additionally, you will be able to retrieve, manipulate, and store these types from a Coherence or Coherence*Extend JVM using the matching Java classes.

Creating an IPortableObject Implementation (.NET)

Each class that implements `IPortableObject` can self-serialize and deserialize its state to and from a POF data stream. This is achieved in the `ReadExternal` (deserialize) and `WriteExternal` (serialize) methods. Conceptually, all user types are composed of zero or more indexed values (properties) which are read from and written to a POF data stream one by one. The only requirement for a portable class, other than the need to implement the `IPortableObject` interface, is that it must

have a default constructor which will allow the POF deserializer to create an instance of the class during deserialization.

Example 15–1 illustrates a user-defined portable class:

Example 15–1 A User-Defined Portable Class

```
public class ContactInfo : IPortableObject
{
    private string name;
    private string street;
    private string city;
    private string state;
    private string zip;
    public ContactInfo()
    {}
    public ContactInfo(string name, string street, string city, string state,
string zip)
    {
        Name   = name;
        Street = street;
        City   = city;
        State  = state;
        Zip    = zip;
    }
    public void ReadExternal(IPofReader reader)
    {
        Name   = reader.ReadString(0);
        Street = reader.ReadString(1);
        City   = reader.ReadString(2);
        State  = reader.ReadString(3);
        Zip    = reader.ReadString(4);
    }
    public void WriteExternal(IPofWriter writer)
    {
        writer.WriteString(0, Name);
        writer.WriteString(1, Street);
        writer.WriteString(2, City);
        writer.WriteString(3, State);
        writer.WriteString(4, Zip);
    }
    // property definitions ommitted for brevity
}
```

Creating a PortableObject Implementation (Java)

An implementation of the portable class in Java is very similar to the one in .NET from the example above:

Example 15–2 illustrates the Java version of the .NET class in Example 15–1.

Example 15–2 A User-Defined Class in Java

```
public class ContactInfo implements PortableObject
    {
    private String m_sName;

    private String m_sStreet;
    private String m_sCity;
    private String m_sState;
```

```
    private String m_sZip;
    public ContactInfo()
        {
        }
    public ContactInfo(String sName, String sStreet, String sCity, String sState,
String sZip)
        {
        setName(sName);
        setStreet(sStreet);
        setCity(sCity);
        setState(sState);
        setZip(sZip);
        }
    public void readExternal(PofReader reader)
            throws IOException
        {
        setName(reader.readString(0));
        setStreet(reader.readString(1));
        setCity(reader.readString(2));
        setState(reader.readString(3));
        setZip(reader.readString(4));
        }
    public void writeExternal(PofWriter writer)
            throws IOException
        {
        writer.writeString(0, getName());
        writer.writeString(1, getStreet());
        writer.writeString(2, getCity());
        writer.writeString(3, getState());
        writer.writeString(4, getZip());
        }
    // accessor methods omitted for brevity
}
```

Registering Custom Types on the .NET Client

Each POF user type is represented within the POF stream as an integer value. As such, POF requires an external mechanism that allows a user type to be mapped to its encoded type identifier (and visa versa). This mechanism uses an XML configuration file to store the mapping information. This is illustrated in Example 15–3. These elements are described in *"POF User Type Configuration Elements* .

Example 15–3 Storing Mapping Informaiton in the POF User Type Configuration File

```
<?xml version="1.0"?>
<pof-config xmlns="http://schemas.tangosol.com/pof">
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->

<include>assembly://Coherence/Tangosol.Config/coherence-pof-config.xml</include>
    <!-- include all application POF user types -->

    <user-type>

      <type-id>1001</type-id>
      <class-name>My.Example.ContactInfo, MyAssembly</class-name>
    </user-type>
    ...
  </user-type-list>
```

```
</pof-config>
```

There are few things to note:

- Type identifiers for your custom types should start from 1001 or higher, as the numbers below 1000 are reserved for internal use.
- You need not specify a fully qualified type name within the class-name element. The type and assembly name is enough.

Once you have configured mappings between type identifiers and your custom types, you must configure Coherence for .NET to use them by adding a serializer element to your cache configuration descriptor. Assuming that user type mappings from Example 15–3 are saved into `my-dotnet-pof-config.xml`, you need to specify a serializer element as illustrated in Example 15–4:

Example 15–4 Using a Serializer in the Cache Configuration File

```
<remote-cache-scheme>
  <scheme-name>extend-direct</scheme-name>
  <service-name>ExtendTcpCacheService</service-name>
  <initiator-config>
    ...
    <serializer>
      <class-name>Tangosol.IO.Pof.ConfigurablePofContext, Coherence</class-name>
      <init-params>
        <init-param>
          <param-type>string</param-type>
          <param-value>my-dotnet-pof-config.xml</param-value>
        </init-param>
      </init-params>
    </serializer>
   </initiator-config>
</remote-cache-scheme>
```

The `ConfigurablePofContext` type will be used for the POF serializer if one is not explicitly specified. It uses a default configuration file (`$AppRoot/coherence-pof-config.xml`) if it exists, or a specific file determined by the contents of the `pof-config` element in the Coherence for .NET application configuration file. For example:

Example 15–5 Specifying a POF Configuration File

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
Coherence"/>
  </configSections>
  <coherence>
    <pof-config>my-dotnet-pof-config.xml</pof-config>
  </coherence>
</configuration>
```

See "Configuring and Using the Coherence for .NET Client Library" for additional details.

Registering Custom Types in the Cluster

Each Coherence node running the TCP/IP Coherence*Extend clustered service requires a similar POF configuration for the custom types to be able to send and receive objects of these types.

The cluster-side POF configuration file looks similar to the one created on the client. The only difference is that instead of .NET class names, you must specify the fully qualified Java class names within the class-name element.

Example 15–6 illustrates a sample cluster-side POF configuration file called `my-java-pof-config.xml`:

Example 15–6 Cluster-side POF Configuration File

```
<?xml version="1.0"?>
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
<pof-config>
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->
    <include>example-pof-config.xml</include>
    <!-- include all application POF user types -->
    <user-type>
      <type-id>1001</type-id>
      <class-name>com.mycompany.example.ContactInfo</class-name>
    </user-type>
    ...
  </user-type-list>
</pof-config>
```

Once your custom types have been added, you must configure the server to use your POF configuration when serializing objects. This is illustrated in Example 15–7:

Example 15–7 Configuring the Server to Use the POF Configuration

```
<proxy-scheme>
  <service-name>ExtendTcpProxyService</service-name>
  <acceptor-config>
    ...
    <serializer>
      <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
      <init-params>
        <init-param>
          <param-type>string</param-type>
          <param-value>my-java-pof-config.xml</param-value>
        </init-param>
      </init-params>
    </serializer>
   </acceptor-config>
  ...
</proxy-scheme>
```

Evolvable Portable User Types

PIF-POF includes native support for both forward- and backward-compatibility of the serialized form of portable user types. In .NET, this is accomplished by making user types implement the `IEvolvablePortableObject` interface instead of the `IPortableObject` interface. The `IEvolvablePortableObject` interface is a marker interface that extends both the `IPortableObject` and IEvolvable interfaces. The `IEvolvable` interface adds three properties to support type versioning.

An `IEvolvable` class has an integer version identifier n, where n >= 0. When the contents and/or semantics of the serialized form of the `IEvolvable` class changes, the version identifier is increased. Two versions identifiers, `n1` and `n2`, indicate the same version if `n1 == n2`; the version indicated by `n2` is newer than the version indicated by `n1` if `n2 > n1`.

The `IEvolvable` interface is designed to support the evolution of types by the addition of data. Removal of data cannot be safely accomplished if a previous version of the type exists that relies on that data. Modifications to the structure or semantics of data from previous versions likewise cannot be safely accomplished if a previous version of the type exists that relies on the previous structure or semantics of the data.

When an `IEvolvable` object is deserialized, it retains any unknown data that has been added to newer versions of the type, and the version identifier for that data format. When the IEvolvable object is subsequently serialized, it includes both that version identifier and the unknown future data.

When an `IEvolvable` object is deserialized from a data stream whose version identifier indicates an older version, it must default and/or calculate the values for any data fields and properties that have been added since that older version. When the `IEvolvable` object is subsequently serialized, it includes its own version identifier and all of its data. Note that there will be no unknown future data in this case; future data can only exist when the version of the data stream is newer than the version of the IEvolvable type.

Example 15–8 demonstrates how the `ContactInfo` .NET type can be modified to support class evolution:

Example 15–8 Modifying a Class to Support Class Evolution

```
public class ContactInfo : IEvolvablePortableObject
{
    private string name;
    private string street;
    private string city;
    private string state;
    private string zip;
    // IEvolvable members
    private int    version;
    private byte[] data;
    public ContactInfo()
    {}
    public ContactInfo(string name, string street, string city, string state,
string zip)
    {
        Name   = name;
        Street = street;
        City   = city;
        State  = state;
        Zip    = zip;
    }
    public void ReadExternal(IPofReader reader)
    {
        Name   = reader.ReadString(0);
        Street = reader.ReadString(1);
        City   = reader.ReadString(2);
        State  = reader.ReadString(3);
        Zip    = reader.ReadString(4);
    }
    public void WriteExternal(IPofWriter writer)
```

```
    {
        writer.WriteString(0, Name);
        writer.WriteString(1, Street);
        writer.WriteString(2, City);
        writer.WriteString(3, State);
        writer.WriteString(4, Zip);
    }
    public int DataVersion
    {
        get { return version; }
        set { version = value; }
    }
    public byte[] FutureData
    {
        get { return data; }
        set { data = value; }
    }
    public int ImplVersion
    {
        get { return 0; }
    }
    // property definitions ommitted for brevity
}
```

Likewise, the `ContactInfo` Java type can also be modified to support class evolution by implementing the `EvolvablePortableObject` interface:

Example 15–9 Modifying a Java Type Class to Support Class Evolution

```
public class ContactInfo
        implements EvolvablePortableObject
    {
    private String m_sName;
    private String m_sStreet;
    private String m_sCity;
    private String m_sState;
    private String m_sZip;

    // Evolvable members
    private int    m_nVersion;
    private byte[] m_abData;

    public ContactInfo()
        {}

    public ContactInfo(String sName, String sStreet, String sCity,
            String sState, String sZip)
        {
        setName(sName);
        setStreet(sStreet);
        setCity(sCity);
        setState(sState);
        setZip(sZip);
        }

    public void readExternal(PofReader reader)
            throws IOException
        {
        setName(reader.readString(0));
        setStreet(reader.readString(1));
```

```
        setCity(reader.readString(2));
        setState(reader.readString(3));
        setZip(reader.readString(4));
        }

    public void writeExternal(PofWriter writer)
            throws IOException
        {
        writer.writeString(0, getName());
        writer.writeString(1, getStreet());
        writer.writeString(2, getCity());
        writer.writeString(3, getState());
        writer.writeString(4, getZip());
        }

    public int getDataVersion()
        {
        return m_nVersion;
        }

    public void setDataVersion(int nVersion)
        {
        m_nVersion = nVersion;
        }

    public Binary getFutureData()
        {
        return m_binData;
        }

    public void setFutureData(Binary binFuture)
        {
        m_binData = binFuture;
        }

    public int getImplVersion()
        {
        return 0;
        }

    // accessor methods omitted for brevity
    }
```

Making Types Portable Without Modification

In some cases, it may be undesirable or impossible to modify an existing user type to make it portable. In this case, you can externalize the portable serialization of a user type by creating an implementation of the `IPofSerializer` in .NET and/or an implementation of the `PofSerializer` interface in Java.

Example 15–10 illustrates, an implementation of the `IPofSerializer` interface for the `ContactInfo` type.

Example 15–10 An Implementation of IPofSerializer for the .NET Type

```
public class ContactInfoSerializer : IPofSerializer
{
    public object Deserialize(IPofReader reader)
    {
        string name   = reader.ReadString(0);
```

```
        string street = reader.ReadString(1);
        string city   = reader.ReadString(2);
        string state  = reader.ReadString(3);
        string zip    = reader.ReadString(4);

        ContactInfo info = new ContactInfo(name, street, city, state, zip);
        info.DataVersion = reader.VersionId;
        info.FutureData  = reader.ReadRemainder();

        return info;
    }

    public void Serialize(IPofWriter writer, object o)
    {
        ContactInfo info = (ContactInfo) o;

        writer.VersionId = Math.Max(info.DataVersion, info.ImplVersion);
        writer.WriteString(0, info.Name);
        writer.WriteString(1, info.Street);
        writer.WriteString(2, info.City);
        writer.WriteString(3, info.State);
        writer.WriteString(4, info.Zip);
        writer.WriteRemainder(info.FutureData);
    }
}
```

An implementation of the `PofSerializer` interface for the `ContactInfo` Java type would look similar:

Example 15–11 An Implementation of PofSerializer for the Java Type Class

```
public class ContactInfoSerializer
        implements PofSerializer
    {
    public Object deserialize(PofReader in)
            throws IOException
        {
        String sName   = in.readString(0);
        String sStreet = in.readString(1);
        String sCity   = in.readString(2);
        String sState  = in.readString(3);
        String sZip    = in.readString(4);

        ContactInfo info = new ContactInfo(sName, sStreet, sCity, sState, sZip);
        info.setDataVersion(in.getVersionId());
        info.setFutureData(in.readRemainder());

        return info;
        }

    public void serialize(PofWriter out, Object o)
            throws IOException
        {
        ContactInfo info = (ContactInfo) o;

        out.setVersionId(Math.max(info.getDataVersion(), info.getImplVersion()));
        out.writeString(0, info.getName());
        out.writeString(1, info.getStreet());
        out.writeString(2, info.getCity());
        out.writeString(3, info.getState());
```

```
        out.writeString(4, info.getZip());
        out.writeRemainder(info.getFutureData());
        }
    }
```

To register the `IPofSerializer` implementation for the `ContactInfo` .NET type, specify the class name of the `IPofSerializer` within a serializer element under the user-type element for the `ContactInfo` user type in the POF configuration file. This is illustrated in Example 15–12:

Example 15–12 Registering the IPofSerializer Implementation of the .NET Type

```
<?xml version="1.0"?>

<pof-config xmlns="http://schemas.tangosol.com/pof">
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->

<include>assembly://Coherence/Tangosol.Config/coherence-pof-config.xml</include>

    <!-- include all application POF user types -->
    <user-type>
      <type-id>1001</type-id>
      <class-name>My.Example.ContactInfo, MyAssembly</class-name>
      <serializer>
        <class-name>My.Example.ContactInfoSerializer, MyAssembly</class-name>
      </serializer>
    </user-type>
      ...
  </user-type-list>
</pof-config>
```

Similarly, you can register the `PofSerializer` implementation for the `ContactInfo` Java type. This is illustrated in Example 15–13.

Example 15–13 Registering the PofSerializer Implementation of the Java Type

```
<?xml version="1.0"?>
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
<pof-config>
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->
    <include>example-pof-config.xml</include>

    <!-- include all application POF user types -->
    <user-type>
      <type-id>1001</type-id>
      <class-name>com.mycompany.example.ContactInfo</class-name>
      <serializer>
        <class-name>com.mycompany.example.ContactInfoSerializer</class-name>
      </serializer>
    </user-type>
    ...
  </user-type-list>
</pof-config>
```

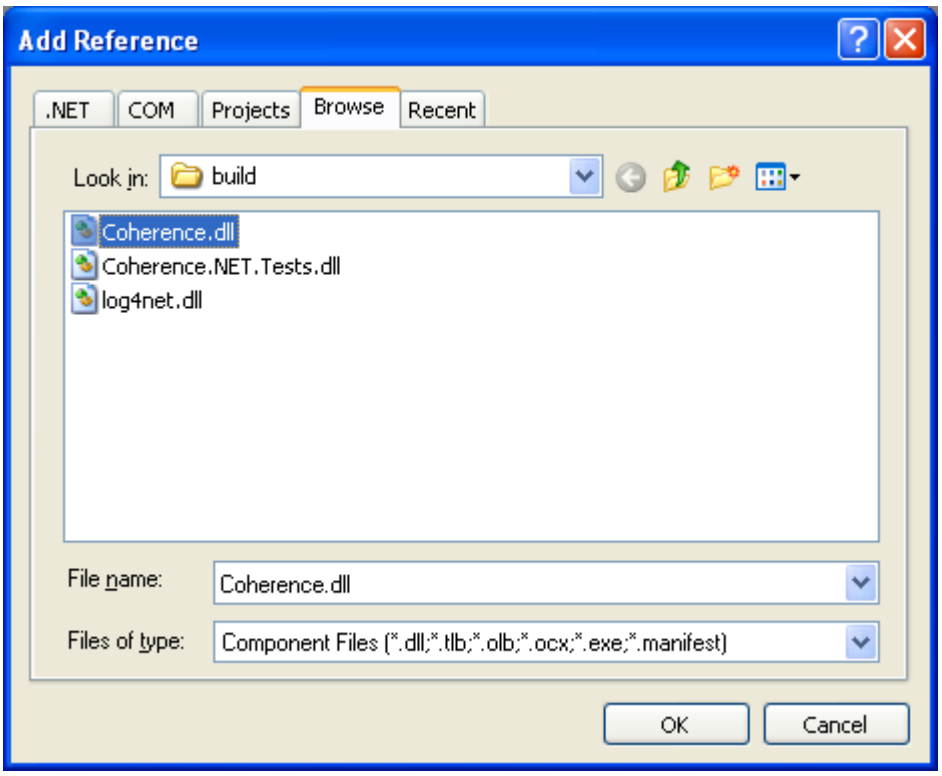
Configuring and Using the Coherence for .NET Client Library

To use the Coherence for .NET library in your .NET applications, you must add a reference to the `Coherence.dll` library in your project and create the necessary configuration files.

Creating a reference to the `Coherence.dll`:

1. In your project go to **Project->Add Reference...** or right click **References** in the Solution Explorer and choose **Add Reference...**
2. In the **Add Reference** window that appears, choose the **Browse** tab and find the `Coherence.dll` library on your file system.

Figure 15–1 Add Reference Window



This illustration is described in the text.

****

3. Click **OK**.

Next, you must create the necessary configuration files and specify their paths in the application configuration settings. This is done by adding an application configuration file to your project (if one was not already created) and adding a Coherence for .NET configuration section (that is, `<coherence/>`) to it.

Example 15–14 Sample Application Configuration File

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
Coherence"/>
```

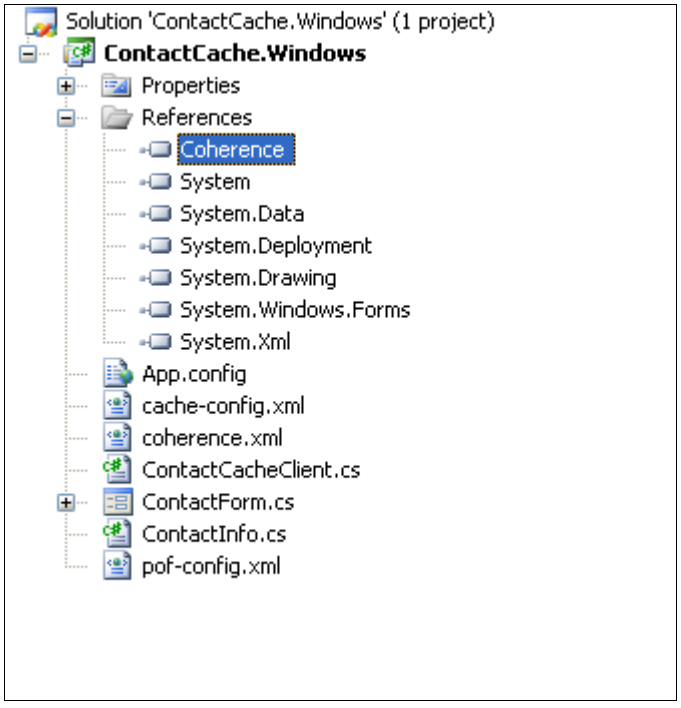
```
  </configSections>
  <coherence>
    <cache-factory-config>my-coherence.xml</cache-factory-config>
    <cache-config>my-cache-config.xml</cache-config>
    <pof-config>my-pof-config.xml</pof-config>
  </coherence>
</configuration>
```

Elements within the Coherence for .NET configuration section are:

- `cache-factory-config`—contains the path to a configuration descriptor used by the CacheFactory to configure the IConfigurableCacheFactory and Logger used by the `CacheFactory`.
- `cache-config`—contains the path to a cache configuration descriptor which contains the cache configuration described earlier (see Configuring Coherence*Extend on the Client). This cache configuration descriptor is used by the DefaultConfigurableCacheFactory.
- `pof-config`—contains the path to the configuration descriptor used by the ConfigurablePofContext to register custom types used by the application.

Figure 15–2 illustrates what the solution should look like after adding the configuration files:

Figure 15–2 File System Displaying the Configuration Files



This illustration is described in the text.

****

CacheFactory

The `CacheFactory` is the entry point for Coherence for .NET client applications. The `CacheFactory` is a factory for `INamedCache` instances and provides various methods for logging. If not configured explicitly, it uses the default configuration file

coherence.xml which is an assembly embedded resource. It is possible to override the default configuration file by adding a cache-factory-config element to the Coherence for .NET configuration section in the application configuration file and setting its value to the path of the desired configuration file.

Example 15–15 Configuring a Factory for INamedCache Instances

```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
Coherence"/>
  </configSections>
  <coherence>
    <cache-factory-config>my-coherence.xml</cache-factory-config>
    ...
  </coherence>
</configuration>
```

This file contains the configuration of two components exposed by the CacheFactory by using static properties:

- CacheFactory.ConfigurableCacheFactory—the IConfigurableCacheFactory implementation used by the CacheFactory to retrieve, release, and destroy INamedCache instances.
- CacheFactory.Logger—the Logger instance used to log messages and exceptions.

When you are finished using the CacheFactory (for example, during application shutdown), the CacheFactory should be shutdown by using the Shutdown() method. This method terminates all services and the Logger instance.

IConfigurableCacheFactory

The IConfigurableCacheFactory implementation is specified by the contents of the <configurable-cache-factory-config> element:

- class-name—specifies the implementation type by it's assembly qualified name.
- init-params—defines parameters used to instantiate the IConfigurableCacheFactory. Each parameter is specified by using a corresponding param-type and param-value child element.

Example 15–16 Configuring a ConfigurableCacheFactory Implementation

```
<coherence>
  <configurable-cache-factory-config>
    <class-name>Tangosol.Net.DefaultConfigurableCacheFactory,
Coherence</class-name>
    <init-params>
      <init-param>
        <param-type>string</param-type>
        <param-value>simple-cache-config.xml</param-value>
      </init-param>
    </init-params>
  </configurable-cache-factory-config>
</coherence>
```

If an `IConfigurableCacheFactory` implementation is not defined in the configuration, the default implementation is used (`DefaultConfigurableCacheFactory`).

DefaultConfigurableCacheFactory

The `DefaultConfigurableCacheFactory` provides a facility to access caches declared in the cache configuration descriptor described earlier (see the Client-side Cache Configuration Descriptor section). The default configuration file used by the `DefaultConfigurableCacheFactory` is `$AppRoot/coherence-cache-config.xml`, where `$AppRoot` is the working directory (in the case of a Windows Forms application) or the root of the application (in the case of a Web application).

If you want to specify another cache configuration descriptor file, you can do so by adding a `cache-config` element to the Coherence for .NET configuration section in the application configuration file with its value set to the path of the configuration file.

Example 15–17 Specifying a Different Cache Configuration Desriptor File

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
Coherence"/>
  </configSections>
  <coherence>
    <cache-config>my-cache-config.xml</cache-config>
    ...
  </coherence>
</configuration>
```

Logger

The `Logger` is configured using the `logging-config` element:

- `destination`—determines the type of LogOutput used by the Logger. Valid values are:
  - `common-logger` for `Common.Logging`
  - `stderr` for `Console.Error`
  - `stdout` for `Console.Out`
  - file path if messages should be directed to a file
- `severity-level`—determines the log level that a message must meet or exceed to be logged.
- `message-format`—determines the log message format.
- `character-limit`—determines the maximum number of characters that the logger daemon will process from the message queue before discarding all remaining messages in the queue.

Example 15–18 Configuring a Logger

```
<coherence>
  <logging-config>
    <destination>log4net</destination>
    <severity-level>5</severity-level>
```

```
    <message-format>(thread={thread}): {text}</message-format>
    <character-limit>8192</character-limit>
  </logging-config>
</coherence>
```

The `CacheFactory` provides several static methods for retrieving and releasing `INamedCache` instances:

- `GetCache(String cacheName)`—retrieves an `INamedCache` implementation that corresponds to the `NamedCache` with the specified `cacheName` running within the remote Coherence cluster.
- `ReleaseCache(INamedCache cache)`—releases all local resources associated with the specified instance of the cache. After a cache is release, it can no longer be used.
- `DestroyCache(INamedCache cache)`—destroys the specified cache across the Coherence cluster.

Methods used to log messages and exceptions are:

- `IsLogEnabled(int level)`—determines if the Logger would log a message with the given severity level.
- `Log(Exception e, int severity)`—logs an exception with the specified severity level.
- `Log(String message, int severity)`—logs a text message with the specified severity level.
- `Log(String message, Exception e, int severity)`—logs a text message and an exception with the specified severity level.

Logging levels are defined by the values of the `CacheFactory.LogLevel enum` values (in ascending order):

- `Always`
- `Error`
- `Warn`
- `Info`
- `Debug`—(default log level)
- `Quiet`
- `Max`

Using the Common.Logging Library

`Common.Logging` is an open source library that enables you to plug in various popular open source logging libraries behind a well-defined set of interfaces. The libraries currently supported are Log4Net (versions 1.2.9 and 1.2.10) and `NLog`. `Common.Logging` is currently used by the Spring.NET framework and will likely be used in the future releases of IBatis.NET and NHibernate, so you might want to consider it if you are using one or more of these frameworks in combination with Coherence for .NET, as it will allow you to configure logging consistently throughout the application layers.

Coherence for .NET does not include the `Common.Logging` library. If you would like to use the `common-logger` Logger configuration, you must download the `Common.Logging` assembly and include a reference to it in your project. You can

download the `Common.Logging` assemblies for both .NET 1.1 and 2.0 from the following location:

http://netcommon.sourceforge.net/

The Coherence for .NET Common.Logging Logger implementation was compiled against the signed release version of these assemblies.

INamedCache

The `INamedCache` interface extends `IDictionary`, so it can be manipulated in ways similar to a dictionary. Once obtained, `INamedCache` instances expose several properties:

- `CacheName`—the cache name.
- `Count`—the cache size.
- `IsActive`—determines if the cache is active (that is, it has not been released or destroyed).
- `Keys`—collection of all keys in the cache mappings.
- `Values`—collection of all values in the cache mappings.

The value for the specified key can be retrieved by using `cache[key]`. Similarly, a new value can be added, or an old value can be modified by setting this property to the new value: `cache[key] = value`.

The collection of cache entries can be accessed by using `GetEnumerator()` which can be used to iterate over the mappings in the cache.

The `INamedCache` interface provides several methods used to manipulate the contents of the cache:

- `Clear()`—removes all the mappings from the cache.
- `Contains(Object key)`—determines if the cache has a mapping for the specified key.
- `GetAll(ICollection keys)`—returns all values mapped to the specified keys collection.
- `Insert(Object key, Object value)`—places a new mapping into the cache. If a mapping for the specified key already exists, its value will be overwritten by the specified value and the old value will be returned.
- `Insert(Object key, Object value, long millis)`—places a new mapping into the cache, but with an expiry period specified by several milliseconds.
- `InsertAll(IDictionary dictionary)`—copies all the mappings from the specified dictionary to the cache.
- `Remove(Object key)`—Removes the mapping for the specified key if it is present and returns the value it was mapped to.

`INamedCache` interface also extends the following three interfaces: IQueryCache, IObservableCache, and IInvocableCache.

IQueryCache

The `IQueryCache` interface exposes the ability to query a cache using various filters.

- `GetKeys(IFilter filter)`—returns a collection of the keys contained in this cache for entries that satisfy the criteria expressed by the filter.
- `GetEntries(IFilter filter)`—returns a collection of the entries contained in this cache that satisfy the criteria expressed by the filter.
- `GetEntries(IFilter filter, IComparer comparer)`—returns a collection of the entries contained in this cache that satisfy the criteria expressed by the filter. It is guaranteed that the enumerator will traverse the collection in the order of ascending entry values, sorted by the specified comparer or according to the natural ordering if the "comparer" is null.

Additionally, the `IQueryCache` interface includes the ability to add and remove indexes. Indexes are used to correlate values stored in the cache to their corresponding keys and can dramatically increase the performance of the `GetKeys` and `GetEntries` methods.

- `AddIndex(IValueExtractor extractor, bool isOrdered, IComparer comparator)`—adds an index to this cache that correlates the values extracted by the given `IValueExtractor` to the keys to the corresponding entries. Additionally, the index information can be optionally ordered.
- `RemoveIndex(IValueExtractor extractor)`—removes an index from this cache.

Example 15–19 illustrates code that performs an efficient query of the keys of all entries that have an age property value greater or equal to 55.

Example 15–19 Querying Keys on a Particular Value

```
IValueExtractor extractor = new ReflectionExtractor("getAge");

cache.AddIndex(extractor, true, null);
ICollection keys = cache.GetKeys(new GreaterEqualsFilter(extractor, 55));
```

IObservableCache

`IObservableCache` interface enables an application to receive events when the contents of a cache changes. To register interest in change events, an application adds a `Listener` implementation to the cache that will receives events that include information about the event type (inserted, updated, deleted), the key of the modified entry, and the old and new values of the entry.

- `AddCacheListener(ICacheListener listener)`—adds a standard cache listener that will receive all events (inserts, updates, deletes) emitted from the cache, including their keys, old, and new values.
- `RemoveCacheListener(ICacheListener listener)`—removes a standard cache listener that was previously registered.
- `AddCacheListener(ICacheListener listener, object key, bool isLite)`—adds a cache listener for a specific key. If `isLite` is `true`, the events may not contain the old and new values.
- `RemoveCacheListener(ICacheListener listener, object key)`—removes a cache listener that was previously registered using the specified key.
- `AddCacheListener(ICacheListener listener, IFilter filter, bool isLite)`—adds a cache listener that receive events based on a filter evaluation. If isLite is true, the events may not contain the old and new values.

- `RemoveCacheListener(ICacheListener listener, IFilter filter)`—removes a cache listener that previously registered using the specified filter.

`Listeners` registered using the filter-based method will receive all event types (inserted, updated, and deleted). To further filter the events, wrap the filter in a `CacheEventFilter` using a `CacheEventMask` enumeration value to specify which type of events should be monitored.

In Example 15–20 a filter evaluates to `true` if an `Employee` object is inserted into a cache with an `IsMarried` property value set to `true`.

Example 15–20 Filtering on an Inserted Object

```
new CacheEventFilter(CacheEventMask.Inserted, new EqualsFilter("IsMarried",
true));
```

In Example 15–21 a filter evaluates to `true` if any object is removed from a cache.

Example 15–21 Filtering on Removed Object

```
new CacheEventFilter(CacheEventMask.Deleted);
```

In Example 15–22 a filter `that` evaluates to `true` if when an `Employee` object `LastName` property is changed from `Smith`.

Example 15–22 Filtering on a Changed Object

```
new CacheEventFilter(CacheEventMask.UpdatedLeft, new EqualsFilter("LastName",
"Smith"));
```

IInvocableCache

An `IInvocableCache` is a cache against which both entry-targeted processing and aggregating operations can be invoked. The operations against the cache contents are executed by (and thus within the localized context of) a cache. This is particularly useful in a distributed environment, because it enables the processing to be moved to the location at which the entries-to-be-processed are being managed, thus providing efficiency by localization of processing.

- `Invoke(object key, IEntryProcessor agent)`—invokes the passed processor against the entry specified by the passed key, returning the result of the invocation.
- `InvokeAll(ICollection keys, IEntryProcessor agent)`—invokes the passed processor against the entries specified by the passed keys, returning the result of the invocation for each.
- `InvokeAll(IFilter filter, IEntryProcessor agent)`—invokes the passed processor against the entries that are selected by the given filter, returning the result of the invocation for each.
- `Aggregate(ICollection keys, IEntryAggregator agent)`—performs an aggregating operation against the entries specified by the passed keys.
- `Aggregate(IFilter filter, IEntryAggregator agent)`—performs an aggregating operation against the entries that are selected by the given filter.

Filters

The `IQueryCache` interface provides the ability to search for cache entries that meet a given set of criteria, expressed using a `IFilter` implementation.

All filters must implement the IFilter interface:

- `Evaluate(object o)`—apply a test to the specified object and return `true` if the test passes, `false` otherwise.

Coherence for .NET includes several `IFilter` implementations in the `Tangosol.Util.Filter` namespace.

The code in Example 15–23 retrieves the keys of all entries that have a value equal to 5.

Example 15–23 Retrieving Keys Equal to a Numeric Value

```
EqualsFilter equalsFilter = new EqualsFilter(IdentityExtractor.Instance, 5);
ICollection  keys         = cache.GetKeys(equalsFilter);
```

The code in Example 15–24 retrieves all keys that have a value greater or equal to 55.

Example 15–24 Retrieving Keys Greater Than or Equal To a Numeric Value

```
GreaterEqualsFilter greaterEquals = new
GreaterEqualsFilter(IdentityExtractor.Instance, 55);
ICollection         keys          = cache.GetKeys(greaterEquals);
```

The code in Example 15–25 retrieves all cache entries that have a value that begins with `Belg`.

Example 15–25 Retrieving Keys Based on a String Value

```
LikeFilter  likeFilter = new LikeFilter(IdentityExtractor.Instance, "Belg%", '\\',
true);
ICollection entries    = cache.GetEntries(likeFilter);
```

The code in Example 15–26 retrieves all cache entries that have a value that ends with an (case sensitive) or begins with `An` (case insensitive).

Example 15–26 Retrieving Keys Based on a Case-Sensitive String Value

```
OrFilter    orFilter = new OrFilter(new LikeFilter(IdentityExtractor.Instance,
"%an", '\\', false), new LikeFilter(IdentityExtractor.Instance, "An%", '\\',
true));
ICollection entries  = cache.GetEntries(orFilter);
```

Extractors

Extractors are used to extract values from an object. All extractors must implement the `IValueExtractor` interface:

- `Extract(object target)`—extract the value from the passed object.

Coherence for .NET includes the following extractors:

- `IdentityExtractor` is a trivial implementation that does not actually extract anything from the passed value, but returns the value itself.
- `KeyExtractor` is a special purpose implementation that serves as an indicator that a query should be run against the key objects rather than the values.
- `ReflectionExtractor` extracts a value from a specified object property.

- `MultiExtractor` is composite `IValueExtractor` implementation based on an array of extractors. All extractors in the array are applied to the same target object and the result of the extraction is a `IList` of extracted values.
- `ChainedExtractor` is composite `IValueExtractor` implementation based on an array of extractors. The extractors in the array are applied sequentially left-to-right, so a result of a previous extractor serves as a target object for a next one.

The code in Example 15–27 retrieves all cache entries with keys greater than 5:

Example 15–27 Retrieving Cache Entries Greater Than a Numeric Value

```
IValueExtractor extractor = new KeyExtractor(IdentityExtractor.Instance);
IFilter         filter    = new GreaterFilter(extractor, 5);
ICollection     entries   = cache.GetEntries(filter);
```

The code inExample 15–28 retrieves all cache entries with values containing a `City` property equal to `city1`:

Example 15–28 Retrieving Cache Entries Based on a String Value

```
IValueExtractor extractor = new ReflectionExtractor("City");
IFilter         filter    = new EqualsFilter(extractor, "city1");
ICollection     entries   = cache.GetEntries(filter);
```

Processors

A processor is an invocable agent that operates against the entry objects within a cache.

All processors must implement the `IEntryProcessor` interface:

- `Process(IInvocableCacheEntry entry)`—process the specified entry.
- `ProcessAll(ICollection entries)`—process a collection of entries.

Coherence for .NET includes several `IEntryProcessor` implementations in the `Tangosol.Util.Processor` namespace.

The code in Example 15–29 demonstrates a conditional put. The value mapped to `key1` is set to `680` only if the current mapped value is greater than `600`.

Example 15–29 Conditional Put of a Key Value Based on a Numeric Value

```
IFilter         greaterThen600 = new GreaterFilter(IdentityExtractor.Instance,
600);
IEntryProcessor processor      = new ConditionalPut(greaterThen600, 680);
cache.Invoke("key1", processor);
```

The code in Example 15–30 uses the `UpdaterProcessor` to update the value of the `Degree` property on a `Temperature` object with key `BGD` to the new value `26`.

Example 15–30 Setting a Key Value Based on a Numeric Value

```
cache.Insert("BGD", new Temperature(25, 'c', 12));
IValueUpdater   updater   = new ReflectionUpdater("setDegree");
IEntryProcessor processor = new UpdaterProcessor(updater, 26);
object          result    = cache.Invoke("BGD", processor);
```

Aggregators

An aggregator represents processing that can be directed to occur against some subset of the entries in an `IInvocableCache`, resulting in an aggregated result. Common examples of aggregation include functions such as minimum, maximum, sum and average. However, the concept of aggregation applies to any process that must evaluate a group of entries to come up with a single answer. Aggregation is explicitly capable of being run in parallel, for example in a distributed environment.

All aggregators must implement the `IEntryAggregator` interface:

- `Aggregate(ICollection entries)`—process a collection of entries to produce an aggregate result.

Coherence for .NET includes several `IEntryAggregator` implementations in the `Tangosol.Util.Aggregator` namespace.

The code in Example 15–31 returns the size of the cache:

Example 15–31 Returning the Size of the Cache

```
IEntryAggregator aggregator = new Count();
object           result     = cache.Aggregate(cache.Keys, aggregator);
```

The code in Example 15–32 returns an `IDictionary` with keys equal to the unique values in the cache and values equal to the number of instances of the corresponding value in the cache:

Example 15–32 Returning an IDictionary

```
IEntryAggregator aggregator =
GroupAggregator.CreateInstance(IdentityExtractor.Instance, new Count());
object           result     = cache.Aggregate(cache.Keys, aggregator);
```

Like cached value objects, all custom `IFilter`, `IExtractor`, `IProcessor` and `IAggregator` implementation classes must be correctly registered in the POF context of the .NET application and cluster-side node to which the client is connected. As such, corresponding Java implementations of the custom .NET types must be created, compiled, and deployed on the cluster-side node. Note that the actual execution of the these custom types is performed by the Java implementation and not the .NET implementation.

See "Configuring a POF Context" for additional details.

Launching a Coherence DefaultCacheServer Process

To start a `DefaultCacheServer` that uses the cluster-side Coherence cache configuration described earlier to allow Coherence for .NET clients to connect to the Coherence cluster by using TCP/IP, you need to do the following:

1. Change the current directory to the Oracle Coherence library directory (`%COHERENCE_HOME%\lib` on Windows and `$COHERENCE_HOME/lib` on UNIX).
2. Make sure that the paths are configured so that the Java command will run.
3. Start the `DefaultCacheServer` command line application with the `-Dtangosol.coherence.cacheconfig` system property set to the location of the cluster-side Coherence cache configuration descriptor described earlier.

Example 15–33 illustrates a sample command line.

Example 15–33 Command to Launch a Coherence Default Cache Server

```
java -cp coherence.jar -Dtangosol.coherence.cacheconfig=file://<path to the
server-side cache configuration descriptor> com.tangosol.net.DefaultCacheServer
```

16

Configuring a Local Cache for .NET Clients

A **Local Cache** is just that: A cache that is local to (completely contained within) a particular .NET application. There are several attributes of the Local Cache that are particularly interesting:

- The Local Cache implements the same standard cache interfaces that a remote cache implements (`ICache`, `IObservableCache`, `IConcurrentCache`, `IQueryCache`, and `IInvocableCache`), meaning that there is no programming difference between using a local and a remote cache.
- The Local Cache can be size-limited. This means that the Local Cache can restrict the number of entries that it caches, and automatically evict entries when the cache becomes full. Furthermore, both the sizing of entries and the eviction policies are customizable, for example allowing the cache to be size-limited based on the memory used by the cached entries. The default eviction policy uses a combination of Most Frequently Used (MFU) and Most Recently Used (MRU) information, scaled on a logarithmic curve, to determine what cache items to evict. This algorithm is the best general-purpose eviction algorithm because it works well for short duration and long duration caches, and it balances frequency versus recentness to avoid cache thrashing. The pure LRU and pure LFU algorithms are also supported, and the ability to plug in custom eviction policies.
- The Local Cache supports automatic expiration of cached entries, meaning that each cache entry can be assigned a time-to-live value in the cache. Furthermore, the entire cache can be configured to flush itself on a periodic basis or at a preset time.
- The Local Cache is thread safe and highly concurrent.
- The Local Cache provides cache "get" statistics. It maintains hit and miss statistics. These runtime statistics can be used to accurately project the effectiveness of the cache, and adjust its size-limiting and auto-expiring settings accordingly while the cache is running.

The Coherence for .NET Local Cache functionality is implemented by the `Tangosol.Net.Cache.LocalCache` class. As such, it can be programatically instantiated and configured; however, it is recommended that a `LocalCache` be configured by using a cache configuration descriptor, just like any other Coherence for .NET cache.

Configuring the Local Cache

The key element for configuring the Local Cache is `<local-scheme>`. Local caches are generally nested within other cache schemes, for instance as the front-tier of a near-scheme. Thus, this element can appear as a subelement of any of these elements

in the coherence-cache-config file: <caching-schemes>, <distributed-scheme>, <replicated-scheme>, <optimistic-scheme>, <near-scheme>, <versioned-near-scheme>, <overflow-scheme>, <read-write-backing-map>, and <versioned-backing-map-scheme>.

The <local-scheme> provides several optional subelements that let you define the characteristics of the cache. For example, the <low-units> and <high-units> subelements allow you to limit the cache in terms of size. Once the cache reaches its maximum allowable size it prunes itself back to a specified smaller size, choosing which entries to evict according to a specified eviction-policy (<eviction-policy>). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (<unit-calculator>).

You can also limit the cache in terms of time. The <expiry-delay> subelement specifies the amount of time from last update that entries will be kept by the cache before being marked as expired. Any attempt to read an expired entry will result in a reloading of the entry from the configured cache store (<cachestore-scheme>). Expired values are periodically discarded from the cache based on the flush-delay.

If a <cachestore-scheme> is not specified, then the cached data will only reside in memory, and only reflect operations performed on the cache itself. See <local-scheme> for a complete description of all of the available subelements.

Example 16–1 illustrates the configuration of a Local Cache. See *"Sample Cache Configurations"* for additional examples.

Example 16–1 Configuring a Local Cache

```
<?xml version="1.0"?>

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example-local-cache</cache-name>
      <scheme-name>example-local</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <local-scheme>
      <scheme-name>example-local</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>32000</high-units>
      <low-units>10</low-units>
      <unit-calculator>FIXED</unit-calculator>
      <expiry-delay>10ms</expiry-delay>
      <flush-delay>1000ms</flush-delay>
      <cachestore-scheme>
        <class-scheme>
          <class-name>ExampleCacheStore</class-name>
        </class-scheme>
      </cachestore-scheme>
      <pre-load>true</pre-load>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

Obtaining a Local Cache Reference for .NET Clients

A reference to a configured Local Cache can be obtained by name by using the CacheFactory class:

Example 16–2 Obtaining a Reference to a Local Cache

```
INamedCache cache = CacheFactory.GetCache("example-local-cache");
```

Cleaning Up Resources Associated with a LocalCache

Instances of all `INamedCache` implementations, including `LocalCache`, should be explicitly released by calling the `INamedCache.Release()` method when they are no longer needed, to free up any resources they might hold.

If the particular `INamedCache` is used for the duration of the application, then the resources will be cleaned up when the application is shut down or otherwise stops. However, if it is only used for a period, the application should call its `Release()` method when finished using it.

Alternatively, you can leverage the fact that `INamedCache` extends `IDisposable` and that all cache implementations delegate a call to `IDisposable.Dispose()` to `INamedCache.Release()`. This means that if you need to obtain and release a cache instance within a single method, you can do so with a `using` block:

Example 16–3 Obtaining and Releasing a Reference to a Local Cache

```
using (INamedCache cache = CacheFactory.GetCache("my-cache"))
{
   // use cache as usual
}
```

After the `using` block terminates, `IDisposable.Dispose()` will be called on the `INamedCache` instance, and all resources associated with it will be released.

17

Configuring a Near Cache for .NET Clients

In Coherence for .NET, the Near Cache is an `INamedCache` implementation that wraps the front cache and the back cache using a read-through/write-through approach. If the back cache implements the `IObservableCache` interface, then the Near Cache can use either the `Listen None`, `Listen Present`, `Listen All`, or `Listen Auto` strategy to invalidate any front cache entries that might have been changed in the back cache

For more information on Near Cache, the `Listen*` invalidation strategies, and the read-through/write-through approach, see *"Near Cache"* in *"Getting Started with Oracle Coherence"*.

The `Tangosol.Net.Cache.NearCache` class enables you to programatically instantiate and configure .NET Near Cache functionality. However, it is recommended that you use a cache configuration descriptor to configure the NearCache.

A typical Near Cache is configured to use a local cache (thread safe, highly concurrent, size-limited and/or auto-expiring local cache) as the front cache and a remote cache as a back cache. A Near Cache is configured by using the `near-scheme` element which has two child elements: `front-scheme` for configuring a local (front) cache and `back-scheme` for defining a remote (back) cache.

Configuring the Near Cache

A Near Cache is configured by using the `<near-scheme>` element in the `coherence-cache-config` file. This element has two required subelements: front-scheme for configuring a local (front-tier) cache and a back-scheme for defining a remote (back-tier) cache. While a local cache (`<local-scheme>`) is a typical choice for the front-tier, you can also use non-JVM heap based caches, (`<external-scheme>` or `<paged-external-scheme>`) or schemes based on Java objects (`<class-scheme>`).

The remote or back-tier cache is described by the `<back-scheme>` element. A back-tier cache can be either a distributed cache (`<distributed-scheme>`) or a remote cache (`<remote-cache-scheme>`). The `<remote-cache-scheme>` element enables you to use a clustered cache from outside the current cluster.

Optional subelements of `<near-scheme>` include `<invalidation-strategy>` for specifying how the front-tier and back-tier objects will be kept synchronized and `<listener>` for specifying a listener which will be notified of events occurring on the cache.

For an example configuration, see *"Sample Near Cache Configuration"*. The elements in the file are described in the `<near-scheme>` section.

Obtaining a Near Cache Reference with .NET

A reference to a configured Near Cache can then be obtained by name by using the CacheFactory class:

Example 17–1 Obtaining a Reference to a Near Cache

```
INamedCache cache = CacheFactory.GetCache("example-near-cache");
```

Cleaning up Resources Associated with a NearCache

Instances of all `INamedCache` implementations, including `NearCache`, should be explicitly released by calling the `INamedCache.Release()` method when they are no longer needed, to free up any resources they might hold.

If the particular `INamedCache` is used for the duration of the application, then the resources will be cleaned up when the application is shut down or otherwise stops. However, if it is only used for a period, the application should call its `Release()` method when finished using it.

Alternatively, you can leverage the fact that `INamedCache` extends `IDisposable` and that all cache implementations delegate a call to `IDisposable.Dispose()` to `INamedCache.Release()`. This means that if you need to obtain and release a cache instance within a single method, you can do so with a using block:

Example 17–2 Obtaining and Releasing a Reference to a Near Cache

```
using (INamedCache cache = CacheFactory.GetCache("my-cache"))
{
   // use cache as usual
}
```

After the using block terminates, `IDisposable.Dispose()` will be call on the `INamedCache` instance, and all resources associated with it will be released.

18

Continuous Query Cache for .NET Clients

While it is possible to obtain a point in time query result from a Coherence for .NET cache, and it is possible to receive events that would change the result of that query, Coherence for .NET provides a feature that combines a query result with a continuous stream of related events to maintain an up-to-date query result in a real-time fashion. This capability is called **Continuous Query**, because it has the same effect as if the desired query had zero latency *and* the query were being executed several times every millisecond!

Coherence for .NET implements the Continuous Query functionality by materializing the results of the query into a Continuous Query Cache, and then keeping that cache up-to-date in real-time using event listeners on the query. In other words, a Coherence for .NET Continuous Query is a cached query result that never gets out-of-date.

Uses of Continuous Query Caching

There are several different general use categories for Continuous Query Caching:

- It is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.
- It is ideal for situations in which an application repeats a particular query, and would benefit from always having instant access to the up-to-date result of that query.
- A Continuous Query Cache is analogous to a *materialized view*, and is useful for accessing and manipulating the results of a query using the standard INamedCache API, and receiving an ongoing stream of events related to that query.
- A Continuous Query Cache can be used in a manner similar to configuring a near cache for .NET clients, because it maintains an up-to-date set of data locally *where it is being used*, for example on a particular server node or on a client desktop; note that a Near Cache is invalidation-based, but the Continuous Query Cache actually maintains its data in an up-to-date manner.

An example use case is a trading system desktop, in which a trader's open orders and all related information must be maintained in an up-to-date manner at all times. By combining the Coherence*Extend functionality with Continuous Query Caching, an application can support literally tens of thousands of concurrent users.

Note: Continuous Query Caches are useful in almost every type of application, including both client-based and server-based applications, because they provide the ability to very easily and efficiently maintain an up-to-date local copy of a specified sub-set of a much larger and potentially distributed cached data set.

The Continuous Query Cache

The Coherence for .NET implementation of Continuous Query is found in the `Tangosol.Net.Cache.ContinuousQueryCache` class. This class, like all Coherence for .NET caches, implements the standard `INamedCache` interface, which includes the following capabilities:

- Cache access and manipulation using the `IDictionary` interface: `INamedCache` extends the standard `IDictionary` interface from the .NET Collections Framework, which is the same interface implemented by the .NET `Hashtable` class.
- Events for all objects modifications that occur within the cache: `INamedCache` extends the `IObservableCache` interface.
- Identity-based clusterwide locking of objects in the cache: `INamedCache` extends the `IConcurrentCache` interface.
- Querying the objects in the cache: `INamedCache` extends the `IQueryCache` interface.
- Distributed Parallel Processing and Aggregation of objects in the cache: `INamedCache` extends the `IInvocableCache` interface.

Since the ContinuousQueryCache implements the `INamedCache` interface, which is the same API provided by all Coherence for .NET caches, it is extremely simple to use, and it can be easily substituted for another cache when its functionality is called for.

Constructing a Continuous Query Cache

There are two items that define a Continuous Query Cache:

- The underlying cache that it is based on;
- A query of that underlying cache that produces the sub-set that the Continuous Query Cache will cache.

The underlying cache is any Coherence for .NET cache, including another Continuous Query Cache. A cache is usually obtained from a `CacheFactory`, which allows the developer to simply specify the name of the cache and have it automatically configured based on the application's cache configuration information; for example:

```
INamedCache cache = CacheFactory.GetCache("orders");
```

The query is the same type of query that would be used to query any other cache; for example:

```
Filter filter = new AndFilter(new EqualsFilter("getTrader", traderid),
                              new EqualsFilter("getStatus", Status.OPEN));
```

Normally, to query a cache, one of the methods from the `IQueryCache` is used; for examples, to obtain a snap-shot of all open trades for this trader:

```
ICollection setOpenTrades = cache.GetEntries(filter);
```

Similarly, the Continuous Query Cache is constructed from those same two pieces:

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
```

Cleaning up Resources Associated with a ContinuousQueryCache

Instances of all `INamedCache` implementations, including `ContinuousQueryCache`, should be explicitly released by calling the `INamedCache.Release()` method when they are no longer needed, to free up any resources they might hold.

If the particular `INamedCache` is used for the duration of the application, then the resources will be cleaned up when the application is shut down or otherwise stops. However, if it is only used for a period, the application should call its `Release()` method when finished using it.

Alternatively, you can leverage the fact that `INamedCache` extends `IDisposable` and that all cache implementations delegate a call to `IDisposable.Dispose()` to `INamedCache.Release()`. This means that if you need to obtain and release a cache instance within a single method, you can do so by using a using block:

Example 18–1 Obtaining and Releasing a Reference to a Continuous Query Cache

```
using (INamedCache cache = CacheFactory.GetCache("my-cache"))
{
   // use cache as usual
}
```

After the using block terminates, `IDisposable.Dispose()` will be call on the `INamedCache` instance, and all resources associated with it will be released.

Semi- and Fully-Materialized Views

When constructing a Continuous Query Cache, it is possible to specify that the cache should only keep track of the keys that result from the query, and obtain the values from the underlying cache only when they are asked for. This feature may be useful for creating a Continuous Query Cache that represents a very large query result set, or if the values are never or rarely requested. To specify that only the keys should be cached, use the constructor that allows the `IsCacheValues` property to be configured; for example:

Example 18–2 Caching Only the Keys in a Continuous Query Cache

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter,
false);
```

If necessary, the `IsCacheValues` property can also be modified after the cache has been instantiated; for example:

```
cacheOpenTrades.IsCacheValues = true;
```

IsCacheValues Property and Event Listeners

If the Continuous Query Cache has any standard (non-lite) event listeners, or if any of the event listeners are filtered, then the `IsCacheValues` property will automatically be set to `true`, because the Continuous Query Cache uses the locally cached values to filter events and to supply the old and new values for the events that it raises.

Listening to a Continuous Query Cache

Since the Continuous Query Cache is itself observable, it is possible for the client to place one or more event listeners onto it. For example:

Example 18–3 Placing a Listener on a Continuous Query Cache

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.AddCacheListener(listener);
```

Assuming some processing has to occur against every item that is already in the cache **and** every item added to the cache, there are two approaches. First, the processing could occur then a listener could be added to handle any later additions:

Example 18–4 Processing Data, then Placing the Listener

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
foreach (ICacheEntry entry in cacheOpenTrades.Entries)
    {
    // .. process the cache entry
    }
cacheOpenTrades.AddCacheListener(listener);
```

However, **that code is incorrect** because it allows events that occur in the split second after the iteration and before the listener is added to be missed! The alternative is to add a listener first, so no events are missed, and then do the processing:

Example 18–5 Placing the Listener, then Processing Data

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.AddCacheListener(listener);
foreach (ICacheEntry entry in cacheOpenTrades.Entries)
    {
    // .. process the cache entry
    }
```

However, it is possible that the same entry will show up in both an event an in the `IEnumerator`, and the events can be asynchronous, so the sequence of operations cannot be guaranteed.

The solution is to provide the listener during construction, and it will receive one event for each item that is in the Continuous Query Cache, whether it was there to begin with (because it was in the query) or if it was added during or after the construction of the cache:

Example 18–6 Providing the Listener During Continuous Query Cache Construction

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter,
listener);
```

Achieving a Stable Materialized View

The Continuous Query Cache implementation faced the same challenge: How to assemble an exact point-in-time snapshot of an underlying cache *while receiving a stream of modification events from that same cache*. The solution has several parts. First, Coherence for .NET supports an option for synchronous events, which provides a set of ordering guarantees. Secondly, the Continuous Query Cache has a two-phase implementation of its initial population that allows it to first query the underlying cache and then subsequently resolve all of the events that came in during the first

phase. Since achieving these guarantees of data visibility without any missing or repeated events is fairly complex, the Continuous Query Cache allows a developer to pass a listener during construction, thus avoiding exposing these same complexities to the application developer.

Support for Synchronous and Asynchronous Listeners

By default, listeners to the Continuous Query Cache will have their events delivered asynchronously. However, the Continuous Query Cache does respect the option for synchronous events as provided by the `CacheListenerSupport.ISynchronousListener` interface.

Making a Continuous Query Cache Read-Only

The Continuous Query Cache can be made into a read-only cache; for example:

Example 18–7 Making a Continuous Query Cache Read-Only

```
cacheOpenTrades.IsReadOnly = true;
```

A read-only Continuous Query Cache will not allow objects to be added to, changed in, removed from or locked in the cache.

Once a Continuous Query Cache has been set to read-only, it cannot be changed back to read/write.

19

Remote Invocation Service for .NET Clients

Coherence for .NET provides a *Remote Invocation Service* which allows execution of single-pass agents (called `IInvocable` objects) within the cluster-side JVM to which the client is connected. Agents are simply runnable application classes that implement the `IInvocable` interface. Agents can execute any arbitrary action and can use any cluster-side services (cache services, grid services, and so on) necessary to perform their work. The agent operations can also be stateful, which means that their state is serialized and transmitted to the grid nodes on which the agent is run.

Configuring and Using the Remote Invocation Service

A Remote Invocation Service is configured using the <remote-invocation-scheme> element in the cache configuration descriptor. For example:

Example 19–1 Configuring a Remote Invocation Service

```
<remote-invocation-scheme>
    <scheme-name>example-invocation</scheme-name>
    <service-name>ExtendTcpInvocationService</service-name>
    <initiator-config>
      <tcp-initiator>
        <remote-addresses>
          <socket-address>
            <address>localhost</address>
            <port>9099</port>
          </socket-address>
        </remote-addresses>
      </tcp-initiator>

      <outgoing-message-handler>
        <request-timeout>30s</request-timeout>
      </outgoing-message-handler>
    </initiator-config>
</remote-invocation-scheme>
```

A reference to a configured Remote Invocation Service can then be obtained by name by using the `CacheFactory` class:

Example 19–2 Obtaining a Reference to a Remote Invocation Service

```
IService service = CacheFactory.GetService("ExtendTcpInvocationService");
```

To execute an agent on the grid node to which the client is connected requires only one line of code:

Example 19–3 Executing an Agent on a Grid Node

```
IDictionary result = service.Query(new MyTask(), null);
```

The single result of the execution will be keyed by the local `Member`, which can be retrieved by calling `CacheFactory.ConfigurableCacheFactory.LocalMember`.

Note: Like cached value objects, all `IInvocable` implementation classes must be correctly registered in the POF context of the .NET application and cluster-side node to which the client is connected. As such, a Java implementation of the `IInvocable` task (a `com.tangosol.net.Invocable` implementation) must be created, compiled, and deployed on the cluster-side node. Note that the actual execution of the task is performed by the Java `Invocable` implementation and not the .NET `IInvocable` implementation.

See Chapter 14, "Configuration and Usage for .NET Clients" for additional details.

20

Special Considerations—Windows Forms Applications for .NET Clients

One of the features of the `INamedCache` interface is the ability to add cache listeners that receive events emitted by a cache as its contents change. These events are sent from the server and dispatched to registered listeners by a background thread.

The .NET Single-Threaded Apartment model prohibits windows form controls created by one thread from being updated by another thread. If one or more controls should be updated as a result of an event notification, you must ensure that any event handling code that must run as a response to a cache event is executed on the UI thread. The `WindowsFormsCacheListener` helper class allows end users to ignore this fact and to handle Coherence cache events (which are always raised by a background thread) as if they were raised by the UI thread. This class will ensure that the call is properly marshalled and executed on the UI thread.

Here is the sample of using this class:

Example 20–1 Marshalling and Executing a Call on the UI Thread

```
public partial class ContactInfoForm : Form
{
    ...
    listener = new WindowsFormsCacheListener(this);
    listener.EntryInserted += new CacheEventHandler(AddRow);
    listener.EntryUpdated  += new CacheEventHandler(UpdateRow);
    listener.EntryDeleted  += new CacheEventHandler(DeleteRow);
    ...
    cache.AddCacheListener(listener);
    ...
}
```

The `AddRow`, `UpdateRow` and `DeleteRow` methods are called in response to a cache event:

Example 20–2 Calling Methods in Response to a Cache Event

```
private void AddRow(object sender, CacheEventArgs args)
{
...
}

private void UpdateRow(object sender, CacheEventArgs args)
{
...
}
```

```
private void DeleteRow(object sender, CacheEventArgs args)
{
...
}
```

The `CacheEventArgs` parameter encapsulates the `IObservableCache` instance that raised the cache event; the `CacheEventType` that occurred; and the `Key`, `NewValue` and `OldValue` of the cached entry.

21

Special Considerations—Web Applications for .NET Clients

By default, session-state values and information are stored in memory within the ASP.NET process. ASP.NET also provides session-state providers that allow you to use a session-state server that keeps session data in a separate process, or you can persist session state data to a SQL database. However, with ASP.NET 2.0, you can create custom session-state providers that allow you to customize how session-state data is stored in your ASP.NET applications.

Coherence for .NET includes a custom `SessionStateStoreProvider` implementation that uses a Coherence cache to store session state. This makes Coherence for .NET the best solution for any large ASP.NET application running within a web farm. Other options in this scenario are to use the `StateServer`, which introduces a single point of failure for the whole web farm, or to use the `SqlServerStateProvider`, which theoretically can be clustered, but is extremely slow and scales only to a certain point. Also, unlike both `StateServer` and `SqlServerStateProvider`, the `CoherenceSessionProvider` supports `Session.End` event through cache events—only the `InProc` one supports this, but it cannot be used in a web farm environment.

The only requirement of the `CoherenceSessionStore` is that all objects stored in the session must be serializable (.NET serializable, not POF). This same requirement applies to both out-of-proc session stores provided by Microsoft, so modifying any existing ASP.NET 2.0 application that uses `StateServer` or `SqlServerStateProvider` to use the `CoherenceSessionStore` is as simple as adding the following to the `Web.config` file:

Example 21–1 Modifying an ASP.NET Application to use CoherenceSessionStore

```
<sessionState mode="Custom" customProvider="CoherenceSessionProvider"
timeout="20">
    <providers>
        <add name="CoherenceSessionProvider"
type="Tangosol.Web.CoherenceSessionStore, Coherence"
cacheName="dist-session-cache"/>
    </providers>
</sessionState>
```

Note that no code changes are required within the application itself.

`CoherenceSessionProvider` doesn't support calling `Session_OnEnd` event by default, so to configure the provider to send this event, the `sessionEndEnabled` attribute should be set to `true`:

Example 21–2 Adding Support for the Session _OnEnd Event

```
<sessionState mode="Custom" customProvider="CoherenceSessionProvider"
timeout="20">
  <providers>
    <add name="CoherenceSessionProvider" type="Tangosol.Web.CoherenceSessionStore,
Coherence" cacheName="dist-session-cache" sessionEndEnabled="true"/>
  </providers>
</sessionState>
```

If your Web application uses Coherence for .NET (either directly, by using the `CoherenceSessionProvider`, or both), you must remember to call `CacheFactory.shutdown()` when your application terminates. To make this easier, Coherence for .NET includes an HTTP module that will automatically call `CacheFactory.shutdown()` when your web application exits. To use the `CoherenceShutdownModule` simply include it in your `Web.config` file, as illustrated in Example 21–3:

Example 21–3 Adding Support for CoherenceShutdownModule

```
<httpModules>
  <add name="CoherenceShutdown" type="Tangosol.Web.CoherenceShutdownModule,
Coherence"/>
</httpModules>
```

22

Network Filters for .NET Clients

A network filter is a mechanism that allows transformation of data sent through TCP/IP sockets to be performed in a pluggable, layered fashion. Coherence for .NET supports custom filters, thus enabling users to modify the contents of the network traffic and is commonly used to add compression and encryption to data.

Custom Filters

To create a new filter, create a .NET class that implements the `Tangosol.IO.IWrapperStreamFactory` interface and optionally implements the `Tangosol.Util.IXmlConfigurable` interface. The `IWrapperStreamFactory` interface defines two methods:

Example 22–1 Methods on the IWrapperStreamFactory Interface

```
Stream GetInputStream(Stream stream);
Stream GetOutputStream(Stream stream);
```

that provide the input/output stream to be wrapped ("filtered") (on input—received message, or output—sending message) and expects a stream back that wraps the original stream. This method is called for each incoming and outgoing message.

Configuring Filters

There are two steps to configuring a filter. The first is to declare the filter in the `<filters>` XML element of the cache factory configuration file. This is illustrated in Example 22–2:

Example 22–2 Configuring a Filter

```
<coherence>
  <cluster-config>
    <filters>
      <filter>
        <filter-name>gzip</filter-name>
        <filter-class>Tangosol.Net.CompressionFilter, Coherence</filter-class>
      </filter>
    </filters>
  </cluster-config>
...
</coherence>
```

Note: GZip compression filter is supported in .NET framework version 2.0 or higher.

The second step is to attach the filter to one or more specific services. To specify the filter for a specific service, for example the `ExtendTcpCacheService` service, add a `<filter-name>` element to the `<use-filters>` element of the service declaration in the cache configuration file.

Example 22–3 Attaching a Filter to a Service

```
<remote-cache-scheme>
  <scheme-name>extend-direct</scheme-name>
  <service-name>ExtendTcpCacheService</service-name>
  <initiator-config>
    <tcp-initiator>
    ...
    </tcp-initiator>

    <outgoing-message-handler>
    ...
    </outgoing-message-handler>

    <use-filters>
      <filter-name>gzip</filter-name>
    </use-filters>

    ...
</remote-cache-scheme>
```

If the filter implements `IXmlConfigurable`, after instantiating the filter, Coherence will set the `Config` property with the following XML element:

Example 22–4 Setting the Config Property for a Filter that Implements IXmlConfigurable

```
<config>
  <param1>value1</param1>
  <param2>value2</param2>
</config>
```

23

Sample Windows Forms Application for .NET Clients

This is a step-by-step user guide that explains how to create a simple Windows Forms Application that uses the Coherence for .NET library.

General Instructions

Developing and configuring a Windows Forms Application that uses Coherence for .NET requires five basic steps:

1. Create a Windows Application Project
2. Add a Reference to the Coherence for .NET Library
3. Create an App.config File
4. Create Coherence for .NET Configuration Files
5. Create and Design the Application
6. Implement the Application

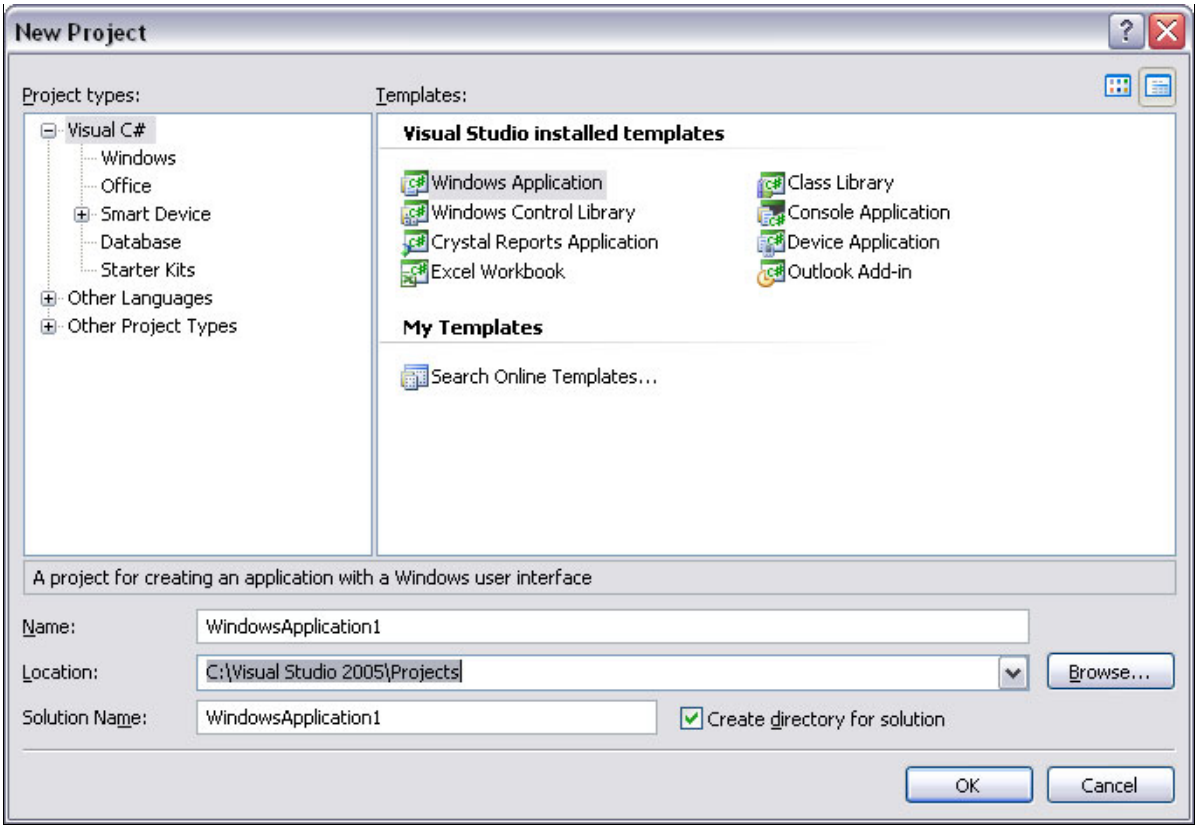
Create a Windows Application Project

To create a new Windows Application, follow these steps:

1. Go to the **File->New->Project...** tab in Visual Studio 2005.
2. In the **New Project** window choose the **Visual C#** project type and **Windows Application** template. Enter the name, location (full path where you want to store your application), and solution for your project.

   Figure 23–1 illustrates the New Project window with the name, location, and solution for the project.

Figure 23–1 New Project Window



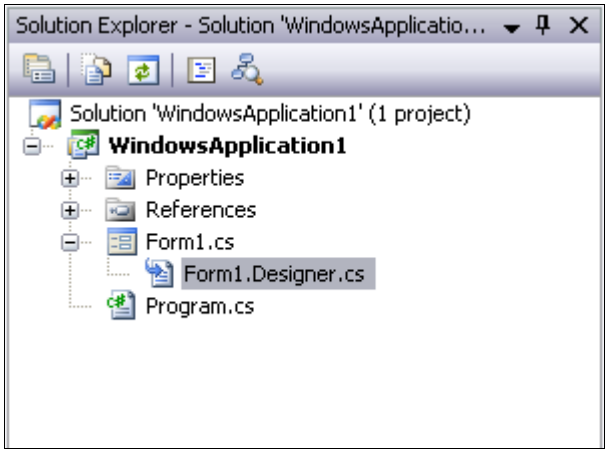
This figure is described in the text.

****

3. Click **OK**.

   Visual Studio should have created the following files: `Program.cs`, `Form1.cs` and `Form1.Designer.cs`. Figure 23–2 illustrates the **Solution Explorer** with the created project files

Figure 23–2 Solution Explorer with the Created Project Files



This figure is described in the text.

****

4. Rename these files if you want.

   In this example they have been renamed to `ContactCacheClient.cs`, `ContactForm.cs`, and `ContactForm.Designer.cs` respectively.

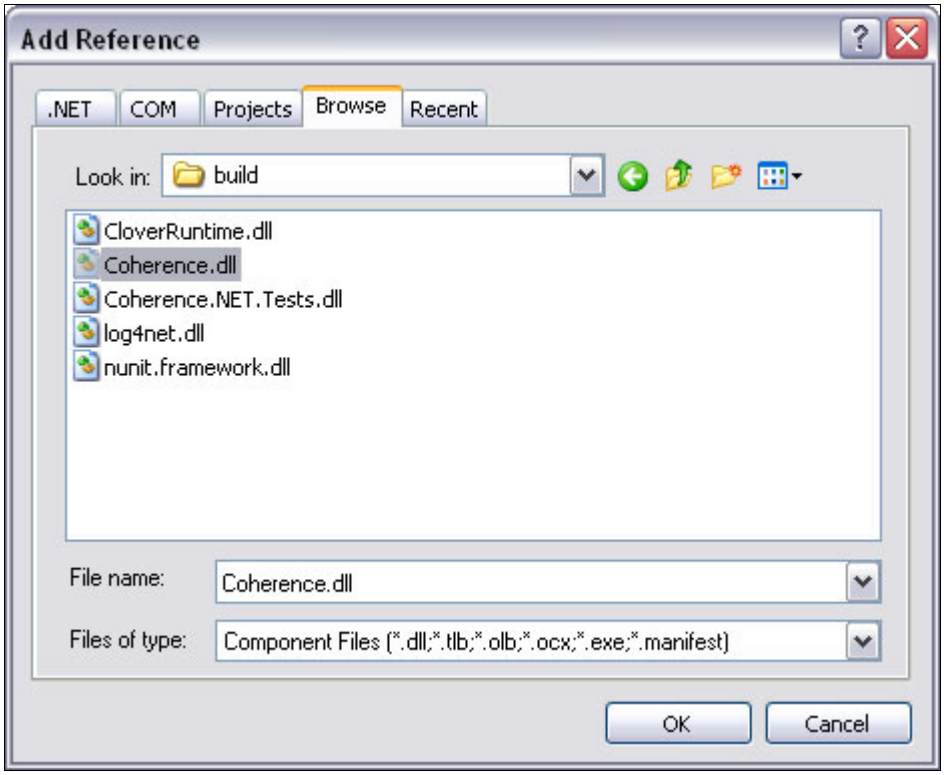
Add a Reference to the Coherence for .NET Library

To use the Coherence for .NET library in your .NET application, you must first add a reference to the `Coherence.dll` library.

Adding a reference to the `Coherence.dll` library:

1. In your project go to **Project->Add Reference...** or right click **References** in the **Solution Explorer** and choose **Add Reference...**
2. In the **Add Reference** window that appears choose the **Browse** tab and find the `Coherence.dll` library on your file system. Figure 23–3 illustrates the `.dll` files in the **Add Reference** window.

Figure 23–3 Add Reference Window



This figure is described in the text.

****

3. Click **OK**.

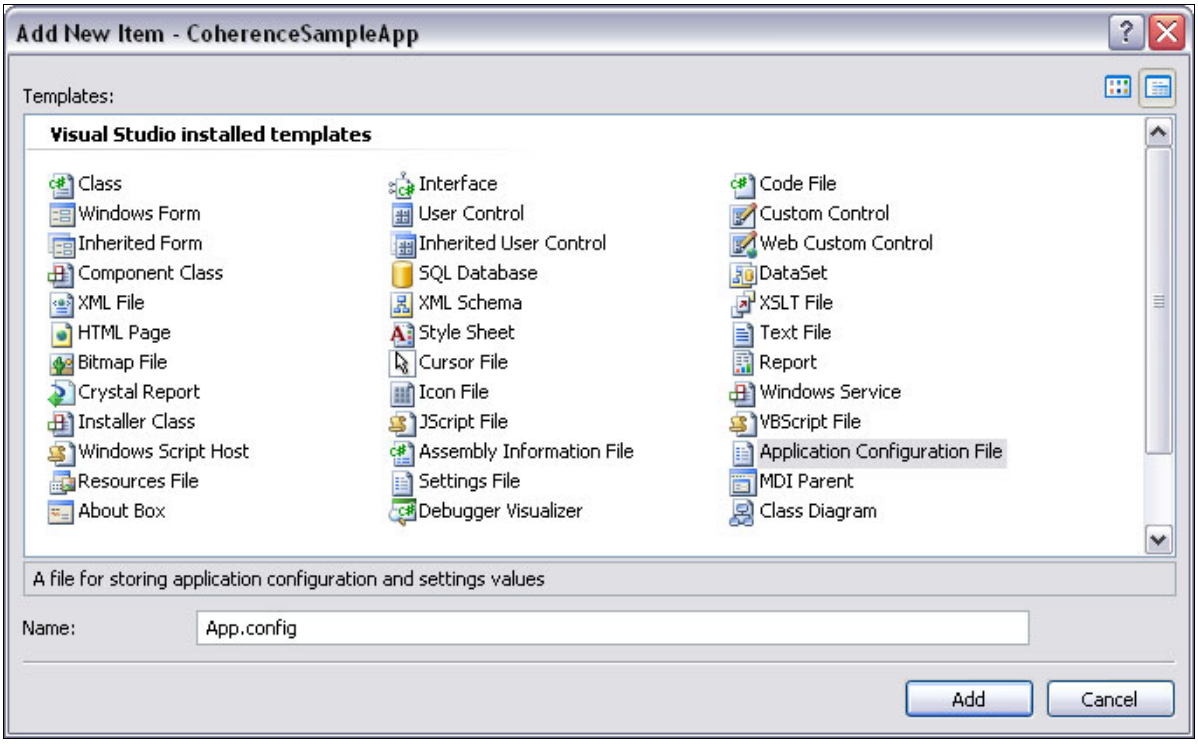
Create an App.config File

To correctly configure the Coherence for .NET library, you must create an `App.config` XML file that contains the appropriate file names for each configuration file used by the library.

1. Right-click the project in the **Solution Explorer** and choose the **Add->New Item...** tab.
2. In the **Add New Item** window select the Application Configuration File.

   Figure 23–4 illustrates the contents of the **Add New Item** window.

Figure 23–4 Add New Item Window



This figure is described in the text.

****

3. Click **OK**.

Example 23–1 illustrates a sample valid `App.config` configuration file.

Example 23–1 Sample App.config File

```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Util.CoherenceConfigHandler,
Coherence"/>
  </configSections>
  <coherence>
    <cache-factory-config>coherence.xml</cache-factory-config>
    <cache-config>cache-config.xml</cache-config>
    <pof-config>pof-config.xml</pof-config>
```

```
  </coherence>
</configuration>
```

In `<configSections>` you must specify a class that handles access to the Coherence for .NET configuration section.

Elements within the Coherence for .NET configuration section are:

- `cache-factory-config`—contains the path to a configuration descriptor used by the `CacheFactory` to configure the (IConfigurableCacheFactory and Logger) used by the `CacheFactory`.
- `cache-config`—contains the path to a cache configuration descriptor which contains the cache configuration described earlier (see "Configuring Coherence*Extend on the Client" on page 14-2). This cache configuration descriptor is used by the DefaultConfigurableCacheFactory.
- `pof-config`—contains the path to a configuration descriptor used by the `ConfigurablePofContext` to register custom types used by the application.

Create Coherence for .NET Configuration Files

Example 23–2 illustrates a sample `coherence.xml` configuration file

Example 23–2 Sample coherence.xml File for .NET

```
<?xml version="1.0"?>

<coherence xmlns="http://schemas.tangosol.com/coherence">
  <logging-config>
    <destination>ContactCache.log</destination>
    <severity-level>5</severity-level>
    <message-format>{date} &lt;{level}&gt; (thread={thread}):
{text}</message-format>
    <character-limit>8192</character-limit>
  </logging-config>
</coherence>
```

Example 23–3 illustrates a sample `cache-config.xml` configuration file.

Example 23–3 Sample cache-config.xml File for .NET

```
<?xml version="1.0"?>

<cache-config xmlns="http://schemas.tangosol.com/cache">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-contact-cache</cache-name>
      <scheme-name>extend-direct</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <remote-cache-scheme>
      <scheme-name>extend-direct</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <initiator-config>
        <tcp-initiator>
          <remote-addresses>
            <socket-address>
              <address>localhost</address>
```

```
            <port>9099</port>
          </socket-address>
        </remote-addresses>
      </tcp-initiator>

      <outgoing-message-handler>
        <request-timeout>30s</request-timeout>
      </outgoing-message-handler>

    </initiator-config>
  </remote-cache-scheme>
 </caching-schemes>
</cache-config>
```

Example 23–4 illustrates a sample `pof-config.xml` configuration file.

Example 23–4 Sample pof-config.xml File for .NET

```
<?xml version="1.0"?>

<pof-config xmlns="http://schemas.tangosol.com/pof">
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->

<include>assembly://Coherence/Tangosol.Config/coherence-pof-config.xml</include>

    <!-- include all application POF user types -->
    <user-type>
      <type-id>1001</type-id>
      <class-name>ContactCache.Windows.ContactInfo,
ContactCacheClient</class-name>
    </user-type>
  </user-type-list>
</pof-config>
```

Having created these configuration files, everything is now in place to connect to a Coherence cluster and perform all operations supported by Coherence for .NET.

Create and Design the Application

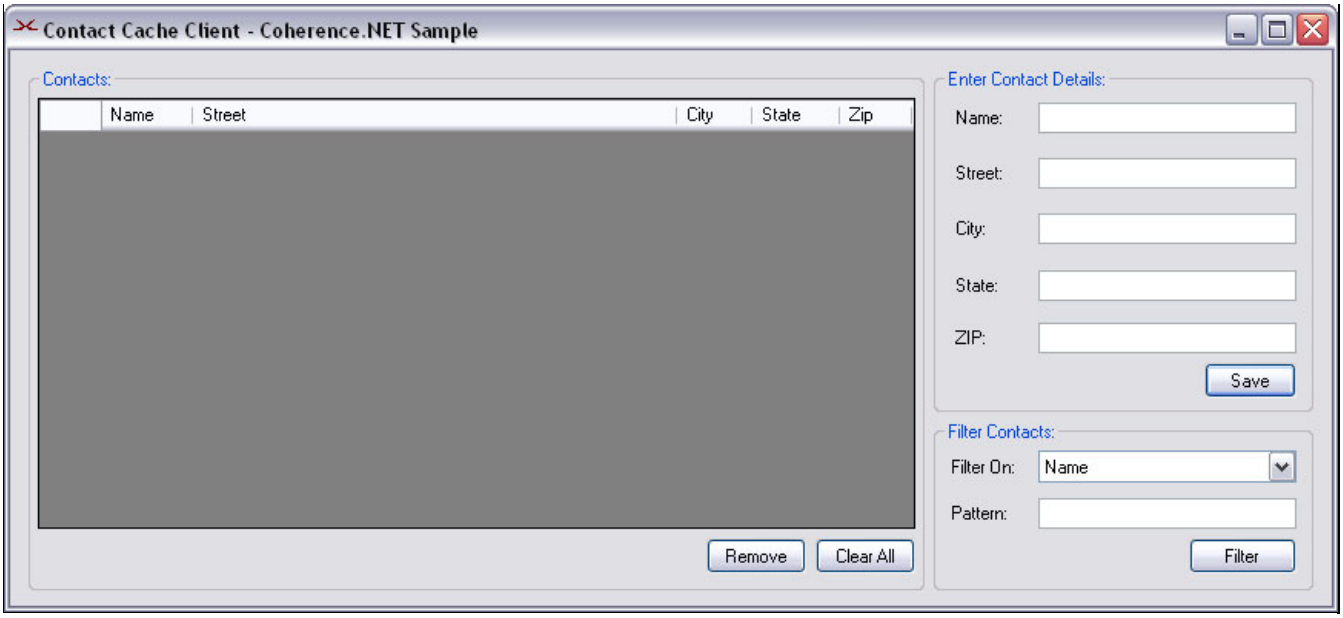
Next, you must add controls to your Windows form. This example shows you how to store objects into a `INamedCache`, read from the cache, query the cache, remove an item from the cache, and clear the cache. For this we're going to use buttons that will raise events when clicked, a couple of `TextBox` components for editing objects, and a `DataGridView` for displaying the current contents of a `INamedCache`. In this example we're going to work with just a `ContactInfo` user type, but a similar approach can be used with any other user defined type.

To add controls in your application follow these steps:

1. Go to **View->Toolbox**.
2. In the **Toolbox** window choose the controls you want to use and drag them on the **Windows** form.
3. For each control, right-click it, choose **Properties** tab, and set the necessary properties.

Figure 23–5 illustrates what the Contact Cache Info application UI should look after you have finished the previous steps.

Figure 23–5 Contact Cache Client UI



This figure is described in the text.

****

Implement the Application

The first step in the implementation of the example Windows application is to create a `ContactInfo` class that implements the `IPortableObject` interface.

Example 23–5 Sample Class that Implements IPortableObject

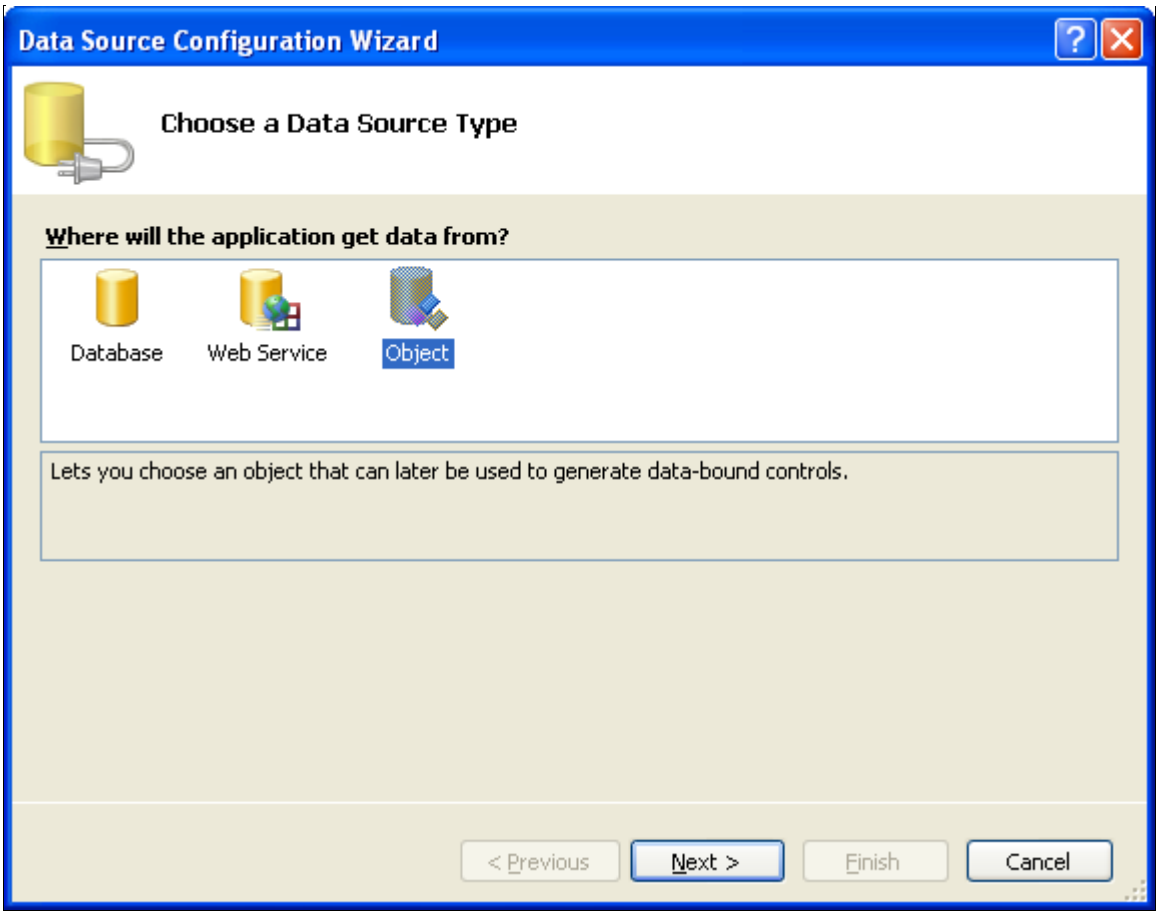
```
public class ContactInfo : IPortableObject
{
    private string name;
    private string street;
    private string city;
    private string state;
    private string zip;

    public ContactInfo()
    { }

    public ContactInfo(string name, string street, string city, string state,
string zip)
    {
        this.name   = name;
        this.street = street;
        this.city   = city;
        this.state  = state;
        this.zip    = zip;
    }

    public void ReadExternal(IPofReader reader)
    {
        name   = reader.ReadString(0);
        street = reader.ReadString(1);
```

```
        city   = reader.ReadString(2);
        state  = reader.ReadString(3);
        zip    = reader.ReadString(4);
    }

    public void WriteExternal(IPofWriter writer)
    {
        writer.WriteString(0, name);
        writer.WriteString(1, street);
        writer.WriteString(2, city);
        writer.WriteString(3, state);
        writer.WriteString(4, zip);
    }

    // property definitions omitted for brevity
}
```

Before the application can start handling events, we must bind the `DataGridView` control with a data source object:

1. In the **Toolbox** window choose the `BindingSource` object and drag it onto the form.
2. Set its properties. Enter `contactsBindingSource` into the **Name** field and then set its data source by clicking the arrow button on the right end of the **DataSource** field. In the drop down window choose **Add Project Data Source...** and the **Data Source Configuration Wizard** will appear. Chose **Object** and find the `ContactInfo` class in your project.

Figure 23–6 Using Data Source Wizard to Bind a Control to a Data Source

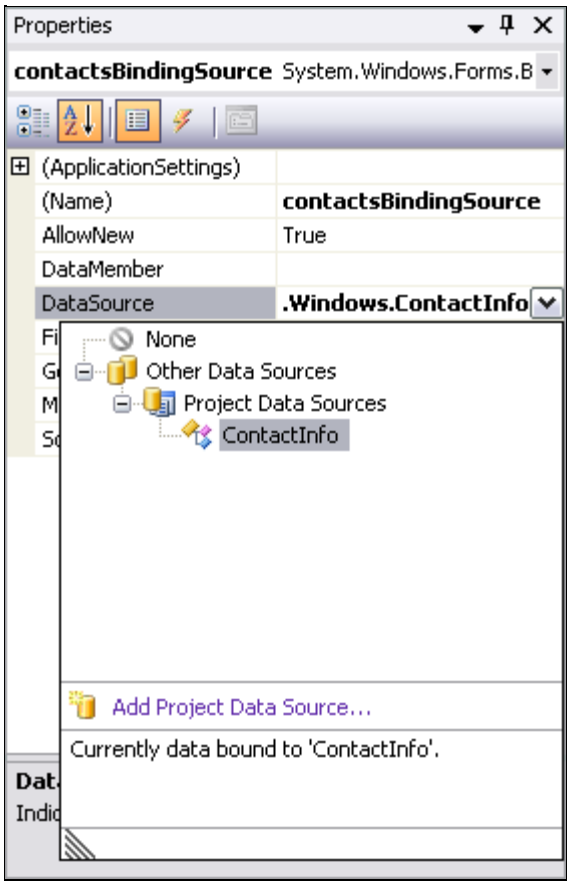


This figure is described in the text.

****

3. The final step is to bind the `DataGridView` control to the `contactBindingSource`. This is done by simply choosing the `contactsBindingSource` in the drop down window in the `DataSource` field of the `DataGridView` properties window. This is illustrated in Figure 23–7.

Figure 23–7 Choosing a Data Source to Bind to the Control



This figure is described in the text.

****

Now we have bound `contactsBindingSource` to our `DataGridView` control and all further interaction with the data, including navigating, sorting, filtering, and updating, is accomplished with calls to the BindingSource component. We also need `IFilter` and `CacheEventFilter` fields to manage filtering and a `WindowsFormsCacheListener` field used to ensure that any event handling code that must run as a response to a cache event is executed on the UI thread. For this to work, we'll have to delegate methods for each cache event we're handling and then register a listener with the cache by using the `AddCacheListener()` method. This is explained in more details in Chapter 20, "Special Considerations—Windows Forms Applications for .NET Clients". In the constructor, we will also obtain the `INamedCache` that we're using in the application by using the `CacheFactory.GetCache()` static method and initialize the **ComboBox** used for choosing the search attribute.

Example 23–6 Adding Listeners

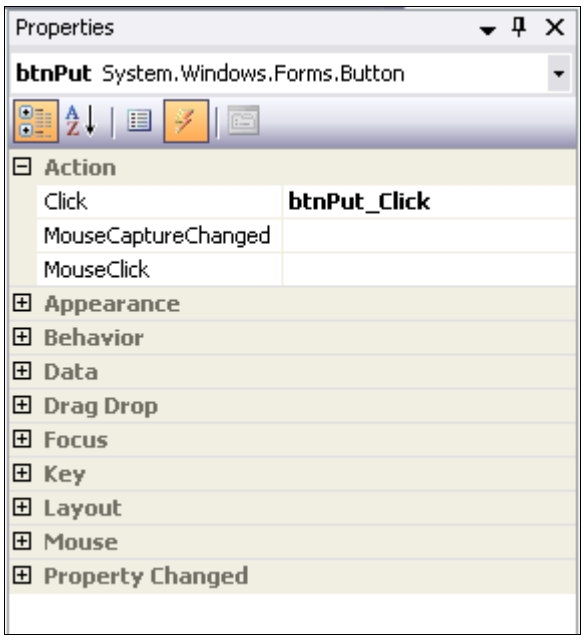
```
/// <summary>
/// Named cache.
/// </summary>
private INamedCache cache;

/// <summary>
/// Listener that allows end users to handle Coherence cache events,
```

```
/// which are always raised from a background thread.
/// </summary>
private WindowsFormsCacheListener listener;

/// <summary>
/// Evaluate the specified extracted value.
/// </summary>
private IFilter filter;

/// <summary>
/// Wrapper filter, used by listeners.
/// </summary>
private CacheEventFilter cacheEventFilter;

/// <summary>
/// Search pattern.
/// </summary>
private string pattern;

/// <summary>
/// Default constructor.
/// </summary>
public ContactForm()
{
    listener = new WindowsFormsCacheListener(this);
    listener.EntryInserted += new CacheEventHandler(AddRow);
    listener.EntryUpdated  += new CacheEventHandler(UpdateRow);
    listener.EntryDeleted  += new CacheEventHandler(DeleteRow);

    cache = CacheFactory.GetCache("dist-contact-cache");
    cache.AddCacheListener(listener);

    InitializeComponent();
    InitializeComboBox();
}

/// <summary>
/// Initialize <b>ComboBox</b> with attribute names.
/// </summary>
/// <remarks>
/// Choosing attribute from the ComboBox allows to search for given
/// pattern in choosen entry attribute inside the named cache.
/// </remarks>
private void InitializeComboBox()
{
    cmbAttribute.Items.Add("Name");
    cmbAttribute.Items.Add("Street");
    cmbAttribute.Items.Add("City");
    cmbAttribute.Items.Add("State");
    cmbAttribute.Items.Add("Zip");

    cmbAttribute.SelectedIndex = 0;
}
```

As with any other Windows application, most of the remaining implementation has to do with event handling. Since each component in the Windows form can raise an event, event handlers must be created to handle each event. Event handlers in Visual Studio can be added to your application by following these steps:

1. Right-click the Window component for which you'd like to implement an event handler and choose Properties.
2. In the upper toolbar of the **Properties** window, select the lighting button and all events that the component can raise will be displayed.

Figure 23–8 Properties Window



This figure is described in the text.

****

3. Choose the event you want to handle and double-click it. Visual Studio will add the necessary code to your application to enable you to handle the event. Next, you must implement the empty event handler method.

Example 23–7 illustrates the event code in the sample Windows application:

Example 23–7 Adding Events

```
/// <summary>
/// Load form event handler.
/// </summary>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void ContactForm_Load(object sender, EventArgs e)
{
    RefreshContactsGrid(true);
}
/// <summary>
/// Closed form event handler.
/// </summary>
/// <remarks>
/// Removes the event handlers.
/// </remarks>
```

```
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void ContactForm_FormClosed(object sender, FormClosedEventArgs e)
{
    cache.RemoveCacheListener(listener, cacheEventFilter);
}

/// <summary>
/// Enter cell event handler for the <b>addressDataGridView</b>.
/// </summary>
/// <remarks>
/// Refreshes the <b>TextBox</b>es with data from selected
/// <b>addressDataGridView</b> row.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void addressDataGridView_CellEnter(object sender,
DataGridViewCellEventArgs e)
{
    DataGridViewCellCollection cells = addressDataGridView.CurrentRow.Cells;

    txtName.Text   = (string) cells[0].Value;
    txtStreet.Text = (string) cells[1].Value;
    txtCity.Text   = (string) cells[2].Value;
    txtState.Text  = (string) cells[3].Value;
    txtZip.Text    = (string) cells[4].Value;
}

/// <summary>
/// Click event handler for <b>Put</b> button.
/// </summary>
/// <remarks>
/// Stores the <see cref="ContactInfo"/> data entered in
/// <b>TextBox</b>es into the INamedCache.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void btnPut_Click(object sender, EventArgs e)
{
    String name  = txtName.Text;
    ContactInfo contact = new ContactInfo(txtName.Text,
                                          txtStreet.Text,
                                          txtCity.Text,
                                          txtState.Text,
                                          txtZip.Text);
    cache.Insert(name, contact);
}
```

```
/// <summary>
/// Click event handler for the <b>Remove</b> button.
/// </summary>
/// <remarks>
/// Removes the <see cref="ContactInfo"/> mapped by the current
/// Name <b>TextBox</b> value. If there is no such entry in the
/// <b>INamedCache</b>, a simple warning box is displayed.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void btnRemove_Click(object sender, EventArgs e)
{
    cache.Remove(txtName.Text);
    ResetTextBoxes();
}

/// <summary>
/// Click event handler for the <b>Clear</b> button.
/// </summary>
/// <remarks>
/// Clears the <b>INamedCache</b>.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void btnClear_Click(object sender, EventArgs e)
{
    cache.RemoveCacheListener(listener, cacheEventFilter);
    cache.Clear();
    cache.AddCacheListener(listener, cacheEventFilter, false);

    contactsBindingSource.Clear();
    ResetTextBoxes();
}

/// <summary>
/// Click event handler for <b>Refresh</b> button.
/// </summary>
/// <remarks>
/// Refreshes the <b>addressDataGridView</b>, filtering named cache
/// entries by a given attribute and string pattern. If empty string
/// is provided as a pattern all entries in the named cache will be
/// accounted and displayed.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void btnRefresh_Click(object sender, EventArgs e)
{
    string newPattern = txtPattern.Text;
```

```
    string attribute = (string) cmbAttribute.SelectedItem;

    if (!newPattern.Equals(pattern))
    {
        pattern = newPattern;
        cache.RemoveCacheListener(listener, cacheEventFilter);

        if (pattern != String.Empty)
        {
            IValueExtractor extractor = new ReflectionExtractor("get" +
attribute);
            filter = new LikeFilter(extractor, pattern, '\\', false);
            cacheEventFilter = new
CacheEventFilter(CacheEventFilter.CacheEventMask.All
                                                    |
CacheEventFilter.CacheEventMask.UpdatedEntered
                                                    |
CacheEventFilter.CacheEventMask.UpdatedLeft,
                                                    filter);
        }
        else
        {
            filter = null;
            cacheEventFilter = null;
        }
        cache.AddCacheListener(listener, cacheEventFilter, false);
    }
    RefreshContactsGrid(true);
}

/// <summary>
/// Click event handler for <b>SelectIndexChanged</b> event.
/// </summary>
/// <remarks>
/// Resets the pattern string to Refresh button click event
/// handler would work properly.
/// </remarks>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="e">
/// An <b>EventArgs</b> that contains no event data.
/// </param>
private void cmbAttribute_SelectedIndexChanged(object sender, EventArgs e)
{
    pattern = "";
}
```

We also have to write cache event handlers, as delegated in the constructor. This is illustrated in Example 23–8:

Example 23–8 Adding Cache Event Handlers

```
/// <summary>
/// Event handler for  <see cref="ICacheListener.EntryInserted"/>
/// event.
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="args">
```

```
/// An <see cref="CacheEventArgs"/>.
/// </param>
private void AddRow(object sender, CacheEventArgs args)
{
    contactsBindingSource.Add(args.NewValue);
}

/// <summary>
/// Event handler for  <see cref="ICacheListener.EntryUpdated"/>
/// event.
/// </summary>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="args">
/// An <see cref="CacheEventArgs"/>.
/// </param>
public void UpdateRow(object sender, CacheEventArgs args)
{
    int index = contactsBindingSource.IndexOf(args.OldValue);
    if (index < 0)
    {
        // updated entered
        contactsBindingSource.Add(args.NewValue);
    }
    else
    {
        if (SatisfiesFilter(args.NewValue))
        {
            contactsBindingSource[index] = args.NewValue;
        }
        else
        {
            contactsBindingSource.RemoveAt(index);
        }
    }
}

/// <summary>
/// Event handler for <see cref="ICacheListener.EntryDeleted"/>
/// event.
/// </summary>
/// <param name="sender">
/// The source of the event.
/// </param>
/// <param name="args">
/// An <see cref="CacheEventArgs"/>.
/// </param>
public void DeleteRow(object sender, CacheEventArgs args)
{
    contactsBindingSource.Remove(args.OldValue);
}
```

Example 23–9 illustrates helper methods used by the event handlers in the previous example:

Example 23–9 Adding Helper Methods for Event Handlers

```
/// <summary>
/// Resets all of the text boxes on the form.
```

```
/// </summary>
private void ResetTextBoxes()
{
    txtName.Text   = "";
    txtStreet.Text = "";
    txtCity.Text   = "";
    txtState.Text  = "";
    txtZip.Text    = "";
}

/// <summary>
/// Initialize <b>ComboBox</b> with attribute names.
/// </summary>
/// <remarks>
/// Choosing attribute from the ComboBox allows to search for given
/// pattern in choosen entry attribute inside the named cache.
/// </remarks>
private void InitializeComboBox()
{
    cmbAttribute.Items.Add("Name");
    cmbAttribute.Items.Add("Street");
    cmbAttribute.Items.Add("City");
    cmbAttribute.Items.Add("State");
    cmbAttribute.Items.Add("Zip");

    cmbAttribute.SelectedIndex = 0;
}

/// <summary>
/// Queries the object with specified filter criteria.
/// </summary>
/// <param name="obj">
/// An object to which the test is applied.
/// </param>
/// <returns>
/// <b>true</b> if the test passes, <b>false</b> otherwise.
/// </returns>
private bool SatisfiesFilter(object obj)
{
    IFilter clientFilter = new LikeFilter(new ReflectionExtractor((string)
cmbAttribute.SelectedItem),
                                          pattern, '\\', false);
    return clientFilter.Evaluate(obj);
}

/// <summary>
/// Refreshes the contacts table.
/// </summary>
/// <param name="updateContacts">
/// Flag specifying whether to query against cache to get
/// the most recent data or not.
/// </param>
private void RefreshContactsGrid(bool updateContacts)
{
    if (updateContacts)
    {
        RefreshContacts();
    }
    contactsBindingSource.ResetBindings(false);
}
```

```
/// <summary>
/// Refreshes the contacts table with the most recent data within the
/// cache.
/// </summary>
private void RefreshContacts()
{
    contactsBindingSource.Clear();
    ICollection cacheEntries = (filter == null ? cache.Values :
cache.GetEntries(filter));
    foreach (object entry in cacheEntries)
    {
        if (entry is DictionaryEntry)
        {
            contactsBindingSource.Add(((DictionaryEntry) entry).Value);
        }
        else
        {
            contactsBindingSource.Add(entry);
        }
    }
}
```

24

Sample Web Application for .NET Clients

This chapter provides step-by-step instructions to create a simple Windows ASP.NET Web application that uses the Coherence for .NET library.

General Instructions

Developing and configuring a Windows ASP.NET web application that uses Coherence for .NET requires six basic steps:

1. Create an ASP.NET Project
2. Add a Reference to the Coherence for .NET Library
3. Configure the Web.config File
4. Create Coherence for .NET Configuration Files
5. Create the Web Form
6. Implement the Web Application.

The following sections describe each of these steps in detail.

Create an ASP.NET Project

To create a new ASP.NET web application, follow these steps:

1. Choose **File->New->Web site** in Visual Studio 2005.
2. Under the "**Templates**", select "**ASP.NET Web Site**".
3. Select the language that you are most familiar with.
4. Select the location (type and full path) where you want to store your application.

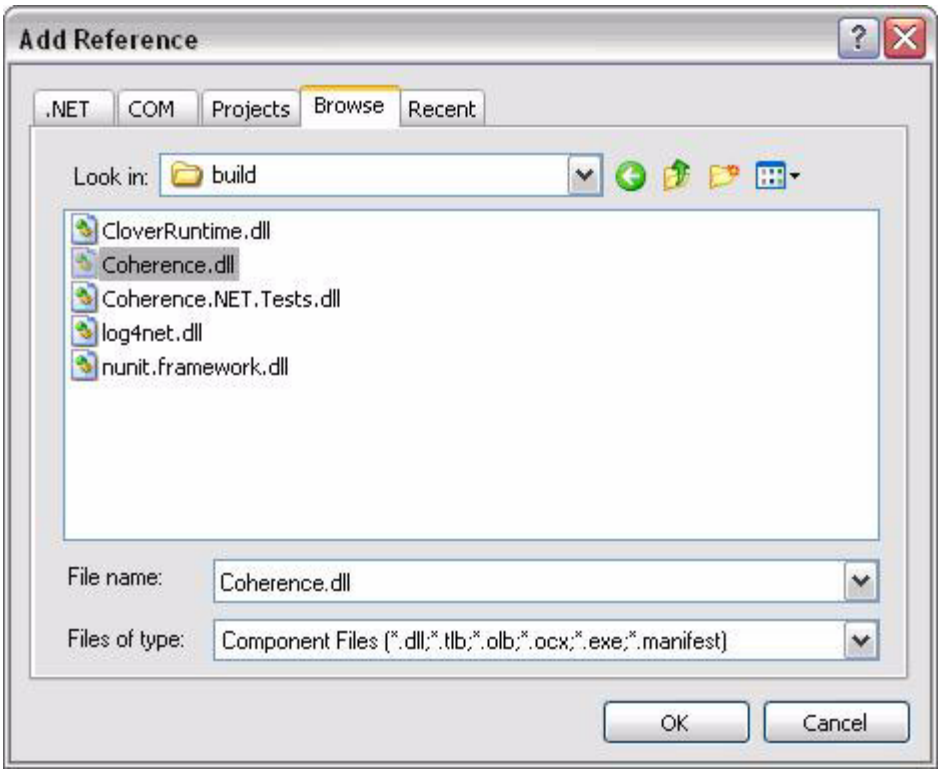
Click the **OK** button to generate a new solution and empty ASP.NET application.

Add a Reference to the Coherence for .NET Library

To use the Coherence for .NET library in your .NET application, you first need to add a reference to the `Coherence.dll` library:

1. In your project go to **Project->Add Reference...** or right click **References** in the **Solution Explorer** and choose **Add Reference...**.
2. In the **Add Reference** window that appears, choose the **Browse** tab and find the `Coherence.dll` library on your file system.

Figure 24–1 Coherence.dll File in the Add Reference Window



This figure is described in the text.

****

3. Click **OK**.

Configure the Web.config File

To correctly configure the Coherence for .NET library, you must configure the `Web.config` XML file with the appropriate file names for each configuration file used by the Coherence for .NET library. Example 24–2 illustrates a valid `Web.config` configuration file:

Example 24–1 Sample Web.config Configuration File

```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Util.CoherenceConfigHandler,
Coherence"/>
  </configSections>

  <coherence>
    <cache-factory-config>coherence.xml</cache-factory-config>
    <cache-config>cache-config.xml</cache-config>
    <pof-config>pof-config.xml</pof-config>
  </coherence>

  <system.web>
    <httpModules>
```

```
      <add name="CoherenceShutdown" type="Tangosol.Web.CoherenceShutdownModule,
Coherence"/>
    </httpModules>
    <compilation debug="true"/>
    <authentication mode="Windows"/>
  </system.web>
</configuration>
```

In the `<configSections>` you must specify a class that handles access to the Coherence for .NET configuration section.

Elements within the Coherence for .NET configuration section are:

- `cache-factory-config`—contains the path to a configuration descriptor used by the `CacheFactory` to configure the (IConfigurableCacheFactory and Logger) used by the `CacheFactory`.
- `cache-config`—contains the path to a cache configuration descriptor which contains the cache configuration described earlier (see "Configuring Coherence*Extend on the Client" on page 14-2). This cache configuration descriptor is used by the DefaultConfigurableCacheFactory.
- `pof-config`—contains the path to a configuration descriptor used by the `ConfigurablePofContext` to register custom types used by the application.

Create Coherence for .NET Configuration Files

Example 24–2 illustrates a sample `coherence.xml` configuration file:

Example 24–2 Sample coherence.xml Configuration File

```
<?xml version="1.0"?>

<coherence xmlns="http://schemas.tangosol.com/coherence">
  <logging-config>
    <destination>stderr</destination>
    <severity-level>5</severity-level>
    <message-format>{date} &lt;{level}&gt; (thread={thread}):
{text}</message-format>
    <character-limit>8192</character-limit>
  </logging-config>
</coherence>
```

Example 24–3 illustrates a sample `cache-config.xml` configuration file:

Example 24–3 Sample cache-config.xml Configuration File

```
<?xml version="1.0"?>

<cache-config xmlns="http://schemas.tangosol.com/cache">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-contact-cache</cache-name>
      <scheme-name>extend-direct</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <remote-cache-scheme>
      <scheme-name>extend-direct</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
```

```
      <initiator-config>
        <tcp-initiator>
          <remote-addresses>
            <socket-address>
              <address>localhost</address>
              <port>9099</port>
            </socket-address>
          </remote-addresses>
          <connect-timeout>5s</connect-timeout>
        </tcp-initiator>

        <outgoing-message-handler>
          <request-timeout>30s</request-timeout>
        </outgoing-message-handler>

      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

Example 24–4illustrates a sample `pof-config.xml` configuration file:

Example 24–4 Sample pof-config.xml Configuration File

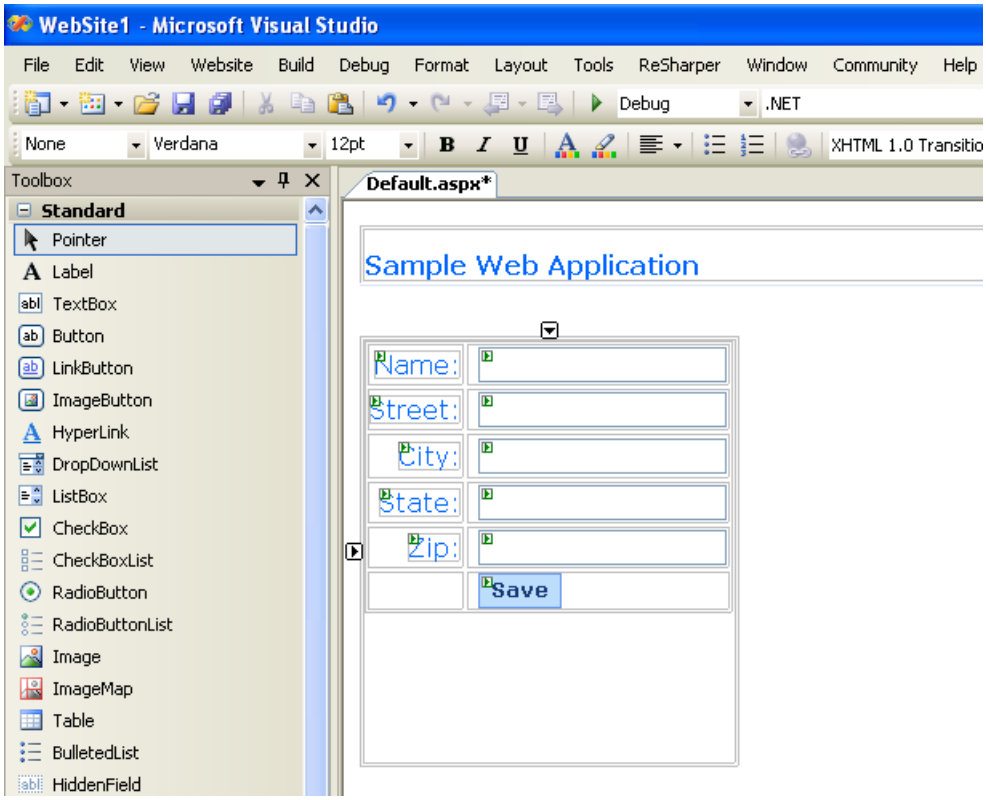
```
<?xml version="1.0"?>

<pof-config xmlns="http://schemas.tangosol.com/pof">
  <user-type-list>
    <!-- include all "standard" Coherence POF user types -->

<include>assembly://Coherence/Tangosol.Config/coherence-pof-config.xml</include>

    <!-- include all application POF user types -->
    <user-type>
      <type-id>1001</type-id>
      <class-name>ContactCache.Web.ContactInfo</class-name>
    </user-type>
  </user-type-list>
</pof-config>
```

Having creating these configuration files, everything is now in place to connect to a Coherence cluster and perform all operations supported by Coherence for .NET.

Create the Web Form

Switch to the **Design** tab for the `Default.aspx` page and from the **Toolbox** pane add the appropriate controls by dragging and dropping them on the page. You will need **TextBox** controls for the `Name`, `Street`, `City`, `State`, and `Zip` fields and corresponding label controls for each. This is illustrated in Figure 24–2.

Figure 24–2 Adding Controls for the .aspx Page

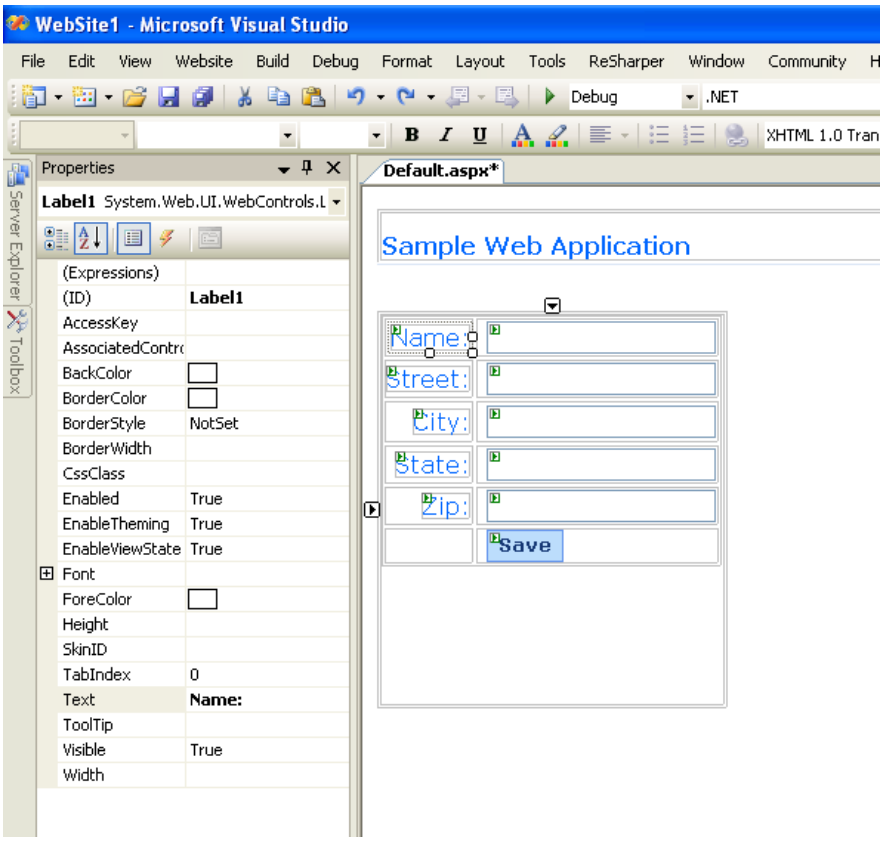


This figure is illustrated in the text.

****

After placing them on the page, you should change the **ID** and **Text** property for each control. As we won't be using labels in the code, you can leave their **ID** property values as generated, and just put appropriate labels in the **Text** property. You should name the **ID** and **TextBox** controls `txtName`, `txtStreet`, and so on. Add one button and rename its **ID** to `btnSave` and Text property to **Save**. This is illustrated in Figure 24–3.

Figure 24–3 Changing IDs and Properties for Data Controls

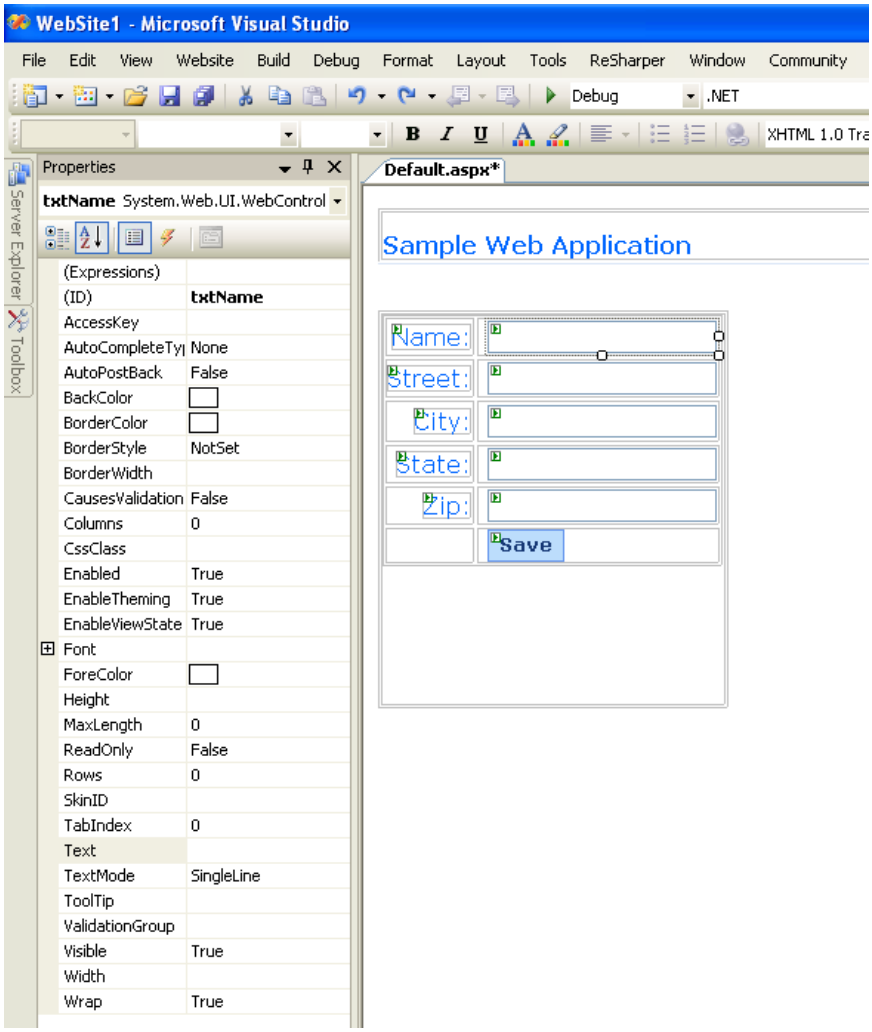


This figure is described in the text.

****

Add one button and rename its **ID** to `btnClear` and **Text** property to `Clear`. This is illustrated in Figure 24–4

Figure 24–4 Adding a "Clear" Button to the Application

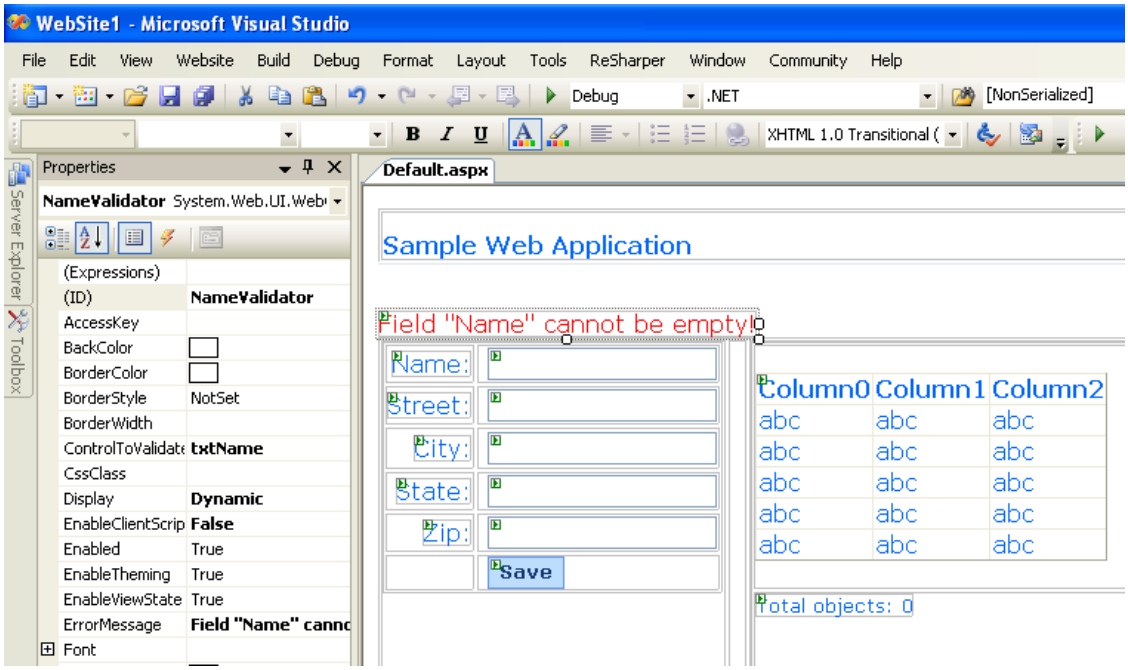


This figure is described in the text.

****

Add `label` and rename its ID to `lblTotal`. This label will be used to display the cache size. We have to add a `RequiredFieldValidator` from the **Validation** list of controls on the **Toolbox** pane and set its properties. This is illustrated in Figure 24–5:

Figure 24–5 Adding a Field Validator and Setting its Properties



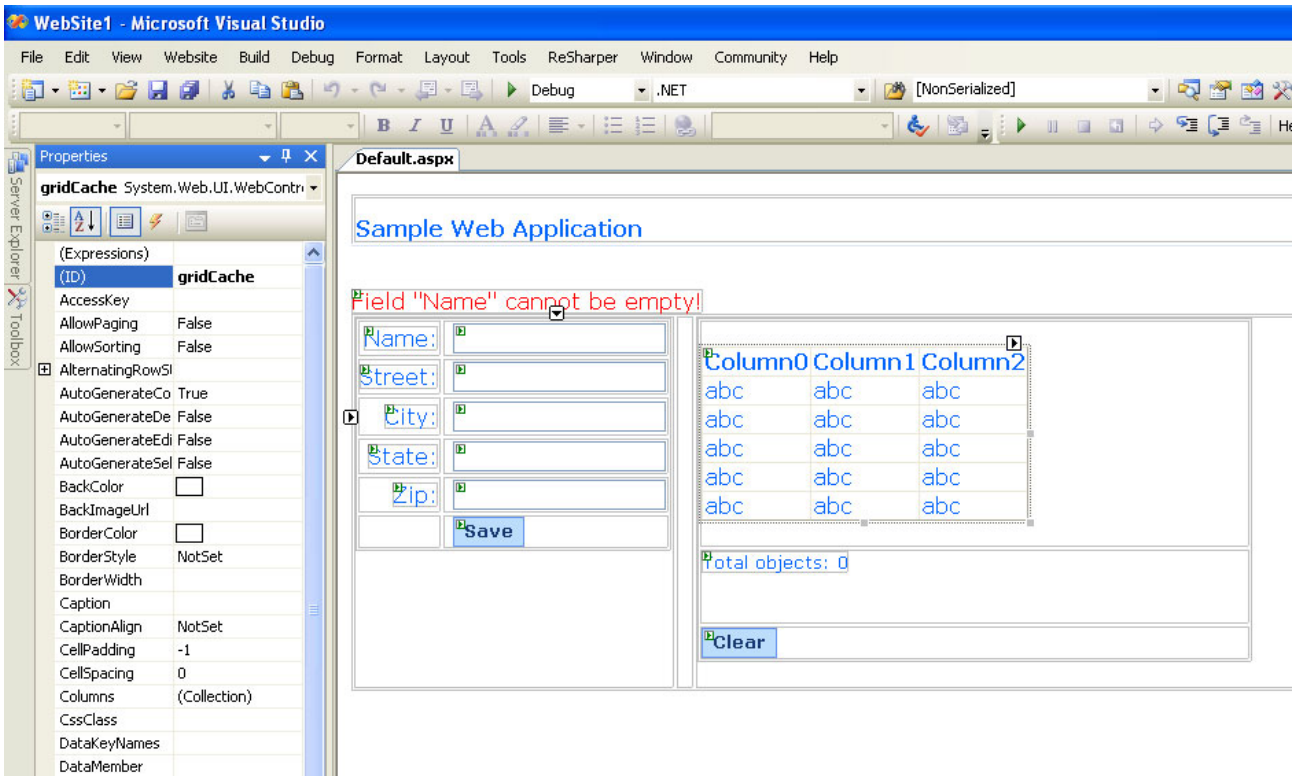
This figure is described in the text.

****

Please note that `ControlToValidate` property is set to the **txtName** control.

From the **Data** list of controls on the **Toolbox** pane, add a **GridView** control and an `ObjectDataSource` (named **dsContact**). This is illustrated in Figure 24–6.

Figure 24–6 Adding a GridView Control and an ObjectDataSource



This figure is described in the text.

****

Example 24–5 illustrates code for the `GridView` control source:

Example 24–5 Code for the GridView Data Control

```
<asp:GridView ID="gridCache" runat="server" DataSourceID="dsContact"
AutoGenerateColumns="False" Font-Names="Verdana">
  <Columns>
    <asp:TemplateField>
      <ItemStyle Font-Size="Small"/>
      <ItemTemplate>
        <asp:HyperLink Text="[Remove]" ID="HyperLink1" runat="server"
NavigateUrl='<%# "?removeKey=" +
              DataBinder.Eval(Container.DataItem, "Name").ToString() %>'/>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:TemplateField HeaderText="Name">
      <HeaderStyle BackColor="#DCE7F7"/>
      <ItemTemplate>
        <asp:HyperLink runat="server" NavigateUrl='<%# "?getKey=" +
DataBinder.Eval(Container.DataItem, "Name").ToString() %>'>
          <%# DataBinder.Eval(Container.DataItem, "Name") %>
        </asp:HyperLink>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:BoundField DataField="Street" HeaderText="Street">
      <HeaderStyle BackColor="#DCE7F7"/>
    </asp:BoundField>
    <asp:BoundField DataField="City" HeaderText="City">
```

```
      <HeaderStyle BackColor="#DCE7F7"/>
    </asp:BoundField>
    <asp:BoundField DataField="State" HeaderText="State">
      <HeaderStyle BackColor="#DCE7F7"/>
    </asp:BoundField>
    <asp:BoundField DataField="Zip" HeaderText="Zip">
      <HeaderStyle BackColor="#DCE7F7"/>
    </asp:BoundField>
  </Columns>
</asp:GridView>
```

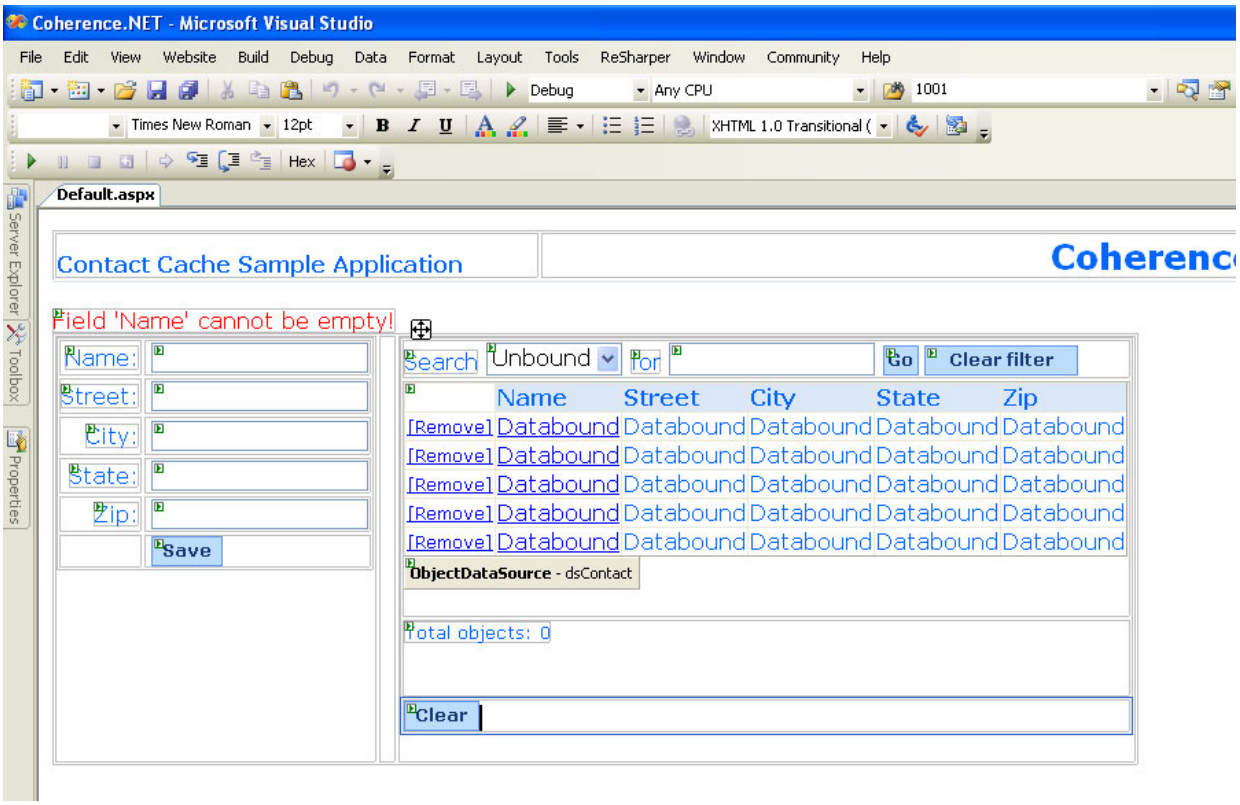
Example 24–6 illustrates the `ObjectDataSource` code.

Example 24–6 ObjectDataSource Code

```
<asp:ObjectDataSource ID="dsContact" runat="server" SelectMethod="GetData"
    TypeName="ContactCache.Web.ContactInfoDataSource"
</asp:ObjectDataSource>
```

Now, let's add a **Search** pane by dragging and dropping a few labels, one **DropDownList** for a filter column, and a **TextBox** for filter criteria. This is illustrated in Figure 24–7.

Figure 24–7 Search Pane



This figure is illustrated in the text.

****

Implement the Web Application

Global.asax File

Example 24–7 illustrates the `Global.asax` file which redirects the user to an error page if an exception occurs. A Coherence for .NET `INamedCache` instance is retrieved by using the `CacheFactory.GetCache(...)` API call. Once it is obtained, it is stored in the `Application` state.

Example 24–7 Redirecting a User to an Error Page

```
<%@ Application Language="C#" %>

<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {
    try
        {
        Application["contactCache"] = CacheFactory.GetCache("dist-contact-cache");
        }
    catch
        {
        }
    }

    void Application_End(object sender, EventArgs e)
    {
        CacheFactory.Log("Application terminated.", CacheFactory.LogLevel.Info);
        INamedCache contactCache = Application["contactCache"] as INamedCache;
        if (contactCache != null)
        {
           contactCache.Release();
        }
    }

    void Application_Error(object sender, EventArgs e)
    {
        Server.Transfer("ConnectionError.html");
    }
</script>
```

Business Object Definition

Example 24–8 illustrates the definition of the `ContactInfo` business object.

Example 24–8 Sample Business Object Definition File

```
public class ContactInfo : IPortableObject
{
    private string name;
    private string street;
    private string city;
    private string state;
    private string zip;

    public ContactInfo()
    { }
```

```
    public ContactInfo(string name, string street, string city, string state,
string zip)
    {
        this.name   = name;
        this.street = street;
        this.city   = city;
        this.state  = state;
        this.zip    = zip;
    }

    public void ReadExternal(IPofReader reader)
    {
        name   = reader.ReadString(0);
        street = reader.ReadString(1);
        city   = reader.ReadString(2);
        state  = reader.ReadString(3);
        zip    = reader.ReadString(4);
    }

    public void WriteExternal(IPofWriter writer)
    {
        writer.WriteString(0, name);
        writer.WriteString(1, street);
        writer.WriteString(2, city);
        writer.WriteString(3, state);
        writer.WriteString(4, zip);
    }
}
```

Service Layer Implementation

Example 24–9 illustrates a class that will provide data to the data bind control. It must have a public `GetData()` method that will return an `ICollection` of data to the data bind control:

Example 24–9 Providing Data to the Data Bind Control

```
public class ContactInfoDataSource
{
    public ICollection Data
    {
        set { m_col = value; }
    }

    public ICollection GetData()
    {
        return m_col;
    }

    public ContactInfoDataSource()
    {}

    public ContactInfoDataSource(ICollection col)
    {
        ArrayList results = new ArrayList();
        if (col is INamedCache)
        {
            INamedCache cache = col as INamedCache;
```

```
            foreach (ContactInfo contactInfo in cache.Values)
            {
                results.Add(contactInfo);
            }
        }
        else if (col is ArrayList)
        {
            foreach (DictionaryEntry entry in col)
            {
                results.Add(entry.Value);
            }
        }
        Data = results;
    }

    private ICollection m_col = null;
}
```

Code-behind the ASP.NET Page

Add an event handler that creates an inner object that provide data to the data bind control. This is illustrated in Example 24–10.

Example 24–10 Event Handler to Provide Data to the Data Bind Control

```
protected void dsContact_ObjectCreating(object sender, ObjectDataSourceEventArgs
e)
{
    ContactInfoDataSource cds = new ContactInfoDataSource(Contacts == null ?
ContactCache : Contacts);
    e.ObjectInstance = cds;
}
```

The method illustrated in Example 24–11 refreshes the `GridView` displayed on the page, refreshes the total label `lblTotal`, and makes the `btnClear` and all buttons visible if there are objects in the cache:

Example 24–11 Method to Refresh the Grid View

```
private void RefreshDataGridAndRenderPage()
{
    gridCache.DataBind();

    int totalObjects = (Contacts == null ? ContactCache.Count : Contacts.Count);
    lblTotal.Text = "Total objects: " + totalObjects;

    if (ContactCache.Count > 0)
    {
        lblTotal.Visible = btnClear.Visible = true;
        lblSearch.Visible = listColumnNames.Visible = lblFor.Visible =
txtFilterCriteria.Visible = btnSearch.Visible = true;
    }
    else
    {
        lblTotal.Visible = btnClear.Visible = false;
        lblSearch.Visible = listColumnNames.Visible = lblFor.Visible =
txtFilterCriteria.Visible = btnSearch.Visible = false;
    }

    btnClearFilter.Visible = (Contacts != null);
```

```
}
```

The method illustrated in Example 24–12 handles page load events. If there is a `getKey` value in the `Request`, then the value mapped to the specified key in the cache is retrieved and the appropriate fields populated with its properties. If there is a `removeKey` value in the `Request`, the value mapped to the specified key is removed from the cache.

Example 24–12 Method to Handle Page Load Events

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Request["getKey"] != null)
    {
        FindObjectInCache(Request["getKey"]);
    }
    else if (Request["removeKey"] != null)
    {
        CacheFactory.Log("Object with key [" + Request["removeKey"] + "] has been
removed from cache.", CacheFactory.LogLevel.Info);
        ContactCache.Remove(Request["removeKey"]);
    }

    RefreshDataGridAndRenderPage();
    PopulateFilterColumns();
}
```

The helper method illustrated in Example 24–13 retrieves an `ContactInfo` object from the cache by a specified key:

Example 24–13 Retrieving a Business Object from the Cache through a Specified Key

```
private void FindObjectInCache(object key)
{
    ContactInfo contactInfo = (ContactInfo)ContactCache[key];

    if (contactInfo == null)
    {
        contactInfo = new ContactInfo();
    }

    txtName.Text = key as String;
    txtStreet.Text = contactInfo.Street;
    txtCity.Text = contactInfo.City;
    txtState.Text = contactInfo.State;
    txtZip.Text = contactInfo.Zip;
}
```

Example 24–14 illustrates an the event handler for the `btnSave` button:

Example 24–14 Event Handler for a "Save" Button

```
protected void btnSave_Click(object sender, EventArgs e)
{
    String name = txtName.Text;

    if (name != null && name != "")
    {
        ContactInfo contactInfo = new ContactInfo(name,
                                      txtStreet.Text,
```

```
                                        txtCity.Text,
                                        txtState.Text,
                                        txtZip.Text);
        ContactCache.Insert(name, contactInfo);

        CacheFactory.Log("Object with key [" + name + "] has been inserted into
cache.", CacheFactory.LogLevel.Info);
        RefreshDataGridAndRenderPage();
    }
}
```

Example 24–15 illustrates the event handler for the `btnClear` button:

Example 24–15 Event Handler for a :Clear" Button

```
protected void btnClear_Click(object sender, EventArgs e)
{
    NameValidator.Enabled = false;

    ContactCache.Clear();
    RefreshDataGridAndRenderPage();

    NameValidator.Enabled = true;
}
```

Example 24–16 illustrates the event handler for the `btnSearch` button:

Example 24–16 Event Handler for a "Search" Button

```
protected void btnSearch_Click(object sender, EventArgs e)
{
    NameValidator.Enabled = false;

    String filterBy = listColumnNames.Items[listColumnNames.SelectedIndex].Text;
    String filterCriteria = txtFilterCriteria.Text.Trim();

    if (filterCriteria != "")
    {
        IValueExtractor extractor = new ReflectionExtractor("get" + filterBy);
        IFilter filter = new LikeFilter(extractor, filterCriteria, '\\', true);

        ICollection results = ContactCache.GetEntries(filter);

        Contacts = results;
        dsContact = new ObjectDataSource();

        RefreshDataGridAndRenderPage();
    }

    NameValidator.Enabled = true;
}
```

Example 24–17 illustrates the event handler for the `btnClearFilter` button:

Example 24–17 Event Handler for a "Clear Filter" Button

```
protected void btnClearFilter_Click(object sender, EventArgs e)
{
    NameValidator.Enabled = false;

    Contacts = null;
```

```
    dsContact = new ObjectDataSource();

    RefreshDataGridAndRenderPage();
    NameValidator.Enabled = true;
}
```

Finally, you should add an `ConnectionError.html` page to the project with an appropriate error message in it.

Part III

Integration with WebLogic Server

This section contains the following chapter:

25

Caching HTTP Sessions for WebLogic

The following example demonstrates how to use Coherence*Web to cache session information for Web application instances that are deployed across WebLogic application servers. In particular, this example creates a Web application and deploys it to two application server instances. The application is a simple counter that stores the current count as a session attribute. Coherence*Web automatically serializes and replicates the attribute across both server instances. Lastly, a browser is used to access each application instance to demonstrate that the same session attribute is used among the instances.

Requirements

To complete this example the following software must be installed:

- Oracle Coherence 3.4
- WebLogic 10.X (This example uses 10.3. WebLogic 8.X and 9.X are also supported.)

Install Coherence*Web on WebLogic 10.X

The Coherence*Web module includes a plug-in installer that supports WebLogic 10.X. Use the instructions located at the following link to install Coherence*Web to WebLogic:

Installing Coherence*Web Session Management Module on BEA WebLogic 10.x

Step three of the install procedure is completed later in this example.

Configure WebLogic

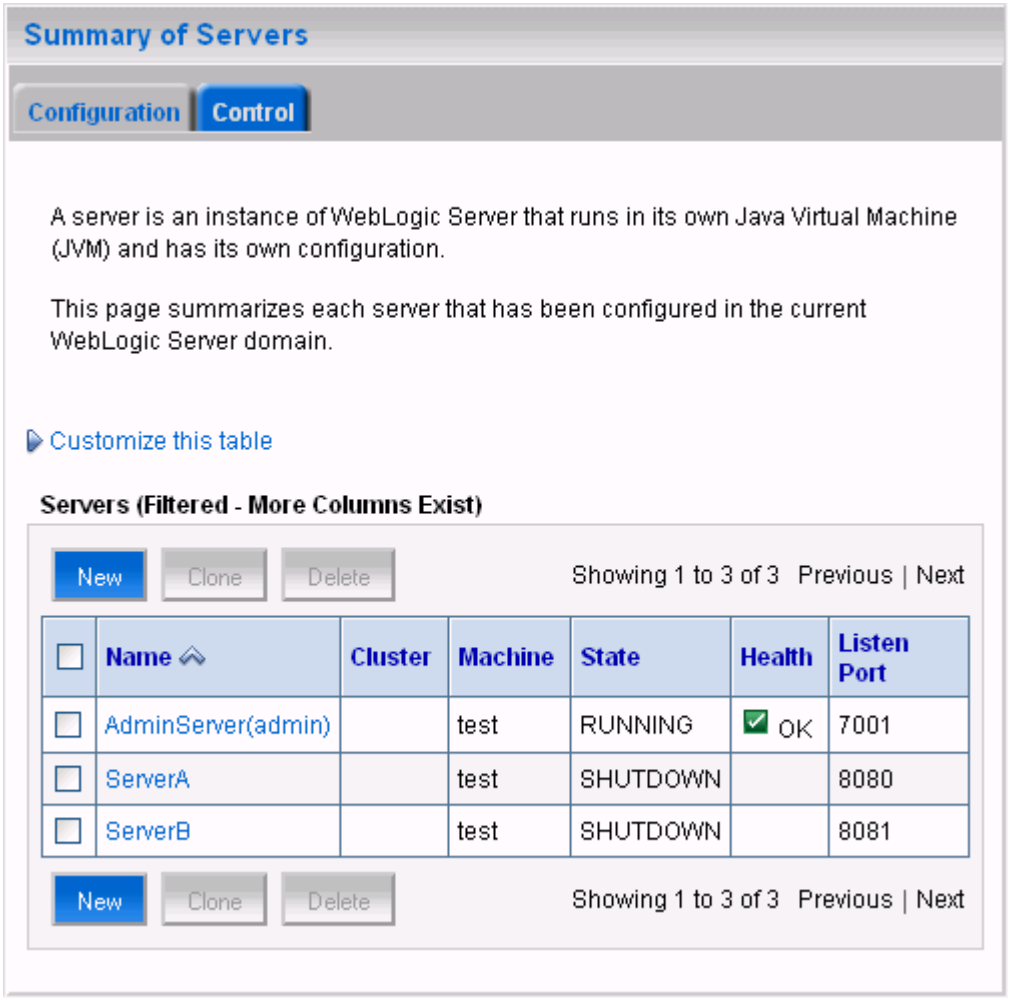
This example requires two application server instances:

1. Run the Oracle WebLogic Configuration Wizard (*BEA_HOME*\wlserver_10.3\common\bin\config.exe or config.sh). Use the wizard to create a new WebLogic domain. From the wizard, customize the domain and configure two managed servers: ServerA using port 8080; and ServerB using port 8081. If you would like to use the WebLogic Node Manager (recommended), configure a Machine to host the application server instances and assign application server instances to the Machine. For this example, the Machine name used is test and the domain name used is test_domain.
2. Before exiting the wizard, click to select the **Start Admin Server** check box, and click **Done**. The configuration wizard automatically starts the administration server. To manually start the administration server, run *BEA_HOME*/user_

`projects/domains/test_domain/startWebLogic.cmd` or `startWebLogic.sh`.

3. From a browser, log in to the Oracle WebLogic Server Administration Console using the following URL:`http://`*`hostname`*`:7001/console`. The console starts, and the domain home page displays.
4. From the **Domain Structure** menu, expand **Environment** and click **Servers**. The **Summary of Servers** page displays and should be similar to Figure 25–1:

Figure 25–1 Summary of Servers Page



This figure is described in the text.

****

5. If you are using the Node Manager, start the manager from a command prompt using *`BEA_HOME`*`\wlserver_10.3\server\bin\startNodeManager.cmd` or `startNodeManager.sh`.
6. From the **Summary of Servers** screen, click the **Control** tab and start both server instances. If you are not using the **Node Manager**, manually start the server instances from the command line. Change directories to *`BEA_HOME`*`\user_projects\domains\test_domain\bin` and issue the following commands:

   To start the two server instances on Windows:

```
startManagedWeblogic.cmd ServerA http://localhost:7001
startManagedWeblogic.cmd ServerB http://localhost:7001
```

To start the two server instances on Linux/UNIX:

```
./startManagedWeblogic.sh ServerA http://localhost:7001
./startManagedWeblogic.sh ServerB http://localhost:7001
```

Create the Counter Web Application

The Counter Web application is a simple counter implemented as a JSP. The counter is stored as an HTTP session attribute and increments each time the page is accessed.

To create the Counter Web application:

1. Create a standard Web application directory as follows:

```
/
/WEB-INF
```

2. Copy the following code to a text file:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
 xmlns="http://java.sun.com/xml/ns/j2ee" version="2.5">
   <description>Empty web.xml file for Web Application</description>
</web-app>
```

3. Save the file as `web.xml` to the `/WEB-INF` directory.
4. Copy the following code to a text file:

```
<h3>
      Counter :
      <%
         Integer counter = new Integer(1);
         HttpSession httpsession = request.getSession(true);
         if (httpsession.isNew()) {
                httpsession.setAttribute("count", counter);
                out.println(counter);
         } else {
                int count = ((Integer)
httpsession.getAttribute("count")).intValue();
                httpsession.setAttribute("count", new Integer(++count));
                out.println(count);
         }
      %>
      </h3>
```

5. Save the file as `counter.jsp` in the root of the Web application directory. The Web application directory should appear as follows:

```
/
/counter.jsp
/WEB-INF/web.xml
```

6. ZIP or JAR the Web application directory and save the file as `counter.war`.

Modify the Counter Web Application to use Coherence*Web

All Web applications that want to take advantage of Coherence*Web must be modified to include the required Coherence*Web libraries and configuration files. Coherence includes an installer that automatically adds the necessary files to a Web application. To run the installer, follow the instructions located at:

General Instructions for Installing Coherence*Web Session Management Module

The installation is a 2-step process:

- The inspect step—In the inspect step, a `coherence-web.xml` file is created for the Counter Web application.
- The install step—In the install step, the Counter Web application's web.xml is modified based on the content of the `coherence-web.xml`. In addition, the application is modified to include the `coherence.jar`, `coherence-web.jar`, and `tangosol.jar`; and the `session-cache-config.xml` configuration file.

The `coherence-web.xml` file can be edited to change the behavior of the session cache. For example, the default session model used is the split model. The `coherence-sessioncollection-class` parameter can be modified to use a different session model (that is, monolithic or traditional).

Note: Changes to the `coherence-web.xml` file must be made before the install step.

For more information on how a Web application is modified during installation, see How the Coherence*Web Installer Instruments a Java EE Application.

Deploy the Application

To deploy the application:

1. From a browser, log in to the Oracle WebLogic Server Administration Console using the following URL:

   `http://`*`host`*`:7001/console`

   The console starts and the domain home page displays.

2. From the **Domain Structure** menu, click **deployments**. The **Summary of Deployments** page displays.
3. Click **Install**. The **Install Application Assistant** screen displays.
4. Use the **Install Application Assistant** to deploy `counter.war` to both `ServerA` and `ServerB`. The **Summary of Deployments** page displays after the application is deployed. Figure 25–2 illustrates the deployments table with the counter Web application.

Figure 25–2 Deployments Window Showing the Deployed Application

This figure is described in the text.

****

Verify the Example

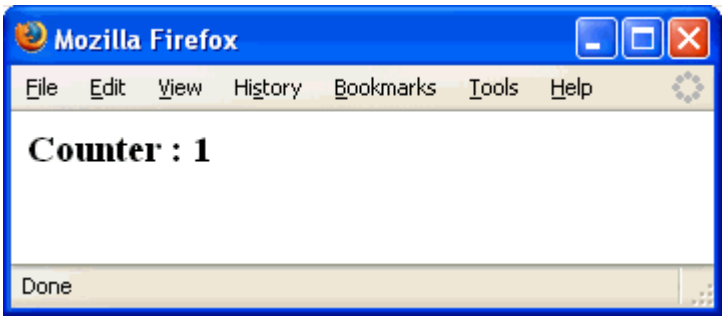
To verify the example:

1. Open a browser and access the `ServerA` Counter instance using the following URL:

   `http://host:8080/counter/counter.jsp`

   The counter page displays and the counter is set to 1 as follows:

Figure 25–3 Counter Page with Counter Set to 1



This figure is described in the text.

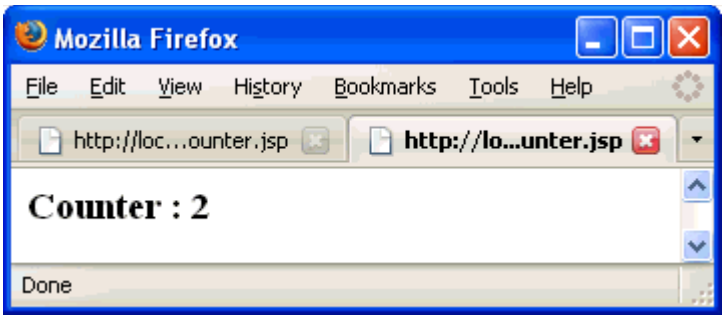
****

2. From the same browser (or in a new browser tab), access the `ServerB` `Counter` instance using the following URL:

   `http://host:8081/counter/counter.jsp`

   The counter page displays and the counter is incremented to 2 based on the session data: If you refresh the page, the counter is incremented to 3. Refresh the instance on `ServerA` and the counter is at 4.

Figure 25–4 Counter Page with Counter Set to 4



This figure is described in the text.

****

Summary

This example demonstrated how Coherence*Web is used to cache session data across multiple Web application instances deployed to multiple WebLogic server instances.

Part IV

Integration with TopLink Essentials

This section contains the following chapters:

26

Configuring Coherence for TopLink Essentials

Oracle TopLink is often described as one of the most flexible and scalable object-relational mapping libraries and performs particularly well for read-intensive applications. A streamlined version called TopLink Essentials became the Reference Implementation for the Java Persistence API (JPA) 1.0 and was open-sourced and donated to the Glassfish project at java.net. TopLink Essentials offers all of the essential functionality for Object-Relational mapping through either JPA or the native TopLink API, but it also provides several other custom and advanced features for more sophisticated application usage. To obtain a free download of TopLink Essentials go to http://www.oracle.com/technology/products/ias/toplink/jpa/index.html.

Coherence and TopLink Essentials

Coherence ships a `CacheStore` implementation that uses TopLink Essentials to load and store objects to the database. This document describes how to configure and use this `CacheStore`. Note that although this `CacheStore` allows the objects to be mapped as JPA entities, it differs from the JPA `CacheStore` in that it uses the TopLink runtime API to load and store the objects.

Limitations

Support is currently limited to TopLink Essentials (not Oracle TopLink). In most cases the actual Oracle TopLink mappings used by applications will also work in TopLink Essentials, but the TopLink project mapping files (deployment XML) are not read by TopLink Essentials. TopLink Essentials will read and process TopLink projects in Java code, though, and all types of JPA mappings and metadata.

Conventions

This chapter refers to the following Java classes and interfaces:

Example 26–1 TopLink Essentials-related Classes and Interfaces

```
com.tangosol.coherence.toplink.TopLinkCacheLoader
com.tangosol.coherence.toplink.TopLinkCacheStore

com.tangosol.net.NamedCache (extends java.util.Map)

com.tangosol.net.cache.CacheLoader
com.tangosol.net.cache.CacheStore
```

```
oracle.toplink.essentials.sessions.Project
oracle.toplink.essentials.threetier.ServerSession
oracle.toplink.essentials.tools.sessionmanagement.SessionManager
```

As the `CacheStore` interface extends `CacheLoader`, the term "`CacheStore`" will be used generically to refer to both interfaces (the appropriate interface being determined by whether read-only or read-write support is required). Similarly, "`TopLinkCacheStore`" will refer to both implementations.

The Coherence cache configuration file is referred to as the `coherence-cache-config.xml` (the default name). TopLink Essentials may be referred to simply as TopLink in this document. The JPA runtime configuration file is referred to as the `persistence.xml` and the JPA mapping file is referred to as the `orm.xml` (the default name).

Using the Coherence TopLinkCacheStore

The TopLink API provides advanced and flexible queries and many relational management features, including referential integrity, cascading deletes and child object fetching. TopLink employs an advanced caching system to properly manage the entities. In many cases the TopLink cache will short-circuit a database operation to minimize the operational latency, while in other cases it will simply use its cache to ensure object identity.

Coherence includes a default entity-based CacheStore implementation, `TopLinkCacheStore` (and a corresponding `CacheLoader` implementation, `TopLinkCacheLoader`). Other information may be found in the Javadoc for the implementing classes.

Mapping the Persistent Classes

The first step in being able to load and store objects through the `CacheStore` is to ensure that the classes are mapped to the database. In TopLink Essentials objects may be mapped using either standard JPA mappings or native TopLink O/R mappings.

JPA mappings are specified either by annotating the entity classes or by adding an orm.xml or other XML mapping files. See the TopLink JPA documentation for more on how to map JPA entities.

TopLink mappings may be used instead of, or in addition to JPA mappings. They may be configured using either a TopLink project class (see the TopLink documentation for more on how to create a Java project class) or a customization class to amend the TopLink descriptors for each class and add the mappings (see the TopLink documentation for more on how to create a customization class and add Java mappings). While a broader set of mappings is available in TopLink the mappings may not be portable to other JPA providers.

Configuring TopLink Essentials

The runtime configuration and startup code will be different depending upon whether JPA mappings or TopLink mappings are used.

Configuration with JPA Mappings

If using JPA mappings then TopLink Essentials is configured using a `persistence.xml` file. Within `persistence.xml` are the properties that dictate runtime operation. The `toplink.session-name` property determines the name given to the TopLink session created to model the persistence unit entity manager

factory. This property may be set to any non-empty value if it is set. Example 26–2 illustrates a sample `persistence.xml` showing the `toplink.session-name` property setting.

Example 26–2 Sample persistence.xml File for TopLink Essentials

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">

   <persistence-unit name="EmpUnit" transaction-type="RESOURCE_LOCAL">

     <provider>oracle.toplink.essentials.PersistenceProvider</provider>

     <class>com.acme.Employee</class>

     <properties>
       <property name="toplink.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
       <property name="toplink.jdbc.url"
value="jdbc:oracle:thin:@localhost:1521:XE"/>
       <property name="toplink.jdbc.user" value="scott"/>
       <property name="toplink.jdbc.password" value="tiger"/>

       <property name="toplink.session-name" value="EmployeeSession"/>
     </properties>

   </persistence-unit>

</persistence>
```

The transaction type should be set to `RESOURCE_LOCAL` and the four JDBC properties should contain the appropriate values for connecting and logging in to the database being used. Classes that are mapped using JPA annotations should be listed in `<class>` elements.

Configuration with TopLink Mappings

When using a Java project class the class may be created either manually or through a tool such as the Mapping Workbench. The class need only be compiled and present on the classpath.

The project class should be instantiated and passed into the session constructor when creating the session. The session must also be added to the `SessionManager`. Example 26–3 illustrates how this might be done.

Example 26–3 Instantiating the Project and Passing to the SessionManager

```
Project project = new EmployeeMappingProject();
ServerSession session = new ServerSession(project);
SessionManager.getManager().addSession("EmployeeSession", session);
```

Configuring Coherence

A `coherence-cache-config.xml` file must be specified to override the default Coherence settings and define the TopLinkCacheStore caching scheme. The caching scheme should include a `<cachestore-scheme>` element that lists the `TopLinkCacheStore` class and includes two parameters. The first parameter is the

entity name, or the alias for the entity being stored. When using JPA this is normally the unqualified name of the entity class, and when mapped using TopLink it is the alias for the class that is set on the descriptor. In the example cache scheme listed below we make use of the built-in Coherence macro `{cache-name}` that translates to the name of the cache that is constructing and using the cache store. This will work because a separate cache should be used for each type of persistent object and we will ensure that the name of each cache will be set to the name of the entity that is being stored in it.

The second parameter is the name of the session that was indicated by the value of the `session-name` property in the `persistence.xml` if using JPA mappings. It is the name that was explicitly given to the session if using a TopLink session directly.

The various named caches are then directed to use the TopLink caching scheme. The following is a sample `coherence-cache-config.xml` used to define a `NamedCache` called "`Employee`" that caches instances of the `Employee` class. To define additional entity caches for more classes then more `<cache-mapping>` elements may be added. In Example 26–4 we are assuming the entities are mapped using JPA mappings.

Example 26–4 Assigning Named Caches to a TopLink Caching Scheme

```
<cache-config>

  <caching-scheme-mapping>

    <!-- Configure a named cache -->
    <cache-mapping>
      <!-- Set the name of the cache to be the entity name  -->
      <cache-name>Employee</cache-name>
      <!-- Configure this cache to use the scheme defined below  -->
      <scheme-name>toplink-distributed</scheme-name>
    </cache-mapping>

  </caching-scheme-mapping>

  <caching-schemes>

    <distributed-scheme>

      <scheme-name>toplink-distributed</scheme-name>
      <service-name>TopLinkDistributedCache</service-name>

      <backing-map-scheme>
        <read-write-backing-map-scheme>

          <internal-cache-scheme>
            <local-scheme/>
          </internal-cache-scheme>

          <!- Define the cache scheme -->
          <cachestore-scheme>
            <class-scheme>
              <class-name>
                com.tangosol.coherence.toplink.TopLinkCacheStore
              </class-name>
              <init-params>
                <!-- This param should be the entity name -->
                <init-param>
                  <param-type>java.lang.String</param-type>
```

```
                <param-value>{cache-name}</param-value>
               </init-param>
               <!-- This param should match the value of the session-name
property -->
              <!-- in persistence.xml file if JPA mappings are used, or the name
-->
              <!-- assigned to the TopLink session if TopLink mappings are used
-->
               <init-param>
                <param-type>java.lang.String</param-type>
                <param-value>EmployeeSession</param-value>
               </init-param>
             </init-params>
           </class-scheme>
         </cachestore-scheme>
       </read-write-backing-map-scheme>
     </backing-map-scheme>

     <autostart>true</autostart>
   </distributed-scheme>

  </caching-schemes>

</cache-config>
```

27

Configuring Coherence for JPA

The Java Persistence API (JPA) is the primary standard for Object-Relational mapping (ORM) and enterprise Java persistence. Several open source and commercial implementations exist and are being developed.

Coherence ships a `CacheStore` implementation that uses JPA to load and store objects to the database. This document describes how to configure and use this `CacheStore`.

Limitations

Only resource-local and bootstrapped entity managers are currently supported. Container-managed entity managers and those that use JTA transactions are not currently supported.

Obtaining a JPA Implementation

A JPA provider is not shipped with Coherence, but is easy to obtain. Although the JPA CacheStore will work with any compliant JPA implementation, we recommend using one of the following:

- TopLink Essentials is the Reference Implementation for the JPA 1.0 specification. It is open source and free, available from the Oracle Technology Network (OTN) at `http://otn.oracle.com/jpa`.
- Eclipse JPA will be the Reference Implementation for the forthcoming JPA 2.0 specification. Oracle is leading the open source EclipseLink project that includes Eclipse JPA. EclipseLink is available from Eclipse at `http://www.eclipse.org/eclipselink`.

Conventions

This document refers to the following Java classes and interfaces:

Example 27–1 JPA-related Classes and Interfaces

```
com.tangosol.coherence.jpa.JpaCacheLoader
com.tangosol.coherence.jpa.JpaCacheStore

com.tangosol.net.NamedCache (extends java.util.Map)

com.tangosol.net.cache.CacheLoader
com.tangosol.net.cache.CacheStore
```

As the `CacheStore` interface extends `CacheLoader`, the term "`CacheStore`" will be used generically to refer to both interfaces (the appropriate interface being determined by whether read-only or read-write support is required). Similarly, "`JpaCacheStore`" will refer to both implementations.

The Coherence cache configuration file is referred to as the `coherence-cache-config.xml` (the default name). The JPA persistence implementation is referred to simply as the JPA provider or JPA vendor. The JPA runtime configuration file is referred to as the `persistence.xml`, and the JPA Object-Relational mapping file is referred to as the `orm.xml` (the default name).

Using the Coherence JpaCacheStore

The JPA is a standard API for mapping, querying and storing Java objects to a database. The characteristics of the different JPA implementations may differ, however, when it comes to caching, threading, and overall performance. TopLink Essentials is a high-performing JPA implementation that meets the performance needs of most applications.

Coherence includes a default entity-based CacheStore implementation, `JpaCacheStore` (and a corresponding `CacheLoader` implementation, `JpaCacheLoader`). Other information may be found in the Javadoc for the implementing classes.

Mapping the Persistent Classes

The first step in being able to load and store objects through the `CacheStore` is to ensure that the classes are mapped to the database. JPA mappings are standard, and hence may be specified the same way for any and all JPA providers.

Entities may be mapped either by annotating the entity classes or by adding an `orm.xml` or other XML mapping file. See the JPA vendor documentation for more on how to map JPA entities.

Configuring JPA

A typical JPA configuration involves making changes to the `persistence.xml` file. Within the `persistence.xml` are the properties that dictate runtime operation. Example 27–2 is a sample `persistence.xml` showing the typical properties that are set.

Example 27–2 Sample persistence.xml File for JPA

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">

   <persistence-unit name="EmpUnit" transaction-type="RESOURCE_LOCAL">

     <provider>oracle.toplink.essentials.PersistenceProvider</provider>

     <class>com.acme.Employee</class>

     <properties>
       <property name="toplink.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
       <property name="toplink.jdbc.url"
```

```
value="jdbc:oracle:thin:@localhost:1521:XE"/>
       <property name="toplink.jdbc.user" value="scott"/>
       <property name="toplink.jdbc.password" value="tiger"/>
     </properties>

   </persistence-unit>

</persistence>
```

The transaction type should be set to `RESOURCE_LOCAL` and the four JDBC properties should contain the appropriate values for connecting and logging into your database. Classes that are mapped using JPA annotations should be listed in `<class>` elements.

Configuring Coherence

A `coherence-cache-config.xml` must be specified to override the default Coherence settings and define the `JpaCacheStore` caching scheme. The caching scheme should include a `<cachestore-scheme>` element that lists the `JpaCacheStore` class and includes three parameters.

- The first parameter is the entity name of the entity being stored. Unless it is explicitly overridden in JPA it will be the unqualified name of the entity class. In Example 27–3, we make use of the built-in Coherence macro `{cache-name}` that translates to the name of the cache that is constructing and using the `CacheStore`. This works because a separate cache should be used for each type of persistent entity and we will ensure that the name of each cache will be set to the name of the entity that is being stored in it.
- The second parameter is the fully qualified name of the entity class. If the classes are all in the same package and use the default JPA entity names then we can once again use the `{cache-name}` macro to fill in the part that is variable across the different entity types. In this way the same caching scheme can be used for all of the entities that are cached within the same persistence unit.
- The third parameter is the persistence unit name, which should be the same as the name specified in the `persistence.xml`.

The various named caches are then directed to use the JPA caching scheme. Example 27–3 is a sample `coherence-cache-config.xml` used to define a `NamedCache` called "`Employee`" that caches instances of the `Employee` class. To define additional entity caches for more classes then more `<cache-mapping>` elements may be added.

Example 27–3 Assigning Named Caches to a JPA Caching Scheme

```
<cache-config>

  <caching-scheme-mapping>

    <cache-mapping>
      <!-- Set the name of the cache to be the entity name  -->
      <cache-name>Employee</cache-name>
      <!-- Configure this cache to use the scheme defined below  -->
      <scheme-name>jpa-distributed</scheme-name>
    </cache-mapping>

  </caching-scheme-mapping>

  <caching-schemes>
```

```
    <distributed-scheme>

      <scheme-name>jpa-distributed</scheme-name>
      <service-name>JpaDistributedCache</service-name>

      <backing-map-scheme>
        <read-write-backing-map-scheme>

          <internal-cache-scheme>
            <local-scheme/>
          </internal-cache-scheme>

          <!- Define the cache scheme -->
          <cachestore-scheme>
            <class-scheme>
              <class-name>
                com.tangosol.coherence.jpa.JpaCacheStore
              </class-name>
              <init-params>

                <!-- This param is the entity name -->
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>{cache-name}</param-value>
                </init-param>

                <!-- This param is the fully qualified entity class -->
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>com.acme.{cache-name}</param-value>
                </init-param>

                <!-- This param should match the value of the -->
                <!-- persistence unit name in persistence.xml -->
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>EmpUnit</param-value>
                </init-param>

              </init-params>
            </class-scheme>
          </cachestore-scheme>

        </read-write-backing-map-scheme>
      </backing-map-scheme>

    </distributed-scheme>

  </caching-schemes>

</cache-config>
```

Part V

Integration with Hibernate

This section contains the following chapters:

28

Using Coherence as the Hibernate L2 Cache

Coherence can be used as the L2 cache provider for Hibernate.

Hibernate and Caching

Hibernate supports three primary forms of caching:

- Session cache
- L2 cache
- Query cache

The `Session` cache is responsible for caching records within a `Session` (a Hibernate transaction, potentially spanning multiple database transactions, and typically scoped on a per-thread basis). As a non-clustered cache (by definition), the `Session` cache is managed entirely by Hibernate. The L2 and Query caches span multiple transactions, and support the use of Coherence as a cache provider. The L2 cache is responsible for caching records across multiple sessions (for primary key lookups). The query cache caches the result sets generated by Hibernate queries. Hibernate manages data in an internal representation in the L2 and Query caches, meaning that these caches are usable only by Hibernate. For more details, see the Hibernate Reference Documentation (shipped with Hibernate), specifically the section on the Second Level Cache.

Configuration and Tuning

To use the Coherence Caching Provider for Hibernate, specify the Coherence provider class in the `hibernate.cache.provider_class` property. Typically this is configured in the default Hibernate configuration file, `hibernate.cfg.xml`.

Example 28–1 Specifying a Coherence Provider Class

```
<property name="hibernate.cache.provider_
class">com.tangosol.coherence.hibernate.CoherenceCacheProvider</property>
```

The file `coherence-hibernate.jar` (found in the `lib/` subdirectory) must be added to the application classpath.

Hibernate provides the configuration property `hibernate.cache.use_minimal_puts`, which optimizes cache access for clustered caches by increasing cache reads and decreasing cache updates. This is enabled by default by the Coherence Cache Provider. Setting this property to false may increase overhead for cache management and also increase the number of transaction rollbacks.

The Coherence Caching Provider includes a setting for how long a lock acquisition should be attempted before timing out. This may be specified by the Java property `tangosol.coherence.hibernate.lockattemptmillis`. The default is one minute.

Specifying a Coherence Cache Topology

By default, the Coherence Caching Provider uses a custom cache configuration located in `coherence-hibernate.jar` named `config/hibernate-cache-config.xml` to define cache mappings for Hibernate L2 caches. If desired, an alternative cache configuration resource may be specified for Hibernate L2 caches by using the `tangosol.coherence.hibernate.cacheconfig` Java property. It is possible to configure this property to point to the application's main `coherence-cache-config.xml` file if mappings are properly configured. It may be beneficial to use dedicated cache service(s) to manage Hibernate-specific caches to ensure that any `CacheStore` modules don't cause re-entrant calls back into Coherence-managed Hibernate L2 caches.

With the scheme mapping section of the Coherence cache configuration file, the `hibernate.cache.region_prefix` property may be used to specify a cache topology. For example, if the cache configuration file includes a wildcard mapping for `near-*`, and the Hibernate region prefix property is set to `near-`, then all Hibernate caches will be named using the `near-` prefix, and will use the cache scheme mapping specified for the `near-*` cache name pattern.

It is possible to specify a cache topology per entity by creating a cache mapping based on the combined prefix and qualified entity name (for example, `near-com.company.EntityName`); or equivalently, by providing an empty prefix and specifying a cache mapping for each qualified entity name.

Also, L2 caches should be size-limited to avoid excessive memory usage. Query caches in particular must be size-limited as the Hibernate API does not provide any means of controlling the query cache other than a complete eviction.

Cache Concurrency Strategies

Hibernate generally emphasizes the use of optimistic concurrency for both cache and database. With optimistic concurrency in particular, transaction processing depends on having accurate data available to the application at the beginning of the transaction. If the data is inaccurate, the commit processing will detect that the transaction was dependent on incorrect data, and the transaction will fail to commit. While most optimistic transactions must cope with changes to underlying data by other processes, the use of caching adds the possibility of the cache itself being stale. Hibernate provides several cache concurrency strategies to control updates to the L2 cache. While this is less of an issue for Coherence due to support for clusterwide coherent caches, appropriate selection of cache concurrency strategy will aid application efficiency.

Note that cache configuration strategies may be specified at the table level. Generally, the strategy should be specified in the mapping file for the class.

For mixed read-write activity, the read-write strategy is recommended. The transactional strategy is implemented similarly to the nonstrict-read-write strategy, and relies on the optimistic concurrency features of Hibernate. Note that nonstrict-read-write may deliver better performance if its impact on optimistic concurrency is acceptable.

For read-only caching, use the nonstrict-read-write strategy if the underlying database data may change, but slightly stale data is acceptable. If the underlying database data never changes, use the read-only strategy.

Query Cache

To cache query results, set the `hibernate.cache.use_query_cache` property to "`true`". Then whenever issuing a cacheable query, use `Query.setCacheable(true)` to enable caching of query results. As `org.hibernate.cache.QueryKey` instances in Hibernate may not be binary-comparable (due to non-deterministic serialization of unordered data members), use a size-limited Local or Replicated cache to store query results (which will force the use of `hashcode()/equals()` to compare keys). The default query cache name is `org.hibernate.cache.StandardQueryCache` (unless a default region prefix is provided, in which case `[`*prefix*`]`. will be prepended to the cache name). Use the cache configuration file to map this cache name to a Local/Replicated topology, or explicitly provide an appropriately-mapped region name when querying.

Fault-Tolerance

The Hibernate L2 cache protocol supports full fault-tolerance during client or server failure. With the read-write cache concurrency strategy, Hibernate will lock items out of the cache at the start of an update transaction, meaning that client-side failures will simply result in uncached entities and an uncommitted transaction. Server-side failures are handled transparently by Coherence (dependent on the specified data backup count).

Deployment

When used with application servers that do not have a unified class loader, the Coherence Cache Provider must be deployed as part of the application so that it can use the application-specific class loader (required to serialize-deserialize objects).

29

Using Hibernate as a CacheStore for Coherence

The functionality in Coherence and Hibernate can be combined in several ways. For example, Hibernate can be used as a `CacheStore` for Coherence.

Using the Coherence HibernateCacheStore

Coherence includes a default entity-based `CacheStore` implementation, `HibernateCacheStore` (and a corresponding `CacheLoader` implementation, `HibernateCacheLoader`. More detailed technical information may be found in the Javadoc for the implementing classes.

Configuring a HibernateCacheStore

The examples below show a simple `HibernateCacheStore` constructor, accepting only an entity name. This will configure Hibernate using the default configuration path, which looks for a `hibernate.cfg.xml` file in the classpath. There is also the ability to pass in a resource name or file specification for the `hibernate.cfg.xml` file as the second `<init-param>` (set the `<param-type>` element to `java.lang.String` for a resource name and `java.io.File` for a file specification). See `HibernateCacheStore` for more details.

The following is a simple `coherence-cache-config.xml` file used to define a `NamedCache` called "`TableA`" which caches instances of a Hibernate entity (`com.company.TableA`). To add additional entity caches, add additional `<cache-mapping>` elements.

Example 29–1 Sample coherence-cache-config.xml File for Hibernate

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>TableA</cache-name>
      <scheme-name>distributed-hibernate</scheme-name>
      <init-params>
        <init-param>
          <param-name>entityname</param-name>
          <param-value>com.company.TableA</param-value>
        </init-param>
      </init-params>
```

```
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

          <cachestore-scheme>
            <class-scheme>
              <class-name>
              com.tangosol.coherence.hibernate.HibernateCacheStore
              </class-name>
              <init-params>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>{entityname}</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

It is also possible to use the pre-defined `{cache-name}` macro to eliminate the need for the `<init-params>` portion of the cache mapping. This is illustrated in Example 29–2:

Example 29–2 Sample coherence-cache-config.xml File that Uses {cache-name} Macro

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>TableA</cache-name>
      <scheme-name>distributed-hibernate</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

          <cachestore-scheme>
            <class-scheme>
```

```
              <class-name>
              com.tangosol.coherence.hibernate.HibernateCacheStore
              </class-name>
              <init-params>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>com.company.{cache-name}</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

And, if naming conventions allow, the mapping may be completely generalized to allow a cache mapping for any qualified class name (entity name). This is illustrated in Example 29–3.

Example 29–3 Sample coherence-cache-config.xml File with Generalized Mappings

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>com.company.*</cache-name>
      <scheme-name>distributed-hibernate</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

          <cachestore-scheme>
            <class-scheme>
              <class-name>
              com.tangosol.coherence.hibernate.HibernateCacheStore
              </class-name>
              <init-params>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>{cache-name}</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
    </distributed-scheme>
```

```
  </caching-schemes>
</cache-config>
```

Configuration Requirements

Hibernate entities accessed by using the `HibernateCacheStore` module must use the "assigned" ID generator and also have a defined ID property.

Be sure to disable the `hibernate.hbm2ddl.auto` property in the `hibernate.cfg.xml` used by the `HibernateCacheStore`, as this may cause excessive schema updates (and possible lockups).

JDBC Isolation Level

In cases where all access to a database is through Coherence, `CacheStore` modules will naturally enforce ANSI-style Repeatable Read isolation as reads and writes are executed serially on a per-key basis (by using the Partitioned Cache Service). Increasing database isolation above Repeatable Read will not yield increased isolation as `CacheStore` operations may span multiple Partitioned Cache nodes (and thus multiple database transactions). Using database isolation levels below Repeatable Read will not result in unexpected anomalies, and may reduce processing load on the database server.

Fault-Tolerance

For single-cache-entry updates, `CacheStore` operations are fully fault-tolerant in that the cache and database are guaranteed to be consistent during any server failure (including failures during partial updates). While the mechanisms for fault-tolerance vary, this is true for both write-through and write-behind caches.

Coherence does not support two-phase `CacheStore` operations across multiple `CacheStore` instances. In other words, if two cache entries are updated, triggering calls to `CacheStore` modules sitting on separate servers, it is possible for one database update to succeed and for the other to fail. In this case, it may be preferable to use a cache-aside architecture (updating the cache and database as two separate components of a single transaction) with the application server transaction manager. In many cases it is possible to design the database schema to prevent logical commit failures (but obviously not server failures). Write-behind caching avoids this issue as "puts" are not affected by database behavior (and the underlying issues will have been addressed earlier in the design process).

Extending HibernateCacheStore

In some cases, it may be desired to extend the `HibernateCacheStore` with application-specific functionality. The most obvious reason for this is to leverage a pre-existing programmatically-configured `SessionFactory` instance.

Creating a Hibernate CacheStore

While the provided `HibernateCacheStore` module provides a solution for most entity-based caches, there may be cases where an application-specific `CacheStore` module is necessary. For example, providing parameterized queries or including or post-processing of query results.

Re-entrant Calls

In a `CacheStore`-backed cache implementation, when the application thread accesses cached data, the cache operations may trigger a call to the associated `CacheStore` implementation by using the managing `CacheService`. The `CacheStore` must not call back into the `CacheService` API. This implies, indirectly, that Hibernate should not attempt to access cache data. Therefore, all methods in `CacheLoader/CacheStore` should be careful to call `Session.setCacheMode(CacheMode.IGNORE)` to disable cache access. Alternatively, the Hibernate configuration may be cloned (either programmatically or by using `hibernate.cfg.xml`), with `CacheStore` implementations using the version with the cache disabled.

It is important that a `CacheStore` implementation does not call back into the hosting cache service. Therefore, in addition to avoiding calls to `NamedCache` methods, you should also ensure that Hibernate itself does not use any cache services. To do this, call `Session.setCacheMode(CacheMode.IGNORE)` each time a session is used. Alternatively, the Hibernate configuration may be cloned (either programmatically or by using `hibernate.cfg.xml`), with `CacheStore` implementations using the version with the cache disabled.

Fully Cached DataSets

Distributed Queries

Distributed queries offer the potential for lower latency, higher throughput and less database server load compared to executing queries on the database server. For set-oriented queries, the dataset must be entirely cached to produce correct query results. More precisely, for a query issued against the cache to produce correct results, the query must not depend on any uncached data.

This means that you can create hybrid caches. For example, it is possible to combine two uses of a NamedCache: a fully cached size-limited dataset for querying (for example, the data for the most recent week), and a partially cached historical dataset used for singleton reads. This is a good approach to avoid data duplication and minimize memory usage.

While fully cached datasets are usually bulk-loaded during application startup (or on a periodic basis), `CacheStore` integration may be used to ensure that both cache and database are kept fully synchronized.

Detached Processing

Another reason for using fully-cached datasets is to provide the ability to continue application processing even if the underlying database goes down. Using write-behind caching extends this mode of operation to support full read-write applications. With write-behind, the cache becomes (in effect) the temporary system of record. Should the database fail, updates will be queued in Coherence until the connection is restored, at which point all cache changes will be sent to the database.

A

Sample C++ Applications

This appendix provides the sample code for the `console`, `contacts`, and `hellogrid` C++ examples.

Sample Code for the console Example

Now that you've run the console example, you are encouraged to have a look at the code. Each sample has a corresponding directory under examples which contains its sample specific source. There is also a common directory which contains source used in all samples.

Example A–1 illustrates the source code for the `console.cpp` command line application that enables you to interact with the cache using simple commands.

Example A–1 Code for the Console Sample Application

```
#include "coherence/lang.ns"

#include "coherence/io/pof/SystemPofContext.hpp"
#include "coherence/net/CacheFactory.hpp"
#include "coherence/net/NamedCache.hpp"
#include "coherence/util/Iterator.hpp"
#include "coherence/util/Map.hpp"
#include "coherence/util/Set.hpp"

#include "StreamParser.hpp"

#include <iostream>
#include <sstream>

using namespace coherence::lang;

using coherence::examples::StreamParser;
using coherence::io::pof::SystemPofContext;
using coherence::net::CacheFactory;
using coherence::net::NamedCache;
using coherence::util::Iterator;
using coherence::util::Map;
using coherence::util::Set;
```

```
/**
* This Coherence for C++ example provides a simple console for playing with
* caches from within C++.
*
* @argc  the number of command line arguments (including the process name)
* @argv  [cache-name]
*/
int main(int argc, char** argv)
    {
    NamedCache::Handle hCache;

    if (argc > 1)
        {
        // load command line specified cache
        try
            {
            hCache = CacheFactory::getCache(argv[1]);
            }
        catch (const std::exception& e)
            {
            std::cerr << e.what() << std::endl;
            }
        }

    while (true)
        {
        try
            {
            // prompt for input
            std::cout << "\nMap (";
            if (NULL == hCache)
                {
                std::cout << '?';
                }
            else
                {
                std::cout << hCache->getCacheName();
                }
            std::cout << "): " << std::flush;

            char achInput[256];
            std::cin.getline(achInput, 256);

            if (std::cin.fail())
                {
                std::cin.clear();
                continue;
                }

            std::stringstream ssInput(achInput);

            // process input
            String::View vsCmd = cast<String::View>(StreamParser::next(ssInput));

            if (vsCmd->equals("bye"))
                {
                // quit
                try
                    {
                    CacheFactory::shutdown();
```

```
                        }
                    catch (const std::exception& e)
                        {
                        std::cerr << e.what() << std::endl;
                        return 1;
                        }
                    return 0;
                    }
                else if (vsCmd->equals("cache"))
                    {
                    // lookup a cache from the CacheFactory
                    String::View vsCacheName =
cast<String::View>(StreamParser::next(ssInput));
                    hCache = CacheFactory::getCache(vsCacheName);
                    }
                else if (vsCmd->equals("classes"))
                    {
                    // output the SystemClassLoader
                    std::cout << SystemClassLoader::getInstance() << std::endl;
                    }
                else if (vsCmd->equals("clear"))
                    {
                    // clear the current cache
                    hCache->clear();
                    }
                else if (vsCmd->equals("destroy"))
                    {
                    // destroy the current cache
                    CacheFactory::destroyCache(hCache);
                    }
                else if (vsCmd->equals("get"))
                    {
                    // perform a get operation on the current cache
                    Object::View vKey   = StreamParser::next(ssInput);
                    Object::View vValue = hCache->get(vKey);

                    // print the current value
                    std::cout << vValue << std::endl;
                    }
                else if (vsCmd->equals("list"))
                    {
                    // obtain the entire cache contents
                    Set::View vSetEntries = hCache->entrySet();

                    // print key value pairs
                    for (Iterator::Handle hIter = vSetEntries->iterator();
                            hIter->hasNext(); )
                        {
                        Map::Entry::View vEntry =
                            cast<Map::Entry::View>(hIter->next());

                        std::cout << vEntry->getKey() << " = "
                                  << vEntry->getValue() << std::endl;
                        }
                    }
                else if (vsCmd->equals("memory"))
                    {
                    // print information about allocated objects
                    HeapAnalyzer::View vAnalyzer = System::getHeapAnalyzer();
```

```
        if (NULL == vAnalyzer)
            {
            std::cout << "analysis disabled" << std::endl;
            }
        else if (cast<String::View>(StreamParser::next(ssInput))->
                equals("delta"))
            {
            static HeapAnalyzer::Snapshot::View vMark;

            // compare current against the heap mark
            std::cout << (NULL == vMark
                    ? vAnalyzer->capture() : vAnalyzer->delta(vMark))
                    << std::endl;

            // reset to mark based on the current heap useage
            vMark = NULL;
            vMark = vAnalyzer->capture();
            }
        else
            {
            // output the current heap useage
            std::cout << vAnalyzer << std::endl;
            }
        }
    else if (vsCmd->equals("pof"))
        {
        // output the SystemPofContext
        std::cout << SystemPofContext::getInstance() << std::endl;
        }
    else if (vsCmd->equals("put"))
        {
        // perform a put operation on the current cache
        Object::View vKey   = StreamParser::next(ssInput);
        Object::View vValue = StreamParser::next(ssInput);
        Object::View vPrev  = hCache->put(vKey, vValue);

        // print the old value
        std::cout << vPrev << std::endl;
        }
    else if (vsCmd->equals("remove"))
        {
        // perform a remove operation on the current cache
        Object::View vKey  = StreamParser::next(ssInput);
        Object::View vPrev = hCache->remove(vKey);

        // print the removed value
        std::cout << vPrev << std::endl;
        }
    else if (vsCmd->equals("release"))
        {
        // release the current cache
        CacheFactory::releaseCache(hCache);
        }
    else if (vsCmd->equals("size"))
        {
        // print the size of the current cache
        size32_t cElements = hCache->size();

        std::cout << cElements << std::endl;
        }
```

```
            else if (vsCmd->equals("threads"))
                {
                // print a stack trace for all threads related to coherence
                Thread::dumpStacks(std::cout);
                }
            else if (vsCmd->equals(""))
                {
                continue;
                }
            else if (vsCmd->equals("help"))
                {
                // print help
                std::cout << "The commands are:"
                    << std::endl << "  bye"
                    << std::endl << "  cache <name>"
                    << std::endl << "  classes"
                    << std::endl << "  clear"
                    << std::endl << "  destroy"
                    << std::endl << "  get <key>"
                    << std::endl << "  help"
                    << std::endl << "  list"
                    << std::endl << "  memory [delta]"
                    << std::endl << "  pof"
                    << std::endl << "  put <key> <value>"
                    << std::endl << "  release"
                    << std::endl << "  remove <key>"
                    << std::endl << "  size"
                    << std::endl << "  threads"
                    << std::endl;
                }
            else
                {
                std::cout << "Unknown command: \"" << vsCmd << "\"\n"
                    << "Entry \"help\" for command list" << std::endl;
                }
            }
        catch (const NullPointerException::Throwable& e)
            {
            if (NULL == hCache)
                {
                std::cerr << "Please specify a cache using the \"cache\" "
                          << "command." << std::endl;
                }
            else
                {
                std::cerr << e << std::endl;
                }
            }
        catch (const std::exception& e)
            {
            std::cerr << "Error: " << e.what() << std::endl;
            }
        }
    }
```

Sample Code for the contacts Example

Now that you've run the contacts example, you are encouraged to have a look at the code. Each sample has a corresponding directory under examples which contains its sample specific source. There is also a common directory which contains source used in all samples.

Example A–2 illustrates the source code for the contacts example, and demonstrates how to store pre-existing (that is, non-Coherence) C++ classes in the grid.

ContactInfo.hpp

Example A–2 Header Code for the ContactInfo Sample Application

```
#ifndef COH_EXAMPLES_CONTACT_INFO_HPP
#define COH_EXAMPLES_CONTACT_INFO_HPP

#include <ostream>
#include <string>

/**
* The ContactInfo class encapsulates common contact information for a person.
*
* This serves as an example data object which does not have direct knowledge
* of Coherence but can be stored in the data grid.
*/
class ContactInfo
    {
    // ----- constructors ---------------------------------------------------

    public:
        /**
        * Create a new ContactInfo object.
        *
        * @param sName    the name of the person
        * @param sStreet  the street on which the person lives
        * @param sCity    the city where the person lives
        * @param sState   the state where the person lives
        * @param sZip     the zip code of the city where the person lives
        */
        ContactInfo(const std::string& sName,
                const std::string& sStreet, const std::string& sCity,
                const std::string& sState, const std::string& sZip);

        /**
        * Copy constructor.
        */
        ContactInfo(const ContactInfo& that);

    protected:
        /**
        * Default constructor.
        */
        ContactInfo();


    // ----- accessors ------------------------------------------------------

    public:
        /**
```

```
* Determine the name of the person for which this ContactInfo object
* contains contact information.
*
* @return the person's name
*/
std::string getName() const;

/**
* Configure the name of the person for which this ContactInfo object
* contains contact information.
*
* @param sName  the person's name
*/
void setName(const std::string& sName);

/**
* Determine the street on which the person lives.
*
* @return the street name
*/
std::string getStreet() const;

/**
* Configure the street on which the person lives.
*
* @param sStreet  the street name
*/
void setStreet(const std::string& sStreet);

/**
* Determine the city in which the person lives.
*
* @return the city name
*/
std::string getCity() const;

/**
* Configure the city in which the person lives.
*
* @param sCity  the city name
*/
void setCity(const std::string& sCity);

/**
* Determine the state in which the person lives.
*
* @return the state name
*/
std::string getState() const;

/**
* Configure the state in which the person lives.
*
* @param sState  the state name
*/
void setState(const std::string& sState);

/**
* Determine the zip code of the city in which the person lives.
*
```

```
        * @return the zip code
        */
        std::string getZip() const;

        /**
        * Configure the zip code of the city in which the person lives.
        *
        * @param sZip  the city's zip code
        */
        void setZip(const std::string& sZip);


    // ----- operators ------------------------------------------------------

    public:
        /**
        * Compare two ContactInfo objects for equality
        *
        * @param that  the ContactInfo to compare against
        *
        * @return true if this referenced contact is equal to this contact
        */
        bool operator==(const ContactInfo& that) const;


    // ----- data members ---------------------------------------------------

    private:
        /**
        * The person's name.
        */
        std::string m_sName;

        /**
        * The street on which the person lives.
        */
        std::string m_sStreet;

        /**
        * The city in which the person lives.
        */
        std::string m_sCity;

        /**
        * The state in which the person lives.
        */
        std::string m_sState;

        /**
        * The zip code of the city in which the person lives.
        */
        std::string m_sZip;
    };

/**
* Output this ContactInfo to the stream
*
* @param out  the stream to output to
*
* @return the stream
```

```
*/
std::ostream& operator<<(std::ostream& out, const ContactInfo& info);

#endif // COH_EXAMPLES_CONTACT_INFO_HPP
```

ContactInfo.cpp

Example A–3 C++ Code for the ContactInfo Sample Application

```
#include "ContactInfo.hpp"


// ----- constructors ---------------------------------------------------

ContactInfo::ContactInfo(const std::string& sName,
        const std::string& sStreet, const std::string& sCity,
        const std::string& sState, const std::string& sZip)
    {
    setName(sName);
    setStreet(sStreet);
    setCity(sCity);
    setState(sState);
    setZip(sZip);
    }

ContactInfo::ContactInfo(const ContactInfo& that)
    {
    setName(that.getName());
    setStreet(that.getStreet());
    setCity(that.getCity());
    setState(that.getState());
    setZip(that.getZip());
    }

ContactInfo::ContactInfo()
    {
    }

// ----- accessors ------------------------------------------------------

std::string ContactInfo::getName() const
    {
    return m_sName;
    }

void ContactInfo::setName(const std::string& sName)
    {
    m_sName = sName;
    }

std::string ContactInfo::getStreet() const
    {
    return m_sStreet;
    }

void ContactInfo::setStreet(const std::string& sStreet)
    {
    m_sStreet = sStreet;
    }
```

```
std::string ContactInfo::getCity() const
    {
    return m_sCity;
    }

void ContactInfo::setCity(const std::string& sCity)
    {
    m_sCity = sCity;
    }

std::string ContactInfo::getState() const
    {
    return m_sState;
    }

void ContactInfo::setState(const std::string& sState)
    {
    m_sState = sState;
    }

std::string ContactInfo::getZip() const
    {
    return m_sZip;
    }

void ContactInfo::setZip(const std::string& sZip)
    {
    m_sZip = sZip;
    }


// ----- operators ------------------------------------------------------------

bool ContactInfo::operator==(const ContactInfo& that) const
    {
    return getName()   == that.getName()   &&
           getStreet() == that.getStreet() &&
           getCity()   == that.getCity()   &&
           getState()  == that.getState()  &&
           getZip()    == that.getZip();
    }

std::ostream& operator<<(std::ostream& out, const ContactInfo& info)
    {
    out << "ContactInfo("
            << "Name="     << info.getName()
            << ", Street=" << info.getStreet()
            << ", City="   << info.getCity()
            << ", State="  << info.getState()
            << ", Zip="    << info.getZip()
            << ')';
    return out;
    }
```

PortableContactInfo.hpp

Example A–4 Header Code for the PortableContactInfo Applications

```
#ifndef COH_EXAMPLES_PORTABLE_CONTACT_INFO_HPP
#define COH_EXAMPLES_PORTABLE_CONTACT_INFO_HPP
```

```
// This set of functions add support for storing ContactInfo objects in
// Coherence without introducing knowledge of Coherence to the ContactInfo
// class.  This portable version of ClassInfo can be referred to as
// Portable<ContactInfo>

#include "ContactInfo.hpp"

#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/Portable.hpp"

/**
* The POF type id for Portable<ContactInfo>
*/
#define POF_CONTACT_INFO_ID 1001

/**
* Serialize a ContactInfo object to a POF stream.
*
* @param info  the ContactInfo to seralize
* @param hOut  the PofWritter to write to
*/
void serialize(const ContactInfo& info, coherence::io::pof::PofWriter::Handle
hOut);

/**
* Deserialize a ContactInfo object from a POF stream.
*
* @param info  the ContactInfo to seralize
* @param hIn   the PofReader to read from
*/
void deserialize(ContactInfo& info, coherence::io::pof::PofReader::Handle hIn);

/**
* Return a hashcode code for a ContactInfo object.
*/
coherence::lang::size32_t hash(const ContactInfo& info);

#endif // COH_EXAMPLES_PORTABLE_CONTACT_INFO_HPP
```

PortableContactInfo.cpp

Example A–5 C++ Code for the PortableContactInfo Application

```
#include "PortableContactInfo.hpp"

#include "coherence/lang.ns"

#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/Portable.hpp"
#include "coherence/io/pof/SystemPofContext.hpp"

using namespace coherence::lang;


// register Portable<ContactInfo> with the SystemPofContext
COH_REGISTER_PORTABLE_CLASS(POF_CONTACT_INFO_ID,
coherence::io::pof::Portable<ContactInfo>);
```

```
void serialize(const ContactInfo& info, coherence::io::pof::PofWriter::Handle
hOut)
    {
    hOut->writeString(0, info.getName());
    hOut->writeString(1, info.getStreet());
    hOut->writeString(2, info.getCity());
    hOut->writeString(3, info.getState());
    hOut->writeString(4, info.getZip());
    }

void deserialize(ContactInfo& info, coherence::io::pof::PofReader::Handle hIn)
    {
    info.setName  (hIn->readString(0));
    info.setStreet(hIn->readString(1));
    info.setCity  (hIn->readString(2));
    info.setState (hIn->readString(3));
    info.setZip   (hIn->readString(4));
    }

size32_t hash(const ContactInfo& info)
    {
    // ContactInfo is not used as a key, use identity hash
    return size32_t(size_t(&info));
    }
```

contacts.cpp

Example A–6 Code for the ContactInfo Data Object

```
#include "coherence/lang.ns"

#include "coherence/io/pof/Portable.hpp"
#include "coherence/net/CacheFactory.hpp"
#include "coherence/net/NamedCache.hpp"

#include "ContactInfo.hpp"
#include "PortableContactInfo.hpp"
#include "StreamParser.hpp"

#include <iostream>
#include <sstream>

using namespace coherence::lang;

using coherence::examples::StreamParser;
using coherence::io::pof::Portable;
using coherence::net::CacheFactory;
using coherence::net::NamedCache;

// ----- prototypes ------------------------------------------------------------

/**
* Create a contact from stdin.
*
* @return the contact
*/
ContactInfo readContact();

/**
```

```
* This Coherence for C++ example illustrates how to use non-Coherence data
* objects in the grid. This example operates on the ContactInfo class which
* is not Coherence aware.
*
* To run this against a remote cache, the proxy node must have the
* corresponding Java ContactInfo.class in its classpath.
*
* @argc  the number of command line arguments (including the process name)
* @argv  [cache-name]
*/
int main(int argc, char** argv)
    {
    try
        {
        String::View       vsCacheName = argc > 1 ? argv[1] : "dist-contacts";
        NamedCache::Handle hCache      = CacheFactory::getCache(vsCacheName);

        while (true)
            {
            // prompt for input
            std::cout << "contacts> " << std::flush;

            char achInput[256];
            std::cin.getline(achInput, 256);
            std::stringstream ssInput(achInput);

            // process input
            String::View vsCmd = cast<String::View>(
                    StreamParser::next(ssInput));

            if (vsCmd->equals("bye"))
                {
                // quit
                CacheFactory::shutdown();
                return 0;
                }
            else if (vsCmd->equals("create"))
                {
                ContactInfo ci = readContact();
                std::cout << "storing: " << ci << std::endl;
                hCache->put(String::create(ci.getName().c_str()),
                        Portable<ContactInfo>::create(ci));
                }
            else if (vsCmd->equals("find"))
                {
                String::View vsPart = cast<String::View>(
                        StreamParser::next(ssInput));

                std::cout << "Name: " << std::flush;
                std::cin.getline(achInput, 256);
                String::View vsName = achInput;

                Portable<ContactInfo>::View vInfo =
                    cast<Portable<ContactInfo>::View>(hCache->get(vsName));

                if (NULL == vInfo)
                    {
                    std::cout << vsName << " not found" << std::endl;
                    continue;
                    }
```

```
                    if (vsPart->equals("all") || vsPart->equals(""))
                        {
                        std::cout << vInfo << std::endl;
                        }
                    else if (vsPart->equals("street"))
                        {
                        std::cout << vInfo->getStreet() << std::endl;
                        }
                    else if (vsPart->equals("city"))
                        {
                        std::cout << vInfo->getCity() << std::endl;
                        }
                    else if (vsPart->equals("state"))
                        {
                        std::cout << vInfo->getState() << std::endl;
                        }
                    else if (vsPart->equals("zip"))
                        {
                        std::cout << vInfo->getZip() << std::endl;
                        }
                    else
                        {
                        std::cerr << "find must be followed by, street, city, "
                            << "state, or zip" << std::endl;
                        }
                    }
                else // output help
                    {
                    std::cout << "commands are:"
                        << std::endl << "bye"
                        << std::endl << "create"
                        << std::endl << "find <street | city | state | zip | all>"
                        << std::endl;
                    }
                }
            }
        catch (const std::exception& e)
            {
            std::cerr << e.what() << std::endl;
            }
        }

    ContactInfo readContact()
        {
        char achInput[256];
        std::cout << "Name: " << std::flush;
        std::cin.getline(achInput, 256);
        std::string sName(achInput);

        std::cout << "Street: " << std::flush;
        std::cin.getline(achInput, 256);
        std::string sStreet(achInput);

        std::cout << "City: " << std::flush;
        std::cin.getline(achInput, 256);
        std::string sCity(achInput);

        std::cout << "State: " << std::flush;
        std::cin.getline(achInput, 256);
```

```
    std::string sState(achInput);

    std::cout << "Zip: " << std::flush;
    std::cin.getline(achInput, 256);
    std::string sZip(achInput);

    return ContactInfo(sName, sStreet, sCity, sState, sZip);
    }
```

Sample Code for the hellogrid Example

Now that you've run the hellogrid samples, you are encouraged to have a look at the code. Each sample has a corresponding directory under examples which contains its sample specific source. There is also a common directory which contains source used in all samples.

Example A–7 illustrates the source code for the `hellogrid.cpp` example, and demonstrates basic cache access.

Example A–7 Code for the HelloGrid Sample Application

```
#include "coherence/lang.ns"

#include "coherence/net/CacheFactory.hpp"
#include "coherence/net/NamedCache.hpp"
#include "coherence/stl/boxing_map.hpp"
#include "coherence/util/aggregator/ComparableMin.hpp"
#include "coherence/util/extractor/IdentityExtractor.hpp"
#include "coherence/util/filter/GreaterFilter.hpp"
#include "coherence/util/processor/NumberIncrementor.hpp"
#include "coherence/util/Iterator.hpp"
#include "coherence/util/Filter.hpp"
#include "coherence/util/Set.hpp"
#include "coherence/util/ValueExtractor.hpp"
#include "coherence/util/ValueManipulator.hpp"

#include <iostream>

using namespace coherence::lang;

using coherence::net::CacheFactory;
using coherence::net::NamedCache;
using coherence::stl::boxing_map;
using coherence::util::aggregator::ComparableMin;
using coherence::util::extractor::IdentityExtractor;
using coherence::util::filter::GreaterFilter;
using coherence::util::processor::NumberIncrementor;
using coherence::util::Iterator;
using coherence::util::Filter;
using coherence::util::Set;
using coherence::util::ValueExtractor;
using coherence::util::ValueManipulator;


/**
* This example demonstrates the basics of accessing a cache by using the
* Coherence C++ API.
*
* @argc  the number of command line arguments (including the process name)
```

```
* @argv  [cache-name]
*/
int main(int argc, char** argv)
    {
    try
        {
```

Basic Cache Access

```
// read optional cache name from command line
        String::View vsCacheName = argc > 1 ? argv[1] : "dist-hello";

        // retrieve the named cache
        NamedCache::Handle hCache = CacheFactory::getCache(vsCacheName);
        std::cout << "retrieved cache \"" << hCache->getCacheName()
                  << "\" containing " << hCache->size() << " entries"
                  << std::endl;

        // create a key, and value
        String::View vsKey   = "hello";
        String::View vsValue = "grid";

        // insert the pair into the cache
        hCache->put(vsKey, vsValue);
        std::cout << "\tput: " << vsKey << " = " << vsValue << std::endl;

        // read back the value, casting to the expected value type
        String::View vsGet = cast<String::View>(hCache->get(vsKey));
        std::cout << "\tget: " << vsKey << " = " << vsGet << std::endl;

        // read a non-existent entry from the cache; result will be NULL
        String::View vsKeyDummy = "dummy";
        Object::View vDummy     = hCache->get(vsKeyDummy);
        std::cout << "\tget: " << vsKeyDummy << " = " << vDummy << std::endl;

        // work with non-string data types
        hCache->put(Integer32::valueOf(12345), Float64::valueOf(6.7));
        hCache->put(Integer32::valueOf(23456), Float64::valueOf(7.8));
        hCache->put(Integer32::valueOf(34567), Float64::valueOf(8.9));

        // iterate and print the cache contents, treating contents abstractly
        std::cout << "entire cache contents:" << std::endl;
        for (Iterator::Handle hIter = hCache->entrySet()->iterator();
             hIter->hasNext(); )
            {
            Map::Entry::View vEntry = cast<Map::Entry::View>(hIter->next());
            Object::View     vKey   = vEntry->getKey();
            Object::View     vValue = vEntry->getValue();
            std::cout << '\t' << vKey << " = " << vValue << std::endl;
            }

        // remove strings to make the cache contents uniform
        hCache->remove(vsKey);
```

STL-like Map Adapter

```
// caches may also be wrapped with an STL-like map adapter
        typedef boxing_map<Integer32, Float64> float_cache;
        float_cache cache(hCache);
        cache[45678] = 9.1;
```

```
        std::cout << "updated cache contents:" << std::endl;
        for (float_cache::iterator i = cache.begin(), e = cache.end(); i != e;
++i)
            {
            std::cout << '\t' << i->first << " = " << i->second << std::endl;
            }
```

InvocableMap Aggregation

```
// perform aggregation, and print the results
        ValueExtractor::View vExtractor = IdentityExtractor::getInstance();
        Float64::View        vFlMin     = cast<Float64::View>(
                hCache->aggregate((Filter::View) NULL,
                ComparableMin::create(vExtractor)));
        std::cout << "minimum: " << vFlMin << std::endl;
```

Query the Cache

```
// query the cache, and print the results
        Filter::View vFilter    = GreaterFilter::create(vExtractor,
                                                        Float64::valueOf(7.0));
        Set::View    vSetResult = hCache->entrySet(vFilter);

        std::cout << "filtered cache contents by " << vFilter << std::endl;
        for (Iterator::Handle hIter = vSetResult->iterator(); hIter->hasNext(); )
            {
            Map::Entry::View vEntry = cast<Map::Entry::View>(hIter->next());
            Object::View     vKey   = vEntry->getKey();
            Object::View     vValue = vEntry->getValue();
            std::cout << '\t' << vKey << " = " << vValue << std::endl;
            }
```

Continuous Query Cache

```
// present a real-time filtered view of the cache
        NamedCache::Handle hCacheCqc =
            ContinuousQueryCache::create(hCache, vFilter);
        std::cout << "ContinuousQueryCache filtered view: " << std::endl;
        for (Iterator::Handle hIter = hCacheCqc->entrySet()->iterator();
             hIter->hasNext(); )
            {
            Map::Entry::View vEntry = cast<Map::Entry::View>(hIter->next());
            Object::View     vKey   = vEntry->getKey();
            Object::View     vValue = vEntry->getValue();
            std::cout << '\t' << vKey << " = " << vValue << std::endl;
            }

        // register MapListener to print changes to stdout
        std::cout << "start listening to events..." << std::endl;
        hCache->addFilterListener(VerboseMapListener::create());
```

InvocableMap Invoke All

```
// invoke entry processor on matching cache contents, incrementing each value
        Float64::Handle vFlIncr = Float64::valueOf(1.0);
        std::cout << "increment results by " << vFlIncr << std::endl;
        hCacheCqc->invokeAll((Filter::View) NULL, NumberIncrementor::create(
                (ValueManipulator::View) NULL, vFlIncr, /*fPost*/ true));
```

```
    // stop the CQC event queue thread and remove listeners
    hCacheCqc->release();
    // disconnect from the grid
    CacheFactory::shutdown();
    }
catch (const std::exception& e)
    {
    std::cerr << "error: " << e.what() << std::endl;
    return 1;
    }
return 0;
}
```