Oracle® Coherence

Client Guide

Release 3.6.1

E15726-03

December 2010

Provides detailed instructions for developing Coherence*Extend clients in various programming languages.

ORACLE®

Oracle Coherence Client Guide, Release 3.6.1

E15726-03

Primary Author: Joseph Ruzzi

Contents

10 Using the Coherence C++ Object Model

List of Examples

List of Figures

List of Tables

Preface

Welcome to *Oracle Coherence Client Guide*. This document provides detailed instructions for developing Coherence*Extend clients in various programming languages.

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 remote caches that reside in a Coherence cluster. The documentation assumes users are familiar with these respective technologies. In addition, users must be familiar with Java when serializing data to the cluster.

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.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

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

- *Oracle Coherence Developer's Guide*
- *Oracle Coherence Getting Started Guide*
- *Oracle Coherence Integration Guide for Oracle Coherence*
- *Oracle Coherence Tutorial for Oracle Coherence*
- *Oracle Coherence User's Guide for Oracle Coherence*Web*
- *Oracle Coherence Java API Reference*
- *Oracle Coherence C++ API Reference*
- *Oracle Coherence .NET API Reference*
- *Oracle Coherence Release Notes 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

Getting Started

Part I contains the following chapters:

1 Introduction

Coherence*Extend "extends" the reach of the core Coherence TCMP cluster to a wider range of consumers, including desktops, remote servers, and machines located across WAN connections. Typical uses of Coherence*Extend include providing desktop applications with access to Coherence caches (including support for Near Cache and Continuous Query) and linking together multiple Coherence clusters connected through a high-latency, unreliable WAN.

The following sections are included in this chapter:

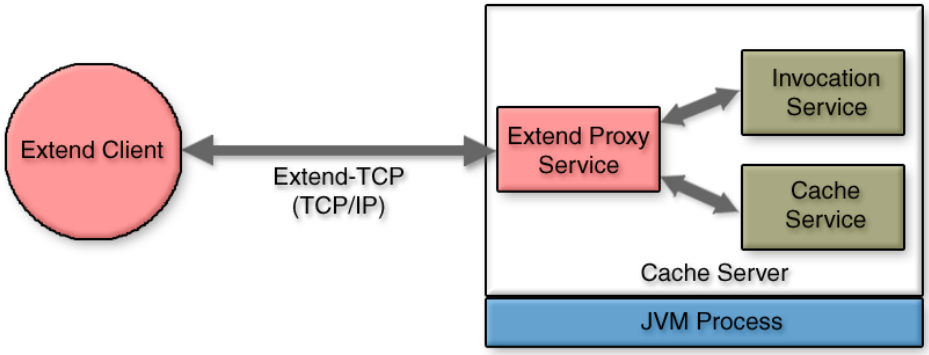
Components Overview

Coherence*Extend consists of two basic components: an extend client running outside the cluster and an extend proxy service running in the cluster hosted by one or more cache servers (`DefaultCacheServer`). The client APIs includes implementations of both the `CacheService` and `InvocationService` interfaces which route all requests to the proxy. The proxy responds to client requests by delegating to an actual Coherence clustered services (for example, a partitioned or replicated cache service or an invocation service).

Coherence*Extend uses the Extend-TCP transport binding (a low-level messaging protocol) to communicate between the client and the cluster. The protocol is a high performance, scalable TCP/IP-based communication layer. The transport binding is configuration-driven and is completely transparent to the client application that uses Coherence*Extend.

Figure 1–1 provides a conceptual view of the Coherence*Extend components and shows an extend client connecting to an extend proxy service using Extend-TCP.

*Figure 1–1 Conceptual View of Coherence*Extend Components*



Like cache clients, an extend client retrieves Coherence clustered service using a cache factory. Once a service is obtained, a client uses the service in the same way as if it were part of the Coherence cluster. The fact that operations are being sent to a remote cluster node is transparent to the client application.

Types Of Clients

Extend clients can be created for the Java, .NET, and C++ platforms and have access to the same rich API as the standard Coherence API without being full data members of the cluster. Typically, client applications are granted only read access to cluster data, although it is possible to enable direct read/write access. There are two categories of clients: Data Clients and Real Time Extend Clients.

Data Clients

Data clients are extend clients that are able to access (put, get, query) data in the cluster and also make invocation service requests using standard Coherence APIs. In particular, data clients provide:

- Key-based cache access through the `NamedCache` interface
- Attribute-based cache access using filters
- Custom processing and aggregation of cluster side entries using the `InvocableMap` interface
- In-Process caching through `LocalCache`
- Remote invocation of custom tasks in the cluster through the Invocation Service

Extend clients are licensed as data clients and can be used with any Coherence server edition (Standard, Enterprise, or Grid). For a complete list of Data Client features, see "Oracle Coherence" in the *Oracle Fusion Middleware Licensing Information* book.

Note: Data clients cannot be notified of changes to data in a cluster. Further, data clients do not have the ability to use Near Caches or Continuous Query caches, as those capabilities also rely on the ability to receive notifications of data changes from the cluster. For these capabilities, real-time clients must be used.

Real Time Clients

Real Time Clients (Extend-TCP) provides the same capabilities associated with data clients; But, unlike data clients, a real-time client also supports:

- **Event Notifications** – Using the standard Coherence event model, data changes that occur within the cluster are visible to the client application. Only events that a client application registers for are delivered over the wire. This model results in efficient use of network bandwidth and client processing.
- **Local Caches** – While the client application can directly access the caches managed by the cluster, that may be inefficient depending on the network infrastructure. For efficiency, a real-time client can use both Near Caching and Continuous Query Caching to maintain cache data locally. If the server to which the client application is attached happens to fail, the connection is automatically reestablished to another server, and any locally cached data is re-synchronized with the cluster.

Extend clients are licensed as Real Time Clients and can only be used with Coherence Grid server edition. For a complete list of Real Time Client features, see "Oracle Coherence" in the *Oracle Fusion Middleware Licensing Information* book.

Client APIs

Java, C++, and .NET (C#) native libraries are available for building extend clients. Each API is delivered in its own distribution and must be installed separately. Extend clients use their respective APIs to perform cache operations such as access, modify, and query data that is in a cluster. The C++ and C# APIs follow the Java API as close as possible in order to provide a consistent experience between platforms.

As an example, a Java client gets a `NamedCache` instance using the `CacheFactory.getCache` method as follows:

```
NamedCache cache = CacheFactory.getCache("dist-extend");
```

For C++, the API is as follows:

```
NamedCache::Handle hCache = CacheFactory::getCache("dist-extend");
```

For C#, the API is as follows:

```
INamedCache cache = CacheFactory.GetCache("dist-extend");
```

This and many other API features are discussed throughout this guide:

- Java – See Part II, "Creating Java Extend Clients" for details on using the API and refer to *Oracle Coherence Java API Reference* for detailed API documentation.
- C++ – See Part III, "Creating C++ Extend Clients" for details on using the API and refer to *Oracle Coherence C++ API Reference* for detailed API documentation.
- .NET – See Part IV, "Creating .NET Extend Clients" for details on using the API and refer to *Oracle Coherence .NET API Reference* for detailed API documentation.

POF Serialization

Like cache clients, extend clients must serialize objects that are to be stored in the cluster. C++ and C# clients use Coherence's Portable Object Format (POF), which is a language agnostic binary format. Java extend clients typically use POF for serialization as well; however, there are several other options for serializing Java objects, such as

Java native serialization and custom serialization routines. See "Serializing Objects" in the *Oracle Coherence Developer's Guide*.

Clients that serialize objects into the cluster can perform get and put based operations on the objects. However, features such as queries and entry processors require Java-based cache servers to interact with the data object, rather then simply holding onto a serialized representation of it. To interact with the object and access its properties, a Java version of the object must be made available to the cache servers.

For detailed information on using POF with Java, see "Using Portable Object Format" in the *Oracle Coherence Developer's Guide*. For more information on using POF with C++ and C#, see Chapter 11, "Building Integration Objects for C++ Clients," and Chapter 20, "Building Integration Objects for .NET Clients," respectively.

Understanding Client Configuration Files

Extend clients are configured using several configurations files. The configuration files are the same as the cluster configuration files. However, client configuration files are deployed with the client. The files include:

- Cache Configuration Deployment Descriptor – This file is used to define client-side cache services and invocation services and must provide the address and port of the cluster-side extend proxy service to which the client connects. The DTD for this file is the `cache-config.dtd` file. For a complete reference of the elements in this file, see Appendix B, "Cache Configuration Elements," in the *Oracle Coherence Developer's Guide*.

  At run time, the first cache configuration file that is found on the classpath is used. The `tangosol.coherence.cacheconfig` system property can also be used to explicitly specify a cache configuration file. The file can also be set programmatically. For general information about the cache configuration deployment descriptor, see "Specifying a Cache Configuration File" in the *Oracle Coherence Developer's Guide*.

- POF Configuration Deployment Descriptor – This file is used to specify custom data types when using POF to serialize objects. The DTD for this file is the `pof-config.dtd` file. For a complete reference of the elements in this file, see Appendix D, "POF User Type Configuration Elements," in the Oracle Coherence Developer's Guide.

  At run time, the first POF configuration file that is found on the classpath is used. The `tangosol.pof.config` system property can also be used to explicitly specify a POF configuration file. When using POF, a client application uses a Coherence-specific POF configuration file together with a POF configuration file that is specific to the user types used in the client. For general information about the POF configuration deployment descriptor, see "Specifying a POF Configuration File" in the *Oracle Coherence Developer's Guide*.

- Operational Override File – This file is used to override the operational deployment descriptor, which is used to specify the operational and runtime settings that are used to create, configure and maintain clustering, communication, and data management services. For extend clients, this file is typically used to override member identity, logging, security, and licensing. The DTD for this file is the `coherence.dtd` file. For a complete reference of the elements in this file, see Appendix A, "Operational Configuration Elements," in the *Oracle Coherence Developer's Guide*.

  At run time, the first operational override file (`tangosol-coherence-override.xml`) that is found on the classpath is used.

The `tangosol.coherence.override` system property can also be used to explicitly specify an operational override file. The file can also be set programmatically. For general information about the operational override file, see "Using the Default Operational Override File" in the *Oracle Coherence Developer's Guide*.

2

Installing a Client Distribution

This chapter provides instructions for installing the C++ and .NET client distributions. There is no separate Java client distribution package. Java extend clients are created using Coherence for Java.

The following sections are included in this chapter:

Installing Coherence for Java

The Coherence for Java distribution is used to build and use Java-based extend clients. To install Coherence for Java, see "Installing Oracle Coherence for Java" in the *Oracle Coherence Developer's Guide*.

Installing the C++ Client Distribution

The Oracle Coherence for C++ distribution is used to develop and run C++ extend clients. The latest version of the distribution can be downloaded from the Coherence product page on the Oracle Technology Network:

`http://www.oracle.com/technology/software/products/ias/htdocs/coherence.html`

This section contains the following topics:

Supported Environments

Table 2–1 lists the supported platforms and operating systems for Coherence for C++:

Table 2–1 Platform and Operating System Support for Coherence for C++

Operating System	Compiler	Architecture
Microsoft Windows 2000+[1]	MSVC 2005 SP1+[2], MSVC 2010	x86
Microsoft Windows Server 2003+[3]	MSVC 2005 SP1+[2], MSVC 2010	x64

Table 2–1 (Cont.) Platform and Operating System Support for Coherence for C++

Operating System	Compiler	Architecture
Sun Solaris 10	SunPro 5.9[4]	SPARC
Sun Solaris 10	SunPro 5.9[5]	x86
Sun Solaris 10	SunPro 5.9[5]	x64
Linux	GCC 3.4+[6]	x86
Linux	GCC 3.4+[6]	x64
Apple OS X 10.4+[7]	GCC 3.4+[6]	x86
Apple OS X 10.4+[8]	GCC 3.4+[6]	x64

[1] Including Windows 32b XP, Vista, 2000, 2003, and 2008.
[2] Specifically MSVC 2005 SP1 (14.00.50727.762+), as well as MSVC 2008 and express versions are supported.
[3] Including Windows 64b XP, Vista, 2003, and 2008.
[4] Specifically Sun C++ 5.9 SPARC Patch 124863-14 or later are supported.
[5] Specifically Sun C++ 5.9 x86/x64 Patch 124864-14 or later are supported.
[6] Specifically GCC 3.4.6-8 and above, as well as GCC 4.x versions are supported.
[7] Including OS X Tiger (10.4), Leopard (10.5), and Snow Leopard (10.6)
[8] Including OS X Leopard (10.5), and Snow Leopard (10.6)

Microsoft-Specific Requirements

When deploying on Microsoft Windows, just as with any MSVC based application, the corresponding MSVC runtime libraries must be installed on the deployment machine.

- **Visual Studio 2005 SP1 and Visual Studio 2008**: The following redistributable runtime libraries for x86 or x64 are required. If developing with Visual Studio 2008, the 2005 SP1 redistributable libraries must still be installed on both the development and deployment machines.

  x86:

  `http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en`

  x64:

  `http://www.microsoft.com/downloads/details.aspx?familyid=EB4EBE2D-33C0-4A47-9DD4-B9A6D7BD44DA&displaylang=en`

  The use of Oracle Coherence for C++ with MSVC 2005 SP1 (x86 and x64) requires both the Microsoft Visual C++ 2005 Service Pack 1 Redistributable and the Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package ATL Security Update. Coherence will not run without the security update. For more information on the security update, see

  `http://support.microsoft.com/?kbid=973544`

  The security update is available from the Microsoft Update Web site or directly from

  `http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=766a6af7-ec73-40ff-b072-9112bab119c2`

- **Visual Studio 2010**: Redistributable runtime libraries for x86 or x64.

  x86:

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=a7b7a05e-6de6-4d3a-a423-37bf0912db84

x64:

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=bd512d9e-43c8-4655-81bf-9350143d5867

Extracting the Coherence for C++ Distribution

Coherence for C++ is distributed as a ZIP file. Use a ZIP utility or the `unzip` command-line utility to extract the ZIP file to a location on the target computer. The extracted files are organized within a single directory called `coherence-cpp`.

The following example uses the `unzip` utility to extract the distribution to the `/opt` directory which is the suggested installation directory on UNIX-based operating systems. Use the ZIP utility provided with the target operating system if the `unzip` utility is not available.

```
unzip /path_to_zip/coherence-cpp-version_number-platform-architecture-compiler.zip
-d /opt
```

The following example extracts the distribution using the `unzip` utility to the `C:\` directory on the Windows operating system.

```
unzip C:\path_to_zip\coherence-cpp-version_
number-platform-architecture-compiler.zip -d C:\
```

The following list describes the directories that are included in installation directory:

- `bin` – This directory includes `sanka.exe`, which is an application launcher that is used to invoke executable classes embedded within a shared library.
- `doc` – This directory contains Coherence for C++ documentation including the API documentation
- `examples` – This directory includes examples that demonstrate basic functionality.
- `include` – This directory contains header files that use the Coherence API and must be compiled together with an application.
- `lib` – This directory includes the Coherence for C++ library. The `coherence.dll` file is the main development and runtime library and is discussed in detail throughout this documentation.

Installing the .NET Client Distribution

The Oracle Coherence for .NET distribution is used to develop and use .NET extend clients. The latest version of the distribution can be downloaded from the Coherence product page on the Oracle Technology Network:

http://www.oracle.com/technology/software/products/ias/htdocs/coherence.html

This section contains the following topics:

Prerequisites

The following are required to use Coherence for .NET:

- Microsoft .NET 2.0, 3.0, or 3.5 Runtime
- Microsoft .NET 2.0, 3.0, or 3.5 SDK
- Supported Microsoft Windows operating system (see the system requirements for the appropriate .NET Runtime above)
- MSHelp 2.x Runtime, which is included in Visual Studio 200x as well as the Microsoft products listed here:

  http://www.helpware.net/mshelp2/h20.htm#MS_H2_Runtime

- Microsoft Visual Studio 2005 is required to build and run the examples included with Coherence for .NET:

Running the Installer

Coherence for .NET is distributed as a ZIP file which contains an installer. Use a ZIP utility or the `unzip` command-line utility to extract the installer to a location on the target computer. The following example extracts the installer using the `unzip` utility to the `C:\` directory:

```
unzip C:\path_to_zip\coherence-net-version_number.zip -d C:\
```

To run the installer:

1. From the directory where the ZIP was extracted, double-click the `coherence-net-version.msi` file.
2. Follow the instructions in the installer to complete the installation.

> **Note:** If the installer indicates that it is rolling back the installation, then run the installer in elevated execution mode. For example, executing the MSI file from a command prompt that was started as an Administrator should enable the installation process to complete. For Windows 7, right-click the command prompt and select **run as Administrator**.

The following list describes the directories that are included in the installation directory:

- `bin` – This directory includes the Coherence for .NET library. The `Coherence.dll` file is the main development and runtime library and is discussed in detail throughout this documentation. A version of the library is included for .NET 2.0 and higher.
- `config` – This directory contains XML schemas for Coherence configuration files and also includes a POF configuration file for Coherence-defined user types.
- `doc` – This directory contains Coherence for .NET documentation including the API documentation. The API documentation is available as a compiled HTML Help (`Coherence.chm`) or as MSHelp 2.0 Help.
- `examples` – This directory includes examples that demonstrate basic functionality.

Deploying Coherence for .NET

Coherence for .NET requires no specialized deployment configuration. Simply add a reference to the `Coherence.dll` found in the `bin\net\2.0` folder to your Microsoft.NET application.

3

Setting Up Coherence*Extend

This chapter provides instructions for configuring Coherence*Extend. The instructions provide basic setup and do not represent a complete configuration reference. In addition, refer to the platform-specific parts of this guide for additional configuration instructions. For a complete Java example that also includes configuration and setup, see Chapter 4, "Building Your First Extend Client."

This chapter includes the following sections:

Overview

Coherence*Extend requires configuration both on the client side and the cluster side. On the cluster side, extend proxy services are setup to accept client requests. Proxy services provide access to cache service instances and invocation service instances that are running on the cluster. On the client side, remote cache services and the remote invocation services are configured and used by clients to access cluster data through the extend proxy service. Extend clients and extend proxy services communicate using TCP/IP.

Extend proxy services are configured in a cache configuration deployment descriptor. This deployment descriptor is often referred to as the cluster-side cache configuration file. It is the same cache configuration file that is used to set up caches on the cluster. Extend clients are also configured using a cache configuration deployment descriptor. This deployment descriptor is deployed with the client and is often referred to as the client-side cache configuration file. For detailed information about the cache configuration deployment descriptor, see "Specifying a Cache Configuration File" in the *Oracle Coherence Developer's Guide*.

Configuring the Cluster Side

A Coherence cluster must include an extend proxy service in order to accept extend client connections and must include a cache that is used by clients to retrieve and store data. Both the extend proxy service and caches are configured in the cluster's cache configuration deployment descriptor. Extend proxy services and caches are started as part of a cache server (`DefaultCacheServer`) process.

The following topics are included this section:

- Setting Up Extend Proxy Services
- Defining Caches for Use By Extend Clients

Setting Up Extend Proxy Services

The extend proxy service (`ProxyService`) is a cluster service that allows extend clients to access a Coherence cluster using TCP/IP. A proxy service includes proxies for two types of cluster services: the `CacheService` cluster service, which is used by clients to access caches; and, the `InvocationService` cluster service, which is used by clients to execute `Invocable` objects on the cluster.

The following topics are included in this section:

- Defining a Proxy Service
- Defining Multiple Proxy Services
- Disabling Cluster Service Proxies
- Specifying Read-Only NamedCache Access
- Specifying NamedCache Locking

Defining a Proxy Service

Extend proxy services are configured within a `<caching-schemes>` node using the `<proxy-scheme>` element. The `<proxy-scheme>` element has a `<tcp-acceptor>` child element that includes the address (IP or DNS name) and port that an extend proxy service listens to for TCP/IP client communication. See the "proxy-scheme" element reference in the *Oracle Coherence Developer's Guide* for a complete list and description of all `<proxy-scheme>` subelements.

Example 3–1 defines a proxy service named `ExtendTcpProxyService` and is set up to listen for client requests on a TCP/IP `ServerSocket` that is bound to `198.168.1.5` and port `9099`. Both the cache and invocation cluster service proxies are enabled for client requests. In addition, the `<autostart>` element is set to `true` so that the service automatically starts at a cluster node.

Example 3–1 Extend Proxy Service Configuration

```
...
<caching-schemes>
   ...
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <local-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </local-address>
         </tcp-acceptor>
      </acceptor-config>
      <proxy-config>
         <cache-service-proxy>
            <enabled>true</enabled>
         </cache-service-proxy>
         <invocation-service-proxy>
            <enabled>true</enabled>
```

```
         </invocation-service-proxy>
      </proxy-config>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

> **Note:** For clarity, the above example explicitly enables the cache and invocation cluster service proxies. However, both proxies are enabled by default and do not require a `<cache-service-proxy>` and `<invocation-service-proxy>` element to be included in the proxy scheme definition.

Defining Multiple Proxy Services

Any number of extend proxy services can be set up to support an expected number of client connections as well as to support fault tolerance. For more information on fault tolerance, see "Configuring Fault Tolerance for Remote Addresses" on page 3-9.

The following example defines two extend proxy services. `ExtendTcpProxyService1` is set up to listen for client requests on a TCP/IP `ServerSocket` that is bound to `198.168.1.5` and port `9099`. `ExtendTcpProxyService2` is set up to listen for client requests on a TCP/IP `ServerSocket` that is bound to `198.168.1.6` and port `9099`.

```
...
<caching-schemes>
   ...
   <proxy-scheme>
      <service-name>ExtendTcpProxyService1</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <local-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </local-address>
         </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
   </proxy-scheme>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService2</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <local-address>
               <address>192.168.1.6</address>
               <port>9099</port>
            </local-address>
         </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

Disabling Cluster Service Proxies

The cache service and invocation service proxies can be disabled within an extend proxy service definition. Both of these proxies are enabled by default and can be explicitly disabled if a client does not require a service.

Cluster service proxies are disabled by setting the `<enabled>` element to `false` within the `<cache-service-proxy>` and `<invocation-service-proxy>` respectively.

The following example disables the inovcation service proxy so that extend clients cannot execute `Invocable` objects within the cluster:

```
<proxy-scheme>
   ...
   <proxy-config>
      <invocation-service-proxy>
         <enabled>false</enabled>
      </invocation-service-proxy>
   </proxy-config>
   ...
</proxy-scheme>
```

Likewise, the following example disables the cache service proxy to restrict extend clients from accessing caches within the cluster:

```
<proxy-scheme>
   ...
   <proxy-config>
      <cache-service-proxy>
         <enabled>false</enabled>
      </cache-service-proxy>
   </proxy-config>
   ...
</proxy-scheme>
```

Specifying Read-Only NamedCache Access

By default, extend clients are allowed to both read and write data to proxied `NamedCache` instances. The `<read-only>` element can be specified within a `<cache-service-proxy>` element to prohibit extend clients from modifying cached content on the cluster. For example:

```
<proxy-scheme>
   ...
   <proxy-config>
      <cache-service-proxy>
         <read-only>true</read-only>
      </cache-service-proxy>
   </proxy-config>
   ...
</proxy-scheme>
```

Specifying NamedCache Locking

By default, extend clients are not allowed to acquire `NamedCache` locks. The `<lock-enabled>` element can be specified within a `<cache-service-proxy>` element to allow extend clients to perform locking. For example:

```
<proxy-scheme>
   ...
```

```
   <proxy-config>
      <cache-service-proxy>
         <lock-enabled>true</lock-enabled>
      </cache-service-proxy>
   </proxy-config>
   ...
</proxy-scheme>
```

If client-side locking is enabled and a client application uses the `NamedCache.lock()` and `unlock()` methods, it is important that a member-based (rather than thread-based) locking strategy is configured when using a partitioned or replicated cache. The locking strategy is configured using the `<lease-granularity>` element when defining cluster-side caches. A granularity value of `thread` (the default setting) means that locks are held by a thread that obtained them and can only be released by that thread. A granularity value of `member` means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release the lock. Because the extend proxy clustered service uses a pool of threads to execute client requests concurrently, it cannot guarantee that the same thread will execute subsequent requests from the same extend client.

The following example demonstrates setting the lease granularity to `member` for a partitioned cache

```
...
<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>
...
```

Defining Caches for Use By Extend Clients

Extend clients read and write data to a cache on the cluster. Any of the cache types can be used to store client data. For extend clients, the cache on the cluster must have the same name as the cache that is being used on the client side; see "Defining a Remote Cache" on page 3-6. For more information on defining caches, see "Using Caches" in the *Oracle Coherence Developer's Guide*.

The following example defines a partitioned cache named `dist-extend`.

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-extend</cache-name>
         <scheme-name>dist-default</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>dist-default</scheme-name>
         <backing-map-scheme>
```

```
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </distributed-scheme>
   </caching-schemes>
</cache-config>
```

Configuring the Client Side

Extend clients use the remote cache service and the remote invocation service to interact with a Coherence cluster. The services must be configured to connect to extend proxy services that are running on the cluster. Both remote cache services and remote invocation services are configured in a cache configuration deployment descriptor that must be found on the classpath when an extend-based client application starts.

The following topics are included in this section:

- Defining a Remote Cache
- Using a Remote Cache as a Back Cache
- Defining Remote Invocation Schemes
- Configuring Fault Tolerance for Remote Addresses
- Detecting Connection Errors

Defining a Remote Cache

A remote cache is specialized cache service that routes cache operations to a cache on the cluster. The remote cache and the cache on the cluster must have the same name. Extend clients use the `NamedCache` interface as normal to get an instance of the cache. At run time, the cache operations are not executed locally but instead are sent using TCP/IP to an extend proxy service on the cluster. The fact that the cache operations are delegated to a cache on the cluster is transparent to the extend client.

A remote cache is defined within a `<caching-schemes>` node using the `<remote-cache-scheme>` element. A `<tcp-initiator>` element is used to define the address (IP or DNS name) and port of the extend proxy service on the cluster to which the client connects. See the "remote-cache-scheme" element reference in the *Oracle Coherence Developer's Guide* for a complete list and description of all `<remote-cache-scheme>` subelements.

Table 3–2 defines a remote cache named `dist-extend` that connects to an extend proxy service that is listening on address `198.168.1.5` and port `9099`. To use this remote cache, there must be a cache defined on the cluster that is also named `dist-extend`. See "Defining Caches for Use By Extend Clients" on page 3-5 for more information on defining caches on the cluster.

Example 3–2 Remote Cache Definition

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <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>198.168.1.5</address>
                     <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>
```

Using a Remote Cache as a Back Cache

Extend clients typically use remote caches as part of a near cache. In such scenarios, a local cache is used as a front cache and the remote cache is used as the back cache. For more information, see "Defining a Near Cache for C++ Clients" on page 8-7 and "Defining a Near Cache for .NET Clients" on page 18-5, respectively.

The following example creates a near cache that uses a local cache together with a remote cache.

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <cache-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-extend-near</cache-name>
         <scheme-name>extend-near</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <near-scheme>
         <scheme-name>extend-near</scheme-name>
         <front-scheme>
            <local-scheme>
               <high-units>1000</high-units>
            </local-scheme>
         </front-scheme>
         <back-scheme>
            <remote-cache-scheme>
               <scheme-ref>extend-dist</scheme-ref>
            </remote-cache-scheme>
         </back-scheme>
         <invalidation-strategy>all</invalidation-strategy>
      </near-scheme>
```

```
      <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>
               <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>
```

Defining Remote Invocation Schemes

A remote invocation scheme defines an invocation service that is used by clients to execute tasks on the remote Coherence cluster. Extend clients use the `InvocationService` interface as normal. At run time, a TCP/IP connection is made to an extend proxy service and an `InvocationService` implementation is returned that executes synchronous `Invocable` tasks within the remote cluster JVM to which the client is connected.

Remote invocation schemes are defined within a `<caching-schemes>` node using the `<remote-invocation-scheme>` element. A `<tcp-initiator>` element is used to define the address (IP or DNS name) and port of the extend proxy service on the cluster to which the client connects. See the "remote-invocation-scheme" element reference in the *Oracle Coherence Developer's Guide* for a complete list and description of all `<remote-invocation-scheme>` subelements.

Example 3–3 defines a remote invocation scheme that is called `ExtendTcpInvocationService` and connects to an extend proxy service that is listening on address `198.168.1.5` and port `9099`.

Example 3–3 Remote Invocation Scheme Definition

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

   <caching-schemes>
      ...
      <remote-invocation-scheme>
         <scheme-name>extend-invocation</scheme-name>
         <service-name>ExtendTcpInvocationService</service-name>
         <initiator-config>
            <tcp-initiator>
               <remote-addresses>
                  <socket-address>
                     <address>198.168.1.5</address>
```

```
                  <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-invocation-scheme>
</caching-schemes>
</cache-config>
```

Configuring Fault Tolerance for Remote Addresses

Remote cache schemes and remote invocation schemes can include multiple extend proxy service addresses to ensure a client can always connect to the cluster. Each address is attempted in a random order until either the list is exhausted or a TCP/IP connection is established. To configure multiple address, add additional `<socket-address>` child elements within the `<tcp-initiator>` element of a `<remote-cache-scheme>` and `<remote-invocation-scheme>` node as required. The following example defines two extend proxy addresses for a remote cache scheme.

```
...
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>ExtendTcpCacheService</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <socket-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </socket-address>
            <socket-address>
               <address>192.168.1.6</address>
               <port>9099</port>
            </socket-address>
         </remote-addresses>
      </tcp-initiator>
   </initiator-config>
</remote-cache-scheme>
...
```

Detecting Connection Errors

When a Coherence*Extend 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`) dispatches a `MemberEvent.MEMBER_LEFT` event to all registered `MemberListeners` and the service is stopped. For cases where the application calls `CacheFactory.shutdown()`, the service implementation dispatches a `MemberEvent.MEMBER_LEAVING` event followed by a `MemberEvent.MEMBER_LEFT` event. In both cases, if the client application attempts to subsequently use the service, the service automatically restarts itself and attempts to reconnect to the cluster. If the connection is successful, the service dispatches a

`MemberEvent.MEMBER_JOINED` event; otherwise, a fatal exception is thrown to the client application.

A Coherence*Extend service has several mechanisms for detecting dropped connections. Some are inherent to the underlying TCP/IP protocol, whereas others are implemented by the service itself. The latter mechanisms are configured within the `<outgoing-message-handler>` element.

The `<request-timeout>` element is the primary mechanism used to detect dropped connections. When a service sends a request to the remote cluster and does not receive a response within the request timeout interval, the service assumes that the connection has been dropped.

WARNING: If a `<request-timeout>` value is not specified, a Coherence*Extend service uses an infinite request timeout. In general, this is not a recommended configuration, as it could result in an unresponsive application. For most use cases, specify a reasonable finite request timeout.

The following example is taken from Example 3–2 and demonstrates setting the request timeout to `5` seconds.

```
...
<initiator-config>
   <tcp-initiator>
      <remote-addresses>
         <socket-address>
            <address>198.168.1.5</address>
            <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>
...
```

The `<heartbeat-interval>` and `<heartbeat-timeout>` can also be used to detect dropped connections. If a service does not receive a response within the configured heartbeat timeout interval, the service assumes that the connection has been dropped.

The following example sets the heartbeat interval to `500` milliseconds and the heartbeat timeout to `5` seconds.

```
...
<initiator-config>
   <tcp-initiator>
      <remote-addresses>
         <socket-address>
            <address>198.168.1.5</address>
            <port>9099</port>
         </socket-address>
      </remote-addresses>
      <connect-timeout>10s</connect-timeout>
   </tcp-initiator>
   <outgoing-message-handler>
```

```
      <heartbeat-interval>500ms</heartbeat-interval>
      <heartbeat-timeout>5s</heartbeat-timeout>
   </outgoing-message-handler>
</initiator-config>
...
```

Using an Address Provider for TCP Addresses

An address provider can be used to dynamically assign TCP address and port settings when binding to a server socket. The address provider must be an implementation of the `com.tangosol.net.AddressProvider` interface. Dynamically assigning addresses is typically used to implement custom load balancing algorithms.

Address providers are defined using the `<address-provider>` element, which can be used within the `<tcp-acceptor>` element for extend proxy schemes and within the `<tcp-initiator>` element for remote cache and remote invocation schemes.

> **Note:** The `<address-provider>` element also supports using a factory for object instantiation. See the `<address-provider>` element reference in the *Oracle Coherence Developer's Guide*.

The following example demonstrates configuring an `AddressProvider` implementation called `MyAddressProvider` for a TCP acceptor when configuring an extend proxy scheme.

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
      <thread-count>5</thread-count>
      <acceptor-config>
         <tcp-acceptor>
            <address-provider>
               <class-name>com.MyAddressProvider</class-name>
            </address-provider>
        </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
    </proxy-scheme>
  </caching-schemes>
</cache-config>
```

The following example demonstrates configuring an `AddressProvider` implementation called `MyClientAddressProvider` for a TCP initiator when configuring a remote cache scheme.

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>ExtendTcpCacheService</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <address-provider>
               <class-name>com.MyClientAddressProvider</class-name>
            </address-provider>
         </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>
```

Using Network Filters with Extend Clients

Like Coherence clustered services, Coherence*Extend services support pluggable network filters. Filters can be used to modify the contents of network traffic before it is placed on the wire. Most standard Coherence network filters are supported, including the compression and symmetric encryption filters. For more information on configuring filters, see "Using Network Filters" in the *Oracle Coherence Developer's Guide*.

To use network filters with Coherence*Extend, a `<use-filters>` element must be added to the `<initiator-config>` element in the client-side cache configuration descriptor and to the `<acceptor-config>` element in the cluster-side cache configuration descriptor.

> **Note:** The contents of the `<use-filters>` element must be the same in the client and cluster-side cache configuration descriptors.

For example, to encrypt network traffic exchanged between an extend client and the clustered service to which it is connected, configure the client-side `<remote-cache-scheme>` and `<remote-invocation-scheme>` elements as follows (assuming the symmetric encryption filter has been named `symmetric-encryption`):

```
<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>
      <connect-timeout>10s</connect-timeout>
    </tcp-initiator>
    <outgoing-message-handler>
      <request-timeout>5s</request-timeout>
    </outgoing-message-handler>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </initiator-config>
</remote-cache-scheme>

<remote-invocation-scheme>
  <scheme-name>extend-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>
      <connect-timeout>10s</connect-timeout>
    </tcp-initiator>
    <outgoing-message-handler>
      <request-timeout>5s</request-timeout>
    </outgoing-message-handler>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </initiator-config>
</remote-invocation-scheme>
```

For the cluster side, add a `<use-filters>` element within the `<proxy-scheme>` element that specifies a filter with the same name as the client-side configuration (for this example, `symmetric-encryption`):

```
<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>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
```

4

Building Your First Extend Client

This chapter demonstrates basic tasks that are required to build and run Coherence*Extend clients. The example client used in this chapter is a Java-based extend client; however, the concepts that are demonstrated are common to both C++ and .NET extend clients as well. See the `/examples` directory in both the C++ and .NET distribution for specific examples for these technologies.

The following sections are included in this chapter:

Overview

This chapter is organized into a set of steps that are used to create, configure, and run a basic Coherence*Extend client. The steps demonstrate many fundamental Coherence*Extend concepts, such as: configuring an extend proxy, configuring a remote cache, configuring the remote invocation service, and using the Coherence API.

Coherence for Java must be installed in order to complete the steps. For simplicity and ease of deployment, the client and cache server in this example are run on the same computer. Typically, extend clients and cache servers are located on separate systems.

Step 1: Configure the Cluster Side

The example extend client requires an extend proxy and cache to be configured in the cluster's cache configuration deployment descriptor. The extend proxy is configured to accept client TCP/IP communication on `localhost` and port `9099`. A distributed cache named `dist-extend` is defined and is used to store client data in the cluster.

To configure the cluster side:

1. Create an XML file named `example-config.xml`.
2. Copy the following XML to the file.

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

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

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>extend</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>
```

3. Save and close the file.

Step 2: Configure the Client Side

The example extend client requires a remote cache scheme and a remote invocation scheme. The remote cache scheme must define a cache on the cluster that is used to cache data and must provide the address and port of the extend proxy to which the client connects. For this example (based on Step 1), the remote cache scheme is configured to use the `dist-extend` cache and connects to an extend proxy that is located on `localhost` and port `9099`.

The example extend client queries the remote cache and therefore requires a remote invocation scheme. The remote invocation scheme must also define the host and port of the extend proxy to which the client connects.

To configure the client side:

1. Create an XML file named `example-client-config.xml`.
2. Copy the following XML to the file.

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

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

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

  <caching-schemes>
    <remote-cache-scheme>
      <scheme-name>remote</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>10s</connect-timeout>
        </tcp-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>

    <remote-invocation-scheme>
      <scheme-name>extend-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>
          <connect-timeout>10s</connect-timeout>
        </tcp-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-invocation-scheme>
  </caching-schemes>
</cache-config>
```

3. Save and close the file.

Step 3: Create the Sample Client

Example 4–1 is a simple client that increments an `Integer` value in a remote cache using the `CacheService` and then retrieves the value from the cache using the `InvocationService`. Lastly, the client writes the value to the system output before exiting.

> **Note:** This example could also be run on a Coherence node (that is, within the cluster) verbatim. The fact that operations are being sent to a remote cluster node over TCP/IP is completely transparent to the client application.

To create the sample application:

1. Create a text file.
2. Copy the following Java code to the file:

Example 4–1 Sample Coherence*Extend Application

```
import com.tangosol.net.AbstractInvocable;
import com.tangosol.net.CacheFactory;
import com.tangosol.net.InvocationService;
import com.tangosol.net.NamedCache;
import java.util.Map;

public class TestClient {
    public static void main(String[] asArgs)
            throws Throwable
        {
        NamedCache cache  = CacheFactory.getCache("dist-extend");
        Integer IValue = (Integer) cache.get("key");
        if (IValue == null)
            {
            IValue = new Integer(1);
            }
        else
            {
            IValue = new Integer(IValue.intValue() + 1);
            }
        cache.put("key", IValue);

        InvocationService service = (InvocationService)
                CacheFactory.getConfigurableCacheFactory()
                    .ensureService("ExtendTcpInvocationService");

        Map map = service.query(new AbstractInvocable()
            {
                public void run()
                    {
                     setResult(CacheFactory.getCache("dist-extend").get("key"));
                    }
            }, null);

        Integer IValue1 = (Integer) map.get(service.getCluster().
            getLocalMember());
        System.out.print("The value of the key is " + IValue1);
        }
}
```

3. Save the file as `TestClient.java` and close the file.
4. Compile `TestClient.java`:

```
javac -cp .;COHERENCE_HOME\lib\coherence.jar TestClient.java
```

Coherence*Extend InvocationService

Since, by definition, a Coherence*Extend client has no direct knowledge of the cluster and the members running within the cluster, the Coherence*Extend `InvocationService` only allows `Invocable` tasks to be executed on the JVM to which the client is connected. Therefore, you should always pass a null member set to the `query()` method. As a consequence of this, the single result of the execution will be keyed by the local `Member`, which will be null if the client is not part of the cluster. This `Member` can be retrieved by calling `service.getCluster().getLocalMember()`. Additionally, the Coherence*Extend `InvocationService` only supports synchronous task execution (that is, the `execute()` method is not supported).

Step 4: Start the Cache Server Process

Extend Proxies are started as part of a cache server process(`DefaultCacheServer`). The cache server must be configured to use the cache configuration that was created in Step 1. In addition, the cache server process must be able to find the `TestClient` application on the classpath at run time.

The following command line starts a cache server process and explicitly names the cache configuration file created in Step 1 by using the `tangosol.coherence.cacheconfig` system property:

```
java -cp COHERENCE_HOME\coherence.jar;PATH_TO_CLIENT
-Dtangosol.coherence.cacheconfig=PATH\example-config.xml
com.tangosol.net.DefaultCacheServer
```

Check the console output to verify that the proxy service is started. The output message is similar to the following:

```
(thread=Proxy:ExtendTcpProxyService:TcpAcceptor, member=1): TcpAcceptor now
 listening for connections on 192.168.1.5:9099
```

Step 5: Run the Application

The `TestClient` application is started using the `java` command and must be configured to use the cache configuration file that was created in Step 2.

The following command line runs the application and assumes that the `TestClient` class is located in the current directory. The cache configuration file is explicitly named using the `tangosol.coherence.cacheconfig` system property:

```
java -cp .;COHERENCE_HOME\lib\coherence.jar
-Dtangosol.coherence.cacheconfig=PATH\example-client-config.xml TestClient
```

The output displays (among other things) that the client successfully connected to the extend proxy TCP address and the current value of the key in the cache. Run the client again to increment the key's value.

5

Securing Coherence*Extend

Coherence*Extend includes a set of features that are used to secure communication between extend clients and extend proxies. The features provide varying levels of security and can be implemented as required. For general Coherence security, see "Securing Coherence" in *Oracle Coherence Developer's Guide*.

The following sections are included in this chapter:

Restricting Client Connections

The extend proxy's default behavior is to accept all extend client connections. This behavior can be changed to only allow client connections based on their host name or IP address. A customized filter can also be created to determine whether or not to accept a particular client. This type of access control is ideal for environments were the clients that are accessing the cluster are known in advance.

The `<authorized-host>` element is used to enter the host name or IP address of the clients that are allowed to access the cluster. Specific addresses are entered using the `<host-address>` element. A range of address can be defined using the `<host-range>` element.

The following example configures an extend proxy to only accept client connections from client's whose IP address is either 192.168.0.5, 192.168.0.6, or within the range of 192.168.0.10 to 192.168.0.20:

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <thread-count>5</thread-count>
   <acceptor-config>
      <tcp-acceptor>
         ...
         <authorized-host>
            <host-address>192.168.0.5</host-address>
            <host-address>192.168.0.6</host-address>
            <host-range>
               <from-address>192.168.0.10</from-address>
```

```
               <to-address>192.168.0.20</to-address>
            </host-range>
         </authorized-host>
         ...
      </tcp-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

A filter class can also be used to determine whether or not to accept a particular client. the class must implement the `com.tangosol.util.Filter` interface. The `evaluate()` method of the interface is passed the `java.net.InetAddress` of the client. Implementations should return `true` to allow the client to connect. To enable a filter, enter a fully qualified class name using the `<class-name>` element within the `<host-filter>` element. Initialization parameters for the implementation class can also be set using the `<init-params>` element.

The following example configures a filter named `MyFilter`, which is used to determine if a client is allowed to connect to the cluster.

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <thread-count>5</thread-count>
   <acceptor-config>
      <tcp-acceptor>
         ...
         <authorized-host>
            <host-address>192.168.0.5</host-address>
            <host-address>192.168.0.6</host-address>
            <host-range>
               <from-address>192.168.0.10</from-address>
               <to-address>192.168.0.20</to-address>
            </host-range>
            <host-filter>
               <class-name>package.MyFilter</class-name>
               <init-params>
                  <init-param>
                     <param-name>sPolicy</param-name>
                     <param-value>strict</param-value>
                  </init-param>
               </init-params>
            </host-filter>
         </authorized-host>
         ...
      </tcp-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Using Identity Tokens to Restrict Client Connections

Extend client can be restricted from accessing a cluster by using an identity token. The token is sent between extend clients and extend proxies whenever a connection is attempted. Only extend clients that pass a valid identity token are allowed to access the cluster.

On the extend client, identity tokens are created by an identity transformer and sent as part of the connection request. On the cluster side, an identity asserter is used to validate the identity token before the connection is accepted. Coherence*Extend

includes a default identity transformer (`DefaultIdentityTransformer`) and identity asserter (`DefaultIdentityAsserter`) that use the Subject (Java) or Principal (.NET) for the identity token. The default behavior can be overridden by providing custom identity transformer and identity asserter implementations and enabling them in the Coherence operational override file.

Note:

- The identity transformer and identity asserter implementation classes must be located on both the extend client's and the extend proxy server's classpath at run time.
- See "Using Custom Security Types" on page 5-5 for more information on using security object types other than the types that are predefined in POF.

The following topics are included in this section:

Creating a Custom Identity Transformer

An identity transformer is a client-side component that converts a Subject, or Principal, into an identity token that can be passed to an extend proxy. The identity token can be any type of object useful for identity validation; it is not required to be a well-known security type. At run time, the token is serialized and sent as part of the extend connection request.

Note: Identity tokens that are of a type that Coherence can serialize are automatically serialized. For .NET and C++ clients, the type must be a POF type. See "Using Custom Security Types" on page 5-5 for more information on using security object types other than the types that are predefined in POF.

For Java and C++, the identity transformer must implement the `IdentityTransformer` interface. C# clients implement the `IIdentityTransformer` interface.

Example 5–1 is a Java implementation that restricts client access by requiring a client to supply a password to access the proxy. The `IdentityTransformer` gets a password from a system property on the client and returns it as an identity token.

Example 5–1 A Sample Identity Transformer Implementation

```
import com.tangosol.net.security.IdentityTransformer;
import javax.security.auth.Subject;

public class PasswordIdentityTransformer
```

```
        implements IdentityTransformer
    {
    public Object transformIdentity(Subject subject)
            throws SecurityException
        {
        return System.getProperty("mySecretPassword");
        }
    }
```

If client authentication is already being done, a new Principal could be added to the Subject, with the Principal name as the password. The password Principal could be added to the Subject during JAAS authentication by modifying an existing JAAS login module or by adding an additional required login module that would add the password Principal. JAAS allows multiple login modules each of which can modify the Subject. Similarly, in .NET a password identity could be added to the Principal. The asserter on the cluster-side would then validate the "normal" Principals plus validate the password Principal. See "Creating a Custom Identity Asserter" below.

Enabling a Custom Identity Transformer

An identity transformer implementation is enabled in the client-side `tangosol-coherence-override.xml` file using the `<identity-transformer>` element within the `<security-config>` node. The element must include the full name of the implementation class. For example:

```
<coherence>
   <security-config>
      <identity-transformer>
        <class-name>com.my.PasswordIdentityTransformer</class-name>
      </identity-transformer>
   </security-config>
</coherence>
```

Creating a Custom Identity Asserter

An identity asserter is a cluster-side component that resides on the cache server that is hosting an extend proxy service. The asserter validates an identity token that is created by an identity transformer on the extend client. The token gets passed when an extend client initiates a connection. If the validation fails, the connection is refused and a security exception is thrown. The transformer and asserter are also invoked when a new channel within an existing connection is created. For Java and C++, the identity asserter must implement the `IdentityAsserter` interface. C# clients implement the `IIdentityAsserter` interface.

Example 5–2 is a Java implementation that checks a security token to ensure a valid password is given. In this case, the password is checked against a system property on the cache server. This asserter implementation is used for the identity transformer sample in Example 5–1.

Example 5–2 A Sample Identity Asserter Implementation

```
import com.tangosol.net.security.IdentityAsserter;
import javax.security.auth.Subject;


public class PasswordIdentityAsserter
        implements IdentityAsserter
    {
```

```
public Subject assertIdentity(Object oToken)
        throws SecurityException
    {
    if (oToken instanceof String)
        {
        if (((String) oToken).equals(System.getProperty("mySecretPassword")))
            {
            return null;
            }
        }
    throw new SecurityException("Access denied");
    }
}
```

There are many possible variations when creating an identity asserter. For example, the asserter could reject connections based on a list of Principals, check role Principals, validate signed Principal name, and so on. The asserter can block any connection attempt that doesn't prove the correct identity.

Enabling a Custom Identity Asserter

An identity asserter implementation is enabled in the cluster-side `tangosol-coherence-override.xml` file using the `<identity-asserter>` element within the `<security-config>` node. The element must include the full name of the implementation class. For example:

```
<coherence>
   <security-config>
      <identity-asserter>
        <class-name>com.my.PasswordIdentityAsserter</class-name>
      </identity-asserter>
   </security-config>
</coherence>
```

Using Custom Security Types

Security objects are automatically serialized/deserialized using Portable Object Format (POF) when they are passed between extend clients and extend proxies. Security objects of types that are predefined in POF can be used without any configuration or programming changes. Security objects of custom types that are not predefined in POF (for example, when using Kerberos authentication) causes an error.

For custom security types, an application can choose to either convert the custom type or define the type in POF:

Convert the Type

In this approach, a custom identity transformer converts a custom security object type to a type that is predefined for POF such as a character array or string before returning it as an object token. On the proxy server, a custom identity asserter would convert it back (after validation) to a subject.

For example, a subject may contain credentials that will not be serialized. The identity transformer would extract the credential and convert it to a character array, returning that array as the token. On the proxy server, the identity asserter would convert the character array to the proper credential type, validate it, and then construct a subject to return.

Define the Custom Type in POF

In this approach, custom security types are defined in a POF configuration file. The type must be defined in both the client's POF configuration file as well as the POF configuration file on the proxy server.

For detailed information on using POF with Java, see "Using Portable Object Format" in the *Oracle Coherence Developer's Guide*. For more information on using POF with C++ and C#, see Chapter 11, "Building Integration Objects for C++ Clients," and Chapter 20, "Building Integration Objects for .NET Clients," respectively.

Understanding Custom Identity Token Interoperability

Solutions that choose to use a custom identity token must always consider what tokens may be sent by an extend client and what tokens may be received by an extend proxy. This is particularly important during rolling upgrades as well as when rolling out a custom identity token solution.

Coherence Upgrades

Interoperability issues may occur during the process of upgrading Coherence. In this scenario, there may be different client versions that interoperate with different proxy server versions. In particular:

- When a 3.5 extend client is connecting to a 3.6 extend proxy, the custom identity asserter that is implemented on the extend proxy must be able to handle identity tokens sent by the 3.5 extend client. A 3.5 extend client will send either a `null` token or a `Subject`. The custom identity asserter must be prepared to handle those token types in addition to any custom tokens originating from a 3.6 extend client.
- Conversely, when a 3.6 extend client is connecting to a 3.5 extend proxy, the client must not use a custom identity transformer that sends a token that the 3.5 extend proxy cannot handle. The client must send either a `null` token or a `Subject`.

Custom Identity Token Rollout

Interoperability issues may occur between extend clients and extend proxies when rolling out a custom identity token solution. In this scenario, as extend proxies are migrated to use a custom identity asserter, there will be proxies that continue to use the default asserter until the roll out is completed. Likewise, as extend clients are migrated to use a custom identity transformer, there will be clients that continue to use the default transformer until the roll out is completed. In both cases, the extend clients and extend proxies must be able to handle a `null` token or a `Subject` until the rollout is complete.

A possible strategy for such scenarios may be to have a custom identity asserter that accepts `null` or `Subject` tokens temporarily as clients are updated. The identity asserter could check an external source for a policy that indicates whether or not those tokens are accepted. Once all clients have been updated to use a custom token, the policy could be changed and the identity asserter will no longer accept those tokens.

Associating Identities with Extend Services

Subject scoping allows remote cache and remote invocation service references that are returned to a client to be associated with the identity from the current security context. Clients use their platform-specific authentication APIs to establish a security context. A subject or principal is obtained from the current security context whenever a client

creates a `NamedCache` and `InvocationService` instance. All requests are then made in the context of the established subject or principal.

> **Note:** See "Using Custom Security Types" on page 5-5 for more information on using security object types other than the types that are predefined in POF.

For example, if the "trader" user calls `CacheFactory.getCache("trade-cache")` and the "manager" user calls `CacheFactory.getCache("trade-cache")`, each user gets a different remote cache reference object. Since an identity will be associated with that remote cache reference, authorization decisions can be made based on the identity of the caller. See "Implementing Authorization" below for details on implementing authorization.

Subject scoping is not enabled by default. This feature is enabled in the client-side `tangosol-coherence-override.xml` file using the `<subject-scope>` element within the `<security-config>` node. The following example enables subject scoping and ensures each subject gets a unique remote cache and remote invocation service reference.

```
<coherence>
   <security-config>
      <subject-scope>true<subject-scope>
   </security-config>
</coherence>
```

Implementing Authorization

Authorization is used to control which actions a particular user is able to perform based on their access control rights. In Coherence*Extend, authorization is implemented through the use of interceptor classes. An extend proxy calls the interceptor classes before a client accesses a proxied resource (cache, cache service, invocation service). Interceptor classes are implementation specific and must provide the necessary authorization logic before passing the request to the proxied resources.

The following topics are included in this section:

- Creating Authorization Interceptor Classes
- Enabling Authorization Interceptor Classes

The code samples in this section are based on the Java authorization example, which is included in the Coherence examples that are delivered as part of the documentation library. The example demonstrates a basic authorization implementation that uses the Principal obtained from a client request and a role-based policy to determine whether or not to allow operations on the requested service. Download the examples for the complete implementation.

Creating Authorization Interceptor Classes

An interceptor class can be created for both a proxied cache service and a proxied invocation service by implementing the `CacheService` and `InvocationService` interfaces, respectively. However, a set of wrapper classes are typically extended when implementing authorization: `com.tangosol.net.WrapperCacheService` (together with `com.tangosol.net.cache.WrapperNamedCache`) and `com.tangosol.net.WrapperInvocationService`. The wrapper classes delegate

to their respective interfaces and provide a convenient way to create interceptor classes that apply access control to the wrapped interface methods.

Example 5–3 is taken from the Coherence examples and demonstrates creating an authorization interceptor class for a proxied cache service by extending `WrapperCacheService`. It wraps all `CacheService` methods on the proxy and applies access controls based on the Subject passed from an extend client. The implementation only allows a Principal with the specified role to access the wrapped `CacheService`.

Example 5–3 Extending the WrapperCacheService Class for Authorization

```
public class EntitledCacheService
   extends WrapperCacheService
   {
   public EntitledCacheService(CacheService service)
       {
       super(service);
       }

   public NamedCache ensureCache(String sName, ClassLoader loader)
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_READER);
      return new EntitledNamedCache(super.ensureCache(sName, loader));
      }

   public void releaseCache(NamedCache map)
      {
      if (map instanceof EntitledNamedCache)
         {
         EntitledNamedCache cache =  (EntitledNamedCache) map;
         SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_READER);
         map = cache.getNamedCache();
         }
      super.releaseCache(map);
      }

   public void destroyCache(NamedCache map)
      {
      if (map instanceof EntitledNamedCache)
         {
         EntitledNamedCache cache =  (EntitledNamedCache) map;
         SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_ADMIN);
         map = cache.getNamedCache();
         }
      super.destroyCache(map);
      }
}
```

Notice that this class requires a named cache implementation. For this example, the `WrapperNamedCache` class is extended and wraps each method of the `NamedCache` instance. This allows access controls to be applied to different cache operations. Example 5–4 is a code excerpt taken from the Coherence examples that demonstrates overriding the `NamedCache` methods and applying access checks before allowing the method to be executed. See the Coherence examples for the complete class.

Example 5–4 Extending the WrapperNamedCache Class for Authorization

```
public class EntitledNamedCache
   extends WrapperNamedCache
```

```
   {
   public EntitledNamedCache(NamedCache cache)
      {
       super(cache, cache.getCacheName());
      }

   public Object put(Object oKey, Object oValue, long cMillis)
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_WRITER);
      return super.put(oKey, oValue, cMillis);
      }

   public Object get(Object oKey)
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_READER);
      return super.get(oKey);
      }

   public void destroy()
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_ADMIN);
      super.destroy();
      }
...
```

Example 5–5 is taken from the Coherence examples and demonstrates creating an authorization interceptor class for a proxied invocation service by extending `WrapperInvocationService`. It wraps all `InvocationService` methods on the proxy and applies access controls based on the Subject passed from an extend client. The implementation only allows Principals with a specified role name to access the wrapped `InvocationService`.

Example 5–5 Extending the WrapperInvocationService Class for Authorization

```
public class EntitledInvocationService
   extends WrapperInvocationService
   {
   public EntitledInvocationService(InvocationService service)
      {
      super(service);
      }

   public void execute(Invocable task, Set setMembers, InvocationObserver
      observer)
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_WRITER)
      super.execute(task, setMembers, observer);
      }

   public Map query(Invocable task, Set setMembers)
      {
      SecurityExampleHelper.checkAccess(SecurityExampleHelper.ROLE_WRITER)
      return super.query(task, setMembers);
      }
}
```

When a client attempts to use a remote invocation service, the proxy calls the `query()` method on the `EntitledInvocationService` class, rather than on the proxied `InvocationService` instance. The `EntitledInvocationService` class

decides to allow or deny the call. If the call is allowed, it then calls the `query()` method on the proxied `InvocationService` instance.

Enabling Authorization Interceptor Classes

Interceptor classes for a proxied cache service and a proxied invocation service are enabled in a proxy scheme definition within the `<cache-service-proxy>` element and `<invocation-service-proxy>` element, respectively. The `<class-name>` element is used to enter the fully qualified name of the interceptor class. Initialization parameters required by the class can be specified using the `<init-params>` element. See "cache-service-proxy" and "invocation-service-proxy" in *Oracle Coherence Developer's Guide* for detailed information on using these elements.

The following example demonstrates enabling both a proxied cache service and proxied invocation service interceptor class. The example uses the interceptor classes that were created in Example 5–3 and Example 5–5.

```
<proxy-scheme>
   ...
   <proxy-config>
      <cache-service-proxy>
         <class-name>
            com.tangosol.examples.security.EntitledCacheService
         </class-name>
         <init-params>
            <init-param>
               <param-type>com.tangosol.net.CacheService</param-type>
               <param-value>{service}</param-value>
            </init-param>
         </init-params>
      </cache-service-proxy>
      <invocation-service-proxy>
      <class-name>
         com.tangosol.examples.security.EntitledInvocationService
      </class-name>
      <init-params>
         <init-param>
            <param-type>com.tangosol.net.InvocationService</param-type>
            <param-value>{service}</param-value>
         </init-param>
      </init-params>
   </invocation-service-proxy>
</proxy-config>
```

Using SSL to Secure Client Communication

Communication between extend clients and extend proxies can be secured using SSL. SSL requires configuration on both the client side as well as the cluster side. SSL is supported for both Java and .NET clients but not for C++ clients. For those new to SSL, see "Using SSL in Coherence" in *Oracle Coherence Developer's Guide*, which provides a brief overview of common SSL concepts as well as links to more detailed SSL resources.

The configuration examples in this section assume that valid digital certificates for all clients and servers have been created as required and that the certificates have been signed by a Certificate Authority (CA). The digital certificates must be found in an identity store, and the trust store must include the signing CA's digital certificate. Self-Signed certificates may be used during development as needed.

The following topics are included in this section:

- Configuring a Cluster-Side SSL Socket Provider
- Configure a Java Client-Side SSL Socket Provider
- Configure a .NET Client-Side Stream Provider

Configuring a Cluster-Side SSL Socket Provider

SSL is configured in the cluster-side cache configuration file by defining a SSL socket provider for a proxy service. There are two options for configuring a SSL socket provider depending on the level of granularity that is required:

- Per Proxy Service – Each proxy service defines a SSL socket provider configuration or references a pre-defined configuration that is included in the operational configuration file.
- All Proxy Services – All proxy services use the same global SSL socket provider configuration. Proxy services that provide their own configuration override the global configuration. The global configuration can also reference a predefined configuration that is included in the operational configuration file.

Configure a SSL Socket Provider Per Proxy Service

To configure a SSL socket provider for a proxy service, add a `<socket-provider>` element within the `<tcp-acceptor>` element of each `<proxy-scheme>` definition. See "socket-provider" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<socket-provider>` element.

Example 5–6 demonstrates a proxy scheme that configures a SSL socket provider that uses the default values for the `<protocol>` and `<algorithm>` element (`TLS` and `SunX509`, respectively). These are shown for completeness but may be left out when using the default values.

Example 5–6 configures both an identity key store (`server.jks`) and a trust key store (`trust.jks`). This is typical of two-way SSL authentication where both the client and proxy must exchange their digital certificate and confirm each other's identity. For one-way SSL authentication, the proxy server configuration must include an identity key store but need not include a trust key store.

Example 5–6 Sample Cluster-Side SSL Configuration

```
...
<caching-schemes>
   ...
   <proxy-scheme>
      <service-name>ExtendTcpSSLProxyService</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <local-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </local-address>
            <socket-provider>
               <ssl>
                  <protocol>TLS</protocol>
                  <identity-manager>
                     <algorithm>SunX509</algorithm>
                     <key-store>
                        <url>file:server.jks</url>
```

```
                        <password>password</password>
                        <type>JKS</type>
                     </key-store>
                     <password>password</password>
                  </identity-manager>
                  <trust-manager>
                     <algorithm>SunX509</algorithm>
                     <key-store>
                        <url>file:trust.jks</url>
                        <password>password</password>
                        <type>JKS</type>
                     </key-store>
                  </trust-manager>
               </ssl>
            </socket-provider>
         </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
   </proxy-scheme>
   ...
</caching-schemes>
...
```

The following example references a SSL socket provider configuration that is defined in the `<socket-providers>` node of the operational deployment descriptor by specifying the configuration's `id` attribute (`ssl`). See "socket-providers" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<socket-providers>` element.

> **Note:** Out-of-box, a pre-defined SSL socket provider is included in the operational deployment descriptor and is named `ssl`. The pre-defined SSL socket provider is configured for two-way SSL connections and is based on peer trust where every trusted peer resides within a single JKS key store. See "Using the Pre-Defined SSL Socket Provider" in *Oracle Coherence Developer's Guide* for more information on using the pre-defined SSL socket provider. To configure a different SSL socket provider, use an operational override file to modify the pre-defined SSL socket provider or to create a new socket provider configuration as required.

```
...
<caching-schemes>
   ...
   <proxy-scheme>
      <service-name>ExtendTcpSSLProxyService</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <local-address>
               <address>192.168.1.5</address>
               <port>9099</port>
            </local-address>
            <socket-provider>ssl</socket-provider>
         </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
```

```
...
```

Configure a SSL Socket Provider for All Proxy Services

To configure a global SSL socket provider for use by all proxy services, use a `<socket-provider>` element within the `<defaults>` element of the cache configuration file. With this approach, no additional configuration is required within a proxy scheme definition. See "defaults" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<default>` element.

The following example uses the same SSL socket provider configuration from Example 5–6 and configures it for all proxy services:

```
<cache-config>
   <defaults>
      <socket-provider>
         <ssl>
            <protocol>TLS</protocol>
            <identity-manager>
               <algorithm>SunX509</algorithm>
               <key-store>
                  <url>file:server.jks</url>
                  <password>password</password>
                  <type>JKS</type>
               </key-store>
               <password>password</password>
            </identity-manager>
            <trust-manager>
               <algorithm>SunX509</algorithm>
               <key-store>
                  <url>file:trust.jks</url>
                  <password>password</password>
                  <type>JKS</type>
               </key-store>
            </trust-manager>
         </ssl>
      </socket-provider>
   </defaults>
   ...
```

The following example configures a global SSL socket provider by referencing a SSL socket provider configuration that is defined in the operational deployment descriptor:

```
<cache-config>
   <defaults>
      <socket-provider>ssl</socket-provider>
   </defaults>
   ...
```

Configure a Java Client-Side SSL Socket Provider

SSL is configured in the client-side cache configuration file by defining a SSL socket provider for a remote cache scheme and, if required, for a remote invocation scheme. There are two options for configuring a SSL socket provider depending on the level of granularity that is required:

- Per Remote Service – Each remote service defines a SSL socket provider configuration or references a pre-defined configuration that is included in the operational configuration file.

- All Remote Services – All remote services use the same global SSL socket provider configuration. Remote services that provide their own configuration override the global configuration. The global configuration can also reference a predefined configuration that is included in the operational configuration file.

Configure a SSL Socket Provider Per Remote Service

To configure a SSL socket provider for a remote service, add a `<socket-provider>` element within the `<tcp-initiator>` element of a remote scheme definition. See "socket-provider" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<socket-provider>` element.

Example 5–7 demonstrates a remote cache scheme that configures a socket provider that uses SSL. The example uses the default values for the `<protocol>` and `<algorithm>` element (`TLS` and `SunX509`, respectively). These are shown for completeness but may be left out when using the default values.

Example 5–7 configures both an identity key store (`server.jks`) and a trust key store (`trust.jks`). This is typical of two-way SSL authentication where both the client and proxy must exchange their digital certificate and confirm each other's identity. For one-way SSL authentication, the client configuration must include a trust key store but need not include an identity key store, which indicates the client will not exchange its digital certificate to the proxy and remains anonymous. The client's trust key store must includes the CA's digital certificate that was used to sign the proxy's digital certificate.

Example 5–7 Sample Java Client-Side SSL Configuration

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <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>ExtendTcpSSLCacheService</service-name>
         <initiator-config>
            <tcp-initiator>
               <remote-addresses>
                  <socket-address>
                     <address>198.168.1.5</address>
                     <port>9099</port>
                  </socket-address>
               </remote-addresses>
               <socket-provider>
                  <ssl>
                     <protocol>TLS</protocol>
                     <identity-manager>
                        <algorithm>SunX509</algorithm>
                        <key-store>
                           <url>file:server.jks</url>
                           <password>password</password>
                           <type>JKS</type>
                        </key-store>
```

```
                     <password>password</password>
                  </identity-manager>
                  <trust-manager>
                     <algorithm>SunX509</algorithm>
                     <key-store>
                        <url>file:trust.jks</url>
                        <password>password</password>
                        <type>JKS</type>
                     </key-store>
                  </trust-manager>
               </ssl>
            </socket-provider>
            <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>
```

The following example references a SSL socket provider configuration that is defined in the `<socket-providers>` node of the operational deployment descriptor by specifying the configuration's `id` attribute (`ssl`). See "socket-providers" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<socket-providers>` element.

> **Note:** Out-of-box, a pre-defined SSL socket provider is included in the operational deployment descriptor and is named `ssl`. The pre-defined SSL socket provider is configured for two-way SSL connections and is based on peer trust where every trusted peer resides within a single JKS key store. See "Using the Pre-Defined SSL Socket Provider" in *Oracle Coherence Developer's Guide* for more information on using the pre-defined SSL socket provider. To configure a different SSL socket provider, use an operational override file to modify the pre-defined SSL socket provider or to create a new socket provider configuration as required.

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <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>ExtendTcpSSLCacheService</service-name>
         <initiator-config>
            <tcp-initiator>
               <remote-addresses>
                  <socket-address>
```

```
                  <address>198.168.1.5</address>
                  <port>9099</port>
               </socket-address>
            </remote-addresses>
            <socket-provider>ssl</socket-provider>
            <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>
```

Configure a SSL Socket Provider for All Remote Services

To configure a global SSL socket provider for use by all remote services, use a `<socket-provider>` element within the `<defaults>` element of the cache configuration file. With this approach, no additional configuration is required within a remote scheme definition. See "defaults" in *Oracle Coherence Developer's Guide* for a detailed reference of the `<default>` element.

The following example uses the same SSL socket provider configuration from Example 5–7 and configures it for all remote services:

```
<cache-config>
   <defaults>
      <socket-provider>
         <ssl>
            <protocol>TLS</protocol>
            <identity-manager>
               <algorithm>SunX509</algorithm>
               <key-store>
                  <url>file:server.jks</url>
                  <password>password</password>
                  <type>JKS</type>
               </key-store>
               <password>password</password>
            </identity-manager>
            <trust-manager>
               <algorithm>SunX509</algorithm>
               <key-store>
                  <url>file:trust.jks</url>
                  <password>password</password>
                  <type>JKS</type>
               </key-store>
            </trust-manager>
         </ssl>
      </socket-provider>
   </defaults>
   ...
```

The following example configures a global SSL socket provider by referencing a SSL socket provider configuration that is defined in the operational deployment descriptor:

```
<cache-config>
   <defaults>
      <socket-provider>ssl</socket-provider>
   </defaults>
   ...
```

Configure a .NET Client-Side Stream Provider

SSL is configured in the .NET client-side cache configuration file by defining an SSL stream provider for remote serviceS. The SSL stream provider is is defined using the `<stream-provider>` element within the `<tcp-initiator>` element.

> **Note:** Certificates are managed on Window servers at the OS level using the Certificate Manager. The sample configuration assumes that the extend proxy's certificate is included in the Certificate Manager and that the CA's certificate that was used to sign the proxy's certificate is included as a trusted certificate authority. For more information on managing certificates, see http://technet.microsoft.com/en-us/library/cc782338(WS.10).aspx.

Example 5–8 demonstrates a remote cache scheme that configures a SSL stream provider. Refer to the cache configuration XML schema (*INSTALL_DIR*\config\cache-config.xsd) for details on the elements used to configure a SSL stream provider.

Example 5–8 Sample .NET Client-Side SSL Configuration

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <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>ExtendTcpSSLCacheService</service-name>
         <initiator-config>
            <tcp-initiator>
               <stream-provider>
                  <ssl>
                     <protocol>Tls</protocol>
                     <local-certificates>
                        <certificate>
                           <url>C:\</url>
                           <password>password</password>
                           <flags>DefaultKeySet</flags>
                        </certificate>
                     </local-certificates>
                  </ssl>
               </stream-provider>
               <remote-addresses>
                  <socket-address>
                     <address>198.168.1.5</address>
                     <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>
```

Managing Rogue Clients

Extend clients that operate outside of acceptable limits are considered rogue clients. Rogue clients can be slow responding clients or abusive clients that attempt to overuse a proxy— as is the case with denial of service attacks. In both cases, the proxy could run out of memory and become unresponsive.

The suspect protocol is used to safeguard against such abuses. The suspect algorithm monitors client connections looking for abnormally slow or abusive clients. When a rouge client connection is detected, the algorithm closes the connection in order to protect the proxy server from running out of memory. The protocol works by monitoring both the size (in bytes) and length (in messages) of the outgoing connection buffer backlog for a client. Different levels are set to determine when a client is suspect, when it has returned to normal, or when it is considered rogue.

The suspect protocol is configured within the `<tcp-acceptor>` element of a proxy scheme definition. See "tcp-acceptor" in the *Oracle Coherence Developer's Guide* for details on using the `<tcp-acceptor>` element. The suspect protocol is enabled by default.

The following example demonstrates configuring the suspect protocol and is similar to the default settings. When the outgoing connection buffer backlog for a client reaches 10 MB or 10000 messages, the client is considered suspect and is monitored. If the connection buffer backlog for a client returns to 2 MB or 2000 messages, then the client is considered safe and the client is no longer monitored. If the connection buffer backlog for a client reaches the 95 MB or 60000 messages, then the client is considered unsafe and the connection with the client is closed:

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <thread-count>5</thread-count>
   <acceptor-config>
      <tcp-acceptor>
         ...
         <suspect-protocol-enabled>true</suspect-protocol-enabled>
         <suspect-buffer-size>10M<suspect-buffer-size>
         <suspect-buffer-length>10000<suspect-buffer-length>
         <nominal-buffer-size>2M<nominal-buffer-size>
         <nominal-buffer-length>2000<nominal-buffer-length>
         <limit-buffer-size>95M<limit-buffer-size>
         <limit-buffer-length>60000<limit-buffer-length>
         ...
      </tcp-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

6

Best Practices for Coherence*Extend

This chapter describes best practices for configuring and running Coherence*Extend. The following sections are included in this chapter:

Run Proxy Servers with Local Storage Disabled

Each server in a partitioned cache, including the proxy server, can store a portion of the data. The proxy server has the responsibility of accepting POF formatted data from the client (either Java, C++, or .NET), deserializing POF data to get the Java objects, serializing the Java objects, then placing the resulting data in the cluster. These tasks can be expensive in terms of CPU and memory. You can preserve resources on the proxy server by disabling its local storage.

There are several ways in which you can disable storage:

Local storage for a proxy server can be enabled or disabled with the `tangosol.coherence.distributed.localstorage` Java property. For example:

```
-Dtangosol.coherence.distributed.localstorage=false
```

You can also disable storage in the cache configuration file. See the description of the `<local-storage>` element in "distributed-scheme" in the *Oracle Coherence Developer's Guide*.

Storage can also be disabled for the proxy server by modifying the `<local-storage>` setting in its `tangosol-coherence.xml` (or `tangosol-coherence-override.xml`) file. Example 6–1 illustrates setting `<local-storage>` to `false` in the `tangosol-coherence-override.xml`) file.

Example 6–1 Disabling Storage in tangosol-coherence-override.xml

```
<!--
Example using tangosol-coherence-override.xml
-->
<coherence>
  <cluster-config>
    <services>
      <!--
      id value must match what's in tangosol-coherence.xml for DistributedCache
service
      -->
      <service id="3">
        <init-params>
          <init-param id="4">
            <param-name>local-storage</param-name>
            <param-value
system-property="tangosol.coherence.distributed.localstorage">false</param-value>
          </init-param>
        </init-params>
      </service>
    </services>
  </cluster-config>
</coherence>
```

Do Not Run a Near Cache on a Proxy Server

By definition, a near cache provides local cache access to recently- and/or often-used data. If a proxy server is configured with a near cache, it will locally cache data accessed by its remote clients. It is unlikely that these clients will be consistently accessing the same subset of data, thus resulting in a low hit ratio on the near cache. This will result in higher heap usage and more network traffic on the proxy nodes with little to no benefit. For these reasons, it is recommended that a near cache not be used on a proxy server. To ensure that the proxy server is not running a near cache, remove all near schemes from the cache configuration being used for the proxy. See "Near Cache" for more information.

Configure Heap NIO Space to be Equal to the Max Heap Size

NIO memory is used for the TCP connection into the proxy and for POF serialization and deserialization. Older Java installations tended to run out of heap memory because it was configured too low. Newer Java JDKs will configure off heap NIO space equal to the max heap space. On Sun JVMs, this can also be set manually with this value:

```
-XX:MaxDirectMemorySize=512M
```

Set Worker Thread Pool Sizes According to the Needs of the Application

Client applications can be classified into two general categories: active and passive.

In active applications, the Coherence*Extend client sends many requests, such as put, get, and so on, to the proxy. These requests are serviced by the proxy service. The proxy will deserialize POF data put into the cache, and serialize data it returns to the client. For these tasks, configure a larger number of daemon (worker) threads for the proxy service.

In passive applications, the client waits on events (such as map listeners) based on some specified criteria. Events are serviced by the DistributedCache service. This service uses worker threads to push events to the client. For these tasks, the thread pool configuration for the DistributedCache service should include a sufficient number of worker threads.

Note that near caches on extend clients will use map listeners under the covers for the invalidation strategies of `ALL`, `PRESENT`, and `AUTO`. Applications that are write-heavy that use near caches will generate many map events.

Be Careful When Making InvocationService Calls

InvocationService allows a member of a service to invoke arbitrary code on any node in the cluster. On Coherence*Extend however, InvocationService calls are serviced by the proxy that the client is connected to by default. When sending the call through a proxy, you cannot choose the particular node on which the code will run.

Be Careful When Placing Collection Classes in the Cache

If a Coherence*Extend client puts a collection object, (such as an `ArrayList`, `HashSet`, `HashMap`, and so on) directly into the cache, it is deserialized as an immutable array. If you then extract it and cast it to its original type, then a `ClassCastExceptions` will be returned. As an alternative, use a Java interface object (such as a `List`, `Set`, `Map`, and so on) or encapsulate the collection object in another object. Both of these techniques are illustrated in the following example:

Example 6–2 Casting an ArrayList Object

```
public class ExtendExample
    {
    @SuppressWarnings({ "unchecked" })
    public static void main(String asArgs[])
        {
        System.setProperty("tangosol.coherence.cacheconfig", "client-config.xml");
        NamedCache cache = CacheFactory.getCache("test");

        // Create a sample collection
        List list  = new ArrayList();
        for (int i = 0; i < 5; i++)
            {
            list.add(String.valueOf(i));
            }
        cache.put("list", list);

        List listFromCache = (List) cache.get("list");

        System.out.println("Type of list put in cache: " + list.getClass());
        System.out.println("Type of list in cache: " + listFromCache.getClass());

        Map map = new TreeMap();
        for (Iterator i = list.iterator(); i.hasNext();)
            {
            Object o = i.next();
            map.put(o, o);
            }
        cache.put("map", map);

        Map mapFromCache = (Map) cache.get("map");
```

```
        System.out.println("Type of map put in cache: " + map.getClass());
        System.out.println("Type of map in cache: " + mapFromCache.getClass());
        }
    }
```

Run Multiple Proxies Instead of Increasing Thread Pool Size

The proxy performs POF/EL conversions in the service thread. A single proxy instance can easily bottleneck on a single core due to POF/EL conversions. Running multiple proxy instances on the same box (instead of increasing the thread pool size) helps spread the load across more cores.

Configure POF Serializers for Cache Servers

One of the tasks the proxy server performs is to deserialize POF data into Java objects. If you run C++ or .NET applications and store data to the cache, then the conversion to Java objects could be viewed as an unnecessary step. In the current release of Coherence, you have the option of configuring a POF serializer for cache servers. This will have the effect of storing POF format data directly in the cache.

This can have the following impact on your applications:

- .NET or C++ clients that only perform puts or gets will not require a Java version of the object. Java versions will still be required if deserializing on the server side (for entry processors, cache stores, and so on).
- POF serializers remove the requirement to serialize/deserialze on the proxy, thus reducing their memory and CPU requirements.

Example 6–3 illustrates a fragment from `example-pof-server.xml`, which configures a POF serializer for the distributed cache.

Example 6–3 Configuring a POFSerializer for a Distributed Cache

```
...
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>

      <serializer>

<class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
                <init-params>
                  <init-param>
                        <param-type>string</param-type>
                        <param-value>custom-types-pof-config.xml</param-value>
                  </init-param>
                </init-params>
          </serializer>

      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>

      <autostart>true</autostart>
    </distributed-scheme>
...
```

Use Node Locking Instead of Thread Locking

Coherence*Extend clients can send lock, put, and unlock requests to the cluster. The proxy holds the locks for the client. The requests for locking and unlocking can be issued at the thread level or the node level. In thread level locking, a particular thread instance belonging to the proxy (Thread 1, for example) issues the lock request. If any other threads (Thread 3, for example) issue an unlock request, they will be ignored. A successful unlock request can be issued only by the thread that issued the initial lock request. This can cause application errors since unlock requests will not succeed unless the original thread that issues the lock is also the one that receives the request to release the lock.

In node level locking, if a particular thread instance belonging to the proxy (Thread 1, for example) issues the lock request, then any other thread (Thread 3, for example) can successfully issue an unlock request.

As an alternative to using locks, Coherence recommends that you use the `EntryProcessor` API instead. `EntryProcessors` are described in "Performing Transactions" in the *Oracle Coherence Developer's Guide*.

Part II

Creating Java Extend Clients

Coherence for Java allows Java applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses for Java extend clients include desktop and Web applications that require access to Coherence caches.

The Coherence for Java library 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).

Like cache clients that are members of the cluster, Java extend clients use the `CacheFactory.getCache()` API call to retrieve a `NamedCache` instance. 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.

Unlike the C++ and .NET distributions, Java does not have a separate client distribution. The API delivered with Coherence for Java is the same API that is used to create extend clients. The API is detailed in the *Oracle Coherence Developer's Guide* and not duplicated in this guide. When building Java extend clients, refer to Part I, "Getting Started" in this guide (for basic setup) and Part IV, "Using the Programming API," in the *Oracle Coherence Developer's Guide*.

Part III

Creating C++ Extend Clients

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:

- Chapter 7, "Setting Up C++ Application Builds"
- Chapter 8, "Configuration and Usage for C++ Clients"
- Chapter 9, "Understanding the Coherence for C++ API"
- Chapter 10, "Using the Coherence C++ Object Model"
- Chapter 11, "Building Integration Objects for C++ Clients"
- Chapter 12, "Perform Continuous Query for C++ Clients"
- Chapter 13, "Query the Cache for C++ Clients"
- Chapter 14, "Remote Invocation Service for C++ Clients"
- Chapter 15, "Deliver Events for Changes as they Occur (C++)"
- Chapter 17, "Sample Applications for C++ Clients"

7

Setting Up C++ Application Builds

The following topics are included in this section:

Setting up the Compiler for Coherence-Based Applications

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 7–1 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++ / SunPro

Table 7–2 Compiler Settings for g++

Setting	Build Type	Required	Description
-O3	Release	No	Enables speed optimizations
-m32 / -m64	All	No	Explicitly set compiler to 32 or 64 bit mode

Including Coherence Header Files

Coherence ships with a set of header files that uses the Coherence API and must be compiled together with your application. The header files are available under the

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

Linking the Coherence Library

Coherence for C++ ships with a release version of the Coherence library. This library is also suitable for linking with debug versions of application code. The library is located in the installation's `lib` directory. During linking, this directory will need to be part of your linkers library path.

Table 7–3 Names of Linking Libraries for Release and Debug Versions

Operating System	Library
Windows	coherence.lib
Solaris	libcoherence.so
Linux	libcoherence.so
Apple OS X	libcoherence.dylib

Setting the 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 7–4 Name of the Coherence for C++ Library and Environment Variables

Operating System	Environment Variable
Windows	PATH
Solaris	LD_LIBRARY_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 programmatically (`CacheFactory::configure()`) from application code, the examples however do not do this.

Table 7–5 Cache Configuration System Property Value for Various Operating Systems

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

Table 7–5 (Cont.) Cache Configuration System Property Value for Various Operating

Operating System	System Property
Solaris	TangosolCoherenceCacheConfig
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. Coherence for C++ includes sample applications that are discussed in Chapter 17, "Sample Applications for C++ Clients," that demonstrate 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.

8

Configuration and Usage for C++ Clients

The following sections are included in this chapter:

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 9, "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 9, "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 8–1.

Example 8–1 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 Chapter 7, "Setting Up C++ Application Builds."

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 8–2 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 11, "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 8–2 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 8–3 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 8–3 A Caching Scheme that Connects to a Remote Coherence Cluster

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

<cache-config>
    <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>
          <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>
```

Defining 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 can be customized, 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 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. 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.

Example 8–4 demonstrates a local cache configuration .

Example 8–4 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>
```

Defining 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 "Configuring a Near Cache" in the *Oracle Coherence Developer's Guide*.

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 (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.

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.

Example 8–5 demonstrates a near cache configuration.

Example 8–5 Near Cache Configuration

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <cache-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-extend-near</cache-name>
         <scheme-name>extend-near</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <near-scheme>
         <scheme-name>extend-near</scheme-name>
         <front-scheme>
            <local-scheme>
               <high-units>1000</high-units>
            </local-scheme>
         </front-scheme>
         <back-scheme>
            <remote-cache-scheme>
               <scheme-ref>extend-dist</scheme-ref>
            </remote-cache-scheme>
         </back-scheme>
         <invalidation-strategy>all</invalidation-strategy>
      </near-scheme>

      <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>
               <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.

Obtaining a Cache Reference with C++

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

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

Cleaning up Resources Associated with a Cache

Instances of all `NamedCache` implementations 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.

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 "Setting the Runtime Library and Search Path" on page 7-2, 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 8–6 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 8–7 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 8–8 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 8–9 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 8–9 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 8-12.
- `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 8–10 illustrates the contents of a sample `tangosol-coherence.xml` file.

Example 8–10 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 8–11 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 8-10.

Example 8–11 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 8–12 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
```

9

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 9-3.

- `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" on page 13-1 for a more in depth look at queries. See also the C++ examples in "Simple Queries" on page 13-1

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 15-5.

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 about processors and aggregators, see "Entry Processors" on page 9-5 and "Entry Aggregators" on page 9-5.

- `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.

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 9-2.

- `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 9–1 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 9–2 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 9–3 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 9–4 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 9–5 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

A value 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 13-3.

Entry Processors

An entry processor is an 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 hellogrid C++ example in Chapter 17, "Sample Applications for C++ Clients."

Entry Aggregators

An entry aggregator 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 11, "Building Integration Objects for C++ Clients," for additional details.

10

Using the Coherence C++ Object Model

The Coherence Extend C++ API contains a robust C++ object model. The object model is the foundation on which Coherence for C++ is built.

This chapter contains the following sections:

Using the Object Model

The following sections 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 10–1 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 10–2 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 10–3 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 10–4 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 10–5 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 10–6 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 10–7 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 defined 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::Arrays`—implementation
- `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 10–8 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 10–9 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 `View` into a `Handle` 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 (specs) enable 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 10–10 illustrates using `interface_spec` to define a `Comparable` interface:

Example 10–10 An Interface Defined by interface_spec

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

Example 10–11 illustrates using `interface_spec` to define a derived interface `Number`:

Example 10–11 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 10–12.

> **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 10–12 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 10–12 is the equivalent the non-spec based definitions in Example 10–13.

Example 10–13 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 10–14 illustrates using the spec'd class:

Example 10–14 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::ThreadLocalreference` 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 10–15 illustrates how to create a new `Runnable` instance and spawn a thread:

Example 10–15 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 10–1.

Figure 10–1 A Bi-Directional Relationship

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.

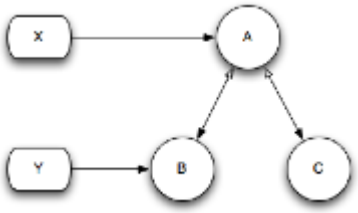
The provided mechanism for dealing with graphs is weak references. A weak reference is one which will reference an object, but not prevent it from being deleted. As illustrated in Figure 10–2, the A->B->A issue could be resolved by changing it to the following.

Figure 10–2 Establishing a Weak Reference

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.

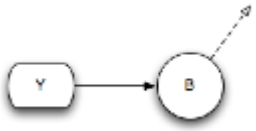
Weak references allow for construction of more complicated structures then this. But it becomes necessary to adopt a convention for which references are weak and which are strong. Consider a tree illustrated in Figure 10–3. The tree consists of nodes A, B, C; and two external references to the tree X, and Y.

Figure 10–3 Weak and Strong References to a Tree



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 10–4.

Figure 10–4 Artifacts after Deleting the Weak References



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 10–1 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:WeakHolder<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`.
coherence:lang:FinalHandle<T>	Yes	No	Thread-safe const transferring read-only `Handle`.
coherence:lang:FinalView<T>	Yes	Yes	Thread-safe read-only `View`.
coherence:lang:FinalHolder<T>	Yes	May	May act a thread-safe read-only `Handle` or `View`.

Thread Safety

Although the base `Object` class is thread-safe, this cannot provide automatic thread safety for the state of derived classes. As is typical it is up to each individual derived class implementation to provide for higher level thread-safety. The object model provides a number of facilities to aid in writing thread-safe code.

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 10–16:

Example 10–16 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 nested types defined on managed classes are intentionally not thread-safe. That is it is not safe to have multiple threads share a single handle. There is an important distinction here, we are discussing the thread-safety of the handle, not the object referenced by the handle. It is safe to have multiple distinct handles reference the same object from different threads without additional synchronization.

This lack of thread-safety for these handle types offers a significant performance optimization as the vast majority of handles are stack allocated. So long as references to these stack allocated handles are not shared across threads, there is no thread-safety issue to be concerned with.

Thread-safe handles are needed any time a single handle may be referenced by multiple threads. Typical cases include:

- Global handles - using the standard handle types as global or static variable is not safe.
- Non-managed multi-threaded application code - Use of standard handles within data structures which may be shared across threads is unsafe.
- Managed classes with handles as data members - It should be assumed that any instance of a managed class may be shared by multiple threads, and thus using standard handles as data members is unsafe. Note that while it may not be strictly true that all managed classes may be shared across threads, if an instance is passed to code outside of your explicit control (for instance put into a cache), there is no guarantee that the object will not made be visible to other threads.

The use of standard handles should be replaced with thread-safe handles in such cases. The object model includes the following set of thread-safe handles.

- `coherence::lang::MemberHandle<T>`—thread-safe version of `T::Handle`
- `coherence::lang::MemberView<T>`—thread-safe version of `T::View`
- `coherence::lang::MemberHolder<T>`—thread-safe version of `T::Holder`
- `coherence::lang::FinalHandle<T>`—thread-safe final version of `T::Handle`
- `coherence::lang::FinalView<T>`—thread-safe final version of `T::View`
- `coherence::lang::FinalHolder<T>`—thread-safe final version of `T::Holder`
- `coherence::lang::WeakHandle<T>`—thread-safe weak handle to `T`
- `coherence::lang::WeakView<T>`—thread-safe weak view to `T`
- `coherence::lang::WeakHolder<T>`—thread-safe weak `T::Holder`

These handle types may be read and written from multiple thread without the need for additional synchronization. They are primarily intended for use as the data-members of other managed classes, each instance is provided with a reference to a guardian managed `Object`. The guardian's internal thread-safe atomic state is used to provide thread-safety to the handle. When using these handle types it is recommended that they be read into a normal stack based handle if they will be accessed more then once within a code block. This assignment to a standard 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 a guardian `Object` must be supplied as the first parameter, this reference can be obtained by calling `self()` on the enclosing object.

Example 10–17 illustrates a trivial example:

Example 10–17 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. Since non-managed classes do not extend `coherence::lang::Object` they cannot be used as the guardian of thread-safe handles. It is possible, however, to use another `Object` as the guardian. When taking this approach it is crucial to ensure that the guardian `Object` outlives the guarded thread-safe handle. To facilitate this, starting with Coherence 3.4.1, you can obtain a random immortal guardian from `coherence::lang::System` via a call to `System::common()`. This is illustrated in Example 10–18:

Example 10–18 Thread-safe Handle as a Non-Managed Class

```
class Employee
    {
    public:
        Employee(String::View vsName, int32_t nId)
            : m_vsName(System::common(), 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:
         MemberView<String> m_vsName;
         const int32_t m_nId;
    };
```

When writing managed classes it is preferable to obtain a guardian via a call to `self()` then to `System::common()`.

> **Note:** In the rare case that one of these handles is declared via the `mutable` keyword, it must be informed of this fact by setting `fMutable` to `true` during construction.

Thread-safe handles can be utilize in non-class shared data as well such as global handles.

```
MemberView<NamedCache> MY_CACHE(System::common());

int main(int argc, char** argv)
    {
    MY_CACHE = CacheFactory::getCache(argv[0]);
    }
```

Escape Analysis

The object model includes escape analysis based optimizations. The escape analysis is used to automatically identify when a managed object is only visible to a single thread and in such cases optimize out unnecessary synchronizations. The following types of operations are optimized for non-escaped objects.

- reference count updates
- `COH_SYNCHRONIZED` acquisition and release
- reading/writing of thread-safe handles
- reading of thread-safe handles from immutables

Escape analysis is automatic and is completely safe so long as you follow the rules of using the object model. Most specifically is that it is not safe to pass a managed object between threads without using one of the provided thread-safe handles. Passing it by an external mechanism will not allow escape analysis to identify the "escape" which could cause memory corruption or other runtime errors.

Shared handles Each managed class type includes nested definitions for a Handles, View, and Holder. These handles are used extensively throughout the Coherence API, and is application code. They are intended for use as stack based references to managed objects. They are not intended to be made visible to multiple threads. That is a single handle should not be shared between two or more threads, though it is safe to have a managed Object referenced from multiple threads, so long as it is by distinct Handles, or a thread-safe MemberHandle/View/Holder.

It is important to remember that global handles to managed Objects should be considered to be "shared", and therefore must be thread-safe, as demonstrated previously. The failure to use thread-safe handles for globals will cause escaped objects to not be properly identified leading to memory corruption.

In 3.4 these non thread-safe handles could be shared across threads so long as external synchronization was employed, or if the handles were read-only. In 3.5 and later this is no longer true, even when used in a read-only mode or enclosed within external synchronization these handles are not thread-safe. This is due to a fundamental change in implementation which drastically reduces the cost of assigning one handle to another, which is an operation which occurs constantly. Any code which was using handles in this fashion should be updated to make use of thread-safe handles. See "Thread Safe Handles" on page 10-15 for more information.

Const Correctness Coherence escape analysis amongst other things leverages the computed mutability of an object to determine if state changes on data members are still possible. Namely once an object is only referenced from views it is assumed that its data members will not undergo further updates. The C++ language provides a number of mechanisms to bypass this const-only access and allow mutation from const methods. For instance the use of the mutable keyword in a data member declaration, or the casting away of constness. The arguably cleaner as well as supported approach for the object model is the mutable keyword. In the case of the Coherence object model, when a thread-safe data member handle is declared as

mutable this information must be communicated to the data member. All thread-safe data members support an optional third parameter fMutable which should be set to true if the data member has been declared with the mutable keyword. This will inform the escape analysis routine to not consider the data member as "const" once the enclosing object is only referenced via Views. Casting away of the constness of managed object is not supported, and can lead to run time errors if the object model believes that the object can no longer undergo state changes.

Thread-Local Allocator

Coherence for C++ includes a thread-local allocator to improve performance of dynamic allocations which are heavily used within the API. By default each thread will grow a pool to contain up to 64KB of reusable memory blocks to satisfy the majority of dynamic object allocations. The pool is configurable using the following system properties:

- `tangosol.coherence.slot.size` controls the maximum size of an object which will be considered for allocation from the pool, the default is 128 bytes. Larger objects will call through to the system's `malloc` routine to obtain the required memory.
- `tangosol.coherence.slot.count` controls the number of slots available to each thread for handling allocations, the default is 512 slots. If there are no available slots allocations will fall back on `malloc`.
- `tangosol.coherence.slot.refill` controls the rate at which slots misses trigger refilling the pool. The default of 10000 causes 1/10000 pool misses to force an allocation which will be eligible for refilling the pool.

The pool allocator can be disabled by setting the size or count to `0`.

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 10–19 illustrates a sample thread dump:

Example 10–19 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. This is the default.
- `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.
- `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.
- `alloc` —Specialization of `coherence::lang::ClassBasedHeapAnalyzer` which additionally tracks the allocation counts at the class level.
- *`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).

Example 10–20 illustrates heap analysis information returned by the class-based analyzer. It returns the heap analysis delta resulting from the insertion of a new entry into a `Map`.

Example 10–20 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
```

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 10–21 illustrates the results from an instance of memory corruption detection:

Example 10–21 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
```

Application Launcher - Sanka

Coherence 3.5 adds an application launcher for invoking executable classes embedded within a shared library. The launcher allows for a shared library to contain a number of utility or test executables without the need to ship individual standalone executable binaries.

Command line syntax

The launcher named `sanka` works similar to `java`, in that it is provided with one or more shared libraries to load, and a fully qualified class name to execute.

```
ge: sanka [-options] <native class> [args...]

available options include:
    -l <native library list>  dynamic libraries to load, separated by : or ;
    -D<property>=<value>      set a system property
    -version                  print the Coherence version
    -?                        print this help message
    <native class>            the fully qualified class e.g.
                              coherence::net::CacheFactory
```

The specified libraries must either be accessible from the operating system library path (`PATH`, `LD_LIBRARY_PATH`, `DYLD_LIBRARY_PATH`), or they may be specified with an absolute or relative path. Library names may also leave off any operating specific prefix or suffix. For instance the UNIX `libfoo.so` or Windows `foo.dll` can be specified simply as `foo`. The Coherence shared library which the application was linked against must be accessible from the system's library path as well.

Built-in Executables

A number of utility executables classes are included in the Coherence shared library:

- `coherence::net::CacheFactory` runs the Coherence C++ console
- `coherence::lang::SystemClassLoader` prints out the registered managed classes

- `coherence::io::pof::SystemPofContext` prints out the registered POF types

The later two executables can be optionally supplied with shared libraries to inspect, in which case they will output the registrations which exist in the supplied library rather then all registrations.

> **Note:** The console which was formerly shipped as an example, is now shipped as a built-in executable class.

Sample Custom Executable Class

Applications can of course still be made executable in the traditional C++ means using a global main function. If desired you can make your own classes executable using Sanka as well. The following is a simple example of an executable class:

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

COH_OPEN_NAMESPACE2(my,test)

using namespace coherence::lang;

class Echo
    : public class_spec<Echo>
    {
    friend class factory<Echo>;

    public:
        static void main(ObjectArray::View vasArg)
            {
            for (size32_t i = 0, c = vasArg->length; i < c; ++i)
                {
                std::cout << vasArg[i] << std::endl;
                }
            }
    };
COH_REGISTER_EXECUTABLE_CLASS(Echo); // must appear in .cpp

COH_CLOSE_NAMESPACE2
```

As you can see the specified class must have been registered as an `ExecutableClass` and have a `main` method matching the following signature:

```
static void main(ObjectArray::View)
```

The supplied `ObjectArray` parameter is an array of `String::View` objects corresponding the the command-line arguments which followed the executable class name.

Once linked into a shared library, for instance `libecho.so` or `echo.dll`, the `Echo` class can be run as follows:

```
> sanka -l echo my::test::Echo Hello World
Hello
World
```

The Coherence examples directory includes a helper script `buildlib` for building simple shared libraries.

11

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. The stream can then be deserialized in an alternate language using a similar POF-based class definition.

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

> **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 10, "Using the Coherence C++ Object Model."

POF Intrinsics

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

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

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

- Map
- Collection
- Exception

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 11-2 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 11-5 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 11-6 for more information.

Table 11–1 lists some of the requirements and limitations of each approach.

Table 11–1 Requirements and Limitations of Serialization Options

Approach	Coherence headers in data-object	Requires derivation from Object	Supports const data-members	External serialization routine	Requires zero-arg constructor
Managed<T>	No	No	Yes	Yes	Yes
PortableObject	Yes	Yes	No	No	Yes
PofSerializer	Yes	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 11–1 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 11–2:

Example 11–2 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 11–3 illustrates instances of the `Address` class wrapped by using `Managed<T>` as `Managed<Address>`, and supplied to the Coherence APIs:

Example 11–3 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 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 11–4, 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 11–4 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)
```

```
      {
       initialize(m_vsCity, hIn->readString(0));
       initialize(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:
     FinalView<String> m_vsCity;
     FinalView<String> m_vsState;
     const int32_tm_nZip;
  };
COH_REGISTER_PORTABLE_CLASS(1234, Address); // type ID registration—this must
                                            // appear in the .cpp not the .hpp
```

Example 11–5 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 11–5 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 11–6, a non-`PortableObject` version of a managed `Address`.

Example 11–6 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(self(), vsCity), m_vsState(self(), vsState), m_nZip(nZip) {}


    Address(const Address& that)
       : super(that), m_vsCity(self(), that.m_vsCity), m_sState(self(), that.m_
vsState), m_nZip(that.m_nZip) {}

  public: // Address interface
    virtual String::View  getCity()  const {return m_vsCity;}
    virtual String::View  getState() const {return m_vsState;}
    virtual int32_t getZip() const {return m_nZip;}

  public: // 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 MemberView<String> m_vsCity;
    const MemberView<String> m_vsState;
    const int32_t      m_nZip;
  };
```

Note that this version uses `const` data members, which makes it not well-suited for `PortableObject`. Example 11–7 illustrates an external class, `AddressSerializer`, which will be registered as being responsible for serialization of `Address` instances.

Example 11–7 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. A POF configuration file is only needed on the nodes where objects are serialized and deserialize.

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. However, to make use of more advanced features of Coherence: such as queries or entry processors; or if you use a key that is not a simple type; or when you use a cache loader and cache store, 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.

12

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 cluster-wide 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 12–1 illustrates how you can use code filters to find a given trader with a given order status:

Example 12–1 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 12–2 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 12–3 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 12–4:

Example 12–4 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 12–5 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.

13

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 data set should be loaded entirely into cache before queries are performed. In cases where the data set 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 13–1 illustrates:

Example 13–1 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 13–2 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 13–3 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 13–4 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 13–5 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 13–6 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 13–7 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 13–8, 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 13–8 Using a ChainedExtractor Implementation

```
ChainedExtractor::Handle hExtractor =

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

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

14

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 need only be concerned with state; the methods themselves can be no-operations.

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 14–1 illustrates a sample remote invocation scheme configuration.

Example 14–1 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 14–2 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 "PortableObject (Self-Serialization)" on page 11-5) 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" on page 11-9 for additional details.

15

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 15–1 illustrates a segment of the `MapListener` API.

Example 15–1 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 15–2 illustrates a segment of the `MapEvent` API.

Example 15–2 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 15–3 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 15–4 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 15–5 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 15–6 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 15–7 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 15–8 illustrates a simplified version of the `EventPrinter` example:

Example 15–8 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 15–9 prints all events that occur against the `Integer` key `5`:

Example 15–9 Printing Events that Occur Against a Specified Integer Key

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

The code in Example 15–10 would only trigger an event when the `Integer` key `5` is inserted or updated:

Example 15–10 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 15–11, a listener is added to the cache with a filter that allows the listener to only receive delete events.

Example 15–11 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 15–12 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 15-10.

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 15–13 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 15–14 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 15–15 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 15–15 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 15–16 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 15–17 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 15–18 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.

16

Performing Transactions for C++ Clients

This chapter provides instructions for using the Transaction Framework API to ensure cache operations are performed within a transaction when using a C++ client. The instructions do not provide detailed transaction API usage. See "Using the Transaction Framework API" in *Oracle Coherence Developer's Guide* for detailed transaction API usage.

The following sections are included in this chapter and are required to perform transactions:

- Using the Transaction API within an Entry Processor
- Creating a Stub Class for a Transactional Entry Processor
- Registering a Transactional Entry Processor User Type
- Configuring the Cluster-Side Transactional Caches
- Configuring the Client-Side Remote Cache
- Using a Transactional Entry Processor from a C++ Client

Using the Transaction API within an Entry Processor

C++ clients perform cache operations within a transaction by leveraging the Transaction Framework API. The transaction API is not supported natively on C++ and must be used within an entry processor. The entry processor is implemented in Java on the cluster and an entry processor stub class is implemented in C++ on the client. Both classes utilize POF to serialize between Java and C++.

Example 16–1 demonstrates an entry processor that performs a simple `update` operation within a transaction using the transaction API. At runtime, the class must be located on the classpath of the extend proxy server.

Example 16–1 Entry Processor for Extend Client Transaction

```
package coherence.tests;

import com.tangosol.coherence.transaction.Connection;
import com.tangosol.coherence.transaction.ConnectionFactory;
import com.tangosol.coherence.transaction.DefaultConnectionFactory;
import com.tangosol.coherence.transaction.OptimisticNamedCache;
import
com.tangosol.coherence.transaction.exception.PredicateFailedException;
import com.tangosol.coherence.transaction.exception.RollbackException;
import
com.tangosol.coherence.transaction.exception.UnableToAcquireLockException;
import com.tangosol.util.Filter;
```

```
import com.tangosol.util.InvocableMap;
import com.tangosol.util.extractor.IdentityExtractor;
import com.tangosol.util.filter.EqualsFilter;
import com.tangosol.util.processor.AbstractProcessor;

public class MyTxProcessor extends AbstractProcessor implements PortableObject
   {
   public Object process(InvocableMap.Entry entry)
   {
      // obtain a connection and transaction cache
      ConnectionFactory connFactory = new DefaultConnectionFactory();
      Connection conn = connFactory.createConnection("TransactionalCache");
      OptimisticNamedCache cache = conn.getNamedCache("MyTxCache");

      conn.setAutoCommit(false);

      // get a value for an existing entry
      String sValue = (String) cache.get("existingEntry");

      // create predicate filter
      Filter predicate = new EqualsFilter(IdentityExtractor.INSTANCE, sValue);

      try
         {
            // update the previously obtained value
            cache.update("existingEntry", "newValue", predicate);
         }
      catch (PredicateFailedException e)
         {
            // value was updated after it was read
            conn.rollback();
            return false;
         }
      catch (UnableToAcquireLockException e)
         {
            // row is being updated by another tranaction
            conn.rollback();
            return false;
         }
      try
         {
            conn.commit();
         }
      catch (RollbackException e)
         {
            // transaction was rolled back
            return false;
         }
      return true;
   }

   public void readExternal(PofReader in)
      throws IOException
   {
   }

    public void writeExternal(PofWriter out)
      throws IOException
   {
   }
```

```
}
```

Creating a Stub Class for a Transactional Entry Processor

An entry processor stub class allows a client to utilize the transactional entry processor on the cluster. The stub class is implemented in C++ and uses POF for serialization. POF allows an entry processor to be serialized between C++ and Java. The entry processor stub class does not need to include any transaction logic and is a skeleton of the transactional entry processor. See Chapter 11, "Building Integration Objects for C++ Clients," for detailed information on using POF with C++.

Example 16–2 and Example 16–3 demonstrate a stub class and associated header file for the transactional entry processor created in Example 16–1. In the example, POF registration is performed within the class.

Example 16–2 Transaction Entry Processor C++ Stub Class

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

COH_OPEN_NAMESPACE2(coherence,tests)
COH_REGISTER_PORTABLE_CLASS(1599, MyTxProcessor);

MyTxProcessor::MyTxProcessor()
   {
   }

void MyTxProcessor::readExternal(PofReader::Handle hIn)
   {
   }

void MyTxProcessor::writeExternal(PofWriter::Handle hOut) const
   {
   }

Object::Holder MyTxProcessor::process(InvocableMap::Entry::Handle hEntry) const
   {
   return NULL;
   }

COH_CLOSE_NAMESPACE2
```

Example 16–3 Transaction Entry Processor C++ Stub Class Header File

```
#ifndef COH_TX_EP_HPP
#define COH_TX_EP_HPP

#include "coherence/lang.ns"
#include "coherence/io/pof/PofReader.hpp"
#include "coherence/io/pof/PofWriter.hpp"
#include "coherence/io/pof/PortableObject.hpp"
#include "coherence/util/InvocableMap.hpp"
#include "coherence/util/processor/AbstractProcessor.hpp";

COH_OPEN_NAMESPACE2(coherence,tests)

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

```
using coherence::io::pof::PortableObject;
using coherence::util::InvocableMap;
using coherence::util::processor::AbstractProcessor;

class MyTxProcessor
   : public class_spec<MyTxProcessor,
      extends<AbstractProcessor>,
      implements<PortableObject> >

      {
      friend class factory<MyTxProcessor>;

      protected:
         MyTxProcessor();

      public:
       virtual Object::Holder process(InvocableMap::Entry::Handle hEntry) const;

      public:
         virtual void readExternal(PofReader::Handle hIn);
         virtual void writeExternal(PofWriter::Handle hOut) const;
      };

COH_CLOSE_NAMESPACE2
#endif // COH_TX_EP_HPP
```

Registering a Transactional Entry Processor User Type

An entry processor class must be registered as a POF user type in the cluster-side POF configuration file. The registration must use the same type ID that was used to register the stub class on the client side. The following example demonstrates registering the `MyTxProcessor` class that was created in Example 16–1 and uses the same type ID that was registered in Example 16–2:

```
<?xml version="1.0"?>
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
<pof-config>
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <user-type>
         <type-id>1599</type-id>
         <class-name>coherence.tests.MyTxProcessor</class-name>
      </user-type>
   </user-type-list>
</pof-config>
```

Configuring the Cluster-Side Transactional Caches

Transactions require a transactional cache to be defined in the cluster-side cache configuration file. Transactional caches are used by the Transaction Framework to provide transactional guarantees. See "Defining Transactional Caches" in *Oracle Coherence Developer's Guide* for details on transactional caches.

The following example creates a transactional cache that is named `MyTxCache`, which is the cache name that was used by the entry processor in Example 16–1. The configuration also includes a proxy scheme and a distributed cache scheme that are required to execute the entry processor from a remote client. The proxy is configured to accept client TCP/IP connections on `localhost` at port `9099`. See Chapter 3,

"Setting Up Coherence*Extend," for detailed information on configuring cluster-side caches when using Coherence*Extend.

```
<cache-config>
   <defaults>
      <serializer>pof</serializer>
   </defaults>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>MyTxCache</cache-name>
         <scheme-name>example-transactional</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>example-distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <transactional-scheme>
         <scheme-name>example-transactional</scheme-name>
         <task-timeout>0</task-timeout>
         <thread-count>7</thread-count>
         <autostart>true</autostart>
         <high-units>15M</high-units>
      </transactional-scheme>

      <distributed-scheme>
         <scheme-name>example-distributed</scheme-name>
         <service-name>DistributedCache</service-name>
         <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>
```

Configuring the Client-Side Remote Cache

Remote clients require a remote cache in order to connect to the cluster's proxy and run a transactional entry processor. The remote cache is defined in the client-side cache configuration file. See Chapter 3, "Setting Up Coherence*Extend," for detailed information on configuring client-side caches.

The following example configures a remote cache to connect to a proxy that is located on `localhost` at port `9099`. In addition, the name of the remote cache (`dist-example`) must match the name of a cluster-side cache that is used when initiating the transactional entry processor.

```
<cache-config>
   <defaults>
      <serializer>pof</serializer>
   </defaults>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>extend</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <remote-cache-scheme>
         <scheme-name>extend</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>30s</connect-timeout>
            </tcp-initiator>
            <outgoing-message-handler>
               <request-timeout>30s</request-timeout>
            </outgoing-message-handler>
         </initiator-config>
      </remote-cache-scheme>
   </caching-schemes>
</cache-config>
```

Using a Transactional Entry Processor from a C++ Client

A client invokes an entry processor stub class the same way any entry processor is invoked. However, at runtime, the cluster-side entry processor is invoked. The client is unaware that the invocation has been delegated to the Java class. The following example demonstrates a client that uses the entry processor stub class and results in an invocation of the transactional entry processor that was created in Example 16–1:

```
String::View vsCacheName = "dist-example";
String::View vsKey       = "AnyKey";

// retrieve the named cache
NamedCache::Handle hCache = CacheFactory::getCache(vsCacheName);

// invoke the cache
Object::View oResult = hCache->invoke(vsKey, MyTxProcessor::create());
std::cout << "Result of extend tx execution: " << oResult << std::endl;
```

17

Sample Applications for C++ Clients

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

- the Java Coherence archive was extracted into the top-level of your file system. For example, it would appear as `C:\coherence` on Windows.
- the C++ Coherence archive was extracted into the Java Coherence 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 "Installing the C++ Client Distribution" on page 2-1 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:

- `hellogrid`—An example of basic cache access.
- `console`—A command line application that enables you to interact with the cache using simple commands.
- `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 Proxy Service and Cache Server; these are Java components.

Starting a Coherence Proxy Service and Cache Server

Coherence for C++ applications communicate with the Coherence cluster using a proxy server. In order to run the examples against a cluster the proxy must first be started.

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 17–1 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 hellogrid
```

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

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

To use this scripts with your own simple applications, just create a new directory under the `examples` directory and place your source files there. Then run `build your_dir_name` to compile your application.

Starting a Sample Application

Now that configuration has been specified and the proxy/cache server has been started, you can start the client. The `examples` directory contains a `run` script which will allow you to run the examples which you've built. This script will perform the basic work of setting environment variables, and library search paths. To use the script just execute the `run` script supplying as the first parameter the name of the example you with to run.

For example, to run the `hellogrid` example on Windows, run the following command from the examples directory:

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

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

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:

```
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>
```

Now that you've run the 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.

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</autostart>
</distributed-scheme>

2008-04-25 09:01:02.264 Oracle Coherence GE 3.4/396 Alpha <D4>
(thread=DistributedCache, member=3): Asking member 1 for 128 out of 128 primary
 partitions

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

Map(dist-hello):
```

4. Now that you've run the 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.

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:

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

1. 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>
```

2. 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>
```

3. Now that you've run the 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.

Part IV

Creating .NET Extend Clients

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:

18

Configuration and Usage for .NET Clients

The following sections are included in this chapter:

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. See "Configuring Coherence*Extend" below.
2. Configure a POF context on the client and on all of the JVMs within the cluster that run the Coherence*Extend clustered service. See "Configuring a POF Context: Overview" on page 20-1.
3. Implement the .NET client application using the Coherence for .NET API. See "Using the Coherence .NET APIs" on page 19-3.
4. Make sure the Coherence cluster is up and running. See "Starting a Coherence DefaultCacheServer Process" on page 18-7.
5. Launch the .NET client application.

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 18–1.

Example 18–1 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 18–2.

Example 18–2 Configuration to Connect to a Remote Coherence Cluster

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

<cache-config>
  <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.

Defining 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 programmatically 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.

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>`). A custom class can be defined using the `<class-scheme>` subelement for both the `<eviction-policy>` and `<unit-calculator>` element to specify custom behavior as required.

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 18–3 demonstrates a near cache configuration.

Example 18–3 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>
```

Defining a Near Cache for .NET Clients

This section describes the Near Cache as it pertains to Coherence for .NET 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 "Configuring a Near Cache" in the *Oracle Coherence Developer's Guide*.

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

The `Tangosol.Net.Cache.NearCache` class enables you to programmatically 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.

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.

Example 18–4 demonstrates a near cache configuration.

Example 18–4 Near Cache Configuration

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
   <cache-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-extend-near</cache-name>
         <scheme-name>extend-near</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <near-scheme>
         <scheme-name>extend-near</scheme-name>
         <front-scheme>
            <local-scheme>
               <high-units>1000</high-units>
            </local-scheme>
         </front-scheme>
         <back-scheme>
            <remote-cache-scheme>
               <scheme-ref>extend-dist</scheme-ref>
            </remote-cache-scheme>
         </back-scheme>
         <invalidation-strategy>all</invalidation-strategy>
      </near-scheme>

      <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>
               <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, `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.

Starting 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 18–5 illustrates a sample command line.

Example 18–5 Command to Start 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
```

Obtaining a Cache Reference with .NET

A reference to a configured cache can be obtained by name by using the `CacheFactory` class:

Example 18–6 Obtaining a Reference to a Cache

```
INamedCache cache = CacheFactory.GetCache("example-local-cache");
```

Cleaning Up Resources Associated with a Cache

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 18–7 Obtaining and Releasing a Reference to a 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.

19

Using the Coherence .NET Client Library

The following sections are included in this chapter:

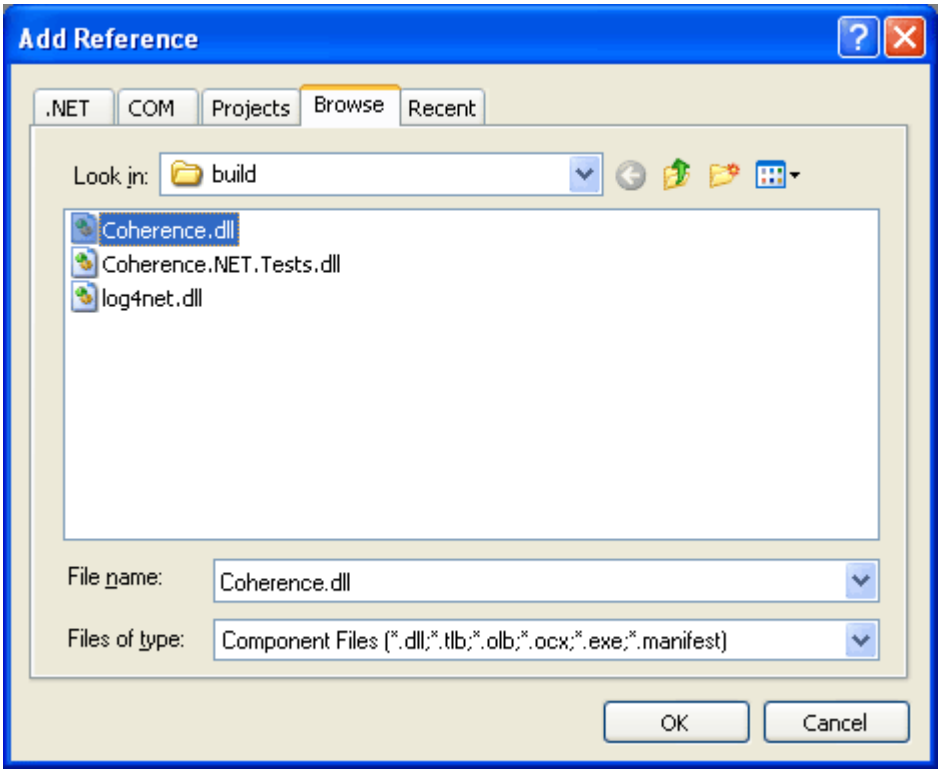
Setting Up the Coherence .NET Client Library

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...**. The Add Reference Window displays.
2. From the **Add Reference** window, choose the **Browse** tab and find the `Coherence.dll` library on your file system as shown in Figure 19–1.

Figure 19–1 Add Reference Window



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.

> **Note:** If these configuration files are not specified in the `app.config/web.config`, Coherence will look for them in both the folder where the application is deployed or, for Web applications, in the root of the Web application.

Example 19–1 Sample Application Configuration File

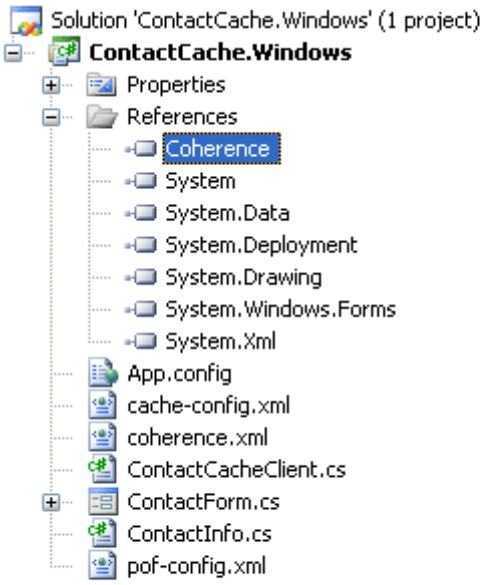
```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
Coherence"/>
  </configSections>
  <coherence>
    <cache-factory-config>my-coherence.xml</cache-factory-config>
    <cache-config>my-cache-config.xml</cache-config>
    <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 operational configuration descriptor used by the `CacheFactory` to configure `IConfigurableCacheFactory` and `Logger`.
- `cache-config`—contains the path to a cache configuration file which contains the cache configuration (see "Configuring Coherence*Extend" on page 18-1). This cache configuration descriptor is used by `DefaultConfigurableCacheFactory`.
- `pof-config`—contains the path to the configuration descriptor used by the `ConfigurablePofContext` to register custom types used by the application. For detailed instructions on using POF, see Chapter 19, "Using the Coherence .NET Client Library."

Figure 19–2 illustrates what the solution should look like after adding the configuration files:

Figure 19–2 File System Displaying the Configuration Files



Using the Coherence .NET APIs

This section highlights the primary Coherence .NET APIs that are used to interact with Coherence caches within a .NET application.

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 19–2 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 19–3 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 19–4 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 19–5 Configuring a Logger

```
<coherence>
  <logging-config>
    <destination>common-logger</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` assembly for .NET 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 19–6 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 19–6 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 Figure 19–7 a filter evaluates to `true` if an `Employee` object is inserted into a cache with an `IsMarried` property value set to `true`.

Example 19–7 Filtering on an Inserted Object

```
new CacheEventFilter(CacheEventMask.Inserted, new EqualsFilter("IsMarried",
true));
```

In Example 19–8 a filter evaluates to `true` if any object is removed from a cache.

Example 19–8 Filtering on Removed Object

```
new CacheEventFilter(CacheEventMask.Deleted);
```

In Example 19–9 a filter `that` evaluates to `true` if when an `Employee` object `LastName` property is changed from `Smith`.

Example 19–9 Filtering on a Changed Object

```
new CacheEventFilter(CacheEventMask.UpdatedLeft, new EqualsFilter("LastName",
"Smith"));
```

Responding to Cache Events

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 19–10 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 19–11 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.

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 19–12 retrieves the keys of all entries that have a value equal to 5.

Example 19–12 Retrieving Keys Equal to a Numeric Value

```
EqualsFilter equalsFilter = new EqualsFilter(IdentityExtractor.Instance, 5);
```

```
ICollection  keys         = cache.GetKeys(equalsFilter);
```

The code in Example 19–13 retrieves all keys that have a value greater or equal to 55.

Example 19–13 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 19–14 retrieves all cache entries that have a value that begins with `Belg`.

Example 19–14 Retrieving Keys Based on a String Value

```
LikeFilter  likeFilter = new LikeFilter(IdentityExtractor.Instance, "Belg%", '\\',
true);
ICollection entries    = cache.GetEntries(likeFilter);
```

The code in Example 19–15 retrieves all cache entries that have a value that ends with `an` (case sensitive) or begins with `An` (case insensitive).

Example 19–15 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);
```

Value 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 19–16 retrieves all cache entries with keys greater than 5:

Example 19–16 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 19–17 retrieves all cache entries with values containing a `City` property equal to `city1`:

Example 19–17 Retrieving Cache Entries Based on a String Value

```
IValueExtractor extractor = new ReflectionExtractor("City");
IFilter         filter    = new EqualsFilter(extractor, "city1");
ICollection     entries   = cache.GetEntries(filter);
```

Entry Processors

An entry processor is an agent that operates against the entry objects within a cache.

All entry 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 19–18 demonstrates a conditional put. The value mapped to `key1` is set to `680` only if the current mapped value is greater than `600`.

Example 19–18 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 19–19 uses the `UpdaterProcessor` to update the value of the `Degree` property on a `Temperature` object with key `BGD` to the new value `26`.

Example 19–19 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);
```

Entry Aggregators

An entry 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 19–20 returns the size of the cache:

Example 19–20 Returning the Size of the Cache

```
IEntryAggregator aggregator = new Count();
object           result     = cache.Aggregate(cache.Keys, aggregator);
```

The code in Example 19–21 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 19–21 Returning an IDictionary

```
IEntryAggregator aggregator =
GroupAggregator.CreateInstance(IdentityExtractor.Instance, new Count());
object           result     = cache.Aggregate(cache.Keys, aggregator);
```

Note: Example 19–20 and Example 19–21 are simple examples and not practical for passing a large amount of keys or keys that are themselves very large. In such scenarios, use the `GroupAggregator.CreateInstance(String, IEntryAggregator, IFilter)` method and pass an `AlwaysFilter` object.

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 Chapter 20, "Building Integration Objects for .NET Clients." for additional details.

20

Building Integration 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.

The following section is included in this chapter:

Configuring a POF Context: Overview

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 20–1 illustrates a user-defined portable class:

Example 20–1 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 20–2 illustrates the Java version of the .NET class in Example 20–1.

Example 20–2 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 20–3. These elements are described in "POF User Type Configuration Elements" in *Oracle Coherence Developer's Guide*.

Example 20–3 Storing Mapping Information 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. As shown in the above example, the `<user-type-list>` includes the `coherence-pof-config.xml` file. This is where Coherence specific user types are defined and should be included in all of your POF configuration files
- 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 20–3 are saved into `my-dotnet-pof-config.xml`, you need to specify a serializer element as illustrated in Example 20–4:

Example 20–4 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>
```

If a serializer is not explicitly specified, the `ConfigurablePofContext` type is used for the POF serializer and uses a default configuration file called `pof-config.xml`. The Coherence .Net application will look for the default POF configuration file in both the folder where the application is deployed and, for Web applications, in the root of the Web application. If a POF configuration file is not found, it will try to be located by the contents of the `pof-config` element in the Coherence for .NET application configuration file. For example:

Example 20–5 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>
```

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 20–6 illustrates a sample cluster-side POF configuration file called `my-java-pof-config.xml`:

Example 20–6 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>coherence-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 20–7:

Example 20–7 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 20–8 demonstrates how the `ContactInfo` .NET type can be modified to support class evolution:

Example 20–8 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 20–9 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 20–10 illustrates, an implementation of the `IPofSerializer` interface for the `ContactInfo` type.

Example 20–10 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 20–11 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 20–12:

Example 20–12 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 20–13.

Example 20–13 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>
```

21

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 cluster-wide 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` class 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 21–1 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 21–2 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 21–3 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 21–4 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 21–5 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 21–6 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 21–7 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.

22

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 22–1 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 22–2 Obtaining a Reference to a Remote Invocation Service

```
IInvocationService service = (IInvocationService)
CacheFactory.GetService("ExtendTcpInvocationService");
```

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

Example 22–3 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 18, "Configuration and Usage for .NET Clients" for additional details.

23

Using 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 23–1 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 23–2:

Example 23–2 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 23–3 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 23–4 Setting the Configuration Property for a Filter that Implements IXmlConfigurable

```
<config>
  <param1>value1</param1>
  <param2>value2</param2>
</config>
```

24

Performing Transactions for .NET Clients

This chapter provides instructions for using the Transaction Framework API to ensure cache operations are performed within a transaction when using a .NET client. The instructions do not provide detailed transaction API usage. See "Using the Transaction Framework API" in *Oracle Coherence Developer's Guide* for detailed transaction API usage.

The following sections are included in this chapter and are required to perform transactions:

- Using the Transaction API within an Entry Processor
- Creating a Stub Class for a Transactional Entry Processor
- Registering a Transactional Entry Processor User Type
- Configuring the Cluster-Side Transactional Caches
- Configuring the Client-Side Remote Cache
- Using a Transactional Entry Processor from a .NET Client

Using the Transaction API within an Entry Processor

.NET clients perform cache operations within a transaction by leveraging the Transaction Framework API. The transaction API is not supported natively on .NET and must be used within an entry processor. The entry processor is implemented in Java on the cluster and an entry processor stub class is implemented in C# on the client. Both classes utilize POF to serialize between Java and C#.

Example 24–1 demonstrates an entry processor that performs a simple `update` operation within a transaction using the transaction API. At runtime, the class must be located on the classpath of the Coherence proxy server.

Example 24–1 Entry Processor for Extend Client Transaction

```
package coherence.tests;

import com.tangosol.coherence.transaction.Connection;
import com.tangosol.coherence.transaction.ConnectionFactory;
import com.tangosol.coherence.transaction.DefaultConnectionFactory;
import com.tangosol.coherence.transaction.OptimisticNamedCache;
import
com.tangosol.coherence.transaction.exception.PredicateFailedException;
import com.tangosol.coherence.transaction.exception.RollbackException;
import
com.tangosol.coherence.transaction.exception.UnableToAcquireLockException;
import com.tangosol.util.Filter;
```

```
import com.tangosol.util.InvocableMap;
import com.tangosol.util.extractor.IdentityExtractor;
import com.tangosol.util.filter.EqualsFilter;
import com.tangosol.util.processor.AbstractProcessor;

public class MyTxProcessor extends AbstractProcessor implements PortableObject
   {
   public Object process(InvocableMap.Entry entry)
   {
      // obtain a connection and transaction cache
      ConnectionFactory connFactory = new DefaultConnectionFactory();
      Connection conn = connFactory.createConnection("TransactionalCache");
      OptimisticNamedCache cache = conn.getNamedCache("MyTxCache");

      conn.setAutoCommit(false);

      // get a value for an existing entry
      String sValue = (String) cache.get("existingEntry");

      // create predicate filter
      Filter predicate = new EqualsFilter(IdentityExtractor.INSTANCE, sValue);

      try
         {
            // update the previously obtained value
            cache.update("existingEntry", "newValue", predicate);
         }
      catch (PredicateFailedException e)
         {
            // value was updated after it was read
            conn.rollback();
            return false;
         }
      catch (UnableToAcquireLockException e)
         {
            // row is being updated by another tranaction
            conn.rollback();
            return false;
         }
      try
         {
            conn.commit();
         }
      catch (RollbackException e)
         {
            // transaction was rolled back
            return false;
         }
      return true;
   }

   public void readExternal(PofReader in)
      throws IOException
   {
   }

    public void writeExternal(PofWriter out)
      throws IOException
   {
   }
```

```
}
```

Creating a Stub Class for a Transactional Entry Processor

An entry processor stub class allows a client to utilize the transactional entry processor on the cluster. The stub class is implemented in C# and uses POF for serialization. POF allows an entry processor to be serialized between C# and Java. The entry processor stub class does not need to include any transaction logic and is a skeleton of the transactional entry processor. See Chapter 20, "Building Integration Objects for .NET Clients," for detailed information on using POF with .NET.

Example 24–2 demonstrate an entry processor stub class for the transactional entry processor created in Example 24–1.

Example 24–2 Transaction Entry Processor .NET Stub Class

```
using Tangosol.IO.Pof;
using Tangosol.Net.Cache;
using Tangosol.Util.Processor;

namespace Coherence.Tests
{
   public class MyTxProcessor : AbstractProcessor, IPortableObject
   {
      public MyTxProcessor()
      {
      }

      public override object Process(IInvocableCacheEntry entry)
      {
         return null;
      }

      public void ReadExternal(IPofReader reader)
      {
      }

      public void WriteExternal(IPofWriter writer)
      {
      }
   }
}
```

Registering a Transactional Entry Processor User Type

Custom user types must be registered for the Java transactional entry processor in the cluster-side POF configuration file and for the client stub in the client-side POF configuration file. Both registrations must use the same type ID. The following example demonstrates registering both the `MyTxProcessor` class that was created in Example 24–1 and the client stub class that was created in Example 24–2, respectively.

Cluster-side POF configuration:

```
<?xml version="1.0"?>
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
<pof-config>
   <user-type-list>
      <include>coherence-pof-config.xml</include>
```

```
      <user-type>
         <type-id>1599</type-id>
         <class-name>coherence.tests.MyTxProcessor</class-name>
      </user-type>
   </user-type-list>
</pof-config>
```

Client-side POF configuration:

```
<?xml version="1.0"?>
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
<pof-config>
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <user-type>
         <type-id>1599</type-id>
         <class-name>Coherence.Tests.MyTxProcessor</class-name>
      </user-type>
   </user-type-list>
</pof-config>
```

Configuring the Cluster-Side Transactional Caches

Transactions require a transactional cache to be defined in the cluster-side cache configuration file. Transactional caches are used by the Transaction Framework to provide transactional guarantees. See "Defining Transactional Caches" in *Oracle Coherence Developer's Guide* for details on transactional caches.

The following example creates a transactional cache that is named `MyTxCache`, which is the cache name that was used by the entry processor in Example 24–1. The configuration also includes a proxy scheme and a distributed cache scheme that are required to execute the entry processor from a remote client. The proxy is configured to accept client TCP/IP connections on `localhost` at port `9099`. See Chapter 3, "Setting Up Coherence*Extend," for detailed information on configuring cluster-side caches when using Coherence*Extend.

```
<cache-config>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>MyTxCache</cache-name>
         <scheme-name>example-transactional</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>example-distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <transactional-scheme>
         <scheme-name>example-transactional</scheme-name>
         <task-timeout>0</task-timeout>
         <thread-count>7</thread-count>
         <autostart>true</autostart>
         <high-units>15M</high-units>
      </transactional-scheme>

      <distributed-scheme>
         <scheme-name>example-distributed</scheme-name>
         <service-name>DistributedCache</service-name>
```

```
         <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>
```

Configuring the Client-Side Remote Cache

Remote clients require a remote cache in order to connect to the cluster's proxy and run a transactional entry processor. The remote cache is defined in the client-side cache configuration file. See Chapter 3, "Setting Up Coherence*Extend," for detailed information on configuring client-side caches.

The following example configures a remote cache to connect to a proxy that is located on `localhost` at port `9099`. In addition, the name of the remote cache (`dist-example`) must match the name of a cluster-side cache that is used when initiating the transactional entry processor.

```
<cache-config>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>extend</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <remote-cache-scheme>
         <scheme-name>extend</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>30s</connect-timeout>
            </tcp-initiator>
            <outgoing-message-handler>
               <request-timeout>30s</request-timeout>
            </outgoing-message-handler>
         </initiator-config>
```

```
      </remote-cache-scheme>
   </caching-schemes>
</cache-config>
```

Using a Transactional Entry Processor from a .NET Client

A client invokes an entry processor stub class the same way any entry processor is invoked. However, at runtime, the cluster-side entry processor is invoked on the cluster. The client is unaware that the invocation has been delegated to the Java class. The following example demonstrates a client that uses the entry processor stub class and results in an invocation of the transactional entry processor that was created in Example 24–1:

```
INamedCache cache  = CacheFactory.GetCache("dist-example");
object      result = cache.Invoke( "AnyKey", new MyTxProcessor());

Console.Out.WriteLine("Result of extend tx execution: " + result );
```

25

Managing ASP.NET Session State

This chapter provides instructions for managing ASP.NET session state in a Coherence cluster. The instructions include how to enable and configure the Coherence session provider.

> **Note:** The Coherence session provider that was included in previous versions of Coherence for .NET is deprecated and has been replaced by the Coherence session provider detailed in this chapter.

The following sections are included in this chapter:

Overview

Coherence for .NET allows you to manage ASP.NET session state in a Coherence cluster, which has a number of benefits compared to out-of-the-box options offered by Microsoft:

- Session state is stored in a highly available Coherence cluster, making sessions resilient to Web server failures
- Sessions are stored in memory which allows for much faster access than when they are serialized to disk using SQL Server session provider
- Unlike relational databases, Coherence cluster is easy to scale out to support additional load
- In some cases, session data can be accessed at in-process speed by leveraging Coherence near caching features

ASP.NET applications are configured to use Coherence for session state management by modifying the `web.config` file and configuring the custom session state provider. In addition, the Coherence session provider includes a number of configuration options that can significantly improve performance and scalability of applications.

Setting Up Coherence Session Management

The following steps are required to use Coherence for ASP.NET session management:

- Configure Coherence for .NET client library by specifying an operational configuration, cache configuration, and POF configuration file (if using POF for session serialization). For details, see "Setting Up the Coherence .NET Client Library" on page 19-1.
- Enable the Coherence Session Provider
- Configure the Cluster-Side ASP Session Caches
- Configure a Client-Side ASP Session Remote Cache

After the ASP.NET application and cluster are configured properly, start the cluster and proxy servers to be used by the application and then start the ASP.NET Web application. The sessions are automatically stored within the Coherence cluster.

Enable the Coherence Session Provider

ASP.NET uses a provider model to allow custom session state management implementations. Coherence for .NET implements a custom provider that fulfils the contract defined by Microsoft. To use the Coherence provider, add the following provider configuration to an application's `web.config` file:

```
<system.web>
   <sessionState mode="Custom"
                 customProvider="CoherenceSessionProvider"
                 cookieless="false"
                 timeout="20">
      <providers>
         <add name="CoherenceSessionProvider"
            type="Tangosol.Web.CoherenceSessionStore, Coherence"/>
      </providers>
   </sessionState>
   ...
</system.web>
```

The above example configures an ASP.NET application to use the `CoherenceSessionStore` provider with the default settings. The Coherence session provider can be customized, as described in this chapter, in order to take full advantage of its included features.

Configure the Cluster-Side ASP Session Caches

The Coherence session provider requires two cache scheme definitions within the cluster's cache configuration file: A storage cache and an overflow cache. The storage cache is used for storing session data and the overflow cache is used if the session size exceeds the limit specified in the `externalAttributeSize` attribute of the `CoherenceSessionProvider` defined in the `Web.config` file.

When defining the session storage cache and the session overflow cache, the service name must be `AspNetSessionCache` and the cache names must be `aspnet-session-storage` and `aspnet-session-overflow`, respectively. In addition, the storage cache must be configured to use the `ConfigurablePofContext` class as the serializer. The scheme name and backing map configuration can be configured as required.

The following cache scheme definition creates two distributed caches that are used by the session provider: one for session storage and one for session overflow .

```
<cache-config>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>aspnet-session-storage</cache-name>
         <scheme-name>aspnet-session-scheme</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>aspnet-session-overflow</cache-name>
         <scheme-name>aspnet-session-overflow-scheme</scheme-name>
      </cache-mapping>
      ...
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>aspnet-session-scheme</scheme-name>
         <service-name>AspNetSessionCache</service-name>
         <serializer>
            <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
            <init-params>
               <init-param>
                  <param-type>string</param-type>
                  <param-value>coherence-pof-config.xml</param-value>
               </init-param>
            </init-params>
         </serializer>
         <backing-map-scheme>
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </distributed-scheme>

      <distributed-scheme>
         <scheme-name>aspnet-session-overflow-scheme</scheme-name>
         <scheme-ref>dist-default</scheme-ref>
         <service-name>AspNetSessionCache</service-name>
         <autostart>true</autostart>
      </distributed-scheme>
      ...
   </caching-schemes>
</cache-config>
```

Configure a Client-Side ASP Session Remote Cache

The Coherence session provider requires an extend client's cache configuration file to include remote cache schemes for the session storage and session overflow caches. As with any remote cache, the cache on the cluster and the cache on the client must use the same name. See "Defining a Remote Cache" on page 3-6 for additional details.

The following example configures a client-side ASP session remote cache scheme that is used by the Coherence session provider to store session data on the cluster.

```
<cache-config>
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>aspnet-session-storage</cache-name>
         <scheme-name>extend-direct</scheme-name>
      </cache-mapping>
      <cache-mapping>
```

```
            <cache-name>aspnet-session-overflow</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>
```

Selecting a Session Model

A session model describes how the Coherence session provider physically represents and stores session state in the cluster. The provider includes three different session model implementations out of the box:

- **Traditional Model** – Stores all session state as a single entity but serializes and deserializes attributes individually
- **Monolithic Model** – Stores all session state as a single entity, serializing and deserializing all attributes as a single operation
- **Split Model** – Extends the Traditional Model but separates the larger session attributes into independent physical entities

The traditional model is the default. It is similar to the `SessionStateItemCollection` provided by ASP.NET - it deserializes session items lazily in order to avoid deserialization penalty for items that are not accessed. However, there are certain scenarios where monolithic or split model are better choices.

Refer to "Session Model" in *Oracle Coherence User's Guide for Oracle Coherence*Web* for details about each model and their pros and cons. The discussion can help determine which model is the best fit for a particular application. The discussion is centered around Coherence*Web; however, the general concepts are exactly the same for ASP.NET Sessions.

Specify the Session Model

The split model is the recommended session model for most applications. However, the traditional model may be more optimal for applications that are known to have small HTTP session objects.

The monolithic model is designed to solve a specific class of problems related to multiple session attributes that have references to the same shared object, and that must maintain that object as a shared object. When migrating to the Coherence session provider from the ASP.NET InProc provider, the monolithic model will ensure that all shared objects are serialized and deserialized properly.

To specify the Coherence session provider's session model, add a `model` attribute within the provider configuration. The following example specifies a `split` model.

```
<system.web>
   <sessionState mode="Custom"
                 customProvider="CoherenceSessionProvider"
                 cookieless="false"
                 timeout="20">
      <providers>
         <add name="CoherenceSessionProvider"
              type="Tangosol.Web.CoherenceSessionStore, Coherence"
              model="split"
              externalAttributeSize="512"/>
      </providers>
   </sessionState>
   ...
</system.web>
```

The valid values for the `model` attribute are `traditional`, `monolithic`, `split`, or a fully qualified name of the class that implements `Tangosol.Web.ISessionModelManager` interface and provides a constructor that accepts a single `Tangosol.IO.ISerializer` argument. The interface allows custom model implementations to be created if necessary.

In the example above, the session provider is configured to use the `split` model. The split model supports `externalAttributeSize` attribute, which specifies the minimum size (in bytes) of the attributes that should be stored separately. If the `externalAttributeSize` attribute is omitted, the default value of `1024` bytes is used.

Registering the Backing Map Listener

Session attributes are partitioned into two regions when utilizing the split session model. Core HTTP session attributes, such as session ID, creation time, last access, and so on, are managed within one partition and large attributes are split out into another partition. This allows support for very large HTTP session objects without incurring overhead for frequently accessed small attributes.

With the .NET session provider implementation, core attributes and large attributes are stored in separate caches. Therefore; the backing map listener (`AspNetSessionStoreProvider$SessionCleanupListener` class) is recommended to keep both caches synchronized. This ensures that if a session is terminated explicitly by the user and/or removed by eviction or expiry, that both the removal of the core and large segments of the session are coherently removed from the two caches.

The following example demonstrates registering the `AspNetSessionStoreProvider$SessionCleanupListener` backing map listener on the cluster-side ASP .NET session cache:

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>aspnet-session-scheme</scheme-name>
      <service-name>AspNetSessionCache</service-name>
```

```
      <serializer>
         <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
         <init-params>
            <init-param>
               <param-type>string</param-type>
               <param-value>coherence-pof-config.xml</param-value>
            </init-param>
         </init-params>
      </serializer>
      <backing-map-scheme>
         <local-scheme>
            <class-name>com.tangosol.net.cache.LocalCache</class-name>
            <listener>
               <class-scheme>
                  <class-name>
       com.tangosol.net.internal.AspNetSessionStoreProvider$SessionCleanupListener
                  </class-name>
                  <init-params>
                     <init-param>
                        <param-type>
                           com.tangosol.net.BackingMapManagerContext
                        </param-type>
                        <param-value>{manager-context}</param-value>
                     </init-param>
                  </init-params>
               </class-scheme>
            </listener>
         </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
```

Specifying a Serializer

The Coherence session provider can be configured to use a specific serializer for serializing session items. To specify a serializer, add a `serializer` attribute within provider definition. The following example specifies the `binary` serializer.

```
<system.web>
   <sessionState mode="Custom"
                 customProvider="CoherenceSessionProvider"
                 cookieless="false"
                 timeout="20">
      <providers>
         <add name="CoherenceSessionProvider"
              type="Tangosol.Web.CoherenceSessionStore, Coherence"
              model="split"
              externalAttributeSize="512"
              serializer="binary"/>
      </providers>
   </sessionState>
   ...
</system.web>
```

The valid values for the `serializer` attribute are `binary` (default), `pof`, or a fully qualified name of the class that implements the `Tangosol.IO.ISerializer` interface. The interface is used to create a custom serializer if necessary. However, the existing serializers are sufficient more often than not.

Using POF for Session Serialization

Portable Object Format (POF) is the recommended serialization format when using Coherence to manage ASP.NET sessions and provides many benefits over standard .NET binary serialization. In particular, POF serialization is faster and has a significantly more compact format. The compact format typically results in a binary form that is 3 to 5 times smaller than the standard binary serializer. This translates directly into a lower memory footprint within the cluster and can result in significant cost savings.

To use POF, ensure that all custom classes that are stored either directly or indirectly within the session are registered within the POF context and either implement the `IPortableObject` interface or have an external `IPofSerializer` configured. For detailed instructions on using POF, see Chapter 20, "Building Integration Objects for .NET Clients."

The following discussion summarizes some of the implementation details that should be considered when using POF. For a detailed description of the POF format, see "The PIF-POF Binary Format" in the appendix of the *Oracle Coherence Developer's Guide*.

When session items are deserialized by the POF serializer, there is no guarantee that the type of the resulting object will be the same as the type of the original value. For example, integer values between -1 and 22 (inclusive) will be returned as `Int32` values, regardless of the original type, so they may need to be cast to the appropriate type.

Collections may also be deserialized to a different type. For example, an `ArrayList` might be stored within the session, but an immutable object array may be received after the object is read back. This is expected behavior and the reason why the `IPofReader` interface can be used to provide a template to read values as an argument to all methods that read collections from the POF stream.

Session items are not typed and there is no way to specify how they should be deserialized. Therefore, a default collection type is always received. This is typically acceptable when reading from the collection. However, if the collection needs to be modified, either of the following two options can be used:

- Create an instance of a mutable collection of a desired type and add elements from the deserialized collection to it. When using this option, don't forget to update corresponding session items with the new collection, or the changes will not be saved.
- Instead of storing "bare" collections directly, create a wrapper class that implements necessary serialization logic and register it within the POF context. This allows full control over collection serialization and can avoid the issues described above.

These steps do require extra work; however, the performance gains and reduced memory footprint are likely worth the trouble.

Sharing Session State Across Applications

In some cases, it is beneficial to be able to share sessions across ASP.NET applications. By default, a session key is determined by combining the application identifier (as returned by the `HostingEnvironment.ApplicationID` property) with the session identifier. This effectively prevents session sharing.

The Coherence session provider can be configured to use a specific application identifier. To specify an application identifier, add an `applicationId` attribute

within a provider definition. The following examples specifies `MyApplication` as the application ID.

```
<system.web>
   <sessionState mode="Custom"
                 customProvider="CoherenceSessionProvider"
                 cookieless="false"
                 timeout="20">
      <providers>
         <add name="CoherenceSessionProvider"
              type="Tangosol.Web.CoherenceSessionStore, Coherence"
              applicationId="MyApplication"
              model="split"
              externalAttributeSize="512"
              serializer="pof"/>
      </providers>
   </sessionState>
   ...
</system.web>
```

To enable session sharing across the applications, configure multiple applications with the same `applicationId` and ensure that they share the cookie containing the session identifier.

26

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

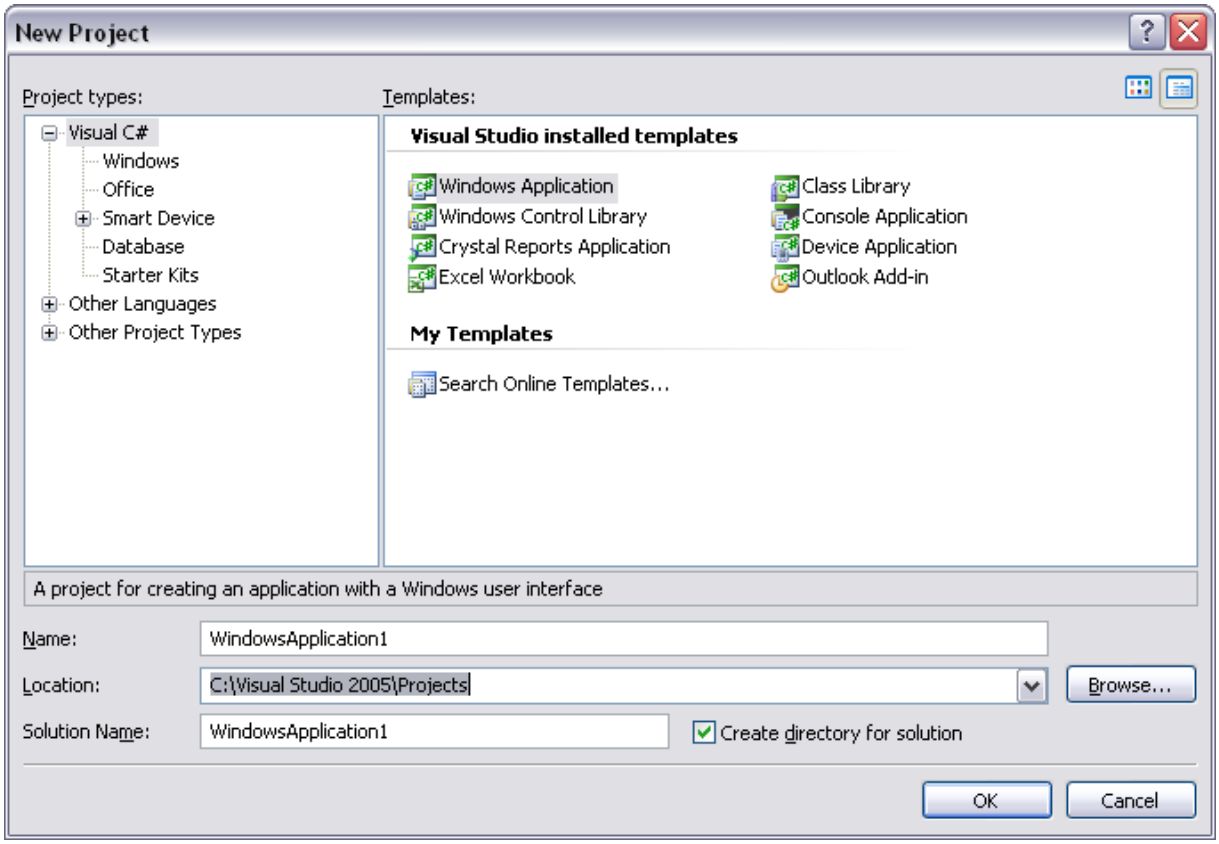
Create a Windows Application Project

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 26–1 illustrates the New Project window with the name, location, and solution for the project.

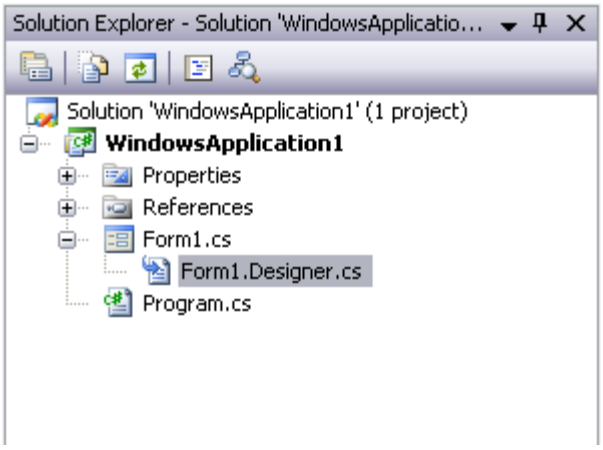
Figure 26–1 New Project Window



3. Click **OK**.

   Visual Studio should have created the following files: `Program.cs`, `Form1.cs` and `Form1.Designer.cs`. Figure 26–2 illustrates the **Solution Explorer** with the created project files

Figure 26–2 Solution Explorer with the Created Project Files



4. Rename these files if you want.

   In this example they have been renamed to `ContactCacheClient.cs`, `ContactForm.cs`, and `ContactForm.Designer.cs` respectively.

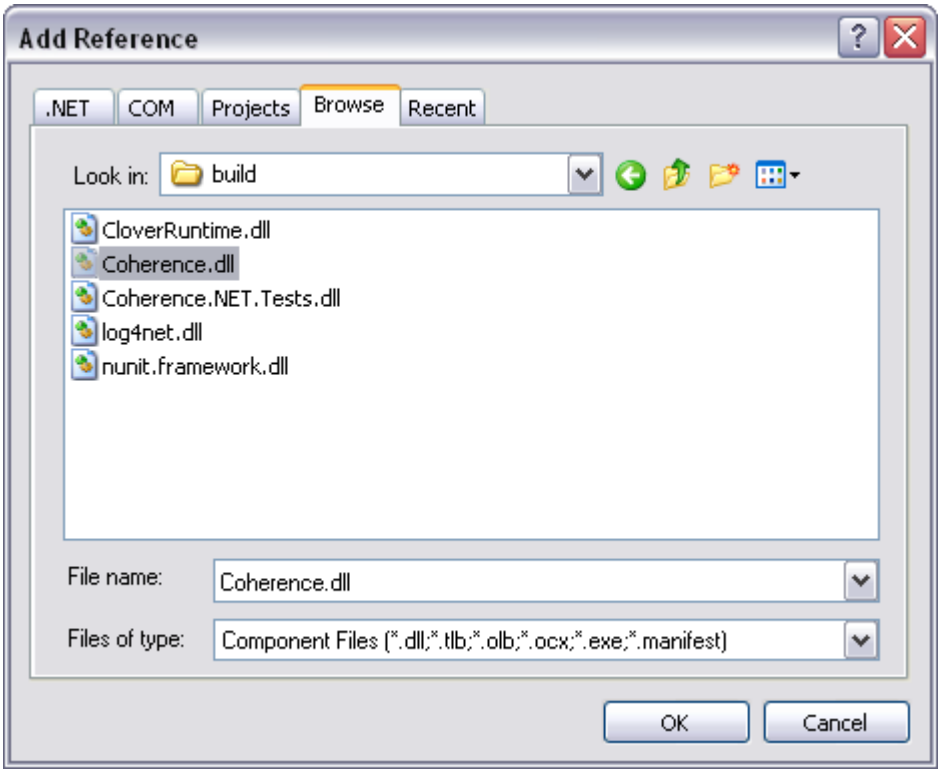
Add a Reference to the Coherence for .NET Library

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 26–3 illustrates the `.dll` files in the **Add Reference** window.

Figure 26–3 Add Reference Window



3. Click **OK**.

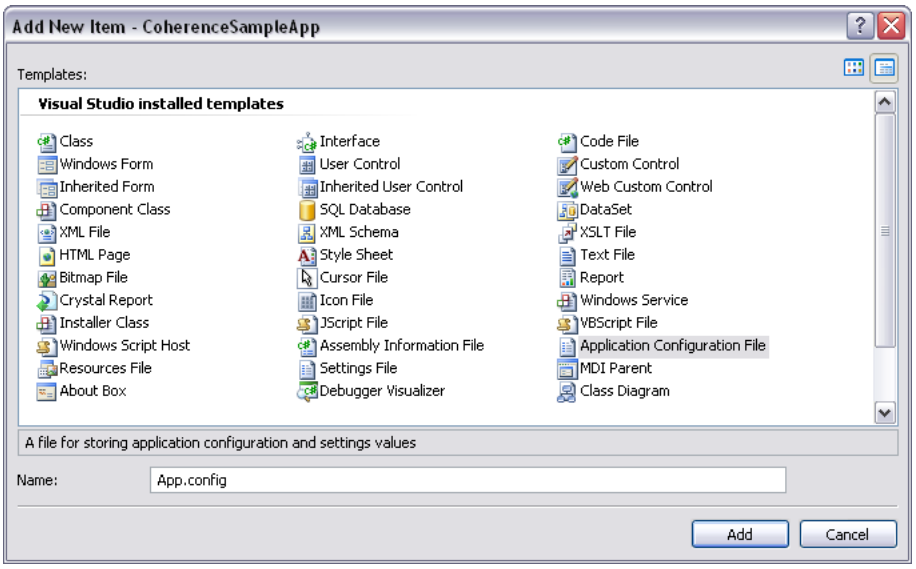
Create an App.config File

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 26–4 illustrates the contents of the **Add New Item** window.

Figure 26–4 Add New Item Window



3. Click **OK**.

Example 26–1 illustrates a sample valid `App.config` configuration file.

Example 26–1 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 18-3). 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 26–2 illustrates a sample `coherence.xml` configuration file

Example 26–2 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 26–3 illustrates a sample `cache-config.xml` configuration file.

Example 26–3 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 26–4 illustrates a sample `pof-config.xml` configuration file.

Example 26–4 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

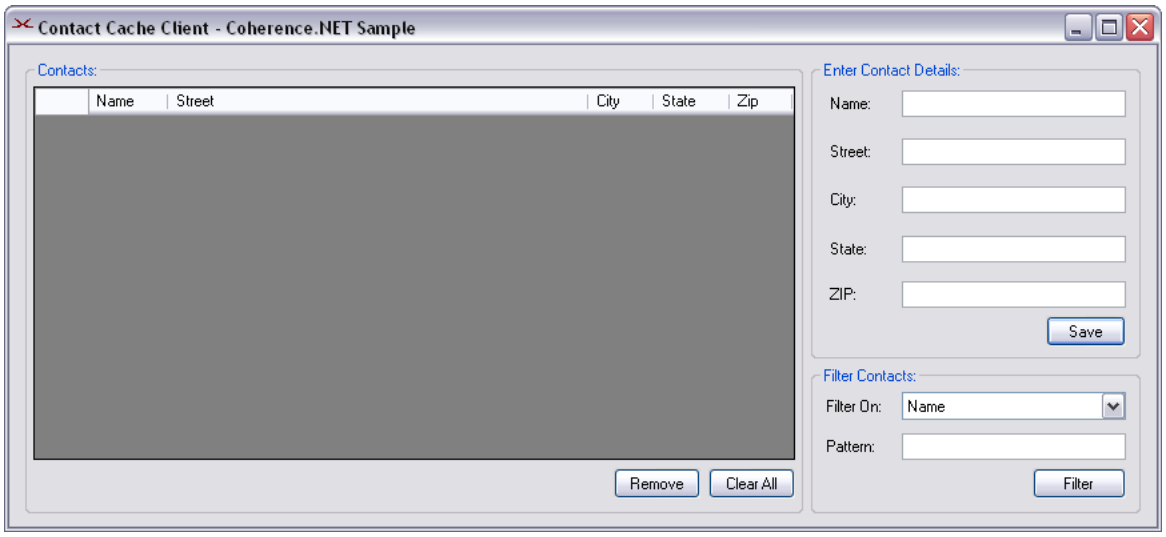
Next, you must add controls to your Windows form. This example shows you how to store objects into a `INamedCache`, read from the cache, query the cache, remove an item from the cache, and clear the cache. For this we're going to use buttons that will raise events when clicked, a couple of `TextBox` 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 26–5 illustrates what the Contact Cache Info application UI should look after you have finished the previous steps.

Figure 26–5 Contact Cache Client UI



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 26–5 Sample Class that Implements IPortableObject

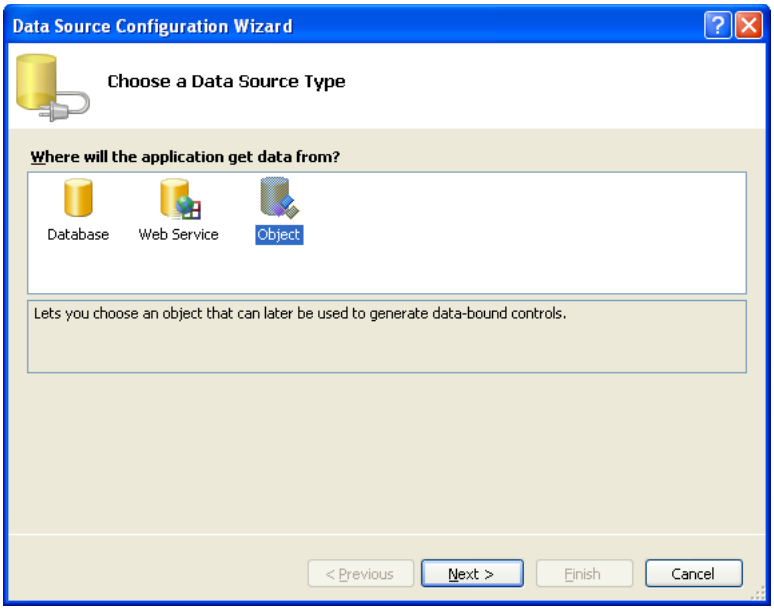
```
public class ContactInfo : IPortableObject
{
    private string name;
    private string street;
    private string city;
    private string state;
    private string zip;

    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:

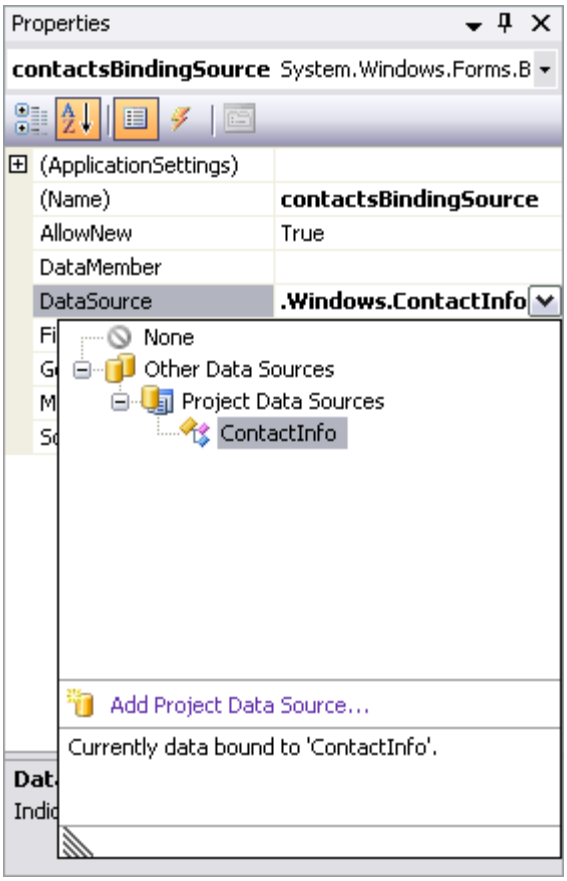
1. In the **Toolbox** window choose the `BindingSource` object and drag it onto the form.
2. Set its properties. Enter `contactsBindingSource` into the **Name** field and then set its data source by clicking the arrow button on the right end of the **DataSource** field. In the drop down window choose **Add Project Data Source...** and the **Data Source Configuration Wizard** will appear. Chose **Object** and find the `ContactInfo` class in your project.

Figure 26–6 Using Data Source Wizard to Bind a Control to a Data Source



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 26–7.

Figure 26–7 Choosing a Data Source to Bind to the Control



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 "Responding to Cache Events" on page 19-9. 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 26–6 Adding Listeners

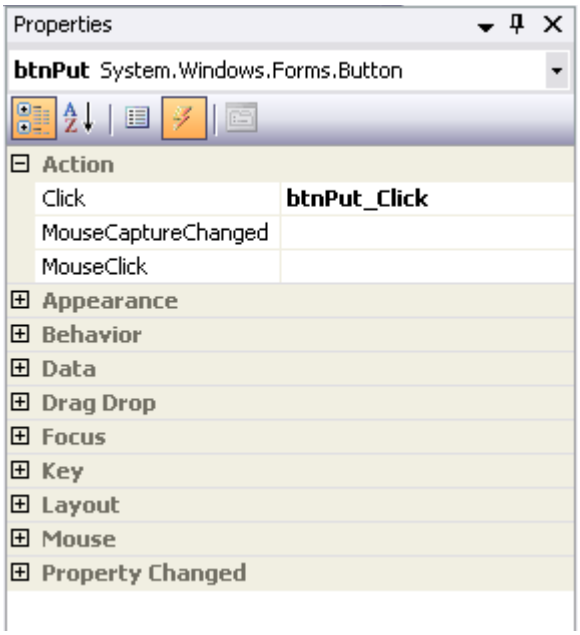
```
/// <summary>
/// Named cache.
/// </summary>
private INamedCache cache;

/// <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 26–8 Properties Window



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 26–7 illustrates the event code in the sample Windows application:

Example 26–7 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 26–8:

Example 26–8 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 26–9 illustrates helper methods used by the event handlers in the previous example:

Example 26–9 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);
        }
    }
}
```

27

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.

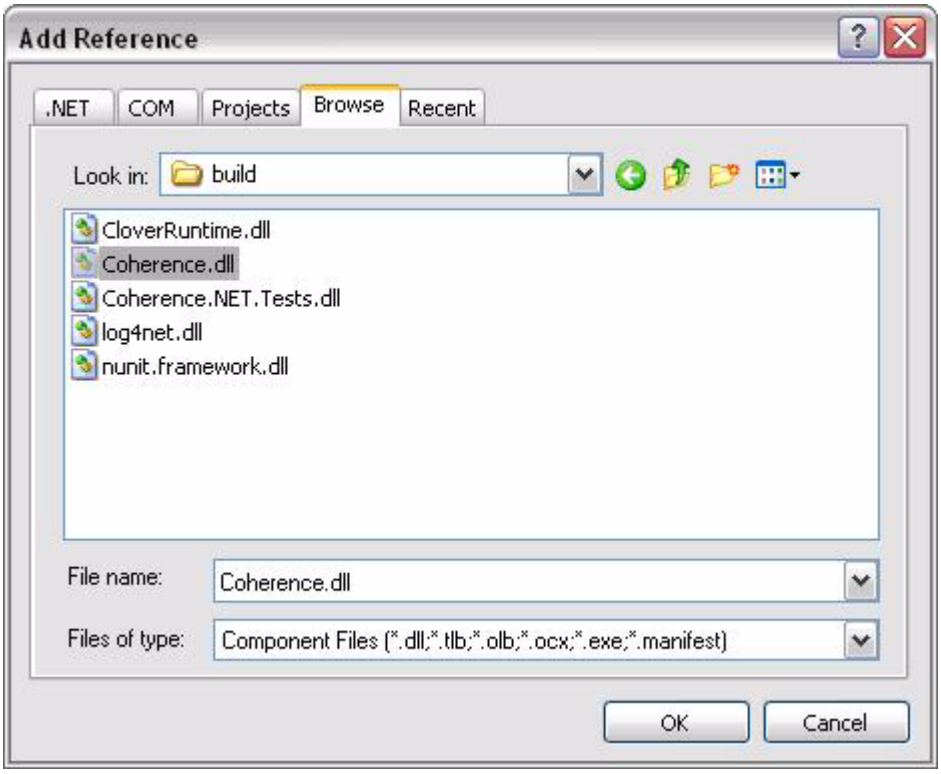
Click the **OK** button to generate a new solution and empty ASP.NET application.

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 27–1 Coherence.dll File in the Add Reference Window



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 27–2 illustrates a valid `Web.config` configuration file:

Example 27–1 Sample Web.config Configuration File

```
<?xml version="1.0"?>
<configuration>
   <configSections>
      <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler,
         Coherence, Version=3.6.0.0, Culture=neutral,
         PublicKeyToken=0ADA89708FDF1F9A"/>
   </configSections>
   <coherence>
         <cache-factory-config>web://~/Config/coherence.xml</cache-factory-config>
         <cache-config>web://~/Config/cache-config.xml</cache-config>
         <pof-config>web://~/Config/pof-config.xml</pof-config>
   </coherence>
   <appSettings/>
   <connectionStrings/>
   <system.web>
      <globalization culture="en-US" uiCulture="en-US"/>
      <sessionState mode="Custom" customProvider="CoherenceSessionProvider"
         timeout="20">
         <providers>
```

```
            <add name="CoherenceSessionProvider"
               type="Tangosol.Web.CoherenceSessionStore, Coherence,
               Version=3.6.0.0, Culture=neutral,
               PublicKeyToken=0ADA89708FDF1F9A"/>
         </providers>
      </sessionState>
      <compilation debug="false" defaultLanguage="c#">
         <assemblies>
            <add assembly="System.Windows.Forms, Version=2.0.0.0, Culture=neutral,
               PublicKeyToken=B77A5C561934E089"/>
            <add assembly="Coherence, Version=3.6.0.0, Culture=neutral,
               PublicKeyToken=0ADA89708FDF1F9A"/>
         </assemblies>
      </compilation>
      <authentication mode="Windows"/>
      <customErrors mode="Off"/>
   </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 18-3). 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 27–2 illustrates a sample `coherence.xml` configuration file:

Example 27–2 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 27–3 illustrates a sample `cache-config.xml` configuration file:

Example 27–3 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>
         <cache-mapping>
            <cache-name>aspnet-session-storage</cache-name>
            <scheme-name>extend-direct</scheme-name>
         </cache-mapping>
         <cache-mapping>
            <cache-name>aspnet-session-overflow</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 27–4illustrates a sample `pof-config.xml` configuration file:

Example 27–4 Sample pof-config.xml Configuration File

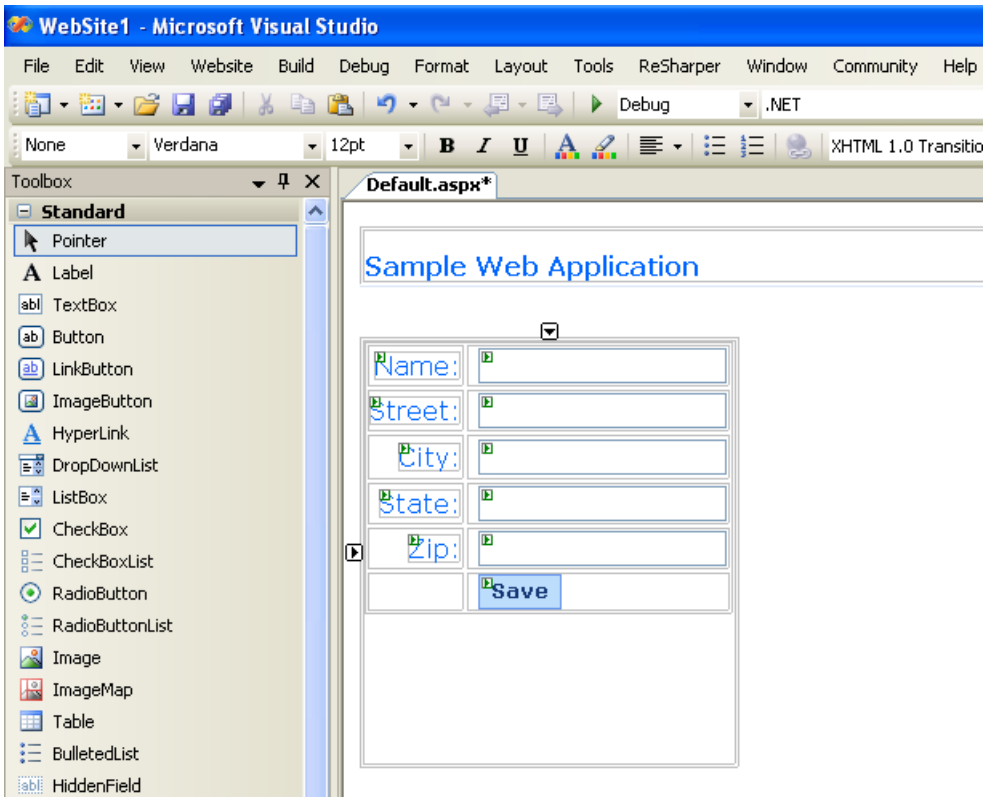
```
<?xml version="1.0"?>

<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

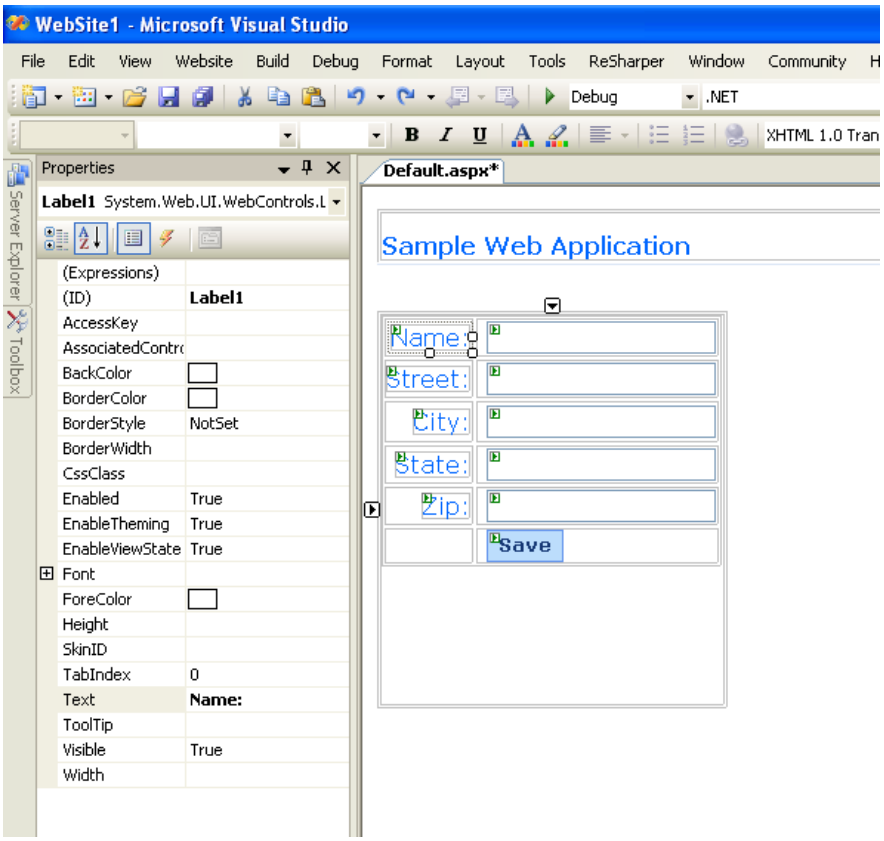
Switch to the **Design** tab for the `Default.aspx` page and from the **Toolbox** pane add the appropriate controls by dragging and dropping them on the page. You will need **TextBox** controls for the `Name`, `Street`, `City`, `State`, and `Zip` fields and corresponding label controls for each. This is illustrated in Figure 27–2.

Figure 27–2 Adding Controls for the .aspx Page



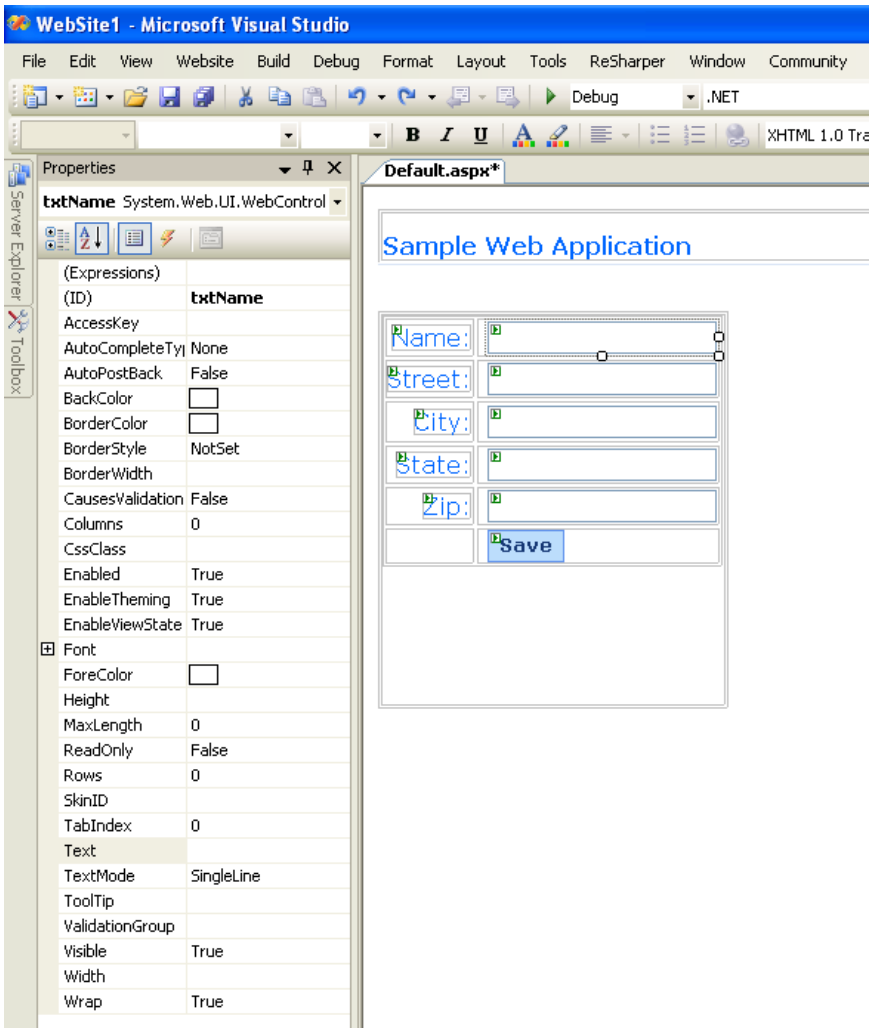
After placing them on the page, you should change the **ID** and **Text** property for each control. As we won't be using labels in the code, you can leave their **ID** property values as generated, and just put appropriate labels in the **Text** property. You should name the **ID** and **TextBox** controls `txtName`, `txtStreet`, and so on. Add one button and rename its **ID** to `btnSave` and Text property to **Save**. This is illustrated in Figure 27–3.

Figure 27–3 Changing IDs and Properties for Data Controls



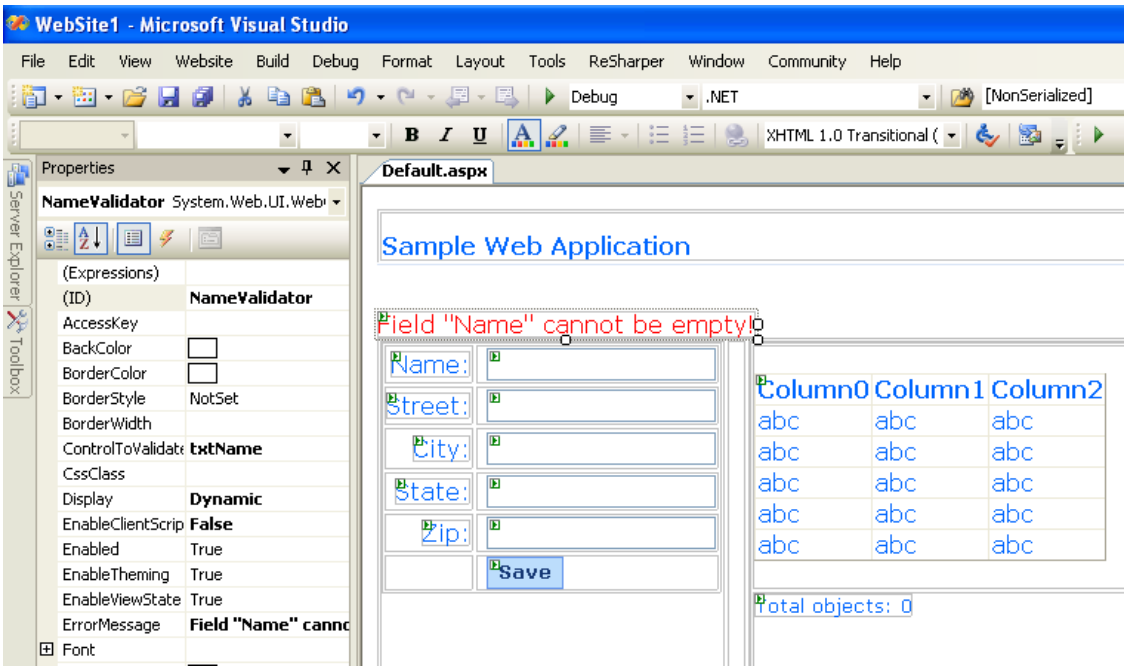
Add one button and rename its **ID** to `btnClear` and **Text** property to `Clear`. This is illustrated in Figure 27–4

Figure 27–4 Adding a "Clear" Button to the Application



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 27–5:

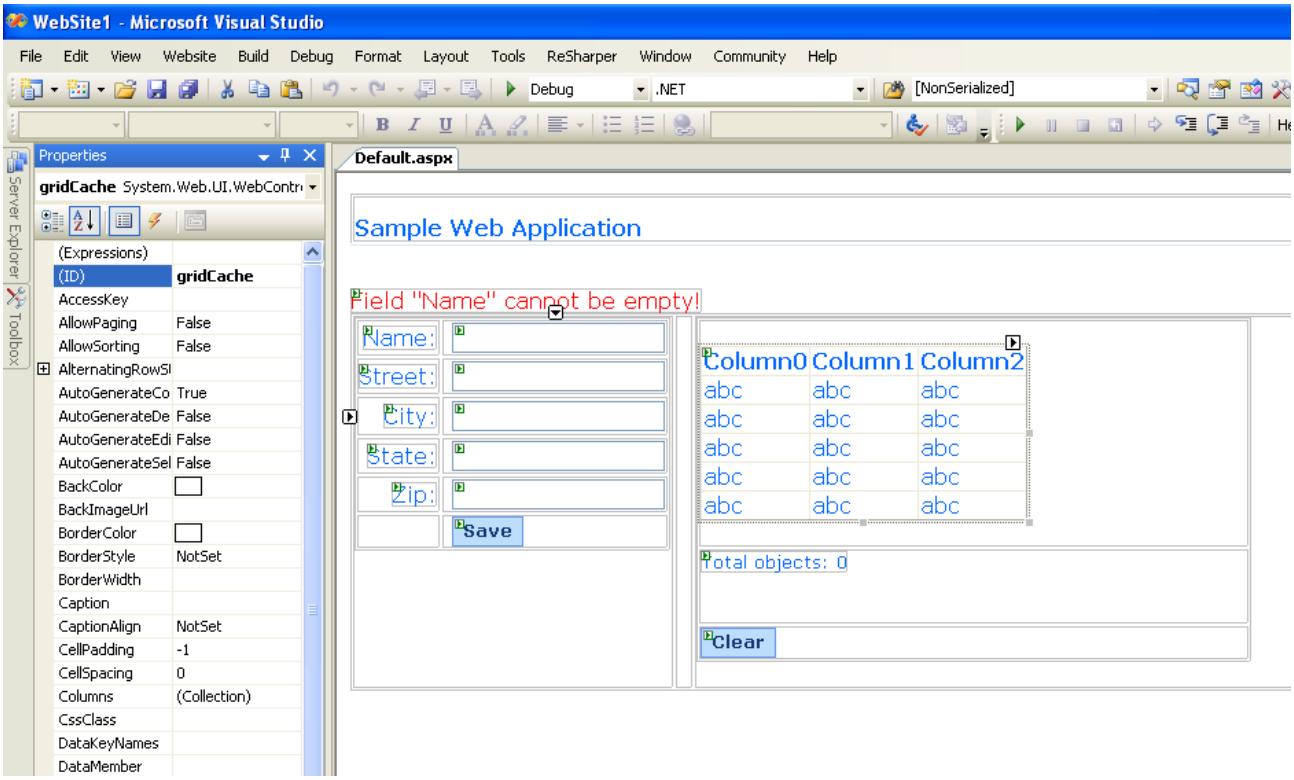
Figure 27–5 Adding a Field Validator and Setting its Properties



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 27–6.

Figure 27–6 Adding a GridView Control and an ObjectDataSource



Example 27–5 illustrates code for the `GridView` control source:

Example 27–5 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>
```

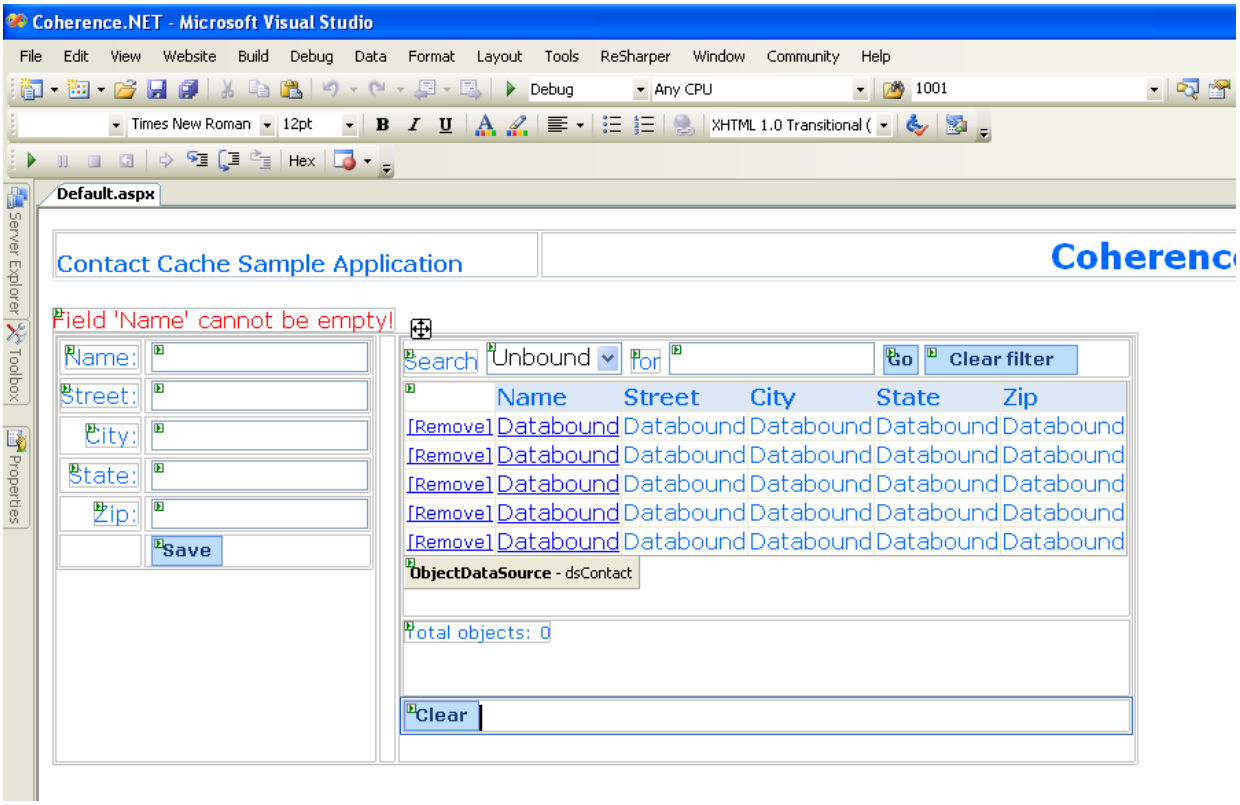
Example 27–6 illustrates the `ObjectDataSource` code.

Example 27–6 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 27–7.

Figure 27–7 Search Pane



Implement the Web Application

Global.asax File

In order to free up resources in the cluster when your ASP.NET application terminates, you need to call `CacheFactory.Shutdown()` within the `Application_End` event handler in `Global.asax`. Example 27–7 illustrates a `Global.asax` file and shows you how to do that, and also adds the call which redirects the user to an error page if an exception occurs.

Example 27–7 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();
        }

        CacheFactory.Shutdown();
    }

    void Application_Error(object sender, EventArgs e)
    {
        Server.Transfer("ConnectionError.html");
    }
</script>
```

Business Object Definition

Example 27–8 illustrates the definition of the `ContactInfo` business object.

Example 27–8 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 27–9 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 27–9 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 27–10.

Example 27–10 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 27–11 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 27–11 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 27–12 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 27–12 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 27–13 retrieves an `ContactInfo` object from the cache by a specified key:

Example 27–13 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 27–14 illustrates an the event handler for the `btnSave` button:

Example 27–14 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 27–15 illustrates the event handler for the `btnClear` button:

Example 27–15 Event Handler for a :Clear" Button

```
protected void btnClear_Click(object sender, EventArgs e)
{
    NameValidator.Enabled = false;

    ContactCache.Clear();
    RefreshDataGridAndRenderPage();

    NameValidator.Enabled = true;
}
```

Example 27–16 illustrates the event handler for the `btnSearch` button:

Example 27–16 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 27–17 illustrates the event handler for the `btnClearFilter` button:

Example 27–17 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.