Oracle® Coherence

Developing Remote Clients for Oracle Coherence

14.1.1.2206
F44672-08
Jan 2025

Oracle Coherence Developing Remote Clients for Oracle Coherence, 14.1.1.2206

F44672-08

Primary Author: Oracle Corporation

Contents

10 Using the Coherence for C++ Client API

11 Building Integration Objects (C++)

12 Querying a Cache (C++)

13 Performing Continuous Queries (C++)

18 Building Integration Objects (.NET)

19 Using the Coherence .NET Client Library

20 Performing Continuous Queries (.NET)

21 Performing Remote Invocations (.NET)

22 Performing Transactions (.NET)

23 Managing ASP.NET Session State

28 Introduction to Coherence REST

29 Building Your First Coherence REST Application

30 Performing Grid Operations with REST

31 Deploying Coherence REST

32 Modifying the Default REST Implementation

A REST Configuration Elements

B Integrating with F5 BIG-IP LTM

Preface

Developing Remote Clients for Oracle Coherence describes how to configure Coherence*Extend and how to develop remote clients in Java, C++, and .NET. This document also includes instructions for developing remote clients using Coherence REST.

This preface includes the following sections:

Audience

Developing Remote Clients for Oracle Coherence is intended for the following audiences:

- **Primary Audience** – Application developers who want to write and deploy clients that use C++, .NET, and REST to interact with remote caches that reside in a Coherence cluster.
- **Secondary Audience** – System architects who want to understand core Oracle Coherence concepts and want to build data grid-based solutions that include remote clients.

The audience must be familiar with the respective client technologies as well as Java to use this guide. In addition, the examples in this guide require the installation and use of the Oracle Coherence product. For details about installing Coherence for Java and the respective client technologies, see *Installing Oracle Coherence*. The use of an IDE is not required to use this guide, but is recommended to facilitate working through the examples.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

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

- *Administering HTTP Session Management with Oracle Coherence*Web*
- *Administering Oracle Coherence*
- *Developing Applications with Oracle Coherence*
- *Installing Oracle Coherence*
- *Integrating Oracle Coherence*
- *Managing Oracle Coherence*
- *Securing Oracle Coherence*
- *Java API Reference for Oracle Coherence*
- *C++ API Reference for Oracle Coherence*
- *.NET API Reference for 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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Part I

Getting Started with Coherence*Extend

Learn about Coherence*Extend proxies, clients, configuration, and best practices. Try creating a simple Coherence*Extend application.

Part I contains the following chapters:

1

Introduction to Coherence*Extend

Coherence*Extend includes support for native Coherence clients (Java, C++, and .NET) and non-native Coherence clients (REST and Memcached).
This chapter includes the following sections:

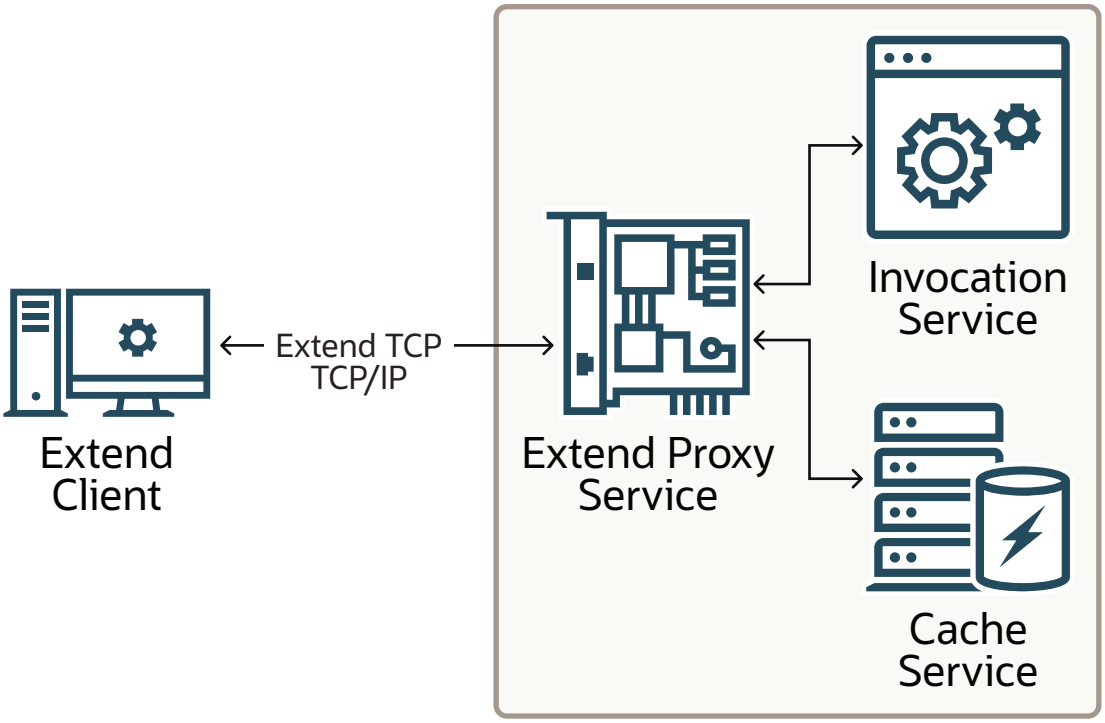
Overview of Coherence*Extend

Coherence*Extend "extends" the reach of the core Coherence TCMP cluster to a wider range of consumers, including desktops, remote servers, and computers 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 multiple Coherence clusters connected through a high-latency, unreliable WAN. 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 include 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. After 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.

Extend Clients

Extend clients (also referred to as real-time clients) can be created for the Java, .NET, and C++ platforms and have access to the same 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.
Extend 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
- 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.
- 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.

For a complete list of real-time client features, see Oracle Coherence Products in *Oracle Fusion Middleware Licensing Information User Manual*.

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

See Using Portable Object Format in *Developing Applications with Oracle Coherence* for detailed information on using POF with Java. For more information on using POF with C++ and C#, see Building Integration Objects (C++), and Building Integration Objects (.NET) , respectively.

Understanding Extend Client Configuration Files

Extend clients use many of the same cluster-side configuration files except they are packaged and deployed with the client.
Extend client configuration 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 schema for this file is the `coherence-cache-config.xsd` file for Java and C++ clients and the `cache-config.xsd` file for .NET clients. See Cache Configuration Elements in *Developing Applications with Oracle Coherence*.

  At run time, the first cache configuration file that is found on the classpath is used. The `coherence.cacheconfig` system property can also be used to explicitly specify a cache configuration file. The file can also be set programmatically. See Specifying a Cache Configuration File in *Developing Applications with Oracle Coherence*.

- POF Configuration Deployment Descriptor – This file is used to specify custom data types when using POF to serialize objects. The schema for this file is the `coherence-pof-config.xsd` file for Java and C++ clients and the `pof-config.xsd` file for .NETclients. See POF User Type Configuration Elements in *Developing Applications with Oracle Coherence*.

  At run time, the first POF configuration file that is found on the classpath is used. The `coherence.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 and a POF configuration file that is specific to the user types used in the client. See Specifying a POF Configuration File in *Developing Applications with Oracle Coherence*.

- Operational Override File – This file is used to override the operational deployment descriptor, which is used to specify the operational and run-time 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 schema for this file is the `coherence-operational-config.xsd` file for Java and C++ clients and the `coherence.xsd` file for .NET clients. See Operational Configuration Elements in *Developing Applications with Oracle Coherence*.

  At run time, the first operational override file (`tangosol-coherence-override.xml`) that is found on the classpath is used. The `coherence.override` system property can also be used to explicitly specify an operational override file. The file can also be set programmatically. See Using the Default Operational Override File in *Developing Applications with Oracle Coherence*.

Non-Native Client Support

Coherence provides remote access to caches from REST-based or Memcached-based clients. As with Coherence*Extend clients, non-native clients use the resources of a cluster without becoming cluster members.
REST and Memcached client APIs are available for many popular programming languages, allowing Coherence to be used in heterogeneous environments. Non-native clients can also be used to ease the migration to a Coherence solution that uses the native Coherence client APIs.

This section includes the following topics:

- REST Client Support

- Memcached Client Support

REST Client Support

Coherence provides a REST implementation that provides access to cache operations over the HTTP protocol. Any REST client API can use Coherence caching. REST support is provided either through an embedded HTTP server that is configured as an extend-like acceptor on a proxy server, or through deployment to any Java EE-based application server. See Using Coherence REST .

Memcached Client Support

Coherence can be used as a drop-in replacement for memcached servers. Any memcached client API that supports the memcached binary protocol can use Coherence distributed caching. Memcached support is provided through a memcached adaptor that is implemented as an extend-like acceptor that runs on a proxy server. See Using Memcached Clients with Oracle Coherence in *Integrating Oracle Coherence*.

2

Building Your First Extend Application

Build and run a simple Coherence*Extend client that accesses and uses a Coherence cache. The example client that is 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. For complete C++ and .NET examples, see the Coherence Examples that are distributed as part of the Coherence for Java distribution.
This chapter includes the following sections:

Overview of the Extend Example

The Coherence*Extend example 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 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 a proxy and cache to be configured in the cluster's cache configuration deployment descriptor. The extend proxy configured in this example is automatically assigned a proxy port to listen for client TCP/IP communication. 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"?>

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <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>ExtendTcpCacheService</service-name>
         <autostart>true</autostart>
      </proxy-scheme>
   </caching-schemes>
</cache-config>
```

3. Save and close the file.

Step 2: Configure the Client Side

The example client queries a remote cache and also invokes a task which is run on a remote cluster node. To complete these operations, the example extend client requires a remote cache scheme and a remote invocation scheme. Invoking tasks is considered a more advanced use case.
The remote cache scheme includes a service name that matches the service name of a proxy service on the cluster to which the client connects. In addition, the cache name that is used in the cluster must also be used as the name of the remote cache scheme. For this example (based on Step 1), the remote cache scheme service name is `ExtendTcpCacheService` and the cache name is `dist-extend`. Lastly, the remote cache scheme includes the address and port of the cluster's name service, which is used to find a proxy. The name service runs on the cluster port which is 7574 by default.

The example extend client invokes a task on the remote cache and therefore requires a remote invocation scheme. The remote invocation scheme defines the `ExtendTcpInvocationService` service, which allows the client to create an `Invocable` instance and send it to the cluster for processing. The remote invocation scheme uses the name service to find a proxy and includes the name of the proxy service to which it 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"?>

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <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>
               <name-service-addresses>
                  <socket-address>
                     <address>127.0.0.1</address>
                     <port>7574</port>
                  </socket-address>
               </name-service-addresses>
            </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>
         <proxy-service-name>ExtendTcpCacheService</proxy-service-name>
         <initiator-config>
            <tcp-initiator>
               <name-service-addresses>
                  <socket-address>
                     <address>127.0.0.1</address>
                     <port>7574</port>
                  </socket-address>
               </name-service-addresses>
            </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

The client application for this example 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`. The client writes the value to the system output before exiting.

> **Note:**
>
> - The client class must be on the classpath for all cache servers in the cluster. The `TestClient$1` class is an anonymous inner class that is generated during compilation. It is serialized and sent to the `InvocationService` running on a cluster member. In this example, the client and cluster member run on a single computer. Therefore, both Java invocations use the same classpath.
> - This example could also be run on a Coherence node (that is, within the cluster) as is. 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:

```
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()
                    {
                    System.out.println("This has been run by
```

```
                        ExtendTcpInvocationService on: " +
                        CacheFactory.getCluster().getLocalMember());
                    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, the single result of the execution is keyed by the local `Member`, which is 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 `coherence.cacheconfig` system property:

```
java -cp COHERENCE_HOME\lib\coherence.jar;PATH_TO_CLIENT -
Dcoherence.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 192.168.1.5:7077
```

Step 5: Run the Application

The `TestClient` application is started using the `java` command and must be configured to use the cache configuration file.
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 `coherence.cacheconfig` system property:

```
java -cp .;COHERENCE_HOME\lib\coherence.jar -Dcoherence.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.

> **Note:**
>
> Check the cache server process output for the message confirming that the invocation task was executed remotely using the `ExtendTcpInvocationService` service.
>
> ```
> This has been run...
> ```

3
Configuring Extend Proxies

Extend proxies must be configured to allow clients to access and use the caches that are defined in a Coherence cluster. The instructions in this chapter provide basic setup and do not represent a complete configuration reference.
This chapter includes the following sections:

- Overview of Configuring Extend Proxies
- Defining Extend Proxy Services
- Defining Caches for Use By Extend Clients
- Disabling Storage on a Proxy Server
- Starting a Proxy Server
  A proxy server can be started using the `DefaultCacheServer` class.

Overview of Configuring Extend Proxies

Extend proxies are Coherence cluster members that host one or more proxy services. A proxy service is the underlying cluster service that extend clients use to access caches in a cluster. Proxies and caches must be configured before extend clients can retrieve and store data in a cluster.
Extend proxies and cache servers run in the same cluster member process (`DefaultCacheServer` process). Collocating extend proxies with cache servers simplifies cluster setup and ensures that proxies automatically scale with the cluster. However, extend proxies can also be configured as separate members of the cluster. In this case, the proxies and cache servers are organized as separate tiers that can scale independently.

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. See Specifying a Cache Configuration File in *Developing Applications with Oracle Coherence*.

Defining 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 proxies 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.
This section includes the following topics:

- Defining a Single Proxy Service Instance
- Defining Multiple Proxy Service Instances
- Defining Multiple Proxy Services
- Explicitly Configuring Proxy Addresses
- Disabling Cluster Service Proxies

- Specifying Read-Only NamedCache Access

Defining a Single Proxy Service Instance

Extend proxy services are configured within a `<caching-schemes>` node using the `<proxy-scheme>` element. Example 3-1 defines a proxy service named `ExtendTcpProxyService` and includes the `<autostart>` element that is set to `true` so that the service automatically starts at a cluster node. See proxy-scheme in *Developing Applications with Oracle Coherence*.

As configured in Example 3-1, a proxy address and ephemeral port is automatically assigned and registered with a cluster name service. Extend clients connect to the name service, which then redirects the client to the address of the requested proxy. The use of the name service allows proxies to run on ephemeral addresses, which simplifies port management and configuration. See Explicitly Configuring Proxy Addresses .

Example 3-1 Extend Proxy Service Configuration

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

Defining Multiple Proxy Service Instances

Multiple extend proxy service instances can be defined in order to support an expected number of client connections and to support fault tolerance and load balancing. Client connections are automatically balanced across proxy service instances. The algorithm used to balance connections depends on the load balancing strategy that is configured. See Load Balancing Connections.

To define multiple proxy service instances, include a proxy service definition in multiple proxy servers and use the same service name for each proxy service. Proxy services that share the same service name are considered peers.

The following examples define two instances of the `ExtendTcpProxyService` proxy service. The proxy service definition is included in each cache server's respective cache configuration file within the `<proxy-scheme>` element. The same configuration can be used on all proxies including proxies that are co-located on the same machine.

On proxy server 1:

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

On proxy server 2:

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
```

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

Defining Multiple Proxy Services

Multiple extend proxy services can be defined in order to provide different applications with their own proxies. Extend clients for a particular application can be directed toward specific proxies to provide a more predictable environment.

The following example defines two extend proxy services: `ExtendTcpProxyService1` and `ExtendTcpProxyService2`:

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService1</service-name>
      <autostart>true</autostart>
   </proxy-scheme>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService2</service-name>
      <autostart>true</autostart>
   </proxy-scheme>
</caching-schemes>
...
```

Explicitly Configuring Proxy Addresses

Older extend clients that predate the name service or clients that have specific firewall constraints may require specific proxy addresses. In this case, the proxy can be explicitly configured to listen on a specific address and port. See Configuring Firewalls for Extend Clients.

The `<tcp-acceptor>` subelement includes the address (IP, or DNS name, and port) that an extend proxy service listens to for TCP/IP client communication. The address can be explicitly defined using the `<address-provider>` element, or the address can be defined within an operational override configuration file and referenced using the `<address-provider>` element. The latter approach decouples the address configuration from the proxy scheme definition and allows the address to change at runtime without having to change the proxy definition. See Using Address Provider References for TCP Addresses.

Example 3-2 defines a proxy service named `ExtendTcpProxyService` and is set up to listen for client requests on a TCP/IP socket that is bound to `198.168.1.5` and port `7077`.

Example 3-2 Explicitly Configured Proxy Service Address

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <address-provider>
               <local-address>
                  <address>192.168.1.5</address>
                  <port>7077</port>
               </local-address>
            </address-provider>
         </tcp-acceptor>
```

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

The specified port should be outside of the computer's ephemeral port range to ensure that it is not automatically assigned to other applications. If the specified port is not available, then the default behavior is to select the next available port. To disable automatic port adjustment, add a `<port-auto-adjust>` element that includes the value `false`. Or, to specify a range of ports from which the port is selected, include a port value that represents the upper limit of the port range. The following example sets a port range from 7077 to 8000:

```
<acceptor-config>
   <tcp-acceptor>
      <address-provider>
         <local-address>
            <address>192.168.1.5</address>
            <port>7077</port>
            <port-auto-adjust>8000</port-auto-adjust>
         </local-address>
      </address-provider>
   </tcp-acceptor>
</acceptor-config>
```

The `<address>` element supports using CIDR notation as a subnet and mask (for example `192.168.1.0/24`). CIDR simplifies configuration by allowing a single address configuration to be shared across computers on the same sub-net. Each cluster member specifies the same CIDR address block and a local NIC on each computer is automatically found that matches the address pattern. The `/24` prefix size matches up to 256 available addresses: from `192.168.1.0` to `192.168.1.255`. The `<address>` element also supports external NAT addresses that route to local addresses; however, both addresses must use the same port number.

For solutions that do not require a firewall, you can omit the IP and port values which causes the proxy to use the same IP address and port as TCMP (7574 by default). The port can also be configured with a listen port of `0`, which causes the proxy to listen on a system assigned ephemeral port. This configuration is the same as omitting the `<acceptor-config>` element as shown in Defining a Single Proxy Service Instance. If the proxy is configured to use ephemeral ports, then clients must use the cluster name service to locate the proxy.

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>
```

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 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. See also Using Caches in *Developing Applications with Oracle Coherence*. This section provides basic examples of three cache types that are commonly used be extend clients.

A Basic Partitioned (distributed) Cache

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

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

A Basic Near Cache

A typical near cache is configured to use a local cache (thread safe, highly concurrent, size-limited and possibly auto-expiring) as the front cache and a remote cache as a back cache. A near ache 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 are kept synchronized and <listener> for specifying a listener which is notified of events occurring on the cache.

Example 3-3 demonstrates a near cache configuration.

Example 3-3 Near Cache Configuration

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-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>
   </caching-schemes>
</cache-config>
```

A Basic Local Cache

A local cache is a cache that is local to (completely contained within) a particular application. There are several attributes of a local cache that are particularly interesting:

- A 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.
- A local cache can be size-limited. Size-limited 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.
- A 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.
- A local cache is thread safe and highly concurrent.
- A local cache provides cache "get" statistics. It maintains hit and miss statistics. These run-time statistics accurately project the effectiveness of the cache and are used to adjust size-limiting and auto-expiring settings accordingly while the cache is running.

The element for configuring a 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. When 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 are kept by the cache before being marked as expired. Any attempt to read an expired entry results 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 only resides 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 3-4 demonstrates a local cache configuration.

Example 3-4 Local Cache Configuration

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
  <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>
      <cachestore-scheme>
        <class-scheme>
          <class-name>ExampleCacheStore</class-name>
        </class-scheme>
      </cachestore-scheme>
      <pre-load>true</pre-load>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

Disabling Storage on a Proxy Server

You must explicitly configure a proxy service to not store any data.

Consider disabling storage on a proxy only if you plan to run proxies and storage nodes in two separate tiers and scale them independently; although, this is generally not necessary and requires more careful planning. A best practice is to run proxy services on cluster members that also store data in the cluster (cache servers) because scaling cache servers increases both cluster storage capacity as well as aggregate proxy bandwidth.

> **Note:**
>
> Storage-enabled proxies bypass the front cache of a near cache and operate directly against the back cache if it is a partitioned cache.

To disable storage on a proxy server, use the `coherence.distributed.localstorage` Java property set to `false` when starting the cluster member. For example:

```
-Dcoherence.distributed.localstorage=false
```

Storage can also be disabled in the cache configuration file as part of a distributed cache definition by setting the `<local-storage>` element to `false`. See distributed-scheme in *Developing Applications with Oracle Coherence*.

```
...
<distributed-scheme>
   <scheme-name>dist-default</scheme-name>
   <local-storage>false</local-storage>
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
...
```

Starting a Proxy Server

A proxy server can be started using the `DefaultCacheServer` class.

To start a proxy server:

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 runs.
3. Run the `DefaultCacheServer` class and include the location of the cache configuration file and the operational configuration file. For example:

```
java -cp path_to_configuration_files;coherence.jar
   com.tangosol.net.DefaultCacheServer
```

4
Configuring Extend Clients

Coherence*Extend clients are configured to connect to a proxy service on the cluster and access to Coherence caches. 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.
This chapter includes the following sections:

Overview of Configuring Extend Clients

Coherence*Extend requires configuration both on the client side and the cluster side. 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. On the cluster side, extend proxy services are setup to accept client requests. Extend clients and extend proxy services communicate using TCP/IP.
Extend clients are 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. 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. See Specifying a Cache Configuration File in *Developing Applications with Oracle Coherence*.

Extend clients use the remote cache service and the remote invocation service to interact with a Coherence 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 client application starts.

Defining a Remote Cache

A remote cache is specialized cache service that routes cache operations to a cache on the Coherence cluster. The remote cache and the cache on the cluster must have the same cache 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. Example 4-1 creates a remote cache scheme that is named

`ExtendTcpCacheService` and connects to the name service, which then redirects the request to the address of the requested proxy service. The use of the name service simplifies port management and firewall configuration. See remote-cache-scheme in *Developing Applications with Oracle Coherence*.

Example 4-1 Remote Cache Definition

```
...
<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>
            <name-service-addresses>
               <socket-address>
                  <address>198.168.1.5</address>
                  <port>7574</port>
               </socket-address>
            </name-service-addresses>
         </tcp-initiator>
         <outgoing-message-handler>
            <request-timeout>5s</request-timeout>
         </outgoing-message-handler>
      </initiator-config>
   </remote-cache-scheme>
</caching-schemes>
...
```

If the `<service-name>` value is different than the proxy scheme `<service-name>` value on the cluster, use the `<proxy-service-name>` element to enter the value of the `<service-name>` element that is configured in the proxy scheme. For example:

```
       <remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <proxy-service-name>SomeOtherProxyService</proxy-service-name>
      ...
```

If the client is in a different cluster than the proxy server, use the <`cluster-name`> element to specify the cluster name of the proxy server. For example:

```
<remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
      <service-name>ExtendTcpCacheService</service-name>
      <cluster-name
```

```
system-property="cache.server.cluster">CacheCluster</cluster-name>
      ...
```

As configured in Example 4-1, the remote cache scheme uses the `<name-service-addresses>` element to define the socket address (IP, or DNS name, and port) of the name service on the cluster. The `<address>` element also supports external NAT addresses that route to local addresses; however, both addresses must use the same port number. The name service listens on the cluster port (7574) by default and is available on all machines running cluster nodes. If the target cluster uses the default cluster port, then the port can be omitted from the configuration. Moreover, extend clients by default use the cluster discovery addresses to find the cluster and proxy. If the extend client is on the same network as the cluster, then no specific configuration is required as long as the client uses a cache configuration file that specifies the same cluster-side cluster name.

The `<name-services-addresses>` element also supports the use of the `<address-provider>` element for referencing a socket address that is configured in the operational override configuration file. See Using Address Provider References for TCP Addresses and Connecting to Specific Proxy Addresses.

> **Note:**
>
> Clients that are configured to use a name service can only connect to Coherence versions that also support the name service. In addition, for previous Coherence releases, the name service automatically listened on a member's unicast port instead of the cluster port.

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

```
...
<caching-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>
            <name-service-addresses>
               <socket-address>
                  <address>198.168.1.5</address>
                  <port>7574</port>
               </socket-address>
            </name-service-addresses>
         </tcp-initiator>
         <outgoing-message-handler>
            <request-timeout>5s</request-timeout>
         </outgoing-message-handler>
      </initiator-config>
   </remote-cache-scheme>
</caching-schemes>
...
```

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. Example 4-2 defines a remote invocation scheme that is called `ExtendTcpInvocationService` and uses the `<name-service-address>` element to configure the address that the name service is listening on. See remote-invocation-scheme in *Developing Applications with Oracle Coherence*.

Example 4-2 Remote Invocation Scheme Definition

```
...
<caching-schemes>
   <remote-invocation-scheme>
      <scheme-name>extend-invocation</scheme-name>
      <service-name>ExtendTcpInvocationService</service-name>
      <initiator-config>
         <tcp-initiator>
            <name-service-addresses>
               <socket-address>
                  <address>198.168.1.5</address>
                  <port>7574</port>
               </socket-address>
            </name-service-addresses>
         </tcp-initiator>
         <outgoing-message-handler>
            <request-timeout>5s</request-timeout>
         </outgoing-message-handler>
      </initiator-config>
   </remote-invocation-scheme>
</caching-schemes>
...
```

If the <service-name> value is different than the proxy scheme <service-name> value on the cluster, then use the <proxy-service-name> element to enter the value of the <service-name> element that is configured in the proxy scheme. For example:

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>ExtendTcpInvocationService</service-name>
   <proxy-service-name>SomeOtherProxyService</proxy-service-name>
   ...
```

Connecting to Specific Proxy Addresses

Clients can connect to specific proxy addresses if the client predates the name service feature or if the client has specific firewall constraints. See Configuring Firewalls for Extend Clients. Example 4-1 uses the <socket-address> element to explicitly configure the address that an extend proxy service is listening on (198.168.1.5 and port 7077). The <address> element also supports external NAT addresses that route to local addresses; however, both addresses must use the same port number. The address can also be defined within an operational override configuration file and referenced using the <address-provider> element. The latter approach decouples the address configuration from the remote cache definition and allows the address to change at runtime without having to change the remote cache definition. See Using Address Provider References for TCP Addresses.

Example 4-3 Remote Cache Definition with Explicit Address

```
...
<caching-scheme-mapping>
   <cache-mapping>
      che-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>7077</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>
...
```

If multiple proxy service instances are configured, then a remote cache scheme or invocation scheme can include each proxy service addresses to ensure a client can always connect to the cluster. The algorithm used to balance connections depends on the load balancing strategy that is configured. See Load Balancing Connections.

To configure multiple addresses, 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:

```
...
<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>192.168.1.5</address>
                  <port>7077</port>
               </socket-address>
               <socket-address>
                  <address>192.168.1.6</address>
                  <port>7077</port>
               </socket-address>
            </remote-addresses>
         </tcp-initiator>
      </initiator-config>
   </remote-cache-scheme>
</caching-schemes>
...
```

While either an IP address or DNS name can be used, DNS names have an additional advantage: any IP addresses that are associated with a DNS name are automatically resolved at runtime. This allows the list of proxy addresses to be stored in a DNS server and centrally managed and updated in real time. For example, if the proxy address list is going to be `192.168.1.1`, `192.168.1.2`, and `192.168.1.3`, then a single DNS entry for hostname `ExtendTcpCacheService` can contain those addresses and a single address named `ExtendTcpCacheService` can be specified for the proxy address:

```
<tcp-initiator>
   <remote-addresses>
      <socket-address>
         <address>ExtendTcpCacheService</address>
         <port>7077</port>
      </socket-address>
   </remote-addresses>
</tcp-initiator>
```

Detecting Connection Errors

Coherence*Extend can detect and notify clients when connection errors occur. Various configuration options are available for controlling dropped connections.
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 irrecoverable error exception is 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>` element. See outgoing-message-handler in *Developing Applications with Oracle Coherence*. In particular, the `<request-timeout>` value controls the amount of time to wait for a response before abandoning the request. The `<heartbeat-interval>` and `<heartbeat-timeout>` values control the amount of time to wait for a response to a ping request before the connection is closed. As a best practice, the heartbeat timeout should be less than the heartbeat interval to ensure other members are not unnecessarily pinged and to not have multiple pings outstanding.

The following example is taken from Example 4-1 and demonstrates setting the request timeout to `5` seconds.

```
...
<initiator-config>
   ...
   <outgoing-message-handler>
      <request-timeout>5s</request-timeout>
   </outgoing-message-handler>
</initiator-config>
...
```

The following example sets the heartbeat interval to `3` seconds and the heartbeat timeout to `2` seconds.

```
...
<initiator-config>
   ...
   <outgoing-message-handler>
      <heartbeat-interval>3s</heartbeat-interval>
      <heartbeat-timeout>2s</heartbeat-timeout>
   </outgoing-message-handler>
</initiator-config>
...
```

Disabling TCMP Communication

Java-based extend clients that are located within the network must disable TCMP communication to exclusively connect to clustered services using extend proxies. If TCMP is not disabled, Java-based extend clients may cluster with each other and may even join an existing cluster. TCMP is disabled in the client-side `tangosol-coherence-override.xml` file. To disable TCMP communication, set the `<enabled>` element within the `<packet-publisher>` element to `false`. For example:

```
...
<cluster-config>
   <packet-publisher>
         <enabled system-property="coherence.tcmp.enabled">false
      </enabled>
   </packet-publisher>
</cluster-config>
...
```

The `coherence.tcmp.enabled` system property is used to specify whether TCMP is enabled instead of using the operational override file. For example:

```
-Dcoherence.tcmp.enabled=false
```

5
Advanced Extend Configuration

There are several advanced configuration options for extend clients and extend proxies that are typically used to change operational defaults or to address specific use cases.
This chapter includes the following sections:

- Using Address Provider References for TCP Addresses
- Using a Custom Address Provider for TCP Addresses
- Load Balancing Connections

Using Address Provider References for TCP Addresses

Proxy service, remote cache, and remote invocation definitions can use the `<address-provider>` element to reference a TCP socket address that is defined in an operational override configuration file instead of explicitly defining an addresses in a cache configuration file. Referencing socket address definitions allows network addresses to change without having to update a cache configuration file.
To use address provider references for TCP addresses:

1. Edit the `tangosol-coherence-override.xml` file (both on the client side and cluster side) and add a `<socket-address>` definition, within an `<address-provider>` element, that includes the socket's address and port. Use the `<address-provider>` elements's `id` attribute to define a unique ID for the socket address. See address-provider in *Developing Applications with Oracle Coherence*. The following example defines an address with `proxy1` ID:

```
...
<cluster-config>
   <address-providers>
      <address-provider id="proxy1">
         <socket-address>
            <address>198.168.1.5</address>
            <port>7077</port>
         </socket-address>
      </address-provider>
   </address-providers>
</cluster-config>
...
```

2. Edit the cluster-side `coherence-cache-config.xml` and create, or update, a proxy service definition and reference a socket address definition by providing the definition's ID as the value of the `<address-provider>` element within the `<tcp-acceptor>` element. The following example defines a proxy service that references the address that is defined in step 1:

```
...
<caching-schemes>
   <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <acceptor-config>
         <tcp-acceptor>
            <address-provider>proxy1</address-provider>
```

```
            </tcp-acceptor>
         </acceptor-config>
         <autostart>true</autostart>
      </proxy-scheme>
   </caching-schemes>
   ...
```

3. Edit the client-side `coherence-cache-config.xml` and create, or update, a remote cache or remote invocation definition and reference a socket address definition by providing the definition's ID as the value of the `<address-provider>` element within the `<tcp-initiator>` element. The following example defines a remote cache that references the address that is defined in step 1:

```
<remote-cache-scheme>
   <scheme-name>extend-dist</scheme-name>
   <service-name>ExtendTcpCacheService</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <address-provider>proxy1</address-provider>
         </remote-addresses>
      </tcp-initiator>
      <outgoing-message-handler>
         <request-timeout>5s</request-timeout>
      </outgoing-message-handler>
   </initiator-config>
</remote-cache-scheme>
```

Using a Custom Address Provider for TCP Addresses

Custom address providers dynamically assigns 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.

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>
   <acceptor-config>
      <tcp-acceptor>
         <address-provider>
            <class-name>com.MyAddressProvider</class-name>
         </address-provider>
      </tcp-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
...
```

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>
      </tcp-initiator>
      <outgoing-message-handler>
         <request-timeout>5s</request-timeout>
      </outgoing-message-handler>
   </initiator-config>
</remote-cache-scheme>
...
```

In addition, the `<address-provider>` element also supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `AddressProvider` instances and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation.

Load Balancing Connections

Extend client connections are load balanced across proxy service members. The default load balancing strategy can be changed as required.
The default proxy-based strategy distributes client connections to proxy service members that are being utilized the least. Custom proxy-based strategies can be created or the default strategy can be modified as required. As an alternative, a client-based load balance strategy can be implemented by creating a client-side address provider or by relying on randomized client connections to proxy service members. The random approach provides minimal balancing as compared to proxy-based load balancing.

Coherence*Extend can be used with F5 BIG-IP Local Traffic Manager (LTM), which provides hardware-based load balancing. See Integrating with F5 BIG-IP LTM.

This section includes the following topics:

- Using Proxy-Based Load Balancing
- Understanding the Proxy-Based Load Balancing Default Algorithm
- Implementing a Custom Proxy-Based Load Balancing Strategy
- Using Client-Based Load Balancing

Using Proxy-Based Load Balancing

Proxy-based load balancing is the default strategy that is used to balance client connections between two or more members of the same proxy service. The strategy is weighted by a proxy's existing connection count, then by its daemon pool utilization, and lastly by its message backlog.

The proxy-based load balancing strategy is configured within a `<proxy-scheme>` definition using a `<load-balancer>` element that is set to `proxy`. For clarity, the following example explicitly specifies the strategy. However, the strategy is used by default if no strategy is specified and is not required in a proxy scheme definition.

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

```
   <service-name>ExtendTcpProxyService</service-name>
   <load-balancer>proxy</load-balancer>
   <autostart>true</autostart>
</proxy-scheme>
...
```

> **Note:**
>
> If multiple proxy address are explicitly specified, clients are not required to list the full set of proxy service members in their cache configuration. However, a minimum of two proxy service members should always be configured for redundancy sake.

Understanding the Proxy-Based Load Balancing Default Algorithm

The proxy-based load balancing algorithm distributes client connections equally across proxy service members. The algorithm redirects clients to proxy service members that are being utilized the least. The following factors are used to determine a proxy's utilization:

- Connection Utilization – this utilization is calculated by adding the current connection count and pending connection count. If a proxy has a configured connection limit and the current connection count plus pending connection count equals the connection limit, the utilization is considered to be infinite.
- Daemon Pool Utilization – this utilization equals the current number of active daemon threads. If all daemon threads are currently active, the utilization is considered to be infinite.
- Message Backlog Utilization – this utilization is calculated by adding the current incoming message backlog and the current outgoing message backlog.

Each proxy service maintains a list of all members of the proxy service ordered by their utilization. The ordering is weighted first by connection utilization, then by daemon pool utilization, and then by message backlog. The list is resorted whenever a proxy service member's utilization changes. The proxy service members send each other their current utilization whenever their connection count changes or every 10 seconds (whichever comes first).

When a new connection attempt is made on a proxy, the proxy iterates the list as follows:

- If the current proxy has the lowest connection utilization, then the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy service members that have a lower connection utilization. The client then attempts to connect to a proxy service member in the order of the returned list.
- If the connection utilizations of the proxies are equal, the daemon pool utilization of the proxies takes precedence. If the current proxy has the lowest daemon pool utilization, then the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy service members that have a lower daemon pool utilization. The client then attempts to connect to a proxy service member in the order of the returned list.
- If the daemon pool utilization of the proxies are equal, the message backlog of the proxies takes precedence. If the current proxy has the lowest message backlog utilization, then the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy service members that have a lower message backlog utilization. The client then attempts to connect to a proxy service member in the order of the returned list.

- If all proxies have the same utilization, then the client remains connected to the current proxy.

Implementing a Custom Proxy-Based Load Balancing Strategy

The `com.tangosol.coherence.net.proxy` package includes the APIs that are used to balance client load across proxy service members.

A custom strategy must implement the `ProxyServiceLoadBalancer` interface. New strategies can be created or the default strategy (`DefaultProxyServiceLoadBalancer`) can be extended and modified as required. For example, to change which utilization factor takes precedence on the list of proxy services, extend `DefaultProxyServerLoadBalancer` and pass a custom `Comparator` object in the constructor that imposes the desired ordering. Lastly, the client's `Member` object (which uniquely defines each client) is passed to a strategy. The `Member` object provides a means for implementing client-weighted strategies. See Specifying a Cluster Member's Identity in *Developing Applications with Oracle Coherence*.

To enable a custom load balancing strategy, include an `<instance>` subelement within the `<load-balancer>` element and provide the fully qualified name of a class that implements the `ProxyServiceLoadBalancer` interface. The following example enables a custom proxy-based load balancing strategy that is implemented in the `MyProxyServiceLoadBalancer` class:

```
...
<load-balancer>
   <instance>
      <class-name>package.MyProxyServiceLoadBalancer</class-name>
   </instance>
</load-balancer>
...
```

In addition, the `<instance>` element also supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `ProxyServiceLoadBalancer` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. See instance in *Developing Applications with Oracle Coherence*.

Using Client-Based Load Balancing

The client-based load balancing strategy relies upon a client address provider implementation to dictate the distribution of clients across proxy service members. If no client address provider implementation is provided, the extend client tries each configured proxy service in a random order until a connection is successful. See Using a Custom Address Provider for TCP Addresses.

The client-based load balancing strategy is configured within a `<proxy-scheme>` definition using a `<load-balancer>` element that is set to `client`. For example:

```
...
<proxy-scheme>
   <service-name>ExtendTcpProxyService1</service-name>
   <load-balancer>client</load-balancer>
   <autostart>true</autostart>
</proxy-scheme>
...
```

The above configuration sets the client strategy on a single proxy service and must be repeated for all proxy services that are to use the client strategy. To set the client strategy as

the default strategy for all proxy services if no strategy is specified, override the `load-balancer` parameter for the proxy service type in the operational override file. For example:

```
...
<cluster-config>
   <services>
      <service id="7">
         <init-params>
            <init-param id="12">
               <param-name>load-balancer</param-name>
               <param-value>client</param-value>
            </init-param>
         </init-params>
      </service>
   </services>
</cluster-config>
...
```

6

Best Practices for Coherence*Extend

There are best practices and guidelines to consider when configuring and running Coherence*Extend.
This chapter includes the following sections:

Do Not Run a Near Cache on a Proxy Server

Running a near cache on a proxy server results in higher heap usage and more network traffic on the proxy nodes with little to no benefit. By definition, a near cache provides local cache access to both recently and often-used data. If a proxy server is configured with a near cache, it locally caches data accessed by its remote clients. It is unlikely that these clients are consistently accessing the same subset of data, thus resulting in a low hit ratio on the near cache. 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.

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

NIO memory is used for TCP connections into the proxy and for POF serialization and deserialization. The amount of off-heap NIO space should be equal to the maximum heap space.
On Oracle JVMs, NIO memory can be set manually if it is not already set:

```
-XX:MaxDirectMemorySize=MAX_HEAP_SIZE
```

Configure Proxy Service Thread Pooling

You can change the thread pool default settings to optimize client performance. Proxy services use a dynamic thread pool for daemon (worker) threads. The thread pool automatically adds and removes threads based on the number of client requests, total backlog of requests, and the total number of idle threads. The thread pool helps ensure that there are enough threads to meet the demand of extend clients and that resources are not waisted on idle threads.
This section includes the following topics:

Understanding Proxy Service Threading

Each application has different thread requirements based on the number of clients and the amount of operations being performed. Performance should be closely monitored to ensure that there are enough threads to service client requests without saturating clients with too many threads. In addition, log messages are emitted when the thread pool is using its maximum amount of threads, which may indicate additional threads are required.

Client applications are classified into two general categories: active applications and passive applications. In active applications, the extend clients send many requests (put, get, and so on) which are handled by the proxy service. The proxy service requires a large number of threads to sufficiently handle these numerous tasks.

In passive applications, the client waits on events (such as map listeners) based on some specified criteria. Events are handled by a distributed cache service. This service uses worker threads to push events to the client. For these tasks, the thread pool configuration for the distributed cache service should include enough worker threads. See distributed-scheme in *Developing Applications with Oracle Coherence*.

> **Note:**
>
> Near caches on extend clients use map listeners when performing invalidation strategies of `ALL`, `PRESENT`, and `AUTO`. Applications that are write-heavy that use near caches generate many map events.

Setting Proxy Service Thread Pooling Thresholds

To set thread pooling thresholds for a proxy service, add the `<thread-count-max>` and `<thread-count-min>` elements within the `<proxy-scheme>` element. See proxy-scheme in *Developing Applications with Oracle Coherence*. The following example changes the default pool settings.

> **Note:**
>
> - The thread pool is enabled by default and does not require configuration. The default setup allows Coherence to automatically tune the thread count based on the load at any given point in time. Consider explicitly configuring the thread pool only if the automatic tuning proves insufficient.
> - Setting a minimum and maximum thread count of zero forces the proxy service thread to handle all requests; no worker threads are used. Using the proxy service thread to handle client requests is not a best practice.

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <thread-count-max>75</thread-count-max>
   <thread-count-min>10</thread-count-min>
   <autostart>true</autostart>
</proxy-scheme>
```

The `coherence.proxy.threads.max` and `coherence.proxy.threads.min` system properties specify the dynamic thread pooling thresholds instead of using the cache configuration file. For example:

```
-Dcoherence.proxy.threads.max=75
-Dcoherence.proxy.threads.min=10
```

Setting an Exact Number of Threads

In most scenarios, dynamic thread pooling is the best way to ensure that a proxy service always has enough threads to handle requests. In controlled applications where client usage is known, an explicit number of threads can be specified by setting the `<thread-count-min>` and `<thread-count-max>` element to the same value. The following example sets 10 threads for use by a proxy service. Additional threads are not created automatically.

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <thread-count-min>10</thread-count-min>
   <thread-count-max>10</thread-count-max>
   <autostart>true</autostart>
</proxy-scheme>
```

Be Careful When Making InvocationService Calls

You cannot choose the particular node on which invocation code runs when sending the call through a proxy. The `InvocationService` allows a service member 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.

Be Careful When Placing Collection Classes in the Cache

Collection objects (such as an `ArrayList`, `HashSet`, `HashMap`, and so on) are deserialized as immutable arrays when cached by Coherence*Extend clients. A `ClassCastExceptions` is returned if the objects are extracted and cast to their original types.
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 6-1 Casting an ArrayList Object

```
public class ExtendExample
    {
    @SuppressWarnings({ "unchecked" })
    public static void main(String asArgs[])
        {
        System.setProperty("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());
        }
    }
```

Configure POF Serializers for Cache Servers

Proxy servers are responsible for deserializing 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.
Coherence provides the option of configuring a POF serializer for cache servers and has 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 do not require a Java version of the object. Java versions are still 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.
- Key manipulation within the proxy is discouraged. This could interfere with the Object decoration used by the POF serializer causing the extend client to not recognize the key.

Example 6-2 illustrates a fragment from a cache configuration file, which configures the default POF serializer that is defined in the operational deployment descriptor.

Example 6-2 Configuring a POFSerializer for a Distributed Cache

```
...
<distributed-scheme>
   <scheme-name>dist-default</scheme-name>
   <serializer>pof</serializer>
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
...
```

Configuring Firewalls for Extend Clients

Firewalls are often used between extend clients and cluster proxies. When using firewalls, the recommended best practice is to configure the proxy to use a range of ports and then open that range of ports in the firewall. In addition, the cluster port (7574 by default) must be opened

for TCP if the name service is used. Alternatively, a fixed (non-ephemeral, non-range) port can be used. In this legacy configuration, only the specific fixed port needs to be opened in the firewall, and clients need to be configured to connect directly to the proxy's IP and port.

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. After 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 that is delivered with Coherence for Java is used to create extend clients. See Performing Data Grid Operations in *Developing Applications with Oracle Coherence*. For basic Coherence*Extend setup, see Getting Started with Coherence*Extend.

Part III
Creating C++ Extend Clients

Learn how to use the Coherence*Extend C++ object model and API to create C++ clients that access Coherence caches on the cluster.

Coherence for C++ contains the following chapters:

7

Introduction to Coherence C++ Clients

Learn about Coherence for C++ and how to set up Coherence C++ application builds. This chapter includes the following sections:

- Overview of Coherence for C++
- Setting Up C++ Application Builds
  Coherence C++ application builds require updating compiler settings, building header files, library linking, and setting environment variables.

Overview of Coherence for C++

Coherence for C++ allows C++ applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses of Coherence for C++ include desktop and web applications that require access to Coherence caches. See Installing the C++ Client Distribution in *Installing Oracle Coherence*.
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. After 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.

Setting Up C++ Application Builds

Coherence C++ application builds require updating compiler settings, building header files, library linking, and setting environment variables.

This section includes the following topics:

- Setting up the Compiler for Coherence-Based Applications
- Including Coherence Header Files
- Linking the Coherence Library
- Setting the run-time Library and Search Path

- Deploying Coherence for C++

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 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 must 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 run-time 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 operating system environment variables, and use the same names as their Java counterparts. The `coherence.cacheconfig` system property specifies 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	coherence.cacheconfig
Linux	CoherenceCacheConfig
Solaris	CoherenceCacheConfig
Apple (Mac) OS X	CoherenceCacheConfig

> **Note:**
>
> Some operating system 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, "`coherence.cacheconfig`" becomes "`CoherenceCacheConfig`".

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

```
c:\coherence\coherence-cpp\examples> set 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. See the C++ examples included in the Coherence Examples for sample build scripts and configuration. The examples are included as part of the Coherence for Java distribution.

8

Configuration and Usage for C++ Clients

Learn the main steps that are required to use Coherence C++ clients.
This chapter includes the following sections:

General Instructions

You can follow a basic set of steps for creating and using Coherence C++ clients.
The general steps include:

1. Implement the C++ Application
2. Compile and Link the Application
3. Configure Paths
4. Defining Extend Proxy Services
5. Defining Caches for Use By Extend Clients
6. Defining a Remote Cache
7. Building Integration Objects (C++)
8. Starting a Proxy Server
9. Launch the client application.

Implement 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.
The 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).

See Using the Coherence for C++ Client API.

Compile and Link the Application

Review a sample Windows build file that demonstrates how to compiles a C++ application. For the list of platforms on which you can compile applications that employ Coherence for C++, see Supported Environments for Coherence C++ Client in the *Installing Oracle Coherence*.

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.

```
@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
```

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.

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 are different depending on your platform environment. See Introduction to Coherence C++ Clients.

Obtaining a Cache Reference with C++

You can obtained a reference to a configured cache by name using the `coherence::net::CacheFactory` class.
For example:

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

Cleaning up Resources Associated with a Cache

`NamedCache` implementations should be explicitly released by calling the `NamedCache::release()` method when they are no longer needed.
If the particular `NamedCache` is used for the duration of the application, then the resources are 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 applications, link the library to your application and provide a cache configuration file. The location of the cache configuration file can be set by an environment variable or programmatically.
This section includes the following topics:

- Setting the Configuration File Location with an Environment Variable
- Setting the Configuration File Location Programmatically

Setting the Configuration File Location with an Environment Variable

The `coherence.cacheconfig` system property specifies the location of the cache configuration file. See Setting the run-time Library and Search Path .

To set the configuration location on Windows execute:

```
c:\coherence_cpp\examples> set 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).

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. For example:

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

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

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

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

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

The `CacheFactory::configure` method is used 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`). For example:

```
...
// Configure the cache
CacheFactory::configure(CacheFactory::loadXmlFile(String::create(
   "C:\coherence-extend-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 run-time settings used by Oracle Coherence to create, configure and maintain its clustering, communication, and data management services. See Using the Default Operational Override File in *Developing Applications with Oracle Coherence*. For a C++ client, the file specifies or overrides 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-operational-config.xsd` schema and contains an element 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 uses. This allows multiple product editions to be used within the same cluster, with each member specifying the edition that it uses. 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 are 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.
- `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, computer name, role, and so on, to which the member belongs. The `member-identity` is an optional subelement of the `cluster-config` element.

The following example illustrates a sample `tangosol-coherence.xml` file.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-config
   coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <site-name>extend site</site-name>
         <rack-name>rack 1</rack-name>
         <machine-name>computer 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>prod</license-mode>
   </license-config>
</coherence>
```

Configuring a Logger

The Coherence logger is configured using the `logging-config` element in the operational configuration file. See Operational Configuration File (tangosol-coherence-override.xml) . 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 processes from the message queue before discarding all remaining messages in the queue.

The following example illustrates an operational configuration that contains a logging configuration:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-config
   coherence-operational-config.xsd">
   <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>
```

9

Using the Coherence C++ Object Model

Learn how to use the C++ object model on which Coherence for C++ is built.
This chapter includes the following sections:

- Using the Object Model
  Coherence C++ clients use the C++ object model which provides a standard approach to building Coherence applications.
- Writing New Managed Classes
- Diagnostics and Troubleshooting
- Application Launcher - Sanka

Using the Object Model

Coherence C++ clients use the C++ object model which provides a standard approach to building Coherence applications.

This section includes the following topics:

- Coherence Namespaces
- Understanding the Base Object
- Automatically Managed Memory
- Managed Strings
- Type Safe Casting
- Managed Arrays
- Collection Classes
- Managed Exceptions
- Object Immutability
- Integrating Existing Classes into the Object Model

Coherence Namespaces

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

- `coherence::lang`—the essential classes that comprise 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. As a best practice include, 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.

This section includes the following topics:

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 are 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 has these nested-types corresponding to these handles. For instance the managed `coherence::lang::String` class defines `String::Handle`, `String::View`, `String::Holder`.

This section includes the following topics:

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 throws 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() << endl;
```

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 creates a new instance of a string:

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

By comparison, these examples are incorrect and do 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.

This section includes the following topics:

- String Instantiation
- Auto-Boxed Strings

String Instantiation

String objects can easily be constructed from `char*` or `std::string` strings. For example:

```
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.

```
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`. Auto-boxing allows the code shown in the prior samples to be rewritten:

```
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 does 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; // does not compile
```

However, the following example *does* compile, as all Strings are Objects.

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

This section includes the following topics:

- Down Casting

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 (run-time 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`:

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

The `cast<H>` function throws a `coherence::lang::ClassCastException` if the supplied object was not of the expected type. The `instanceof<H>` function tests 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. For example:

```
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:

```
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::HashMap`—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. For example:

```
Map::Handle  hMap = HashMap::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. Managed Exceptions allow caught exceptions to be held 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 sets stack information and then calls 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:

```
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 run-time 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 when 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 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, existing classes must be integrated into the object model. A typical example would be 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 automatically converts a non-managed plain old C++ class instance into a managed class instance at run time.

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

You can write new managed classes, which are classes that extend the `Object` class.

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.

> **Note:**
>
> For Microsoft Visual Studio 2017, Coherence managed classes cannot be declared in an anonymous namespace. There are two helper macros for using pseudo anonymous namespaces: `COH_OPEN_NAMESPACE_ANON(NAME)` and `COH_CLOSE_NAMESPACE_ANON`. Further details on the macros are provided in `compatibility.hpp`.

This section includes the following topics:

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 the specs above. 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 for `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.

The following example illustrates using `interface_spec` to define a `Comparable` interface:

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

The following example illustrates using `interface_spec` to define a derived interface `Number`:

```
class Number
    : public interface_spec<Number,
        implements<Comparable> >
    {
    public:
        virtual int32_t getValue() const = 0;
    };
```

The following example uses `cloneable_spec` to produce an implementation.

> **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.

```
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 can also be defined without the use of specs. For example:

```
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;
};
```

The following example illustrates using the spec'd 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

Equality, Hashing, Cloning, Immutability, and Serialization all identify the state of an object and generally have similar implementation concerns. Simply put, all data members referenced in one of these methods, are likely 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. 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. If `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 `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`.

For clone, only the key and value (not all the data members) require cloning. To clone 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 a 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.

The following example illustrates how to create a `Runnable` instance and spawn a thread:

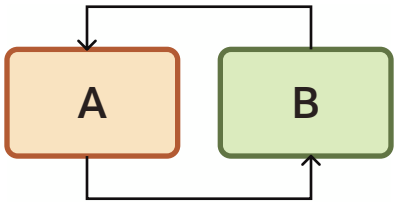
```
class HelloRunner
      : public class_spec<HelloRunner,
          extends<Object>,
          implements<Runnable> >
     {
     friend class factory<HelloRunner>;

     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 9-1.

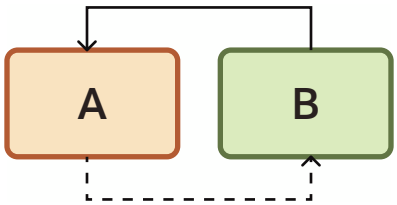
Figure 9-1 A Bi-Directional Relationship



In this picture, both A and B have a reference count of one, which keeps them active. What they do not realize is that they are the only things keeping each other active, and that no external references to them exist. Reference counting alone is 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 references an object, but not prevent it from being deleted. As illustrated in Figure 9-2, the A->B->A issue could be resolved by changing it to the following.

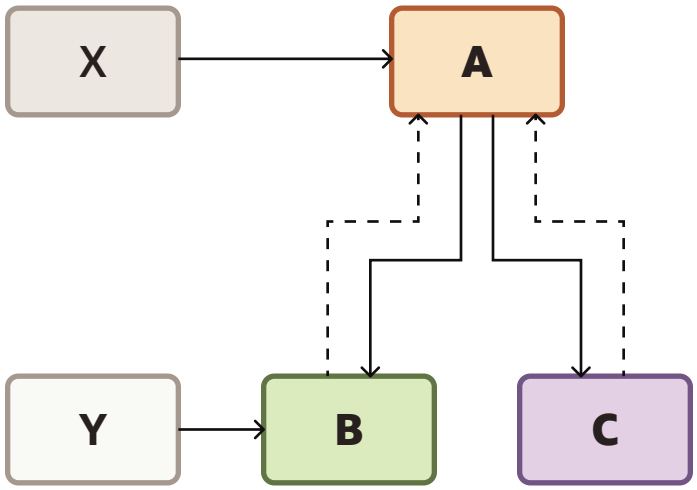
Figure 9-2 Establishing a Weak Reference



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.

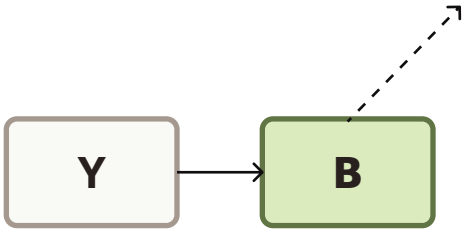
Weak references allow for construction of more complicated structures then this. But it becomes necessary to adopt a convention for which references are weak and which are strong. Consider a tree illustrated in Figure 9-3. The tree consists of nodes A, B, C; and two external references to the tree X, and Y.

Figure 9-3 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 9-4.

Figure 9-4 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 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 usually causes the object to be destructed before it ever finishes being created. Instead, 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 do not appear in normal application code.

Table 9-1 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`

Table 9-1 (Cont.) Advanced Handle Types Supported by Coherence for C++

Type	Thread-safe?	View	Notes
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 some facilities to aid in writing thread-safe code.

This section includes the following topics:

- Synchronization and Notification
- Thread Safe Handles
- Escape Analysis
- Thread-Local Allocator

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. For example:

```
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: thread-safety of the handle is being discussed not the object referenced by the handle. It is safe to have multiple distinct handles that 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 is not 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 are continually accessed within a code block. This assignment to a standard stack based handle is thread-safe, and, after completed, allows for essentially free dereferencing of the stack based handle. Note that when initializing thread-safe handles a reference to a guardian `Object` must be supplied as the first parameter, this reference can be obtained by calling `self()` on the enclosing object.

The following example demonstrates a 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 to use another `Object` as the guardian. However, it is crucial to ensure that the guardian `Object` outlives the guarded thread-safe handle. When using another object as the guardian, obtain a random immortal guardian from `coherence::lang::System` through a call to `System::common()`. For example:

```
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 through a call to `self()` then to `System::common()`.

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

Thread-safe handles can also be used in non-class shared data as well. For example, 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 a provided thread-safe handle. Passing it by an external mechanism does not allow escape analysis to identify the "escape" which could cause memory corruption or other run-time errors.

This section includes the following topics:

- Shared handles
- Const Correctness

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 causes 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.

Const Correctness

Coherence escape analysis, among other things, leverages the computed mutability of an object to determine if state changes on data members are still possible. Namely, when an object is only referenced from views, it is assumed that its data members do not undergo further updates. The C++ language provides some 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 and supported approach for the object model is the mutable keyword. For 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 informs the escape analysis routine to not consider the data member as "const" when the enclosing object is only referenced using 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 grows 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:

- `coherence.heap.slot.size` controls the maximum size of an object which is considered for allocation from the pool, the default is 128 bytes. Larger objects call through to the system's `malloc` routine to obtain the required memory.
- `coherence.heap.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 fall back on `malloc`.
- `coherence.heap.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 is eligible for refilling the pool.

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

Diagnostics and Troubleshooting

Learn how to diagnosing issues in applications that use the Coherence C++ object model. This section includes the following topics:

Thread-Local Allocator Logs

Logs can be enabled to view the efficiency of the thread-local allocator pool. To enable the logs, set the `coherence.heap.logging` system property to `true`.

The log entries indicate the memory location of the pool, the size of the pool, how many allocation areas are in the pool and the fraction of successful hits on the pool (the rate of finding a slot within the pool). The following example demonstrates a typical allocator log entry:

```
(thread=main): Allocator hit: pool=0x7f8e5ac039d0, size=128, slots=512, hit rate=0.62963
```

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). The following output illustrates a 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 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. After 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 `coherence.heap.analyzer system` property. The property can be set to the following values:

- `none`—No heap analysis is performed. This is the default.
- `object`—The `coherence::lang::ObjectCountHeapAnalyzer` is 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` is 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).

The following output 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`.

```
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 is typically used along side non-managed code which could cause corruption. Therefore, the object model includes memory corruption detection support. When enabled, the object model's memory allocator pads 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 affects a pad, subsequent validations 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 increases the chances of corruption affecting a pad, and thus the chance of detecting corruption.

The size of the pad can be configured by using the `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 nonzero value enables the feature, and is available even in release builds.

The following output illustrates the results from an instance of memory corruption detection:

```
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 uses an application launcher for invoking executable classes embedded within a shared library. The launcher allows for a shared library to contain utility or test executables without shipping individual standalone executable binaries.
This section includes the following topics:

- Command line syntax
- Built-in Executables
- Sample Custom Executable Class

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. For example,
                              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

Several 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 output the registration which exists 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 to the command-line arguments which followed the executable class name.

When 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
```

10

Using the Coherence for C++ Client API

The Coherence for C++ API allows C++ applications to use Coherence clustered services from outside the Coherence cluster.
Coherence for C++ API documentation is available at *C++ API Reference for Oracle Coherence* and in the `doc` directory of the Coherence for C++ distribution.

This chapter includes the following sections:

- CacheFactory
- NamedCache
- QueryMap
- ObservableMap
- InvocableMap
- Filter
- Value Extractors
- Entry Processors
  An entry processor is an agent that operates against the entry objects within a cache.
- Entry Aggregators

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.

This section includes the following topics:

NamedCache

A `NamedCache` is a map of resources shared among members of a cluster. The `NamedCache` provides 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. See Filter .

- `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 traverses 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`. The index correlates 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 Querying a Cache (C++) and Performing Simple Queries.

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 Signing Up for all Events.

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. See Entry Processors and Entry Aggregators.

- `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

The `Filter` API 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 QueryMap.

- `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. The following example creates an `EqualsFilter` to test that an object equals 5:

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

- `GreaterEqualsFilter` is used to test a "Greater or Equals" condition. The following example creates a `GreaterEqualsFilter` that tests that an objects value is `>= 55`:

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

- `LikeFilter` is used for pattern matching. The followiung example creates a `LikeFilter` that tests that the string representation of an object begins with "`Belg`":

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

- `Some filters combine two filters to create a compound condition. AndFilter` is used to combine two filters to create an "`AND`" condition. The following example creates an `AndFilter` that tests that an objects value is greater than 10 and less than 20:

```
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. The following example create an `OrFilter` that tests that an object's value is less than `10` or greater than `20`:

```
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 Understanding Query Concepts.

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 C++ examples that are part of the Coherence Java distribution.

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 these custom types is performed by the Java implementation and not the C++ implementation. See Building Integration Objects (C++).

11

Building Integration Objects (C++)

You can use Portable Object Format (POF) serialization when creating C++ clients.

> **Note:**
>
> This document assumes familiarity with the Coherence C++ Object Model, including advanced concepts such as specification-based class definitions. See Using the Coherence C++ Object Model.

This chapter includes the following sections:

Overview of Building Integration Objects (C++)

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. See The PIF-POF Binary Format in *Developing Applications with Oracle Coherence*.
While the Coherence C++ API includes several POF serializable classes, custom data types require serialization support as described in this chapter.

POF Intrinsics

POF supports many internal types that do not require special handing.

POF intrinsic types include:

- 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.
This section includes the following topics:

Overview of Serialization Options

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) .
- Modify the data object to extend `Object`, and implement the `PortableObject` interface, to allow for object to self-serialize. See PortableObject (Self-Serialization) .
- Modify the data object to extend `Object`, and produce a `PofSerializer` class to perform external serialization. See PofSerializer (External Serialization) .

Table 11-1lists some 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:

- Serialization routines that allow the data items to be encoded to POF must be implemented.
- The data object's fields are identified by using numeric indexes.
- 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

```
#include <iostream>
#include <string>
using namespace std;

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. However, they are not required as 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 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, the above `Address` example can be rewritten as a managed class, and implement the `PortableObject` interface, which fully embraces the Coherence object model as part of the definition of the class. For example, 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;
using coherence::io::pof::PortableObject;

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_vsState(self(),
       that.m_vsState), m_nZip(that.m_nZip) {}

    Address() // required by PortableObject
       : m_vsCity(self()),
         m_vsState(self()),
         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;
     int32_t     m_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 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, an `AddressSerializer` is written 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. Typically, it is not necessary to use this approach, as either the `Managed<T>` or `PortableObject` approaches 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.getCity()), m_vsState(self(),
         that.getState()), m_nZip(that.getZip()) {}

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

#include "Address.hpp"

using namespace coherence::lang;

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

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));
```

Using POF Object References

POF supports the use of object identities and references for objects that occur more than once in a POF stream. Objects are labeled with an identity and subsequent instances of a labeled object within the same POF stream are referenced by its identity.
Using references avoids encoding the same object multiple times and helps reduce the data size. References are typically used when a large number of sizeable objects are created multiple times or when objects use nested or circular data structures. However, for applications that contain large amounts of data but only few repeats, the use of object references provides minimal benefits due to the overhead incurred in keeping track of object identities and references.

The use of object identity and references has the following limitations:

- Object references are only supported for user defined object types.
- Object references are not supported for `Evolvable` objects.
- Object references are not supported for keys.
- Objects that have been written out with a POF context that does not support references cannot be read by a POF context that supports references. The opposite is also true.
- POF objects that use object identity and references cannot be queried using POF extractors. Instead, use the `ValueExtractor` API to query object values or disable object references.
- The use of the `PofNavigator` and `PofValue` API has the following restrictions when using object references:
  - Only read operations are allowed. Write operations result in an `UnsupportedOperationException`.

- User objects can be accessed in non-uniform collections but not in uniform collections.
- For read operations, if an object appears in the data stream multiple times, then the object must be read where it first appears before it can be read in the subsequent part of the data. Otherwise, an `IOException: missing identity: <ID>` may be thrown. For example, if there are 3 lists that all contain the same person object, `p`. The `p` object must be read in the first list before it can be read in the second or third list.

This section includes the following topics:

Enabling POF Object References

Object references are not enabled by default and must be enabled using `setReferenceEnabled` when creating a POF context. For example:

```
SystemPofContext::Handle hCtx = SystemPofContext::getInstance();
hCtx->setReferenceEnabled(true);
```

Registering POF Object Identities for Circular and Nested Objects

Circular or nested objects must manually register an identity when creating the object. Otherwise, a child that references the parent will not find the identity of the parent in the reference map. Object identities can be registered from a serializer during the deserialization routine using the `PofReader.registerIdentity` method.

The following examples demonstrate two objects (`Customer` and `Product`) that contain a circular reference and a serializer implementation that registers an identity on the `Customer` object.

The `Customer` object is defined as follows:

```
class Customer
   : public class_spec<Customer,
      extends<Object> >
   {
   friend class factory<Customer>;

   protected:
      Customer()
         : m_vsName(self(), String::null_string),
         m_vProduct(self(), NULL)
         {
         }

         Customer(String::View vsName)
            : m_vsName(self(), vsName),
            m_vProduct(self(), NULL)
            {
         }

         Customer(String::View vsName, Product::View vProduct)
         : m_vsName(self(), vsName),
         m_vProduct(self(), vProduct)
         {
         }

   public:
```

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

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

      Product::View getProduct() const
         {
         return m_vProduct;
         }

      void setProduct(Product::View vProduct)
         {
         m_vProduct = vProduct;
         }

   private:
      MemberView<String> m_vsName;
      MemberView<Product> m_vProduct;
   };
```

The `Product` object is defined as follows:

```
class Product
   : public class_spec<Product,
      extends<Object> >
   {
   friend class factory<Product>;

   protected:
      Product()
         : m_vCustomer(self(), NULL)
         {
         }

      Product(Customer::View vCustomer)
         : m_vCustomer(self(), vCustomer)
         {
         }

   public:
      Customer::View getCustomer() const
         {
         return m_vCustomer;
         }

      void setCustomer(Customer::View vCustomer)
         {
         m_vCustomer= vCustomer;
         }

   private:
         MemberView<Customer> m_vCustomer;
   };
```

The serializer implementation registers an identity during deserialization and is defined as follows:

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

   public:
      void serialize(PofWriter::Handle hOut, Object::View v) const
         {
         Customer::View vCustomer = cast<Customer::View>(v);
         hOut->writeString(0, vCustomer->getName());
         hOut->writeObject(1, vCustomer->getProduct());
         hOut->writeRemainder(NULL);
         }

      Object::Holder deserialize(PofReader::Handle hIn) const
         {
         String::View vsName = cast<String::View>(hIn->readString(0));
         Customer::Holder ohCustomer = Customer::create(vsName);

         hIn->registerIdentity(ohCustomer);
         ohCustomer->setProduct(cast<Product::View>(hIn->readObject(1)));
         hIn->readRemainder();
         return ohCustomer;
         }
   };
```

Registering Custom C++ Types

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.

Implementing a Java Version of a C++ Object

A C++ object must have a parallel Java implementation on the cluster if direct access to the deserialized object is required.

The use of POF allows key and value objects to be stored within the cluster without the need for parallel Java implementations. This is ideal for performing basic get and put based operations. In addition, the `PofExtractor` and `PofUpdater` APIs add flexibility in working with non-primitive types in Coherence. For many extend client cases, a corresponding Java classes in the grid is not required. Because POF extractors and POF updaters can navigate the binary, the entire key and value does not have to be deserialized into object form. This implies that indexing can be achieved by simply using POF extractors to pull a value to index on.

When to Include a Parallel Java Implementation

A parallel Java implementation is required whenever the Java-based cache servers must directly interact with a data object rather then simply holding onto a serialized representation of it. For example, a Java class is still required when using a cache store. In this case, the deserialized version of the key and value is passed to the cache store to write to the back end. In addition, queries, filters, entry processors, and aggregators require a Java implementation if direct access to the object is desired.

If a Java implementation is required, then the implementation must be located on the cache servers. The approach to making the Java version serializable over POF is similar to the above examples, see `PortableObject` and `PofSerializer` for details. These APIs are compatible with all three of the C++ approaches.

Deferring the Key Association Check

Key classes do not require a cluster-side Java implementation even if the key class specifies data affinity using `KeyAssociation`. Key classes are checked on the client side and a decorated binary is created and used by the cluster. However, existing client implementations that do rely on a Java key class for key association must set the `defer-key-association-check` parameter in order to force the use of the Java key class. Existing client applications that use key association but want to leverage client-side key binaries, must port the `getAssociatedKey()` implementation from the existing Java class to the corresponding client class.

To force key association processing to be done on the cluster side instead of by the extend client, set the `<defer-key-association-check>` element, within a `<remote-cache-scheme>` element, in the client-side cache configuration to `true`. For example:

```
<remote-cache-scheme>
   ...
   <defer-key-association-check>true</defer-key-association-check>
</remote-cache-scheme>
```

> **Note:**
>
> If the parameter is set to `true`, a Java key class implementation must be found on the cluster even if key association is no being used.

Understanding Serialization Performance

Both `Managed<T>` and `PortableObject` use `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. For `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.

Using POF Annotations to Serialize Objects

POF annotations provide an automated way to implement the serialization and deserialization routines for an object. POF annotations are serialized and deserialized using the `PofAnnotationSerializer` class which is an implementation of the `PofSerializer` interface. Annotations offer an alternative to using the `Managed<T>` adapter, `PortableObject` interface, and `PofSerializer` interface and reduce the amount of time and code that is required to make objects serializable.

This section includes the following topics:

- Annotating Objects for POF Serialization
- Registering POF Annotated Objects
- Enabling Automatic Indexing
- Providing a Custom Codec

Annotating Objects for POF Serialization

Two annotations are available to indicate that a class and its methods are POF serializable:

- `Portable` – Marks the class as POF serializable. The annotation is only permitted at the class level and has no members.
- `PortableProperty` – Marks a method accessor as a POF serialized property. Annotated methods must conform to accessor notation (`get`, `set`, `is`). Members can be used to specify POF indexes as well as custom codecs that are executed before or after serialization or deserialization. Index values may be omitted and automatically assigned. If a custom codec is not entered, the default codec is used.

The following example demonstrates annotating a class and method and also explicitly assigns property index values. Note that the class must be registered with the system class loader `COH_REGISTER_CLASS`.

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

   Public:
      String::View getFirstName() const
         {
         return m_vsFirstName;
         }

      void setFirstName(String::View vsFirstName)
         {
         m_vsFirstName = vsFirstName;
         }

   private: String m_firstName;
      MemberView<String> m_vsFirstName;
      MemberView<String> m_vsLastName;
      int32_t            m_nAge;
```

```
   public:
      static const int32_t FIRST_NAME = 0;
      static const int32_t LAST_NAME  = 1;
      static const int32_t AGE        = 2;
   };

COH_REGISTER_CLASS(TypedClass<Person>::create()
   ->annotate(Portable::create())
   ->declare(COH_PROPERTY(Person, FirstName, String::View)
      ->annotate(PortableProperty::create(Person::FIRST_NAME)))
   ->declare(COH_PROPERTY(Person, LastName, String::View)
      ->annotate(PortableProperty::create(Person::LAST_NAME)))
   ->declare(COH_PROPERTY(Person, Age, BoxHandle<const Integer32>)
      ->annotate(PortableProperty::create(Person::AGE)))
      );
```

Registering POF Annotated Objects

POF annotated objects must be registered as a user type using the `COH_REGISTER_POF_ANNOTATED_CLASS` macro. The following example registers a user type for an annotated `Person` object:

```
COH_REGISTER_POF_ANNOTATED_CLASS(1001, Person);
```

Enabling Automatic Indexing

POF annotations support automatic indexing which alleviates the need to explicitly assign and manage index values. The index value can be omitted whenever defining the `PortableProperty` annotation. Any property that does assign an explicit index value is not assigned an automatic index value. The automatic index algorithm can be described as follows:

Name	Explicit Index	Determined Index
c	1	1
a	omitted	0
b	omitted	2

> **Note:**
>
> Automatic indexing does not currently support evolvable classes.

To enable automatic indexing, use the `COH_REGISTER_POF_ANNOTATED_CLASS_AI` pre-processor macro when registering the user type. The following example registers a user type for an annotated `Person` object that uses automatic indexing:

```
COH_REGISTER_POF_ANNOTATED_CLASS_AI(1001, Person);
```

Providing a Custom Codec

Codecs allow code to be executed before or after serialization or deserialization. The codec defines how to encode and decode a portable property using the `PofWriter` and `PofReader` interfaces. Codecs are typically used for concrete implementations that could get lost when

being deserialized or to explicitly call a specific method on the `PofWriter` interface before serializing an object.

To create a codec, create a class that implements the `Codec` interface. The following example demonstrates a codec that defines the concrete implementation of a linked list type:

```
class LinkedListCodec
   : public class_spec<LinkedListCodec,
      extends<Object>,
      implements<Codec> >
   {
   friend class factory<LinkedListCodec>;

   public:
      void encode(PofWriter::Handle hOut, int32_t nIndex, Object::View ovValue)
         const
         {
            hOut->writeCollection(nIndex, cast<Collection::View>(ovValue));
         }

      Object::Holder decode(PofReader::Handle hIn, int32_t nIndex) const
         {
            LinkedList::Handle hLinkeList = LinkedList::create();
            return hIn->readCollection(nIndex, hLinkeList);
         }
   };
COH_REGISTER_TYPED_CLASS(LinkedListCodec);
```

To assign a codec to a property, enter the codec as a member of the `PortableProperty` annotation. If a codec is not specified, a default codec (`DefaultCodec`) is used. The following example demonstrates assigning the above `LinkedListCodec` codec:

```
COH_REGISTER_CLASS(TypedClass<Person>::create()
   ->annotate(Portable::create())
   ->declare(COH_PROPERTY(Person, FirstName, String::View)
      ->annotate(PortableProperty::create(Person::FIRST_NAME)))
   ->declare(COH_PROPERTY(Person, LastName, String::View)
      ->annotate(PortableProperty::create(Person::LAST_NAME)))
   ->declare(COH_PROPERTY(Person, Age, BoxHandle<const Integer32>)
      ->annotate(PortableProperty::create(Person::ALIASES,
SystemClassLoader::getInstance()->loadByType(typeid(LinkedListCodec)) )))
);
```

12

Querying a Cache (C++)

You can query Coherence caches from C++ clients.
This chapter includes the following sections:

Overview of Query Functionality

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. 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 do 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; For 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).

Performing Simple Queries

You can use a value extractor and filter to query a cache.
The following example uses an a value extractor and filter to query a cache.

```
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:

```
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:

```
// 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 provides 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) exists. It is expected that an application calls this method to suggest an index even if the index exists, just so that the application is certain that index has been suggested. For example, in a distributed environment each server likely suggests 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 requested the same indexes.

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

The Partitioned Cache implements the QueryMap interface using the Parallel Query feature and results in high performance queries even for large data sets.

Querying Near Caches

Although queries can be executed through a near cache, the query does 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);
```

Understanding Query Concepts

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 only need 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 function-based 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:

```
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:

```
// 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:

```
// 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();
        }
    }
```

Performing Queries Involving Multi-Value Attributes

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

Using a Chained Extractor in a Query

The `ChainedExtractor` implementation allows chained invocation of zero-argument (accessor) methods.
The following example extractor first uses 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.

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

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

POF extractors and POF updaters offer the same functionality as `ChainedExtractors` through the use of the `SimplePofPath` class. See Using POF Extractors and POF Updaters in *Developing Applications with Oracle Coherence*.

Using a Query Recorder

The `QueryRecorder` class produces an explain or trace record for a given filter. The class is an implementation of a parallel aggregator that is capable querying all nodes in a cluster and aggregating the results. The class supports two record types: an `QueryRecorder::explain` record that provides the estimated cost of evaluating a filter as part of a query operation and a `QueryRecorder::trace` record that provides the actual cost of evaluating a filter as part of a query operation. Both query records take into account whether or not an index can be used by a filter. See Interpreting Query Records in *Developing Applications with Oracle Coherence*.
To create a query record, create a new `QueryRecorder` instance that specifies a `RecordType` parameter. Include the instance and the filter to be tested as parameters of the `Aggregate` method. The following example creates an explain record:

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

IdentityExtractor::View hExtract = IdentityExtractor::getInstance();
OrFilter::Handle hFilter = OrFilter::create(
   GreaterEqualsFilter::create(hExtract, Integer32::create(50)),
   LessEqualsFilter::create(hExtract, Integer32::create(20)));

QueryRecord::View vRecord = cast<QueryRecord::View>(hCache->aggregate(
   (Filter::View) hFilter, QueryRecorder::create(QueryRecorder::explain)));

cout << vRecord;
```

To create a trace record, change the `RecordType` parameter to `trace`:

```
QueryRecord::View vRecord = cast<QueryRecord::View>(hCache->aggregate(
   (Filter::View) hFilter, QueryRecorder::create(QueryRecorder::trace)));
```

13

Performing Continuous Queries (C++)

You can use Continuous Query Caching in a C++ client to ensure that a query always retrieves the latest results from a cache in real-time.
This chapter includes the following sections:

Overview of Performing Continuous Queries (C++)

Queries provide the ability to obtain a point in time query result from a Coherence cache and it is possible to receive events that would change the result of that query. However, the continuous query feature 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. Materialized views enable 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, to monitor, in real time, all sales orders for several customers. You can create a continuous query cache and set up an event listener that listens for any events pertaining to the customers. Coherence queries for all of the data objects on the grid that pertain to a particular customer and copies them to a local cache. The event listener on the query listens 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.

Understanding the Use Cases for Continuous Query Caching

Continuous Query Caching is ideal for many use cases, such as event processing and instant access to up-to-date query results.
Consider using Continuous Query Caching in the following situations:

- A Continuous Query Cache is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.

- A Continuous Query Cache 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.

Understanding the Continuous Query Caching Implementation

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.
- 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

Continuous query caching requires an underlying cache and a query filter.
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 is created automatically and its configuration is 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. The following example illustrates how you can use code filters to find a given trader with a given order status:

```
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 a method 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 Continuous Query Cache Resources

A Continuous Query Cache places one or more event listeners on its underlying cache. If a Continuous Query Cache is used for the duration of the application, then the resources is cleaned up when the node is shut down or otherwise stops. If a Continuous Query Cache is only used for a period of time, then the application must call the `release` method.

Caching Only Keys Versus 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 is automatically 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 a Continuous Query Cache

A client can place one or more event listeners onto a Continuous Query Cache.
For example:

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

If your application has to perform some processing against every item that is in the cache **and** every item added to the cache, then provide the listener during construction. The resulting cache receives 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:

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

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

This section includes the following topics:

- Avoiding Unexpected Results
- Achieving a Stable Materialized View

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 are missed. For example:

```
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. Although, the same entry may appear in both an event and in the `Iterator`. The events can be asynchronous, so the sequence of operations cannot be guaranteed.

```
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.

Making a Continuous Query Cache Read-Only

A 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 does not allow objects to be added to, changed in, removed from, or locked in the cache.

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

14

Performing Remote Invocations (C++)

You can perform remote invocations on Coherence caches from C++ clients.
This chapter includes the following sections:

- Overview of Performing Remote Invocations (C++)
- Configuring and Using the Remote Invocation Service
- Registering Invocable Implementation Classes

Overview of Performing Remote Invocations (C++)

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.
The following example illustrates a 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>7077</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:

```
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 and cluster-side node to which the client is connected. See PortableObject (Self-Serialization) . 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 Registering Custom C++ Types.

15

Using Cache Events (C++)

You can use map event listeners to receive cache events and events from any class in Coherence that implements the `ObservableMap` interface.
This chapter includes the following sections:

Overview of Map Events (C++)

The event model is comprised of 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.
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 an `addMapListener()` method.

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.

Caches and Classes that Support Events

All Coherence caches implement the `ObservableMap` interface, which allows an application to receive cache events. Any cache can receive events, 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 are 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, pass an object that implements the `MapListener` interface to an `addMapListener` method on `ObservableMap`.
For example:

```
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:

```
#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 simplifies printing all events from any given cache (since all caches implement the `ObservableMap` interface):

```
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:

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

Later, to remove the 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.

Using a Multiplexing Map Listener

The `MultiplexingMapListener` class routes all events to a single method for handling. The following example illustrates a simple `EventPrinter` class:

```
#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

You can register a listener on a cache using the `<listener>` element in the cache configuration. If configured, then Coherence automatically adds the listener when it configures the cache. Registering a listener in the configuration is useful when a listener should always be on a particular cache.

Signing Up for Events on Specific Identities

You can sign up for events that occur against specific identities (keys). The following code in prints all events that occur against the `Integer` key `5`:

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

The following code only triggers an event when the `Integer` key `5` is inserted or updated:

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

Filtering Events

You can use a filter to listen for specific events. In the following example, a listener is added to the cache with a filter that allows the listener to only receive delete 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:

```
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}
```

See Listening to Queries .

Filtering Events Versus Filtering Cached Data

When building a `Filter` for querying, the object that is passed to the evaluate method of the `Filter` is 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 is passed to the `evaluate` method of the `Filter` is always of type `MapEvent`.

Using Lite Events

You can save resources by using lite events if an application does not require the old and the new value to be included in the event. By default, Coherence provides both the old and the new value as part of an event. Consider the following example:

```
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 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:

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

> **Note:**
>
> 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 might 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 require knowledge of the old and new values for the event.

Listening to Queries

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 are 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:**
>
> The following example uses the `coherence::util::extractor::ReflectionExtractor` class. While the C++ client does not support reflection, `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.

```
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:

```
// 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 is passed to the `evaluate` method of the Filter is 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 is passed to the `evaluate` method of the `Filter` is 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:

```
// 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);
```

Using Synthetic Events

An application can listen for synthetic events, which originate from operations within a cache. Synthetic events are different than client 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; while, fifty threads on each server in the cluster is accessing data from the same cache. All the modifying actions produce events that any server within the cluster can choose to receive. These actions are referred to 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:

```
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`.

Using Backing Map Events

For some advanced use cases, you can listen to events on the map behind a service. Replication, partitioning and other approaches to managing data in a distributed environment are all distribution *services*. The data structure that actually manages the data for a service 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 is 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`.)

Using 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. A `MapListener` implementation can use the `SynchronousListener` marker interface to guarantee that the cache API operations and the events are ordered as if the local view of the clustered system were single-threaded.

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

16

Performing Transactions (C++)

You can use 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 *Developing Applications with Oracle Coherence*.

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 use 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 run time, 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 use 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 require any transaction logic and is a skeleton of the transactional entry processor. See Building Integration Objects (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"?>

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <include>txn-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 *Developing Applications with Oracle Coherence*.

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 `7077`. See Configuring Extend Proxies .

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <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>
         <service-name>TransactionalCache</service-name>
         <thread-count-min>2</thread-count-min>
         <thread-count-max>10</thread-count-max>
         <high-units>15M</high-units>
         <task-timeout>0</task-timeout>
         <autostart>true</autostart>
      </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>
         <autostart>true</autostart>
      </proxy-scheme>
   </caching-schemes>
</cache-config>
```

Configuring the Client-Side Remote Cache

Remote clients require a remote cache 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 Configuring Extend Proxies .

The following example configures a remote cache to connect to a proxy that is located on `localhost` at port `7077`. 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.

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <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>7077</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>
```

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 run time, 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 transaction execution: " << oResult << std::endl;
```

Part IV
Creating .NET Extend Clients

Learn how to use the Coherence*Extend .NET client library to create .NET clients that access Coherence caches on the cluster.

Coherence for .NET contains the following chapters:

- Introduction to Coherence .NET Clients
- Building Integration Objects (.NET)
- Using the Coherence .NET Client Library
- Performing Continuous Queries (.NET)
- Performing Remote Invocations (.NET)
- Performing Transactions (.NET)
- Managing ASP.NET Session State

17

Introduction to Coherence .NET Clients

Learn about Coherence for .NET and how to set up Coherence .NET applications.
This chapter includes the following sections:

- Overview of Coherence for .NET
- Configuration and Usage for .NET Clients

Overview of Coherence for .NET

Coherence for .NET allows .NET applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses of Coherence for .NET include desktop and web applications that require access to Coherence caches. See Installing the .NET Client Distribution in *Installing Oracle Coherence*. 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. After 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.

Configuration and Usage for .NET Clients

Learn the main steps that are required to use Coherence .NET clients.
This section includes instructions for setting up .NET applications to use Coherence. This section includes the following topics:

- General Instructions
- Configuring Coherence*Extend for .NET
- Obtaining a Cache Reference with .NET
- Cleaning Up Resources Associated with a Cache

General Instructions

You can follow a basic set of steps for creating and using Coherence .NET clients. The general steps include:

1. Configuring Coherence*Extend for .NET
2. Building Integration Objects (.NET)
3. Using the Coherence .NET APIs
4. Starting a Proxy Server

5. Launching the .NET client application

Configuring Coherence*Extend for .NET

Coherence for .NET clients use a specific XML schema for the Coherence cache configuration file. Make sure the cache configuration file uses the following schema:

```
<cache-config xmlns="http://schemas.tangosol.com/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.tangosol.com/cache
  assembly://Coherence/Tangosol.Config/cache-config.xsd">
  ...
```

For general instructions on setting up and configuring Coherence*Extend, refer to:

- Defining Extend Proxy Services
- Defining Caches for Use By Extend Clients
- Defining a Remote Cache

Obtaining a Cache Reference with .NET

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

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

Cleaning Up Resources Associated with a Cache

`INamedCache` instances, including `LocalCache`, should be explicitly released by calling the `INamedCache.Release` method when they are no longer needed. If the particular `INamedCache` is used for the duration of the application, then the resources are cleaned up when the application is shut down or otherwise stops. However, if the instance is only used for a period of time, then 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`. If you want to obtain and release a cache instance within a single method, you can do so with a `using` block:

```
using (INamedCache cache = CacheFactory.GetCache("my-cache"))
{
   // use cache as usual
}
```

After the `using` block terminates, `IDisposable.Dispose` is called on the `INamedCache` instance, and all resources associated with it are released.

18
Building Integration Objects (.NET)

You can use Portable Object Format (POF) serialization when creating .NET clients. This chapter includes the following sections:

Overview of Building Integration Objects (.NET)

Coherence caches are used to cache value objects. .NET clients require a platform-independent serialization format that allows both .NET clients and Coherence JVMs to properly serialize and deserialize value objects that are 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. See The PIF-POF Binary Format in *Developing Applications with Oracle Coherence*.
POF supports all common .NET types out-of-the-box. Custom .NET classes can also be serialized to a POF stream by completing the following steps:

1. Create a .NET class that implements the `IPortableObject` interface. See Creating an IPortableObject Implementation.
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.

After 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 can retrieve, manipulate, and store these types from a Coherence or Coherence*Extend JVM using the matching Java classes.

Creating an IPortableObject Implementation

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 requirement to implement the `IPortableObject` interface, is that it must have a default constructor which allows the POF deserializer to create an instance of the class during deserialization.
Example 18-1 illustrates a user-defined portable class:

Example 18-1 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
}
```

Implementing a Java Version of a .NET Object

A .NET object must have a parallel Java implementation on the cluster if direct access to the deserialized object is required.
The use of POF allows key and value objects to be stored within the cluster without the need for parallel Java implementations. This is ideal for performing basic get and put based operations. In addition, the `PofExtractor` and `PofUpdater` APIs add flexibility in working with non-primitive types in Coherence. For many extend client cases, a corresponding Java classes

in the grid is not required. Because POF extractors and POF updaters can navigate the binary, the entire key and value does not have to be deserialized into object form. This implies that indexing can be achieved by simply using POF extractors to pull a value to index on.

When to Include a Parallel Java Implementation

A parallel Java implementation is required whenever the Java-based cache servers must directly interact with a data object rather then simply holding onto a serialized representation of it. For example, a Java class is still required when using a cache store. In this case, the deserialized version of the key and value is passed to the cache store to write to the back end. In addition, queries, filters, entry processors, and aggregators require a Java implementation if direct access to the object is desired.

If a Java implementation is required, then the implementation must be located on the cache servers. The approach to making the Java version serializable over POF is demonstrated in Creating a PortableObject Implementation (Java).

Deferring the Key Association Check

Key classes do not require a cluster-side Java implementation even if the key class specifies data affinity using the `IKeyAssociation` interface. Key classes are checked on the client side and a decorated binary is created and used by the cluster. However, existing client implementations that do rely on a Java key class for key association must set the `defer-key-association-check` parameter in order to force the use of the Java key class. Existing client applications that use key association but want to leverage client-side key binaries, must port the `getAssociatedKey()` implementation from the existing Java class to the corresponding client class (see I`KeyAssociation.AssociatedKey`.

To force key association processing to be done on the cluster side instead of by the extend client, set the `<defer-key-association-check>` element, within a `<remote-cache-scheme>` element, in the client-side cache configuration to `true`. For example:

```
<remote-cache-scheme>
   ...
   <defer-key-association-check>true</defer-key-association-check>
</remote-cache-scheme>
```

> **Note:**
>
> If the parameter is set to `true`, a Java key class implementation must be found on the cluster even if key association is no being used.

This section includes the following topic:

- Creating a PortableObject Implementation (Java)

Creating a PortableObject Implementation (Java)

An implementation of the portable class in Java is very similar to the one in .NET. Example 18-2 illustrates the Java version of the .NET class in Example 18-1.

Example 18-2 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 the opposite is true as well). The POF XML configuration file maps user types to a type identifier. See POF User Type Configuration Elements in *Developing Applications with Oracle Coherence*.
The following example demonstrates a POF 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.

After 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. The following examples assumes that the user type mappings are saved in the `my-dotnet-pof-config.xml` 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 looks 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 tries to located the file by the contents of the `pof-config` element in the Coherence for .NET application configuration file. For example:

```
<?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 file 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.
The following illustrates a sample cluster-side POF configuration file called `my-java-pof-config.xml`:

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

```
<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <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>
```

After your custom types have been added, you must configure the server to use your POF configuration when serializing objects:

```
<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

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, or semantics, or both 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 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 is 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 18-3 demonstrates how the `ContactInfo` .NET type can be modified to support class evolution:

Example 18-3 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 18-4 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. You can externalize the portable serialization of a user type by creating an `IPofSerializer` implementation.
Example 18-5 illustrates, an implementation of the `IPofSerializer` interface for the `ContactInfo` type.

Example 18-5 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 18-6 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. For example:

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

<pof-config xmlns="http://schemas.tangosol.com/pof"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.tangosol.com/pof
   assembly://Coherence/Tangosol.Config/pof-config.xsd">
   <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:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <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>
```

Using POF Object References

POF supports the use of object identities and references for objects that occur more than once in a POF stream. Objects are labeled with an identity and subsequent instances of a labeled object within the same POF stream are referenced by its identity.
Using references avoids encoding the same object multiple times and helps reduce the data size. References are typically used when a large number of sizeable objects are created multiple times or when objects use nested or circular data structures. However, for applications that contain large amounts of data but only few repeats, the use of object references provides minimal benefits due to the overhead incurred in keeping track of object identities and references.

The use of object identity and references has the following limitations:

- Object references are only supported for user defined object types.
- Object references are not supported for `IEvolvable` objects.
- Object references are not supported for keys.
- Objects that have been written out with a POF context that does not support references cannot be read by a POF context that supports references. The opposite is also true.
- POF objects that use object identity and references cannot be queried using POF extractors. Instead, use the `IValueExtractor` API to query object values or disable object references.
- The use of the `IPofNavigator` and `IPofValue` API has the following restrictions when using object references:
  - Only read operations are allowed. Write operations result in an `UnsupportedOperationException`.
  - User objects can be accessed in non-uniform collections but not in uniform collections.
  - For read operations, if an object appears in the data stream multiple times, then the object must be read where it first appears before it can be read in the subsequent part

of the data. Otherwise, an `IOException: missing identity: <ID>` may be thrown. For example, if there are 3 lists that all contain the same person object, `p`. The `p` object must be read in the first list before it can be read in the second or third list.

This section includes the following topics:

- Enabling POF Object References
- Registering POF Object Identities for Circular and Nested Objects

Enabling POF Object References

Object references are not enabled by default and must be enabled either within a `pof-config.xml` configuration file or programmatically when using the `SimplePofContext` class.

To enable object references in the POF configuration file, include the `<enable-references>` element, within the `<pof-config>` element, and set the value to `true`. For example:

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

<pof-config xmlns="http://schemas.tangosol.com/pof"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.tangosol.com/pof
   assembly://Coherence/Tangosol.Config/pof-config.xsd">
   ...
   <enable-references>true</enable-references>
</pof-config>
```

To enable object references when using the `SimplePofContext` class, call the `setReferenceEnabled` method and set it to `true`. For example:

```
SimplePofContext ctx = new SimplePofContext();
ctx.IsReferenceEnabled = true;
```

Registering POF Object Identities for Circular and Nested Objects

Circular or nested objects must manually register an identity when creating the object. Otherwise, a child that references the parent will not find the identity of the parent in the reference map. Object identities can be registered from a serializer during the deserialization routine using the `Tangosol.IO.Pof.IPofReader.RegisterIdentity` method.

The following examples demonstrate two objects (`Customer` and `Product`) that contain a circular reference and a serializer implementation that registers an identity on the `Customer` object.

The `Customer` object is defined as follows:

```
public class Customer
   {
      String m_name;
      Product m_product;

   public Customer(String name)
      {
      m_name = name;
      }

   public Customer(String name, Product product)
      {
      m_name = name;
```

```
      m_product = product;
      }

   public String getName()
      {
      return m_name;
      }

   public Product getProduct()
      {
      return m_product;
      }

   public void setProduct(Product product)
      {
         m_product = product;
      }
   }
```

The `Product` object is defined as follows:

```
public class Product
   {
      private Customer m_customer;

   public Product(Customer customer)
      {
      m_customer = customer;
      }

   public Customer getCustomer()
      {
      return m_customer;
      }
   }
```

The serializer implementation registers an identity during deserialization and is defined as follows:

```
public class CustomerSerializer : IPofSerializer
   {
   public void Serialize(IPofWriter pofWriter, object o)
      {
      var c = (Customer) o;
      pofWriter.WriteString(0, c.getName());
      pofWriter.WriteObject(1, c.getProduct());
      pofWriter.WriteRemainder(null);
   }

   public object Deserialize(IPofReader pofReader)
      {
      String name = pofReader.ReadString(0);
      var customer = new Customer(name);

      pofReader.RegisterIdentity(customer);
      customer.setProduct((Product) pofReader.ReadObject(1));
      pofReader.ReadRemainder();
      return customer;
      }
   }
```

Using POF Annotations to Serialize Objects

POF annotations provide an automated way to implement the serialization and deserialization routines for an object. POF annotations are serialized and deserialized using the `PofAnnotationSerializer` class which is an implementation of the `IPofSerializer` interface. Annotations offer an alternative to using the `IPortableObject` and `IPofSerializer` interfaces and reduce the amount of time and code that is required to make objects serializable.

This section includes the following topics:

Annotating Objects for POF Serialization

Two annotations are available to indicate that a class and its properties are POF serializable:

- `[Portable]` – Marks the class as POF serializable. The annotation is only permitted at the class level and has no members.
- `[PortableProperty]` – Marks a property, accessor, or member variable as a POF serialized property. Annotated methods must conform to accessor notation (`Get`, `Set`, `Is`). Members can be used to specify POF indexes as well as custom codecs that are executed before or after serialization or deserialization. Index values may be omitted and automatically assigned. If a custom codec is not entered, the default codec is used.

The following example demonstrates annotating a class, property, and member variable. In addition `PortableProperty` indexes are explicitly specified.

```
[Portable]
public class Person
{
   [PortableProperty(0)]
   public string GetFirstName()
   {
      return m_firstName;
   }

   private String m_firstName;

   [PortableProperty(1)]
   public string LastName;
   {
      get; set;
   }

   [PortableProperty(2)]
   private int m_age;
}
```

Registering POF Annotated Objects

POF annotated objects must be registered in a `pof-config.xml` file within a `<user-type>` element. See POF User Type Configuration Elements in *Developing Applications with Oracle*

Coherence. POF annotated objects use the `PofAnnotationSerializer` serializer if an object does not implement `IPortableObject` and is annotated as `Portable`; however, the serializer is automatically assumed if an object is annotated and does not need to be included in the user type definition. The following example registers a user type for an annotated `Person` object:

```
<?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
      <!-- User types must be above 1000 -->
      <user-type>
         <type-id>1001</type-id>
         <class-name>My.Examples.Person, MyAssembly</class-name>
      </user-type>
   </user-type-list>
</pof-config>
```

Enabling Automatic Indexing

POF annotations support automatic indexing which alleviates the need to explicitly assign and manage index values. Omit the index value when defining the `[PortableProperty]` annotation. Index allocation is determined by the property name. Any property that does assign an explicit index value is not assigned an automatic index value. The following table demonstrates the ordering semantics of the automatic index algorithm. Notice that automatic indexing maintains explicitly defined indexes (as shown for property `c`) and assigns an index value if an index is omitted.

Name	Explicit Index	Determined Index
c	1	1
a	omitted	0
b	omitted	2

> **Note:**
>
> Automatic indexing does not currently support evolvable classes.

To enable automatic indexing, the `PofAnnotationSerializer` serializer class must be explicitly defined when registering the object as a user type in the POF configuration file. The `autoIndex` boolean parameter in the constructor enables automatic indexing and must be set to `true`. For example:

```
<user-type>
   <type-id>1001</type-id>
   <class-name>Examples.Person</class-name>
   <serializer>
      <class-name>Tangosol.IO.Pof.PofAnnotationSerializer, Coherence</class-name>
         <init-params>
         <init-param>
            <param-type>int</param-type>
            <param-value>{type-id}</param-value>
         </init-param>
         <init-param>
            <param-type>class</param-type>
```

```
            <param-value>{class}</param-value>
         </init-param>
         <init-param>
            <param-type>bool</param-type>
            <param-value>true</param-value>
         </init-param>
      </init-params>
   </serializer>
</user-type>
```

Providing a Custom Codec

Codecs allow code to be executed before or after serialization or deserialization. A codec defines how to encode and decode a portable property using the `IPofWriter` and `IPofReader` interfaces. Codecs are typically used for concrete implementations that could get lost when being deserialized or to explicitly call a specific method on the `IPofWriter` interface before serializing an object.

To create a codec, create a class that implements the `ICodec` interface. The following example demonstrates a codec that defines the concrete implementation of a linked list type:

```
public class LinkedListCodec<T> : ICodec
{
   public object Decode(IPofReader reader, int index)
      {
         return reader.ReadCollection(index, (ICollection)new LinkedList<T>());
      }

   public void Encode(IPofWriter writer, int index, object value)
      {
         writer.WriteCollection(index, (ICollection)value);
      }
}
```

To assign a codec to a property, enter the codec as a member of the `[PortableProperty]` attribute. If a codec is not specified, a default codec (`DefaultCodec`) is used. The following example demonstrates assigning the above `LinkedListCodec` codec:

```
[PortableProperty(typeof(LinkedListCodec<string>))]
```

19

Using the Coherence .NET Client Library

Learn about the Coherence for .NET API and how to add the .NET client library to an application. Coherence for .NET API documentation is available at .NET API Reference for Oracle Coherence and in the `doc` directory of the Coherence for .NET distribution.
This chapter includes the following sections:

- Setting Up the Coherence .NET Client Library
- Using the Coherence .NET APIs
- Configuring .NET Clients Programmatically

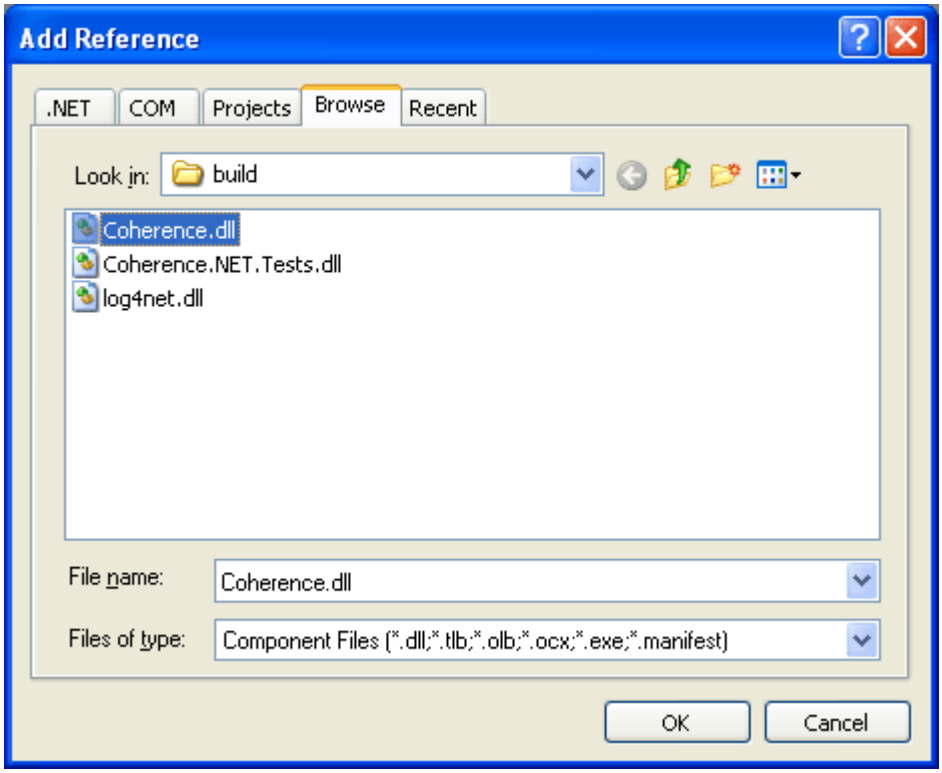
Setting Up the Coherence .NET Client Library

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.

To create 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**.
4. 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 does not exist) 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 looks for them in both the folder where the application is deployed or, for Web applications, in the root of the Web application. You can also specify the cache configuration file programmatically. See Configuring .NET Clients Programmatically.

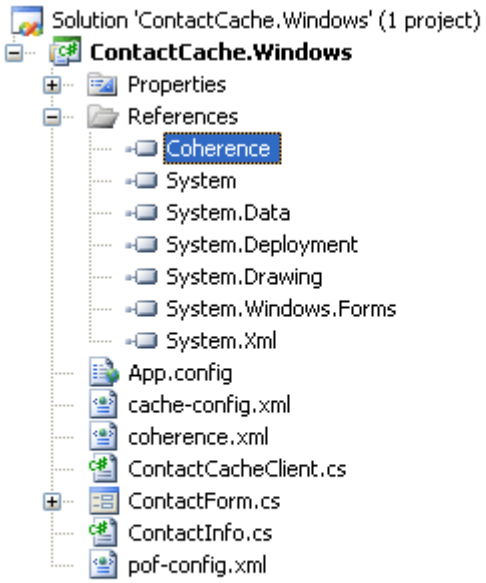
```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="coherence" type="Tangosol.Config.CoherenceConfigHandler, Coherence"/>
  </configSections>
  <coherence>
    <cache-factory-config>my-coherence.xml</cache-factory-config>
    <cache-config>my-cache-config.xml</cache-config>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

The Coherence .NET API includes many classes that are used to interact with Coherence caches within a .NET application.
This section includes the following topics:

- CacheFactory
- IConfigurableCacheFactory
- DefaultConfigurableCacheFactory
- Logger
- Using the Common.Logging Library
- INamedCache
- IQueryCache
- QueryRecorder
- IObservableCache
- IInvocableCache
- Filters
- Value Extractors
- Entry Processors
- Entry Aggregators

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. You can also specify the cache configuration file programmatically. See Configuring .NET Clients Programmatically.

```
<?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.

```
<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. The default configuration file used by the `DefaultConfigurableCacheFactory` is `$AppRoot/coherence-cache-config.xml`, where `$AppRoot` is the working directory (for a Windows Forms application) or the root of the application (for 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. You can also specify the cache configuration file programmatically. See Configuring .NET Clients Programmatically.

```
<?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`—specifies the log level that a message must meet or exceed to be logged.
- `logger-name`—specifies the name of the logger. The default value is `Coherence`.
- `message-format`—determines the log message format.
- `character-limit`—determines the maximum number of characters that the logger daemon processes from the message queue before discarding all remaining messages in the queue.

```
...
<logging-config>
   <destination>common-logger</destination>
   <logger-name>Coherence</logger-name>
   <severity-level>5</severity-level>
   <message-format>(thread={thread}): {text}</message-format>
   <character-limit>8192</character-limit>
</logging-config>
...
```

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 are likely to 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 allows logging to be consistently configured throughout the application layers.

Coherence for .NET does not include the `Common.Logging` library. To use the `common-logger` Logger configuration, 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. When 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 iterates 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 exists, its value is overwritten by the specified value and the old value is 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 traverses 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.

The following example performs an efficient query of the keys of all entries that have an age property value greater or equal to 55.

```
IValueExtractor extractor = new ReflectionExtractor("getAge");

cache.AddIndex(extractor, true, null);
ICollection keys = cache.GetKeys(new GreaterEqualsFilter(extractor, 55));
```

QueryRecorder

The `QueryRecorder` class produces an explain or trace record for a given filter. The class is an implementation of a parallel aggregator that is capable querying all nodes in a cluster and aggregating the results. The class supports two record types: an `Explain` record that provides the estimated cost of evaluating a filter as part of a query operation and a `Trace` record that provides the actual cost of evaluating a filter as part of a query operation. Both query records take into account whether or not an index can be used by a filter. See Interpreting Query Records in *Developing Applications with Oracle Coherence*.

To create a query record, create a new `QueryRecorder` instance that specifies a `RecordType` parameter. Include the instance and the filter to be tested as parameters of the `Aggregate` method. The following example creates an explain record:

```
INamedCache cache = CacheFactory.GetCache(MyCache);

IFilter filter = new OrFilter(
   new GreaterFilter(IdentityExtractor.Instance, 100),
   new LessFilter(IdentityExtractor.Instance, 30));

QueryRecorder aggregator = new QueryRecorder(QueryRecorder.RecordType.Explain);
IQueryRecord record = (IQueryRecord) cache.Aggregate(filter, aggregator);

Console.WriteLine(record.ToString());
```

To create a trace record, change the `RecordType` parameter to `Trace`:

```
QueryRecorder aggregator = new QueryRecorder(QueryRecorder.RecordType.Trace);
```

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 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 receives 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 that are registered using the filter-based method receives 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.

The following example filter evaluates to `true` if an `Employee` object is inserted into a cache with an `IsMarried` property value set to `true`.

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

The following example filter evaluates to `true` if any object is removed from a cache.

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

The following example filter evaluates to `true` when an `Employee` object `LastName` property is changed from `Smith`.

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

This section includes the following topic:

- Responding to Cache Events

Responding to Cache Events

A feature 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 because 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 ensures that the call is properly marshalled and executed on the UI thread.

Here is the sample of using this class:

```
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:

```
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 following example retrieves the keys of all entries that have a value equal to 5.

```
EqualsFilter equalsFilter = new EqualsFilter(IdentityExtractor.Instance, 5);
ICollection  keys         = cache.GetKeys(equalsFilter);
```

The following example retrieves all keys that have a value greater or equal to 55.

```
GreaterEqualsFilter greaterEquals = new GreaterEqualsFilter(IdentityExtractor.Instance,
55);
ICollection         keys          = cache.GetKeys(greaterEquals);
```

The following example retrieves all cache entries that have a value that begins with `Belg`.

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

The following example retrieves all cache entries that have a value that ends with `an` (case sensitive) or begins with `An` (case insensitive).

```
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.

POF extractors and POF updaters offer the same functionality as `ChainedExtractors` through the use of the `SimplePofPath` class. See Using POF Extractors and POF Updaters in *Developing Applications with Oracle Coherence*.

The following example retrieves all cache entries with keys greater than 5:

```
IValueExtractor extractor = new KeyExtractor(IdentityExtractor.Instance);
IFilter         filter    = new GreaterFilter(extractor, 5);
ICollection     entries   = cache.GetEntries(filter);
```

The following example retrieves all cache entries with values containing a `City` property equal to `city1`:

```
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 following example demonstrates a conditional put. The value mapped to `key1` is set to `680` only if the current mapped value is greater than `600`.

```
IFilter         greaterThen600 = new GreaterFilter(IdentityExtractor.Instance, 600);
IEntryProcessor processor      = new ConditionalPut(greaterThen600, 680);
cache.Invoke("key1", processor);
```

The following example uses the `UpdaterProcessor` to update the value of the `Degree` property on a `Temperature` object with key `BGD` to the new value `26`.

```
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 following example returns the size of the cache:

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

The following example 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:

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

> **Note:**
>
> The above examples 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 these custom types is performed by the Java implementation and not the .NET implementation. See Building Integration Objects (.NET) .

Configuring .NET Clients Programmatically

Clients can load Coherence configuration files programmatically at runtime. The configuration files overwrite any configuration files that are specified in the application configuration file. See Setting Up the Coherence .NET Client Library.
The following example loads the `pofConfig.xml`, `cacheConfig.xml`, and `coherenceConfig.xml` files.

```
using System;
using System.IO;
using Tangosol.IO.Pof;
using Tangosol.Net;
using Tangosol.Run.Xml;

namespace configExample
{
    internal class TestPofContext : ConfigurablePofContext
    {
        public TestPofContext()

            : base("config/pofConfig.xml")
        {
        }
    }

    internal class TestClient
    {
        private static void Main(string[] args)
        {
```

```
                try
                {
                    CacheFactory.Configure("config/cacheConfig.xml",
                       "config/coherenceConfig.xml");
                    var cache = CacheFactory.GetCache("dist-test");
                    cache["key"] = new TestValue(1, "Test");
                    Console.Out.WriteLine("key=" + cache["key"]);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
                Console.ReadLine();
            }
        }
    }
```

20

Performing Continuous Queries (.NET)

You can use Continuous Query Caching in a .NET client to ensure that a query always retrieves the latest results from a cache in real-time.
This chapter includes the following sections:

Overview of Performing Continuous Queries (.NET)

Queries provide the ability to obtain a point in time query result from a Coherence cache and it is possible to receive events that would change the result of that query. However, the continuous query feature 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.

Understanding Use Cases for Continuous Query Caching

Continuous Query Caching is ideal for many use cases, such as event processing and instant access to up-to-date query results.
Consider using Continuous Query Caching in the following situations:

- A Continuous Query Cache is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.
- A Continuous Query Cache 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 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 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 always be maintained 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.

Understanding the Continuous Query Caching Implementation

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.
- 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

The `ContinuousQueryCache` class is used for continuous query caching and requires an underlying cache and a query filter.
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, a method 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 Continuous Query Cache Resources

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 is 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()`. If you want to obtain and release a cache instance within a single method, you can do so by using a using block:

```
using (INamedCache cache = CacheFactory.GetCache("my-cache"))
{
   // use cache as usual
}
```

After the using block terminates, `IDisposable.Dispose()` is called on the `INamedCache` instance, and all resources associated with it are released.

Caching Only Keys Versus Keys and Values

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:

```
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 is automatically 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.

This section includes the following topics:

Listening to a Continuous Query Cache

A client can place one or more event listeners onto a Continuous Query Cache. For example:

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.AddCacheListener(listener);
```

Assuming some processing has to occur against every item that is 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:

```
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:

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.AddCacheListener(listener);
foreach (ICacheEntry entry in cacheOpenTrades.Entries)
    {
    // .. process the cache entry
    }
```

However, the same entry may appear 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 receives 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:

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter, listener);
```

This section includes the following topics:

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 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

A Continuous Query Cache can be made into a read-only cache.
For example:

```
cacheOpenTrades.IsReadOnly = true;
```

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

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

21

Performing Remote Invocations (.NET)

You can perform remote invocations on Coherence caches from .NET clients. This chapter includes the following sections:

Overview of Performing Remote Invocations

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:

```
...
<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>7077</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:

```
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:

```
IDictionary result = service.Query(new MyTask(), null);
```

The single result of the execution are 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 Introduction to Coherence .NET Clients.

22

Performing Transactions (.NET)

You can use 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 *Developing Applications with Oracle Coherence*.

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 use POF to serialize between Java and C#.
Example 22-1 demonstrates an entry processor that performs a simple `update` operation within a transaction using the transaction API. At run time, the class must be located on the classpath of the Coherence proxy server.

Example 22-1 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 use 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 required any transaction logic and is a skeleton of the transactional entry processor. See Building Integration Objects (.NET) .

Example 22-2 demonstrate an entry processor stub class for the transactional entry processor created in Example 22-1.

Example 22-2 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 22-1 and the client stub class that was created in Example 22-2, respectively.
Cluster-side POF configuration:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <include>txn-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"?>

<pof-config xmlns="http://schemas.tangosol.com/pof"
```

```
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.tangosol.com/pof
   assembly://Coherence/Tangosol.Config/pof-config.xsd">
   <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 *Developing Applications with Oracle Coherence*.

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 22-1. 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 `7077`. See Configuring Extend Proxies .

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <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>
         <service-name>TransactionalCache</service-name>
         <thread-count-min>2</thread-count-min>
         <thread-count-max>10</thread-count-max>
         <high-units>15M</high-units>
         <task-timeout>0</task-timeout>
         <autostart>true</autostart>
      </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>
         <autostart>true</autostart>
      </proxy-scheme>
   </caching-schemes>
</cache-config>
```

Configuring the Client-Side Remote Cache

Remote clients require a remote cache 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 Configuring Extend Proxies .

The following example configures a remote cache to connect to a proxy that is located on `localhost` at port `7077`. 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.

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

<cache-config xmlns="http://schemas.tangosol.com/cache"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.tangosol.com/cache
  assembly://Coherence/Tangosol.Config/cache-config.xsd">
   <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>7077</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>
```

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 run time, 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 22-1:

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

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

23
Managing ASP.NET Session State

You can manage ASP.NET session state in a Coherence cluster by using a Coherence session provider.
This chapter includes the following sections:

Overview of ASP.NET Session State

Coherence for .NET allows ASP.NET session state to be managed in a Coherence cluster, which has some benefits as compared to the 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 configuration options that can significantly improve performance and scalability of applications.

Setting Up Coherence ASP.NET Session Management

To manage ASP .NET sessions, you must enable a Coherence session provider and configure ASP session caches.
This section includes the following topics:

Overview of 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). See Setting Up the Coherence .NET Client Library.
- Enable the Coherence Session Provider
- Configure the Cluster-Side ASP Session Caches
- Configure a Client-Side ASP Session Remote Cache
- Overriding the Default Session Cache Name

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, 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 .

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

<cache-config xmlns="http://schemas.tangosol.com/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.tangosol.com/cache
   assembly://Coherence/Tangosol.Config/cache-config.xsd">
   <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.

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.

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

<cache-config xmlns="http://schemas.tangosol.com/cache"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.tangosol.com/cache
   assembly://Coherence/Tangosol.Config/cache-config.xsd">
```

```
   <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>7077</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>
```

Overriding the Default Session Cache Name

The Coherence session provider's default behavior is to use a remote session cache named `aspnet-session-storage`. The remote cache example in Configure a Client-Side ASP Session Remote Cache demonstrates creating a remote cache with the default name. However, a session provider can be configured to use a remote cache with a name other than the default.

To override the default session cache name, add a `cacheName` attribute within the provider configuration. The following example specifies a cache named `my-session-cache`.

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

Selecting a Session Model

You can configure a Coherence session provider to store session state using different models depending on an application's requirements.

This section includes the following topics:

Overview of Session Models

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 to avoid deserialization penalty for items that are not accessed. However, there are certain scenarios where monolithic or split model are better choices. See Session Model in *Administering HTTP Session Management with Oracle Coherence*Web*. The topic can help determine which model is the best fit for a particular application. The topic focuses on Coherence*Web; however, the general concepts are 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 ensures 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 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>
```

Configuring a Serializer

You can configure a Coherence session provider to use a specific serializer, including a Coherence POF serializer.
This section includes the following topics:

- Specifying a Serializer
- Using POF for Session Serialization

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. See Building Integration Objects (.NET) .

The following discussion summarizes some implementation details that should be considered when using POF. See The PIF-POF Binary Format in *Developing Applications with Oracle Coherence*.

When session items are deserialized by the POF serializer, there is no guarantee that the type of the resulting object equals the type of the original value. For example, integer values between -1 and 22 (inclusive) are returned as `Int32` values, regardless of the original type, so they may require a 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 provides 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 must 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, do not forget to update corresponding session items with the new collection, or the changes are not 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 ASP.NET Session State Across Applications

In some cases, it is beneficial 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.

Part V
Getting Started with gRPC

Learn how to use the Coherence gRPC library to interact with a Coherence data management services using Java, JavaScript, Python, and Go clients.

This part contains the following chapters:

- Introduction to gRPC
  Coherence provides the ability for clients in various languages to connect to a cluster using gRPC (`https://grpc.io/`) as the network transport.
- Using the Coherence gRPC Proxy Server
  The Coherence gRPC proxy is the server-side implementation of the gRPC services defined within the Coherence gRPC module. The gRPC proxy uses standard gRPC Java libraries to provide Coherence APIs over gRPC to the Java, JavaScript, Python, and Go gRPC clients.
- Using the Coherence Java gRPC Client
  The Coherence gRPC Java client allows Java applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses for Java gRPC clients include desktop and web applications that require access to remote Coherence resources.
- Using the JavaScript, Python, and Go gRPC Clients
  To connect to Coherence from other clients which use gRPC, such as JavaScript, Python, and Go, you should refer to the relevant open source repositories for details and examples.

24

Introduction to gRPC

Coherence provides the ability for clients in various languages to connect to a cluster using gRPC (`https://grpc.io/`) as the network transport.

For Java clients, connecting using gRPC provides an alternative to Coherence*Extend connections and can be advantageous when you need to connect through a load balancer as gRPC uses HTTP/2 under the covers and is more load balancer friendly.

If you want to connect to Coherence from JavaScript, Python, or Go clients, then gRPC is the only protocol supported.

For any of the language options, from a cluster perspective, you must include the `coherence-grpc-proxy` module, with which the server-side gRPC proxy will accept the gRPC connections and carry out work on behalf of the clients.

See Using the Coherence gRPC Proxy Server for setting up a gRPC Proxy server.

25

Using the Coherence gRPC Proxy Server

The Coherence gRPC proxy is the server-side implementation of the gRPC services defined within the Coherence gRPC module. The gRPC proxy uses standard gRPC Java libraries to provide Coherence APIs over gRPC to the Java, JavaScript, Python, and Go gRPC clients.

This chapter includes the following sections:

- Setting Up the Coherence gRPC Proxy Server
  To set up and start using the Coherence gRPC Server, you should declare it as a dependency of your project.
- Configuring the Server
  Configuring the gRPC server includes configuring the server listen address and port, SSL/TLS, and server thread pool.
- Disabling the gRPC Proxy Server
- Deploying the Proxy Service with Helidon Microprofile gRPC Server
  If you use the Helidon Microprofile server with the microprofile gRPC server enabled, you can deploy the Coherence gRPC proxy into the Helidon gRPC server instead of the Coherence default gRPC server.

Setting Up the Coherence gRPC Proxy Server

To set up and start using the Coherence gRPC Server, you should declare it as a dependency of your project.

For example:

If using Maven, declare the server as follows:

pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>${coherence.group.id}</groupId>
            <artifactId>coherence-bom</artifactId>
            <version>${coherence.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>${coherence.groupId}</groupId>
        <artifactId>coherence</artifactId>
    </dependency>
    <dependency>
        <groupId>${coherence.groupId}</groupId>
```

```
        <artifactId>coherence-grpc-proxy</artifactId>
    </dependency>
<dependencies>
```

In the `pom.xml` file, `coherence.version` property is the version of Coherence being used, and `coherence.groupId` property is either the Coherence commercial group id, `com.oracle.coherence`, or the CE group id, `com.oracle.coherence.ce`.

If using Gradle, declare the server as follows:

```
build.gradle

dependencies {
    implementation platform("${coherenceGroupId}:coherence-bom:$
{coherenceVersion}")

    implementation "${coherenceGroupId}:coherence"
    implementation "${coherenceGroupId}:coherence-grpc-proxy"
}
```

In the `build.gradle` file, `coherenceVersion` property is the version of Coherence being used, and `coherenceGroupId` property is either the Coherence commercial group id, `com.oracle.coherence` or the CE group id, `com.oracle.coherence.ce`.

This section includes the following topic:

- Starting the Server
  The gRPC server starts automatically when you run `com.tangosol.net.Coherence` (or `com.tangosol.net.DefaultCacheServer`). Typically, `com.tangosol.net.Coherence` class should be used as the application's main class. Alternatively, you can start an instance of `com.tangosol.net.Coherence` by using the Bootstrap API.

Starting the Server

The gRPC server starts automatically when you run `com.tangosol.net.Coherence` (or `com.tangosol.net.DefaultCacheServer`). Typically, `com.tangosol.net.Coherence` class should be used as the application's main class. Alternatively, you can start an instance of `com.tangosol.net.Coherence` by using the Bootstrap API.

By default, the gRPC server listens on all local addresses using an ephemeral port. Like Coherence*Extend, the endpoints that the gRPC server is bound to can be discovered by a client using the Coherence NameService. So, using ephemeral ports allows the gRPC server to start without any port clashes.

When reviewing the log output, the two log messages appear as follows:

```
In-Process GrpcAcceptor is now listening for connections using name "default"
GrpcAcceptor now listening for connections on 0.0.0.0:55550
```

The service is ready to process requests from one of the Coherence gRPC client implementations.

Configuring the Server

Configuring the gRPC server includes configuring the server listen address and port, SSL/TLS, and server thread pool.

The Coherence gRPC proxy is configured using an internal default cache configuration file named `grpc-proxy-cache-config.xml` that contains only a single `<proxy-scheme>` configuration for the gRPC proxy. You need not override the configuration file as the server can be configured with system properties and environment variables.

This section includes the following topics:

Configuring the Server Listen Address

At runtime, you can configure the address that the gRPC server binds to by setting the `coherence.grpc.server.address` system property or `COHERENCE_GRPC_SERVER_ADDRESS` environment variable.

By default, the server binds to the address `0.0.0.0` which equates to all the local host's network interfaces.

For example, if the host had a local IP address `192.168.0.25`, you can configure the server to bind to this specific address as follows:

Using system properties:

```
-Dcoherence.grpc.server.address=192.168.0.25
```

Using environment variables:

```
export COHERENCE_GRPC_SERVER_ADDRESS=192.168.0.25
```

Configuring the Server Listen Port

At runtime, you can configure the port that the gRPC server binds to by setting the `coherence.grpc.server.port` system property or `COHERENCE_GRPC_SERVER_PORT` environment variable .

For example, to configure the server to listen on port `1408`:

Using system properties:

```
-Dcoherence.grpc.server.port=1408
```

Using environment variables:

```
export COHERENCE_GRPC_SERVER_PORT=1408
```

Configuring SSL/TLS

Like other Coherence services, you can configure the Coherence gRPC server to use SSL by specifying the name of a socket provider.

Named socket providers are configured in the Coherence operational configuration file (override file), `tangosol-coherence-override.xml`.

`tangosol-coherence-override.xml` file looks similar to:

```
<socket-providers>
      <socket-providerid="tls">
        <ssl>
          <identity-manager>
            <keysystem-property="coherence.security.key">server.key</key>
            <certsystem-property="coherence.security.cert">server.cert</cert>
          </identity-manager>
          <trust-manager>
            <certsystem-property="coherence.security.ca.cert">server-ca.cert</
cert>
          </trust-manager>
        </ssl>
      </socket-provider>
    </socket-providers>
```

After the named socket provider has been configured, you can configure the gRPC server to use that provider by setting the `coherence.grpc.server.socketprovider` system property or `COHERENCE_GRPC_SERVER_SOCKETPROVIDER` environment variable.

For example, if a socket provider named `tls` has been configured in the operational configuration file, `tangosol-coherence-override.xml`, you can configure the gRPC server to use `tls` as follows:

Using system properties:

```
-Dcoherence.grpc.server.socketprovider=tls
```

Using environment variables:

```
export COHERENCE_GRPC_SERVER_SOCKETPROVIDER=tls
```

For more information about socket providers and how to configure them, see Using SSL to Secure Communication in *Securing Oracle Coherence*.

Configuring the Server Thread Pool

Like other Coherence services, the gRPC server uses a dynamically sized thread pool to process requests. If the dynamic sizing algorithm proves to not be optimal, you can configure the thread pool size by adjusting the minimum thread count and the maximum thread count.

This section includes the following topics:

- Setting the Minimum Thread Count
- Setting the Maximum Thread Count

Setting the Minimum Thread Count

Adjusting the minimum number of threads can be useful to deal with bursts in load.

At times, when the dynamic pool quickly deals with an increase in load, it takes some time to increase the thread count to a suitable number. Setting the minimum size ensures that there are always a certain number of threads to service load.

You can set the minimum number of threads in the pool by using the `coherence.grpc.server.threads.min system` property, or the `COHERENCE_GRPC_SERVER_THREADS_MIN` environment variable.

For example, you can set the minimum thread count to 10 as follows:

Using system properties:

```
-Dcoherence.grpc.server.threads.min=10
```

Using environment variables:

```
export COHERENCE_GRPC_SERVER_THREADS_MIN=10
```

Setting the Maximum Thread Count

Adjusting the maximum number of threads can be useful to stop the dynamic pool going too high and consuming too much CPU resource.

You can set the maximum number of threads in the pool by using the `coherence.grpc.server.threads.max system` property, or the `COHERENCE_GRPC_SERVER_THREADS_MAX` environment variable.

> **Note:**
>
> When you specify both maximum and minimum thread counts, you must set the maximum thread count at a higher value than the minimum thread count.

For example, you can set the maximum thread count to 20 as follows:

Using system properties:

```
Dcoherence.grpc.server.threads.max=20
```

Using environment variables:

```
export COHERENCE_GRPC_SERVER_THREADS_MAX=20
```

Disabling the gRPC Proxy Server

The Coherence gRPC server starts automatically if the `coherence-grpc-proxy` module is on the class path (or module path). However, you can disable the server by setting the `coherence.grpc.enabled` system property to *false*.

Deploying the Proxy Service with Helidon Microprofile gRPC Server

If you use the Helidon Microprofile server with the microprofile gRPC server enabled, you can deploy the Coherence gRPC proxy into the Helidon gRPC server instead of the Coherence default gRPC server.

For this behavior to happen automatically, set the `coherence.grpc.enabled` system property to `false`, which disables the built-in server. A built-in `GrpcMpExtension` implementation then deploys the proxy services to the Helidon gRPC server.

For more information about Helidon, see the Helidon Documentation.

> **Note:**
>
> When using the Helidon MP gRPC server, if you have not set the `coherence.grpc.enabled` system property to `false`, then both the Helidon gRPC server and the Coherence default gRPC server will start. This event can cause port binding issues unless you have configured both the servers to use different ports.

26

Using the Coherence Java gRPC Client

The Coherence gRPC Java client allows Java applications to access Coherence clustered services, including data, data events, and data processing from outside the Coherence cluster. Typical uses for Java gRPC clients include desktop and web applications that require access to remote Coherence resources.

This provides an alternative to using Coherence*Extend when writing client applications.

> **Note:**
>
> The Coherence gRPC client and Coherence*Extend client feature sets do not match exactly; some functionality in Coherence gRPC is not available in Coherence*Extend and vice versa.

Like cache clients that are members of the cluster, Java gRPC clients use the Session API call to retrieve resources, such as NamedMap, NamedCache, and such. After it is obtained, a client accesses these resources in the same way as it would if it were part of the Coherence cluster. The fact that operations on Coherence resources are being sent to a remote cluster node (over gRPC) is completely transparent to the client application. For the Java gRPC client, you need to include the `coherence-java-client` module. For more information, see the topics in the following sections.

This chapter includes the following sections:

- Setting Up the Coherence gRPC Client
  To set up and start using the Coherence gRPC Java client, you should declare it as a dependency of your project. The gRPC client is provided in the `coherence-java-client` module.
- Configuring the Coherence gRPC Client
  Like Coherence*Extend, a Coherence gRPC client accesses remote clustered resources by configuring remote schemes in the applications cache configuration file.
- Accessing Coherence Resources
  As the gRPC client is configured as a remote scheme in the cache configuration file, you can access the Coherence resources using the same Coherence APIs as used on cluster members or Extend clients.

Setting Up the Coherence gRPC Client

To set up and start using the Coherence gRPC Java client, you should declare it as a dependency of your project. The gRPC client is provided in the `coherence-java-client` module.

For example:

If using Maven, declare the server as follows:

```
pom.xml

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>${coherence.group.id}</groupId>
            <artifactId>coherence-bom</artifactId>
            <version>${coherence.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>${coherence.groupId}</groupId>
        <artifactId>coherence</artifactId>
    </dependency>
    <dependency>
        <groupId>${coherence.groupId}</groupId>
        <artifactId>coherence-java-client</artifactId>
    </dependency>
<dependencies>
```

In the `pom.xml` file, `coherence.version` property is the version of Coherence being used, and `coherence.groupId` property is either the Coherence commercial group id, `com.oracle.coherence`, or the CE group id, `com.oracle.coherence.ce`.

If using Gradle, declare the server as follows:

```
build.gradle

dependencies {
    implementation platform("${coherenceGroupId}:coherence-bom:$
{coherenceVersion}")

    implementation "${coherenceGroupId}:coherence"
    implementation "${coherenceGroupId}:coherence-java-client"
}
```

In the `build.gradle` file, `coherenceVersion` property is the version of Coherence being used, and `coherenceGroupId` property is either the Coherence commercial group id, `com.oracle.coherence` or the CE group id, `com.oracle.coherence.ce`.

Configuring the Coherence gRPC Client

Like Coherence*Extend, a Coherence gRPC client accesses remote clustered resources by configuring remote schemes in the applications cache configuration file.

This section includes the following topics:

- Overview of Configuring gRPC Clients
- Defining a Remote gRPC Cache

Overview of Configuring gRPC Clients

You can configure a gRPC client using the following two approaches:

- NameService - The simplest configuration in which the gRPC client uses the Coherence NameService to discover the gRPC endpoints in the cluster. In this configuration, Coherence discovers all the endpoints in the cluster that the gRPC proxy is listening on and the gRPC Java library's standard client-side load balancer is used to load balance connections from the client to those proxy endpoints.
- Fixed Endpoints - In this configuration, a fixed set of gRPC endpoints can be supplied via a custom `AddressProvider` configuration or the endpoints can be included in the software code. If multiple endpoints are provided, the gRPC Java library's standard client-side load balancer is used to load balance connections from the client to those proxy endpoints.

These approaches work only in some deployment environments. For example, the NameService configurations works if both clients and cluster are inside the same containerized environment. In containerized environments such as Kubernetes, NameService configuration is typically configured with a single ingress point which load balances connections to the Coherence cluster Pods. The address of this ingress point is then used as a single fixed address in the remote gRPC cache configuration. But, the NameService configuration does not work in containerized environments where the cluster is inside a containerized environment, such as Kubernetes, and the client is external to the containerized environment.

Defining a Remote gRPC Cache

A remote cache is specialized cache service that routes cache operations to a cache on the Coherence cluster. The remote cache and the cache on the cluster must have the same cache name. Coherence gRPC clients use the `NamedMap` or `NamedCache` interface as normal to get an instance of the cache. At runtime, the cache operations are not executed locally but instead are sent using gRPC to a gRPC proxy service on the cluster. The fact that the cache operations are delegated to a cache on the cluster is transparent to the client.

A remote cache is defined within a `<caching-schemes>` node using the `<remote-grpc-cache-scheme>` element.

In the case of a minimal NameService configuration (simplest configuration), the gRPC client uses the NameService to locate the gRPC proxy endpoints, but without adding any address or port information in the `<remote-grpc-cache-scheme>` in the configuration file. As this configuration uses Coherence's default cluster discovery mechanism to locate the Coherence cluster's NameService and look up the gRPC endpoints, you must configure the client with the same cluster name and well-known address list (or multicast configuration) as the cluster being connected to.

The following example shows an absolute minimum, required configuration, in which a `<remote-grpc-cache-scheme>` is configured with `<scheme-name>` and `<service-name>` elements:

```
coherence-cache-config.xml

<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
         <scheme-name>remote-grpc</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
   <remote-grpc-cache-scheme>
      <scheme-name>remote-grpc</scheme-name>
      <service-name>RemoteGrpcCache</service-name>
   </remote-grpc-cache-scheme>
</caching-schemes>
```

In the case of minimal NameService configuration with different cluster name, wherein, the client is configured with a different cluster name to the cluster being connected to (that is, the client is in a different Coherence cluster), you can configure the `<remote-grpc-cache-scheme>` with a cluster name.

The following example shows `<remote-grpc-cache-scheme>` configured with `<cluster-name>test-cluster</cluster-name>`, so Coherence uses the NameService to discover the gRPC endpoints in the Coherence cluster named `test-cluster`:

```
coherence-cache-config.xml

<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
         <scheme-name>remote-grpc</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
   <remote-grpc-cache-scheme>
      <scheme-name>remote-grpc</scheme-name>
      <service-name>RemoteGrpcCache</service-name>
      <cluster-name>test-cluster</cluster-name>
   </remote-grpc-cache-scheme>
</caching-schemes>
```

Configuring the NameService Endpoints

If the client cannot use the standard Coherence cluster discovery mechanism to look up the target cluster, you can specify the NameService endpoints in the `<grpc-channel>` node of the `<remote-grpc-cache-scheme>` configuration.

The following example shows how to create a remote cache scheme that is named `RemoteGrpcCache`, which connects to the Coherence NameService on `198.168.1.5:7574`, and then redirects the request to the address of the gRPC proxy service:

```
coherence-cache-config.xml

<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
         <scheme-name>remote-grpc</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
    <remote-grpc-cache-scheme>
        <scheme-name>remote-grpc</scheme-name>
        <service-name>RemoteGrpcCache</service-name>
        <grpc-channel>
            <name-service-addresses>
               <socket-address>
                  <address>198.168.1.5</address>
                  <port>7574</port>
               </socket-address>
            </name-service-addresses>
        </grpc-channel>
    </remote-grpc-cache-scheme>
</caching-schemes>
```

Configuring the Fixed Endpoints

If the NameService cannot be used to discover the gRPC endpoints, you can configure a fixed set of addresses by specifying a `<remote-addresses>` element containing one or more `<socket-address>` elements in the `<grpc-channel>` node.

The following example shows the client configuration that connects to a gRPC proxy listening on the endpoint `test-cluster.svc:1408`:

```
coherence-cache-config.xml

<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
         <scheme-name>remote-grpc</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
    <remote-grpc-cache-scheme>
        <scheme-name>remote-grpc</scheme-name>
        <service-name>RemoteGrpcCache</service-name>
        <grpc-channel>
            <remote-addresses>
               <socket-address>
                  <address>test-cluster.svc</address>
                  <port>1408</port>
               </socket-address>
            </remote-addresses>
        </grpc-channel>
```

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

Configuring SSL

Like other Coherence services, to configure the client to use SSL, configure the socket provider in the `<grpc-channel>` node of the `<remote-grpc-cache-scheme>` configuration . The `<socket-provider>` element can either contain the name of a socket provider configured in the operational override file, or can be configured with an inline socket provider configuration.

The following example shows the configuration of `<remote-grpc-cache-scheme>` with a reference to the socket provider named `ssl` that is configured in the operational override file:

coherence-cache-config.xml

```
<remote-grpc-cache-scheme>
    <scheme-name>remote-grpc</scheme-name>
    <service-name>RemoteGrpcCache</service-name>
    <grpc-channel>
        <remote-addresses>
           <socket-address>
              <address>test-cluster.svc</address>
              <port>1408</port>
           </socket-address>
        </remote-addresses>
        <socket-provider>ssl</socket-provider>
    </grpc-channel>
</remote-grpc-cache-scheme>
```

The following example shows the configuration of `<remote-grpc-cache-scheme>` with an inline socket provider:

coherence-cache-config.xml

```
<remote-grpc-cache-scheme>
    <scheme-name>remote-grpc</scheme-name>
    <service-name>RemoteGrpcCache</service-name>
    <grpc-channel>
        <remote-addresses>
           <socket-address>
              <address>test-cluster.svc</address>
              <port>1408</port>
           </socket-address>
        </remote-addresses>
        <socket-provider>
            <ssl>
                <identity-manager>
                    <key>server.key</key>
                    <cert>server.cert</cert>
                </identity-manager>
                <trust-manager>
                    <cert>server-ca.cert</cert>
                </trust-manager>
            </ssl>
        </socket-provider>
```

```
    </grpc-channel>
</remote-grpc-cache-scheme>
```

For more information about socket providers and how to configure them, see Using SSL to Secure Communication in *Securing Oracle Coherence*.

Configuring the Client Thread Pool

Unlike an Extend client, the gRPC client is built on top of a gRPC asynchronous client that is configured with a thread pool to allow the client to process multiple parallel requests and responses. The gRPC client uses a standard Coherence dynamically sized thread pool where the number of threads automatically adjust depending on the load. Sometimes if Coherence does not adjust the thread pool optimally for an application use case, you can configure the pool size by adjusting the minimum thread count and the maximum thread count. The thread count must be greater than or equal to the minimum count, and less than or equal to the maximum count, and the maximum count must be greater than or equal to the minimum count.

If you want to configure a fixed size pool, set the minimum and maximum to the same value.

The following example shows how to configure all three thread counts, fixed, minimum, and maximum. The pool starts with 10 threads and is automatically sized between 5 and 15 threads depending on load.

coherence-cache-config.xml

```
<remote-grpc-cache-scheme>
    <scheme-name>remote-grpc</scheme-name>
    <service-name>RemoteGrpcCache</service-name>
    <grpc-channel>
        <remote-addresses>
           <socket-address>
              <address>test-cluster.svc</address>
              <port>1408</port>
           </socket-address>
        </remote-addresses>
    </grpc-channel>
    <thread-count>10</thread-count>
    <thread-count-max>15</thread-count-max>
    <thread-count-min>5</thread-count-min>
</remote-grpc-cache-scheme>
```

Accessing Coherence Resources

As the gRPC client is configured as a remote scheme in the cache configuration file, you can access the Coherence resources using the same Coherence APIs as used on cluster members or Extend clients.

If you started the client using the Coherence Bootstrap API, running a `com.tangosol.net.Coherence` instance, you can access a `Session` and `NamedMap` as shown below:

```
Sessionsession = Coherence.getInstance().getSession();
NamedMap<String, String> map = session.getMap("test-cache");
```

This section includes the following topic:

- Using a Remote gRPC Cache As a Back Cache

Using a Remote gRPC Cache As a Back Cache

The gRPC client uses remote gRPC cache as a back cache of a near cache or a view cache in the same way as other types of caches.

The following example shows the configuration of a near cache that uses a `<remote-grpc-cache-scheme>` as the back cache:

coherence-cache-config.xml

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
         <scheme-name>near</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
    <near-scheme>
      <scheme-name>near</scheme-name>
      <front-scheme>
        <local-scheme>
          <high-units>10000</high-units>
        </local-scheme>
      </front-scheme>
      <back-scheme>
        <remote-grpc-cache-scheme>
          <scheme-ref>remote-grpc</scheme-ref>
        </remote-grpc-cache-scheme>
      </back-scheme>
    </near-scheme>

    <remote-grpc-cache-scheme>
      <scheme-name>remote-grpc</scheme-name>
      <service-name>RemoteGrpcCache</service-name>
    </remote-grpc-cache-scheme>
</caching-schemes>
```

27

Using the JavaScript, Python, and Go gRPC Clients

To connect to Coherence from other clients which use gRPC, such as JavaScript, Python, and Go, you should refer to the relevant open source repositories for details and examples.

- JavaScript: https://github.com/oracle/coherence-js-client
- Python: https://github.com/oracle/coherence-py-client
- Go: https://github.com/oracle/coherence-go-client

Part VI
Using Coherence REST

Learn how to use Coherence REST to allow applications written in any programming language to interact with cached data. Try creating a simple Coherence REST application.
Part V contains the following chapters:

- Introduction to Coherence REST
- Building Your First Coherence REST Application
- Performing Grid Operations with REST
- Deploying Coherence REST
- Modifying the Default REST Implementation

28

Introduction to Coherence REST

Before using Coherence REST, take some time learn how Coherence REST is implemented. Users should be familiar with Web services and JAX-RS to use Coherence REST.
This chapter includes the following sections:

Overview of Coherence REST

Coherence REST provides easy access to Coherence caches and cache entries over the HTTP protocol. It is similar to Coherence*Extend, as it allows remote clients to access data stored in Coherence without being members of the cluster themselves. However, unlike Coherence*Extend, which is a proprietary protocol, Coherence REST uses HTTP as the underlying protocol and can marshal data in both JSON and XML representation formats.The benefit of Coherence REST is that it allows applications written in others languages, such as Ruby and Python (that are not natively supported by Coherence), to interact with cached data.

Coherence REST Example

The Coherence distribution includes an end-to-end example of a REST application. See Coherence REST Examples in *Installing Oracle Coherence*.

Dependencies for Coherence REST

The Coherence REST implementation is packaged in the *COHERENCE_HOME*`/lib/coherence-rest.jar` library and depends on the `coherence.jar` library. In addition, the Coherence REST implementation has many library dependencies and also supports various HTTP server implementations (Netty HTTP Server, Simple HTTP Server, Jetty HTTP Server, and Grizzly HTTP Server). To manage these dependencies, it is strongly recommended that applications use Maven. If you are new to Maven, see: `https://maven.apache.org/`.
To use Coherence REST with the Netty HTTP server, add the following dependencies in the in the Maven `pom.xml` file:

> **Note:**
>
> When copying this `pom.xml` for your use, update all `<coherence.version>` elements and `<coherence.groupId>` elements to match the edition and Coherence version you are using.

```
<dependencies>
   <dependency>
```

```
      <groupId>${coherence.groupId}</groupId>
      <artifactId>coherence</artifactId>
      <version>${coherence.version}</version>
   </dependency>
   <dependency>
      <groupId>${coherence.groupId}</groupId>
      <artifactId>coherence-rest</artifactId>
      <version>${coherence.version}</version>
   </dependency>
    <dependency>
      <groupId>${coherence.groupId}</groupId>
      <artifactId>coherence-http-netty</artifactId>
      <version>${coherence.version}</version>
   </dependency>
</dependencies>
```

All the required libraries are automatically downloaded. To see the complete list of libraries, run the following Maven command:

```
mvn dependency:list
```

Refer to the Coherence REST examples for a complete `pom.xml` file.

Overview of Configuration for Coherence REST

Coherence REST is configured using the cache configuration file and the REST configuration file.

> **Note:**
>
> When deploying Coherence REST to a JavaEE server, configuration of the `web.xml` file is also required. See Deploying to a Java EE Server (Generic).

- Cache Configuration Deployment Descriptor – This file is used to define client-side cache services and the HTTP acceptor which accepts connections from remote REST clients over HTTP. The acceptor includes the address and port of the cluster-side HTTP server to which clients connects. The schema for this file is the `coherence-cache-config.xsd` file. See http-acceptor in *Developing Applications with Oracle Coherence*.

  At run time, the first cache configuration file that is found on the classpath is used. The `coherence.cacheconfig` system property can also be used to explicitly specify a cache configuration file. The file can also be set programmatically. See Specifying a Cache Configuration File in *Developing Applications with Oracle Coherence*.
- REST Configuration Deployment Descriptor – This file is used to configure the Jersey resource configuration class as well as custom aggregators and custom entry processors. The default name of the descriptor is `coherence-rest-config.xml` and the schema is defined in the `coherence-rest-config.xsd` file. The file must be found on the classpath and the name can be overridden using the `coherence.rest.config` system property. See REST Configuration Elements.

Understanding Data Format Support

Coherence REST supports both XML and JSON formats as input and output. To use these formats, the correct bindings are required when creating a user type. Both formats are demonstrated in this section.
This section includes the following topics:

Using XML as the Data Format

Objects that are represented in XML must have the appropriate JAXB bindings defined in order to be stored in a cache. The following example creates an object that uses annotations to add JAXB bindings:

```
@XmlRootElement(name="Address")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Address implements Serializable{
    private String street;
    private String city;
    private String country;

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }
}

@XmlRootElement(name="Person")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Person implements Serializable {
    private Long id;
    private String name;
    private Address address;
    public Long getId() {
        return id;
    }
```

```
    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @XmlElement(name = "address")
    public Address getAddr() {
        return address;
    }

    public void setAddr(Address addr) {
        this.addr = addr;
    }
}
```

Using JSON as the Data Format

Objects that are represented in JSON must have the appropriate Jackson bindings or JAXB bindings defined in order to be stored in a cache. The default Coherence REST JSON marshaller gives priority to Jackson bindings. If Jackson bindings are not found, JAXB bindings are used instead. Using Jackson annotations gives user more power on controlling the output JSON format. However, in case when both XML and JSON formats are needed, JAXB annotations can be enough for both formats.

The following example creates an object that uses annotations to add Jackson bindings:

```
@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS, include= JsonTypeInfo.As.PROPERTY,
   property="@type")
public class Address implements Serializable {
   private String street;
   private String city;
   private String country;

   public String getStreet() {
       return street;
   }

   public void setStreet(String street) {
       this.street = street;
   }

   public String getCity() {
      return city;
   }

   public void setCity(String city) {
      this.city = city;
   }

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
```

```
      this.country = country;
   }
}

@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS, include= JsonTypeInfo.As.PROPERTY,
   property="@type")
public class Person implements Serializable {
   private Long id;
   private String name;
   private Address address;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   @JsonProperty("address")
   public Address getAddr() {
      return address;
   }

   public void setAddr(Address addr) {
      this.addr = addr;
   }
}
```

Authenticating and Authorizing Coherence REST Clients

Coherence REST provides both authentication and authorization to restrict access to cluster resources. Authentication support includes both HTTP basic authentication and SSL authentication. Authorization is implemented using Coherence*Extend-styled authorization, which relies on interceptor classes that provide fine-grained access for named cache and invocation service operations. See Securing Oracle Coherence REST in *Securing Oracle Coherence*.

29

Building Your First Coherence REST Application

Build and run a simple Coherence REST application that accesses and uses a Coherence cache.
The Coherence examples that ship with the distribution also include an end-to-end example of a REST application. See Coherence REST Examples in *Installing Oracle Coherence*.

This chapter includes the following sections:

Overview of the Basic Coherence REST Example

The Coherence REST example is organized into a set of steps that are used to configure and run a basic Coherence REST application. The steps demonstrate fundamental concepts, such as: configuring a proxy server responsible for handling HTTP request, configuring a remote cache, and using the Coherence REST API.
The example in this chapter uses an embedded HTTP server in order to deploy a standalone application that does not require an application server. Additional deployment options are available. See Deploying Coherence REST .

Coherence for Java must be installed to complete the steps in this chapter. In addition, the following user-defined variables are used in this example:

- *`DEV_ROOT`* - The path to root folder where user is performing all of the listed steps, or in other words all of the following folders are relative to *`DEV_ROOT`*.
- *`COHERENCE_HOME`* - The path to folder containing Coherence JARs (`coherence.jar` and `coherence-rest.jar`)

Step 1: Configure the Cluster Side

Coherence REST requires both a cache and a proxy scheme. The proxy scheme must define an HTTP acceptor to handle an incoming HTTP request.
The cluster-side cache configuration deployment descriptor configures a cache and proxy. For this example, the proxy is configured to accept client HTTP requests on `localhost` and port `8080`. A distributed cache named `dist-http-example` is defined and is used to store client data in the cluster.

To configure the cluster side:

1. Create an XML file named `example-server-config.xml` in the *DEV_ROOT*`\config` folder.
2. Copy the following XML to the file:

```
<?xml version="1.0"?>
<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-http-example</cache-name>
         <scheme-name>dist-http</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>dist-http</scheme-name>
         <backing-map-scheme>
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </distributed-scheme>

      <proxy-scheme>
         <service-name>ExtendHttpProxyService</service-name>
         <acceptor-config>
            <http-acceptor>
               <local-address>
                  <address>localhost</address>
                  <port>8080</port>
               </local-address>
            </http-acceptor>
         </acceptor-config>
         <autostart>true</autostart>
      </proxy-scheme>
   </caching-schemes>
</cache-config>
```

3. Save and close the file.

Step 2: Create a User Type

Create the `Person` user type, which is stored in the cache and used to demonstrate basic REST operations.

To create the Person object:

1. Create a text file in a *DEV_ROOT*`\example` folder.
2. Copy the following Java code to the file:

```
package example;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="person")
@XmlAccessorType(XmlAccessType.PROPERTY)
```

```
public class Person implements Serializable {

    public Person() {}

    public Person(String name, int age)
        {
        m_name = name;
        m_age  = age;
        }

    public String getName() { return m_name; }

    public void setName(String name) { m_name = name; }

    public int getAge() { return m_age; }

    public void setAge(int age) {  m_age = age; }

    protected String m_name;
    protected int    m_age;
}
```

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

```
javac example\Person.java
```

Step 3: Configure REST Services

The Coherence REST services require metadata about the cache that it exposes. The metadata includes the cache entry's key and value types as well as key converters and value marshallers. The key and value types are required in order for Coherence to be able to use built-in converters and marshallers (XML and JSON supported).
To configure the REST services:

1. Create an XML file named `coherence-rest-config.xml` in *`DEV_ROOT`*`\config` folder.
2. Copy the following XML to the file:

```
<?xml version="1.0"?>
<rest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-rest-config"
xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-rest-config
   coherence-rest-config.xsd">
   <resources>
      <resource>
         <cache-name>dist-http-example</cache-name>
         <key-class>java.lang.String</key-class>
         <value-class>example.Person</value-class>
         <direct-query enabled="true"/>
      </resource>
   </resources>
</rest>
```

> **Note:**
>
> The `<key-class>` and `<value-class>` element can either be defined within the `<resource>` element or within the `<cache-mapping>` element in the cache configuration file.

3. Save and close the file

Step 4: Start the Cache Server Process

REST services are exposed as part of a cache server process (`DefaultCacheServer`). The cache server's classpath must be configured to find all the configuration files that were created in the previous steps as well as the `Person.class`. The classpath must also contain the required dependency libraries. See Dependencies for Coherence REST. For the sake of brevity, all of the dependencies are placed in *`DEV_ROOT`*`\libs` folder and are not individually listed.
The *`DEV_ROOT`* folder should appear as follows:

```
\
\config
\config\example-server-config.xml
\config\coherence-rest-config.xml
\example
\example\Person.class
\libs
\libs\*
```

The following command line starts a cache server process and explicitly names the cache configuration file created in Step 1 by using the `coherence.cacheconfig` system property. In addition it sets all the needed libraries and configuration files (replace *`dependencies`* with all the required library dependencies):

```
java -cp DEV_ROOT\config;DEV_ROOT;DEV_ROOT\libs\dependencies;
COHERENCE_HOME\coherence-rest.jar -Dcoherence.clusterport=8090
-Dcoherence.ttl=0
-Dcoherence.cacheconfig=DEV_ROOT\config\example-server-config.xml
com.tangosol.net.DefaultCacheServer
```

An example script for UNIX-based system follows:

```
#!/bin/bash

export CLASSPATH=${DEV_ROOT}/config:${DEV_ROOT}:
${DEVROOT}/lib/dependencies:${COHERENCE_HOME}/lib/coherence.jar:
${COHERENCE_HOME}/lib/coherence-rest.jar

java -cp ${CLASSPATH} -Dcoherence.clusterport=8090
-Dcoherence.ttl=0 -Dcoherence.cacheconfig=
${DEV_ROOT}/config/example-server-config.xml com.tangosol.net.DefaultCacheServer
```

Check the console output to verify that the proxy service has started. The output message should include the following:

```
(thread=Proxy:ExtendHttpProxyService:HttpAcceptor, member=1): Started:
HttpAcceptor{Name=Proxy:ExtendHttpProxyService:HttpAcceptor, State=(SERVICE_STARTED),
HttpServer=com.tangosol.coherence.rest.server.DefaultHttpServer, LocalAddress=localhost,
```

```
LocalPort=8080, ResourceConfig=com.tangosol.coherence.rest.server.DefaultResourceConfig,
RootResource=com.tangosol.coherence.rest.DefaultRootResource}
```

Step 5: Access REST Services From a Client

Client applications use Coherence REST services to perform cache operations. There are many application platforms that provide client libraries to build HTTP-based clients. For example, the Jersey project provides Java support for client-side communication with HTTP-based REST Web services.
The following sections demonstrate the semantics for `PUT`, `GET`, and `Post` operations that a client would use to access the `dist-http-example` cache. An example Java client built using Jersey follows and requires the `Jersey-client-2.12.jar` library. See Performing Grid Operations with REST .

Put Operations

```
PUT http://localhost:8080/api/dist-http-example/1
Content-Type=application/json
Request Body: {"name":"chris","age":30}

PUT http://localhost:8080/api/dist-http-example/2
Content-Type=application/json
Request Body: {"name":"adam","age":26}
```

GET Operations

```
GET http://localhost:8080/api/dist-http-example/1.json

GET http://localhost:8080/api/dist-http-example/1.xml

GET http://localhost:8080/api/dist-http-example/entries?q=name is 'chris'

GET http://localhost:8080/api/dist-http-example/1.json;p=name

GET http://localhost:8080/api/dist-http-example/count()

GET http://localhost:8080/api/dist-http-example/double-average(age)
```

Post Operation

```
POST http://localhost:8080/api/dist-http-example/increment(age,1)
```

Sample Jersey REST Client

```
package example;

import java.io.IOException;

import java.net.MalformedURLException;

import java.net.URISyntaxException;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
```

```
public class RestExample {
    public static void PUT(String url, MediaType mediaType, String data) {
        process(url, "put", mediaType, data);
    }

    public static void GET(String url, MediaType mediaType) {
        process(url, "get", mediaType, null);
    }

    public static void POST(String url, MediaType mediaType, String data) {
        process(url, "post", mediaType, data);
    }

    public static void DELETE(String url, MediaType mediaType) {
        process(url, "delete", mediaType, null);
    }

    public static void process(String sUrl, String action,
            MediaType mediaType,
            String data) {
        Client client = ClientBuilder.newClient();
        Response response = null;

        WebTarget webTarget = client.target(sUrl);
        String responseType = MediaType.APPLICATION_XML;
        if (mediaType == MediaType.APPLICATION_JSON_TYPE) {
            responseType = MediaType.APPLICATION_JSON;
        }

        if (action.equalsIgnoreCase("get")) {
            response = webTarget.request(responseType).get();
        } else if (action.equalsIgnoreCase("post")) {
            Entity<String> person = Entity.entity(data, responseType);
            response = webTarget.request(responseType).post(person);
        } else if (action.equalsIgnoreCase("put")) {
            Entity<String> person = Entity.entity(data, responseType);
            response = webTarget.request(responseType).put(person);
        } else if (action.equalsIgnoreCase("delete")) {
            Entity<String> person = Entity.entity(data, responseType);
            response = webTarget.request(responseType).delete();
        }
        System.out.println(response.readEntity(String.class));
    }

    public static void main(String[] args) throws URISyntaxException,
            MalformedURLException, IOException {
        PUT("http://localhost:8080/api/dist-http-example/1",
                MediaType.APPLICATION_JSON_TYPE,
                "{\"name\":\"chris\",\"age\":32}");
        PUT("http://localhost:8080/api/dist-http-example/2",
                MediaType.APPLICATION_JSON_TYPE,
                "{\"name\":\"\ufeff\u30b8\u30e7\u30f3A\",\"age\":66}");
        PUT("http://localhost:8080/api/dist-http-example/3",
                MediaType.APPLICATION_JSON_TYPE,
                "{\"name\":\"adm\",\"age\":88}");
        POST("http://localhost:8080/api/dist-http-example/increment(age,1)",
                MediaType.APPLICATION_XML_TYPE, null);
        GET("http://localhost:8080/api/dist-http-example/1",
                MediaType.APPLICATION_JSON_TYPE);
        GET("http://localhost:8080/api/dist-http-example/1",
                MediaType.APPLICATION_XML_TYPE);
        GET("http://localhost:8080/api/dist-http-example/count()",
```

```
                    MediaType.APPLICATION_XML_TYPE);
        }
```

30
Performing Grid Operations with REST

You can perform grid operations using the Coherence REST API. The Coherence REST API pre-defines many operations that can be used to interact with a cache. In addition, custom operations such aggregators and entry processors can be created as required.
This chapter includes the following sections:

Specifying Key and Value Types

The Coherence REST services require metadata about the cache that they expose. The metadata includes the cache entry's key and value types as well as key converters and value marshallers. The key and value types are required in order for Coherence to be able to use built-in converters and marshallers (both XML and JSON are supported).
To define the key and value types for a cache entry, edit the `coherence-rest-config.xml` file and include the `<key-class>` and the `<value-class>` elements within the `<resource>` element whose values are set to key and value types, respectively. See resource.

> **Note:**
>
> The `<key-class>` and `<value-class>` element can either be defined within the `<resource>` element or within the `<cache-mapping>` element in the cache configuration file.

The following example defines a `String` key class and a value class for a `Person` user type:

```
<resources>
   <resource>
      <cache-name>person</cache-name>
      <key-class>java.lang.String</key-class>
      <value-class>example.Person</value-class>
   </resource>
</resources>
```

Performing Single-Object REST Operations

The REST API includes support for performing GET, PUT, and DELETE operations on a single object in a cache.

GET Operation

`GET http://host:port/cacheName/key`

Returns a single object from the cache based on a key. A `404 (Not Found)` status code returns if the object with the specified key does not exist. The get operation supports partial results. See Performing Partial-Object REST Operations. Conditional gets are supported if an object implements the `com.tangosol.util.Versionsable` interface. The version is added to the response and used to determine if a client has the latest version of an object. If a client already has the latest version of an object, a `304 (Not Modified)` status code returns.

The following sample output demonstrates the response of a GET operation:

```
* Client out-bound request
> GET http://127.0.0.1:8080/dist-http-example/1
> Accept: application/xml

* Client in-bound response
< 200
< Content-Length: 212
< Content-Type: application/xml
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><Person><id>1</id><name>
Mark</name><address><street>500 Oracle Parkway</street><city>Redwood Shores</city>
<country>United States</country></address></Person>

* Client out-bound request
> GET http://127.0.0.1:8080/dist-http-example/1
> Accept: application/json

* Client in-bound response
< 200
< Content-Type: application/json
<
{"@type":"rest.Person","address":{"@type":"rest.Address","city":"Redwood Shores",
"country":"United States","street":"500 Oracle Parkway"},"id":1,"name":"Mark"}
```

PUT Operations

`PUT http://host:port/cacheName/key`

Creates or updates a single object in the cache. A `200 (OK)` status code returns if the object was updated. If optimistic concurrency check fails, a `409 (Conflict)` status code returns with the current object as an entity. See Understanding Concurrency Control.

The following sample output demonstrates the response of a PUT operation:

```
* Client out-bound request
> PUT http://127.0.0.1:8080/dist-test-sepx/1
> Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><Person><id>1</id><name>
Mark</name><address><street>500 Oracle Parkway</street><city>Redwood Shores</city>
<country>United States</country></address></Person>

* Client in-bound response
```

```
< 200
< Content-Length: 0
<

* Client out-bound request
> PUT http://127.0.0.1:8080/dist-test-sepj/1
> Content-Type: application/json
{"@type":"rest.Person","id":1,"name":"Mark","address":{"@type":"rest.Address","str
eet":"500 Oracle Parkway","city":"Redwood Shores","country":"United States"}}

* Client in-bound response
< 200
< Content-Length: 0
<
```

Delete Operation

`DELETE http://host:port/cacheName/key`

Deletes a single object from the cache based on a key. A `200 (OK)` status code returns if the object is successfully deleted, or a `404 (Not Found)` status code returns if the object with the specified key does not exist.

Performing Multi-Object REST Operations

Multi-object operations allow users to retrieve or delete multiple objects in a single network request and can significantly reduce the network usage and improve network performance.

> **Note:**
>
> PUT operations are not supported as it may produce tainted data. Specifically, it would require that individual objects (in serialized form) within the entity body to be in the same order as the corresponding keys in the URL. In addition, since updates result in a replacement, an entire object serialized form must be provided which can lead to overhead.

GET Operations

`GET http://host:port/cacheName/(key1, key2, ...)`

Returns a set of objects from the cache based on the specified keys. The ordering of returned objects is undefined and does not need to match the key order in the URL. Missing objects are silently omitted from the results. A `200 (OK)` status code always returns. An empty result set is returned if there are no objects in the result set. The get operation supports partial results. See Performing Partial-Object REST Operations.

DELETE Operations

`DELETE http://host:port/cacheName/(key1, key2, ...)`

Deletes multiple objects from the cache based on the specified keys. A `200 (OK)` status code always returns even if no objects for the specified keys were present in the cache.

Performing Partial-Object REST Operations

You can specify which object attributes to retrieve when performing GET operations. An application may not want (or need) to retrieve a whole object. For example, in order to populate a drop down with a list of options, the application may only need two properties of a potentially large object with many other properties. In order to support this use case, each read operation should accept a list of object properties that the user is interested in as a matrix parameter `p`. The following example performs a get operation that retrieves just the `id` and `name` attributes for a person:

```
GET http://localhost:8080/people/123;p=id,name
```

To include a `country` attribute of the address as well, the request URL is as follows:

```
GET http://localhost:8080/people/123;p=id,name,address:(country)
```

This approach allows an application to selectively retrieve only the properties that are required using a simple, URL-friendly notation.

The following sample output demonstrates the response of a GET operation:

```
* Client out-bound request
> GET http://127.0.0.1:8080/dist-test-sepj/1;p=name
> Accept: application/json

* Client in-bound response
< 200
< Transfer-Encoding: chunked
< Content-Type: application/json
<
{"name":"Mark"}
```

Performing Queries with REST

Coherence REST allows users to query a cache. CohQL is the default query syntax; however, additional query syntaxes can be created and used as required.
The section includes the following topics:

- Using Direct Queries
- Using Named Queries
- Specifying a Query Sort Order
- Limiting Query Result Size
- Retrieving Only Keys
- Using Custom Query Engines

Using Direct Queries

Direct queries are query expression that are submitted as the value of the parameter `q` in a REST URL. By default, the query expression must be specified as a URL-encoded CohQL expression (the predicate part of CohQL). See Filtering Entries in a Result Set in *Developing Applications with Oracle Coherence*. The syntax of a direct query is as follows:

```
GET http://host:port/cacheName?q=query
```

For example, to query the `person` cache for person objects where `age` is less than 18:

```
GET http://host:port/person?q=age%3C18
```

Direct queries are disabled by default. To enabled direct queries, edit the `coherence-rest-config.xml` file and add a `<direct-query>` element for each resource to be queried and set the `enabled` attribute to `true`. For example:

```
<resource>
   <cache-name>persons</cache-name>
   <key-class>java.lang.Integer</key-class>
   <value-class>example.Person</value-class>
   <direct-query enabled="true"/>
</resource>
```

A `403 (Forbidden)` response code is returned if a query is performed on a resource that does not have direct queries enabled.

Using Named Queries

Named queries are query expression that are configured for a resource in the `coherence-rest-config.xml` file. By default, the query expression must be specified as a CohQL expression (the predicate part of CohQL). Since this expression is configured in an XML file, any special characters (such as < and >) must be escaped using the corresponding entity. See Filtering Entries in a Result Set in *Developing Applications with Oracle Coherence*. In addition, named queries can include context values as required. The syntax of a named query is as follows:

```
GET http://host:port/cacheName/namedQuery?param1=value1,param2=value2...
```

To specify named queries, add any number of `<query>` elements, within a `<resource>` element, that each contain a query expression and name binding. See query. For example:

```
<resource>
   <cache-name>persons</cache-name>
   <key-class>java.lang.Integer</key-class>
   <value-class>example.Person</value-class>
   <query>
      <name>minors</name>
      <expression>age &lt; 18</expression>
   </query>
   <query>
      <name>first-name</name>
      <expression>name is :name</expression>
   </query>
</resource>
```

To use a named query, enter the name of the query within the REST URL. The following example uses the `minors` named query that is defined in the above example.

```
GET http://host:port/persons/minors
```

Parameters provide flexibility by allowing context values to be replaced in the query expression. The following example uses the `:name` parameter that is defined in the `first-name` query expression above to only query entries whose `name` property is `Mark`.

```
http://host:port/persons/first-name?name=Mark
```

Parameter names must be prefixed by a colon character (`:paramName`). Parameter bindings do not have access to type information, so it's possible to get a false where a true is expected on

the comparison operators. To avoid such behavior, specify type hints as part of a query parameter (`:paramName;int`). Table 30-1 lists the supported type hints.

Table 30-1 Parameter Type Hints

Hint	Type
i, int	`java.lang.Integer`
s, short	`java.lang.Short`
l, long	`java.lang.Long`
f, float	`java.lang.Float`
d, double	`java.lang.Double`
I	`java.math.BigInteger`
D	`java.math.BigDecimal`
date	`java.util.Date`
uuid	`com.tagosol.util.UUID`
uid	`com.tangosol.util.UID`
package.MyClass	`package.MyClass`

Named queries can also be used in conjunction with aggregation and entry processing. See Performing Aggregations with REST and Performing Entry Processing with REST, respectively. For example:

```
http://host:port/persons/first-name?name=Mark/long-max(age)

http://host:port/persons/first-name?name=Mark/increment(age,1)
```

Specifying a Query Sort Order

The `sort` matrix parameter is an optional parameter used within a REST URL that provides the ability to order the returned results of a query. The `sort` parameter is available for both direct queries and named queries. The value of the `sort` parameters is a comma-separated list of properties to sort on, each of which can have an optional `:asc` (default) or `:desc` qualifier that determines the order of the sort. For example, to sort a list of people by last name with family members sorted from the oldest to the youngest, the `sort` parameter is defined as follows:

```
GET http://host:port/persons/minors;sort=lastName,age:desc
```

The following example uses the `sort` parameter as part of a direct query.

```
GET http://host:port/persons;sort=lastName,age:desc?q=age%3C18
```

Limiting Query Result Size

Queries against large caches can potentially return large result sets that may cause out-of-memory errors. You should always use keys when querying large caches even though the use of keys in queries is optional. If keys are omitted, then the query may return all cache entries.

There are two ways to limit the number of results that are returned to a client: the `start` and `count` matrix parameters and the `max-results` attribute. Both ways are supported for direct and named queries.

The `start` and `count` parameters are optional integer arguments that determine the subset of the results to return. The following example uses the parameters as part of a named query and returns the first 10 entries sorted by name.

```
http://host:port/persons/minors;start=0;count=10;order=name:asc
```

The following example uses the parameters as part of a direct query.

```
GET http://host:port/persons;start=0;count=10?q=age%3C18
```

The `max-results` attribute is used within the `coherence-rest-config.xml` file and explicitly limits how many results are returned to the client. Note that this attribute does not limit the number of entries that are returned from a cache. The following example sets the `max-results` attribute:

```
<resource max-results="50">
   <cache-name>persons</cache-name>
   <key-class>java.lang.Integer</key-class>
   <value-class>example.Person</value-class>
   <direct-query enabled="true" max-results="25">
   <query max-results="25">
      <name>minors</name>
      <expression>age &lt; 18</expression>
   </query>
</resource>
```

The `max-results` value for a direct or named query overrides the resource's `max-results` value if both are specified. If a query includes a `count` parameter and a `max-results` element is also specified, the lesser value is used.

Retrieving Only Keys

It is possible to retrieve just keys of entries stored in cache. Key operations do not support paging and sorting, therefore those query parameters, if submitted, are ignored. The following key retrieval operations are supported:

```
GET http://host:port/cacheName/keys
```

Returns the keys of all entries in the cache.

```
GET http://host:port/cacheName/keys?q=query
```

Returns the keys of all entries satisfying the direct query criteria.

```
GET http://host:port/cacheName/namedQuery/keys
```

Returns the keys of all entries that satisfy the named query criteria.

Using Custom Query Engines

A query engine executes queries for both direct and named queries. The default query engine executes queries that are expressed using a CohQL syntax (the predicate part of CohQL).

> **Note:**
>
> The default query engine for Coherence REST uses MvelExtractor. Coherence REST users who use the default query engine should use the MvelExtractor to create cache index.

Implementing a custom query engine allows the use of different query expression syntaxes or the ability to execute queries against data sources other than Coherence (for example, to query a database for entries that are not present in a cache).

This section includes the following topics:

- Implementing Custom Query Engines
- Enabling Custom Query Engines

Implementing Custom Query Engines

Custom query engines must implement the `com.tangosol.coherence.rest.query.QueryEngine` and `com.tangosol.coherence.rest.query.Query` interfaces. Custom implementations can also extend the `com.tangosol.coherence.rest.query.AbstractQueryEngine` base class which provides convenience methods for parsing query expression and handling parameter bindings. The base class also supports parameter replacement at execution time and type hints that are submitted as part of the query parameter value. Both parameter names and type hints follow the CohQL specification and can be used for other query engine implementations. See Using Named Queries.

The following example is a simple query engine implementation that executes SQL queries directly against a database and forces cache read-through. In reality, a query engine implementation would probably support runtime parameter binding, which is not shown in the example.

```
public class SqlQueryEngine
   extends AbstractQueryEngine
   {
   protected Connection m_con;
   private static final String DB_DRIVER = "oracle.jdbc.OracleDriver";
   private static final String DB_URL = "jdbc:oracle:thin:@localhost:1521:orcl";
   private static final String DB_USERNAME = "username";
   private static final String DB_PASSWORD = "password";

   public SqlQueryEngine()
      {
      configureConnection();
      }

   @Override
   public Query prepareQuery(String sQuery, Map<String, Object> mapParams)
      {
      ParsedQuery parsedQuery = parseQueryString(sQuery);
      String  sSQL            = createSelectPKQuery(parsedQuery.getQuery());
      return new SqlQuery(sSQL);
      }

   protected void configureConnection()
      {
      try
```

```
            {
            Class.forName(DB_DRIVER);
            m_con = DriverManager.getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);
            m_con.setAutoCommit(true);
            }
         catch (Exception e)
            {
            throw new RuntimeException(e);
            }
         }

      protected String createSelectPKQuery(String sSQL)
         {
         return "SELECT id,name,age FROM " +
            sSQL.substring(sSQL.toUpperCase().indexOf("FROM") + 4);
         }

      private class SqlQuery
         implements Query
         {
         protected String m_sql;

         public SqlQuery(String sql)
            {
            m_sql = sql;
            }

         @Override
         public Collection values(NamedCache cache, String sOrder, int nStart,
            int cResults)
            {
            // force read through
            Set setKeys = keySet(cache);
            return cache.getAll(setKeys).values();
            }

         @Override
         public Set keySet(NamedCache cache)
            {
            Set setKeys = new HashSet();
            try
               {
               PreparedStatement stmt   = m_con.prepareStatement(m_sql);
               ResultSet         result = stmt.executeQuery();
               while (result.next())
                  {
                  Object oKey = result.getLong(1);
                  setKeys.add(oKey);
                  Person person = new Person(result.getString("name"),
                     result.getInt("age"));
                  cache.put(oKey, person);
                  }
               stmt.close();
               }
            catch (SQLException e)
               {
               throw new RuntimeException(e);
               }
            return setKeys;
            }
      }
   }
```

Enabling Custom Query Engines

Custom query engines are enabled in the `coherence-rest-config.xml` file. To enable a custom query engine, first register the implementation by adding an `<engine>` element, within the `<query-engines>` element, that includes a name for the query engine and the fully qualified name of the implementation class. See engine. For example:

```
<query-engines>
   <engine>
      <name>SQL-ENGINE</name>
      <class-name>package.SqlQueryEngine</class-name>
   </engine>
</query-engines>
```

To explicitly specify a custom query engine for a named query or a direct query, add the `engine` attribute, within a `<direct-query>` element or a `<query>` element, that refers to the custom query engine's registered name. For example:

```
<resource>
   <cache-name>persons</cache-name>
   <key-class>java.lang.Integer</key-class>
   <value-class>example.Person</value-class>
   <query engine="SQL-ENGINE">
      <name>less-than-1000</name>
      <expression>select * from PERSONS where id &lt; 1000</expression>
   </query>
   <direct-query enabled="true" engine="SQL-ENGINE"/>
 </resource>
```

To make a custom query engine the default query engine, use `DEFAULT` (uppercase mandatory) as the registered name. The following definition overrides the default CohQL-based query engine and is automatically used whenever an `engine` attribute is not specified.

```
<query-engines>
   <engine>
      <name>DEFAULT</name>
      <class-name>package.SqlQueryEngine</class-name>
   </engine>
</query-engines>
```

Performing Aggregations with REST

Aggregations can be performed on data in a cache. Coherence REST includes a set of pre-defined aggergators and custom aggregators can be created as required.

This section includes the following topics:

- Aggregation Syntax for REST
- Listing of Pre-Defined Aggregators
- Creating Custom Aggregators

Aggregation Syntax for REST

The following examples demonstrate how to perform aggregations using REST. If the aggregation succeeds, a `200 (OK)` status code returns with the aggregation result as an entity.

- Aggregates all entries in the cache.

```
GET http://host:port/cacheName/aggregator(args, ...)
```

- Aggregates query results. The query must be specified as a URL-encoded CohQL expression (the predicate part of CohQL).

```
GET http://host:port/cacheName/aggregator(args, ...)?q=query

GET http://host:port/cacheName/namedQuery/aggregator(args, ...)?param1=value1
```

- Aggregates specified entries.

```
GET http://host:port/cacheName/(key1, key2, ...)/aggregator(args, ...)
```

Coherence REST provides a simple strategy for aggregator creation (out of aggregator related URL segments). Out-of-box, Coherence REST can resolve any registered (either built-in or user registered) aggregator with a constructor that accepts a single parameter of type `com.tangosol.util.ValueExtractor` (such as `LongMax`, `DoubleMax`, and so on). If an aggregator call within a URL doesn't contain any parameters, the aggregator is created using `com.tangosol.util.extractor.IdentityExtractor`.

If an aggregator segment within the URL doesn't contain any parameters nor a constructor accepting a single `ValueExtractor` exists, Coherence REST tries to instantiate the aggregator using a default constructor which is the desired behavior for some built-in aggregators (such as `count`).

The following example retrieves the oldest person in a cache:

```
GET http://host:port/people/long-max(age)
```

The following example calculates the max number in a cache containing only numbers:

```
GET http://host:port/numbers/comparable-max()
```

The following example calculates the size of the people cache:

```
GET http://host:port/people/count()
```

Listing of Pre-Defined Aggregators

The following pre-defined aggregators are supported:

Aggregator Name	Aggregator
big-decimal-average	BigDecimalAverage.class
big-decimal-max	BigDecimalMax.class
big-decimal-min	BigDecimalMin.class
big-decimal-sum	BigDecimalSum.class
double-average	DoubleAverage.class
double-max	DoubleMax.class
double-min	DoubleMin.class
double-sum	DoubleSum.class
long-max	LongMax.class
long-min	LongMin.class
long-sum	LongSum.class
comparable-max	ComparableMax.class

Aggregator Name	Aggregator
comparable-min	ComparableMin.class
distinct-values	DistinctValues.class
count	Count.class

Creating Custom Aggregators

Custom aggregator types can be defined by specifying a name to be used in the REST URL and a class implementing either the `com.tangosol.util.InvocableMap.EntryAggregator` interface or the `com.tangosol.coherence.rest.util.aggregator.AggregatorFactory` interface.

An `EntryAggregator` implementation is used for simple scenarios when aggregation is either performed on single property or on cache value itself (as most of the pre-defined aggregators do).

The `AggregatorFactory` interface is used when a more complex creation strategy is required. The implementation must be able to resolve the URL segment containing aggregator parameters and use the parameters to create the appropriate aggregator.

Custom aggregators are configured in the `coherence-rest-config.xml` file within the `<aggregators>` elements. See aggregator. The following example configures both a custom `EntryAggregator` implementation and a custom `AggregatorFactory` implementation:

```
<aggregators>
   <aggregator>
      <name>my-simple-aggr</name>
      <class-name>com.foo.MySimpleAggregator</class-name>
   </aggregator>
   <aggregator>
      <name>my-complex-aggr</name>
      <class-name>com.foo.MyAggreagatorFactory</class-name>
   </aggregator>
</aggregators>
```

Performing Entry Processing with REST

Entry Processors can be invoked on one or more objects in a cache. Coherence REST includes a set of pre-defined entry processors and custom entry processors can be created as required.
This section includes the following topics:

- Entry Processor Syntax for REST
- Listing of Pre-defined Entry Processors
- Creating Custom Entry Processors

Entry Processor Syntax for REST

The following examples demonstrate how to perform entry processing using REST. If the processing succeeds, a `200 (OK)` status code returns with the processing result as an entity.

- Process all entries in the cache.

```
POST http://host:port/cacheName/processor(args, ...)
```

- Process query results.

```
POST http://host:port/cacheName/processor(args, ...)?q=query

POST http://host:port/cacheName/namedQuery?param1=value1/processor(args, ...)
```

- Process specified entries.

```
POST http://host:port/cacheName/(key1, key2, ...)/processor (args, ...)
```

Unlike aggregators, processors (even the pre-defined processors) have more diverse creation patterns, so Coherence REST does not assume anything about processor creation. Instead, for each entry processor implementation, there needs to be an implementation of the `com.tangosol.coherence.rest.util.processor.ProcessorFactory` interface that can handle an input string from a URL section and instantiate the processor instance. Out-of-box, Coherence REST provides two such factories for `NumberIncrementor` and `NumberMultiplier`.

The following example increments each person's age in a cache by 5:

```
POST http://localhost:8080/people/increment(age, 5)
```

The following example multiplies each number in a cache containing only numbers by the factor 10:

```
POST http://localhost:8080/numbers/multiply(10)
```

Listing of Pre-defined Entry Processors

The following pre-defined processors are supported:

Processor Name	Processor
`increment`	A `NumberIncrementor` instance that always returns the new (incremented) value
`post-increment`	A `NumberIncrementor` instance that always returns the old (not incremented) value
`multiply`	A `NumberMultiplier` instance that always returns the new (multiplied) value
`post-multiply`	A `NumberMultiplier` instance that always returns the old (not multiplied) value

Creating Custom Entry Processors

Custom entry processors can be defined by specifying a name to be used in a REST URL and a class that implements the `com.tangosol.coherence.rest.util.processor.ProcessorFactory` interface.

Custom entry processors are configured in the `coherence-rest-config.xml` file within the `<processors>` elements. See processors. The following example configures a custom `ProcesorFactory` implementation:

```
<processors>
   <processor>
      <name>my-processor</name>
      <class-name>com.foo.MyProcessorFactory</class-name>
   </processor>
</processors>
```

Understanding Concurrency Control

Coherence REST supports optimistic concurrency only as it maps cleanly to the HTTP protocol. When an application submits a `GET` request for an object that implements the `com.tangosol.util.Versionable` interface, the current version identifier is returned in an HTTP `ETag` (as well as in the representation of the object, assuming the version identifier is included in the JSON/XML serialized form). If the application then submits the same `GET` request for the resource, but this time with an `If-None-Match` header with the same `ETag` value, Coherence REST returns a status of `304`, indicating that the application has the latest version of the resource.
Likewise, when an application submits a `PUT` request to update an object that implements the `com.tangosol.util.Versionable` interface, Coherence REST performs an update only if the existing and new object versions match; otherwise a `409 Conflict` status is returned along with the current object so that the client can reapply the changes and retry.

The following example illustrates these concepts:

```
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import javax.ws.rs.core.MediaType;
import org.codehaus.jettison.json.JSONObject;

public class ConcurrencyTests
   {
   public static void main(String[] asArg)
      throws Exception
      {
      Client      client      = Client.create();
      String      url         = "http://localhost:" + getPort() + "/dist-test1/2";
      WebResource webResource = client.resource(url);

      // perform a GET of a server-side resource that implements Versionable
      ClientResponse response = webResource
         .accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
      assert 200 == response.getStatus(); /* OK */

      // verify that the current version of the resource is 1
      JSONObject json    = new JSONObject(response.getEntity(String.class));
      String     version = json.getString("versionIndicator");
      assert "1".equals(version);
      assert new EntityTag("1").equals(response.getEntityTag());

      // perform a conditional GET of the same resource and verify that we
      // get a response status of 304: Not Modified
      response = webResource
         .accept(MediaType.APPLICATION_JSON)
         .header ("If-None-Match", '"' + version + '"').get(ClientResponse.class);
      assert 304 == response.getStatus(); /* Not Modified */

      // simulate a version change on the server-side by rolling back the
      // version indicator on our representation of the resource
      json.put("versionIndicator", String.valueOf(0));

      // perform a conditional PUT of the same resource and verify that we
      // get a response status of 409: Conflict
      response = webResource
         .accept(MediaType.APPLICATION_JSON)
         .put(ClientResponse.class, json);
```

```
      assert 409 == response.getStatus(); /* Conflict */

      // retry again with the returned value and verify that we now get a
      // response status of 200: OK
      json = new JSONObject(response.getEntity(String.class));
      response = webResource
         .accept(MediaType.APPLICATION_JSON)
         .put(ClientResponse.class, json);
      assert 200 == response.getStatus(); /* OK */
      }
   }
```

Specifying Cache Aliases

Cache aliases are used to specify simplified cache names that are used when a cache name is not ideal for the REST URL path segment. The simplified names are mapped to the real cache names.
To define a cache alias, edit the `coherence-rest-config.xml` file and include the `<name>` attribute within the `<resource>` element whose value is set to a simplified cache name.

The following example creates a cache alias named `people` for a cache with the name `dist-extend-not-ideal-name-for-a-cache*`:

```
<resources>
   <resource name="people">
      <cache-name>dist-extend-not-ideal-name-for-a-cache*</cache-name>
      ...
   </resource>
</resources>
```

Using Server-Sent Events

Server-sent events allow Coherence REST applications to automatically receive cache events from the Coherence cluster. For example, events can be received when cache entries are inserted or deleted. For a complete example of using server-sent events, see the Coherence REST examples in Coherence REST Examples in *Installing Oracle Coherence*.
Server-sent events require the use of either the Grizzly HTTP server or the Jetty HTTP server. See Using Grizzly HTTP Server and Using Jetty HTTP Server, respectively. In addition, server-sent events must be supported by your web browser. Refer to your browser documentation for support details.

This section includes the following topic:

- Receiving Server-Sent Events

Receiving Server-Sent Events

Web pages use the `EventSource` object to receive server-sent events. The `EventSource` object connects to a specified URI where events are generated and custom `EventListeners` are added to listen and process the incoming server-sent events. The following code from the Coherence REST example uses JavaScript to create a new `EventSource` object that listens to the `/cache/contacts` URI and adds event listeners for `insert`, `update`, `delete`, and `error` events.

```
$scope.startListeningContacts = function() {
   $scope.contacts.listening = true;
   $scope.contacts.started   = true;
```

```
   if ($scope.contacts.filter == 'all') {
      query = '';
   }
   else if ($scope.contacts.filter == '>=45') {
      query = '?q=age%20>=%2045';
      $scope.contacts.filter = 'age >= 45';
   }
   else {
      query = '?q=age%20<%2045';
      $scope.contacts.filter = 'age < 45';
   }

   $scope.contacts.status  = 'Listening: ' + $scope.contacts.filter;
   var eventSourceContacts = new EventSource('/cache/contacts' + query);

   eventSourceContacts.addEventListener('insert', function(event) {
      $scope.contacts.insertCount++;
      $scope.contacts.allCount++;
      $scope.updateContactEvent(JSON.parse(event.data), 'insert');
      $scope.$apply();
   });

   eventSourceContacts.addEventListener('update', function(event) {
      $scope.contacts.updateCount++;
      $scope.contacts.allCount++;
      $scope.updateContactEvent(JSON.parse(event.data), 'update');
      $scope.$apply();
   });

   eventSourceContacts.addEventListener('delete', function(event) {
      $scope.contacts.deleteCount++;
      $scope.contacts.allCount++;
      $scope.updateContactEvent(JSON.parse(event.data), 'delete');
      $scope.$apply();
   });

   eventSourceContacts.addEventListener('error', function(event) {
      var eventData = JSON.parse(event.data);
      alert('error');
   });
};
```

When an event is received, an application can choose take some meaningful action based on the event. For example:

```
$scope.updateContactEvent = function(eventData, eventType) {
   $scope.contacts.eventType = eventType;
   $scope.contacts.eventKey  = eventData.key.firstName + ' ' +
      eventData.key.lastName;

   $scope.contacts.eventNewValue = 'N/A';
   $scope.contacts.eventOldValue = 'N/A';

   if (eventType == 'insert' || eventType == 'update') {
      $scope.contacts.eventNewValue = $scope.getContactString(eventData.newValue);
   }
   if (eventType == 'delete' || eventType == 'update') {
      $scope.contacts.eventOldValue = $scope.getContactString(eventData.oldValue);
   }
};
```

31
Deploying Coherence REST

You can deploy Coherence REST to an embedded HTTP server, WebLogic Server, and any generic servlet container.
This chapter includes the following sections:

Deploying with the Embedded HTTP Server

Coherence provides multiple embedded HTTP server implementations that can be used to host RESTful Web services. See Changing the Embedded HTTP Server.

- `DefaultHttpServer` (backed by Oracle's lightweight HTTP server)
- `NettyHttpServer` (backed by Netty HTTP server and recommended for production)
- `SimpleHttpServer` (backed by Simple HTTP server)
- `JettyHttpServer` (backed by Jetty HTTP server)
- `GrizzlyHttpServer` (backed by Grizzly HTTP server)

The HTTP server must be enabled on a Coherence proxy server. To enable the HTTP server, edit the proxy's cache configuration file and add an `<http-acceptor>` element, within the `<proxy-scheme>` element, and include the host and port for the HTTP server. The `<address>` element also supports external NAT addresses that route to local addresses; however, both addresses must use the same port number.

The following example configures the HTTP server to accept requests on localhost `127.0.0.1` and port `8080`. The example explicitly defines the HTTP server class and Jersey resource configuration class and uses `/` as the context path for the Coherence REST application. However; these are default values and need not be included. The context path can be changed as required and additional Coherence REST applications can be defined with different context paths. See http-acceptor in *Developing Applications with Oracle Coherence*.

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         <class-name>
            com.tangosol.coherence.rest.server.DefaultHttpServer</class-name>
         <local-address>
            <address>127.0.0.1</address>
            <port>8080</port>
         </local-address>
         <resource-config>
            <context-path>/</context-path>
            <instance>
               <class-name>
```

```
                    com.tangosol.coherence.rest.server.DefaultResourceConfig
                </class-name>
            </instance>
        </resource-config>
    </http-acceptor>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
```

If you are using POF, make sure that the `pof-config.xml` file includes the location of the REST POF types. See Configuring REST Server Access to POF-Enabled Services.

Deploying to WebLogic Server

WebLogic Server includes a Coherence integration that standardizes the way Coherence applications are packaged, deployed, and managed within a WebLogic Server domain. Coherence REST must follow the integration standards. See Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server*. In addition, Coherence applications must be packaged as a Grid ARchive (GAR). See Packaging Coherence Applications in *Developing Applications with Oracle Coherence*.
This section includes the following topics:

- Task 1: Configure a WebLogic Server Domain for Coherence REST
- Task 2: Package the Coherence REST Web Application
- Task 3: Package the Coherence Application
- Task 4: Package the Enterprise Application
- Task 5: Deploy the Enterprise Application

Task 1: Configure a WebLogic Server Domain for Coherence REST

Create a managed Coherence server in your WebLogic Server domain that will host Coherence REST. The server should be configured as a storage disabled member of a Coherence cluster. If more than one managed Coherence server is required for a Coherence REST solution, then the servers should be managed as a tier in a WebLogic Server cluster. See Setting Up a Coherence Cluster in *Administering Clusters for Oracle WebLogic Server*.

Task 2: Package the Coherence REST Web Application

To package the Coherence REST Web application:

1. Create a Web application directory structure as follows:

   ```
   /
   /WEB-INF/
   /WEB-INF/classes/
   /WEB-INF/lib/
   ```

2. Create a Web application deployment descriptor (`web.xml`) and include a servlet definition for the REST application as follows:

> **Note:**
>
> The WebLogic Server classpath contains the `coherence-rest.jar` library which includes a default servlet context listener that shuts down the cluster member during the REST application shutdown. The listener is registered as shown below. If the cluster member is not shut down, a variety of exceptions are thrown post shutdown.

```
<web-app>
   ...
   <listener>
      <listener-class>
         com.tangosol.coherence.rest.servlet.DefaultServletContextListener
      </listener-class>
   </listener>
   <servlet>
      <servlet-name>Coherence REST</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer
      </servlet-class>
      <init-param>
         <param-name>javax.ws.rs.Application</param-name>
         <param-value>
            com.tangosol.coherence.rest.server.ContainerResourceConfig
         </param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Coherence REST</servlet-name>
      <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
      ...
</web-app>
```

3. Save the `web.xml` file to the `/WEB-INF/` directory.
4. Create a WAR file using the `jar` utility. For example, issue the following command from a command prompt at the root of the Web application directory:

```
jar -cvf coherence_rest.war *
```

Task 3: Package the Coherence Application

To package the Coherence application:

1. Copy the `coherence-rest-config.xml` file to the root of your Coherence application. The structure should be as follows:

```
/
/com/myco/MyClass.class
/lib/
/META-INF/
/META-INF/coherence-application.xml
/META-INF/coherence-cache-config.xml
/META-INF/pof-config.xml
coherence-rest-config.xml
```

2. Edit the `pof-config.xml` file to include the `coherence-rest-pof-config.xml` POF configuration file that contains the Coherence REST default user types. See Configuring REST Server Access to POF-Enabled Services.
3. Create a GAR file using the `jar` utility. For example, issue the following command from a command prompt at the root of the GAR directory:

```
jar -cvf MyCohApp.gar *
```

Task 4: Package the Enterprise Application

To package the enterprise application:

1. Create an enterprise application directory structure and copy the Coherence REST WAR file and the Coherence application GAR file to the root of the EAR. For example:

```
/
/META-INF/
/META-INF/application.xml
/META-INF/weblogic-application.xml
/coherence_rest.war
/MyCohApp.gar
```

2. Edit the `application.xml` file and add a module definition for the Coherence REST Web application. For example:

```
<application>
   <module>
      <web>
         <web-uri>coherence_rest.war</web-uri>
         <context-root>/</context-root>
      </web>
   </module>
</application>
```

3. Edit the `weblogic-application.xml` file and add a module reference for the Coherence application GAR file. For example:

```
<weblogic-application>
   <module>
      <name>person</name>
      <type>GAR</type>
      <path>MyCohApp.gar</path>
   </module>
</weblogic-application>
```

4. Create the EAR file using the `jar` utility. For example, issue the following command from a command prompt at the root of the EAR directory:

```
jar -cvf MyCohRestApp.ear *
```

Task 5: Deploy the Enterprise Application

To deploy the Enterprise application:

1. Use the WebLogic Server Administration Console or WLST tool to deploy the EAR to the managed Coherence server created in Task 1.
2. From a browser, verify the deployment by navigating to the managed Coherence server's listening port and include the cache name as part of the URL. For example: `http://host:port/rest/{cacheName}`.

Deploying to a Java EE Server (Generic)

Coherence REST can be deployed to any standard Java EE environment. This section includes the following topics:

- Packaging Coherence REST for Deployment
- Deploying to a Servlet Container

Packaging Coherence REST for Deployment

To package a Coherence REST application:

1. Create a basic Web application directory structure as follows:

```
/
/WEB-INF
/WEB-INF/classes
/WEB-INF/lib
```

2. Copy the `coherence.jar` and `coherence-rest.jar` libraries from the *`COHERENCE_HOME`*`/lib` directory to the `/WEB-INF/lib` directory.
3. Copy the Coherence REST dependencies from the *`ORACLE_HOME`*`/oracle_common/modules/` directory to the `/WEB-INF/lib` directory. See Dependencies for Coherence REST.
4. Create a Web application deployment descriptor (`web.xml`) and include a servlet definition for the REST application as follows:

> **Note:**
>
> A default servlet context listener is included in the `coherence-rest.jar` that shuts down the cluster member during the REST application shutdown. The listener is registered as shown below. If the cluster member is not shut down, a variety of exceptions are thrown post shutdown.

```
<web-app>
   ...
   <listener>
      <listener-class>
         com.tangosol.coherence.rest.servlet.DefaultServletContextListener
      </listener-class>
   </listener>
   <servlet>
      <servlet-name>Coherence REST</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer
      </servlet-class>
      <init-param>
         <param-name>javax.ws.rs.Application</param-name>
         <param-value>
            com.tangosol.coherence.rest.server.ContainerResourceConfig
         </param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
```

```
    <servlet-mapping>
       <servlet-name>Coherence REST</servlet-name>
       <url-pattern>/*</url-pattern>
     </servlet-mapping>
       ...
</web-app>
```

5. Save the `web.xml` file to the `/WEB-INF/` directory.
6. Copy the `coherence-rest-config.xml` file to the `WEB-INF/classes` directory.
7. Copy your `coherence-cache-config.xml` file and `tangosol-coherence-override.xml` file to the `WEB-INF/classes` directory.
8. If you are using POF, copy the `pof-config.xml` file to the `WEB-INF/classes` directory. Make sure that the `pof-config.xml` file includes the location of the REST POF types. See Configuring REST Server Access to POF-Enabled Services.
9. Create a Web ARchive file (WAR) using the `jar` utility. For example, issue the following command from a command prompt at the root of the Web application directory:

   ```
   jar -cvf coherence_rest.war *
   ```

   The archive should contain the following files:

   ```
   /WEB-INF/web.xml
   /WEB-INF/classes/coherence-rest-config.xml
   /WEB-INF/classes/tangosol-coherence-override.xml
   /WEB-INF/classes/coherence-cache-config.xml
   /WEB-INF/lib/coherence.jar
   /WEB-INF/lib/coherence-rest.jar
   /WEB-INF/lib/coherence_dependencies
   ```

Deploying to a Servlet Container

Coherence REST can be deployed to any servlet container by packaging Coherence REST as a WAR file. See Packaging Coherence REST for Deployment. Refer to your vendors documentation for details on deploying WAR files. In addition, See the Jersey user guide for additional servlet container deployment options:

http://jersey.java.net/nonav/documentation/latest/user-guide.html#d4e194

Configuring REST Server Access to POF-Enabled Services

POF-enabled services must include the defined Coherence REST POF user types. The user types are defined in the `coherence-rest-pof-config.xml` file that is located in the `coherence-rest.jar` library and is automatically loaded at runtime.
To configure the REST default user types, edit the `pof-config.xml` file to include the `coherence-rest-pof-config.xml` POF configuration file. For example:

```
<pof-config>
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <include>coherence-rest-pof-config.xml</include>
      ...
   </user-type-list>
</pof-config>
```

32
Modifying the Default REST Implementation

You can change the default behavior of the Coherence REST implementation. This chapter includes the following sections:

Using the Pass-Through Resource

Coherence REST includes a resource implementation that enables pass-through access to caches. The resource allows static binaries such as graphics to be cached. The resource is implemented in the `PassThroughRootResource` class and is registered using the `PassThroughResourceConfig` class.

To use the pass-through resource in an application, modify the proxy service definition in the cache configuration file and add the fully qualified name of the `PassThroughResourceConfig` class within the `<resource-config>` element. The resource is mapped to a specific context path or the default path (`/`) if no context is defined. The following example registers the resource and uses `/cache` as the context path. Any cache resources that are defined in the `coherence-rest-config.xml` configuration file are prefixed with `/cache/` in the URL.

```
<proxy-scheme>
   <service-name>HttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         ...
         <resource-config>
            <context-path>/cache</context-path>
            <instance>
               <class-
name>com.tangosol.coherence.rest.server.PassThroughResourceConfig</class-name>
            </instance>
         </resource-config>
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Likewise, the `ContainerPassThroughResourceConfig` class, which is an extension of the `PassThroughResourceConfig` class, is used for container deployments of Coherence REST when pass-through is required. The resource is configured in the Web application deployment descriptor included in the Coherence REST application.

```
<web-app>
   ...
   <listener>
      <listener-class>
         com.tangosol.coherence.rest.servlet.DefaultServletContextListener
```

```
      </listener-class>
   </listener>
   <servlet>
      <servlet-name>Coherence REST</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer
      </servlet-class>
      <init-param>
         <param-name>javax.ws.rs.Application</param-name>
         <param-value>
            com.tangosol.coherence.rest.server.ContainerPassThroughResourceConfig
         </param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Coherence REST</servlet-name>
      <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
      ...
</web-app>
```

Using Custom Providers and Resources

Custom providers and resources can be created as required. This section demonstrates how to register custom providers, and how to override Coherence's default root resource.
The `com.tangosol.coherence.rest.server.DefaultResourceConfig` class supports package scanning, which can be used to register custom providers or resources. The following example demonstrates registering a custom provider and resource using package scanning:

```
public class MyResourceConfig  extends DefaultResourceConfig
   {
   public MyResourceConfig()
      {
      super("com.my.providers;com.my.resources");
      }
   }
```

As an alternative, the following example demonstrates how to override one or more of the `register` methods defined in the `DefaultResourceConfig` class in order to use custom providers, a custom root resource, or to add filters and filter factories.

> **Note:**
>
> Never override (unregister) Coherence default Providers without overriding the root resource class as well (the `DefaultRootResource` class depends on the default providers to provide the necessary dependencies and configuration).

```
public class MyResourceConfig  extends DefaultResourceConfig
    {
    protected void registerRootResource()
        {
        // remove if you don't want Coherence defaults to be registered
        super.registerRootResource();
        getClasses().add(MyRootResource.class);
        }

    protected void registerProviders()
```

```
        {
        // remove if you don't want Coherence defaults to be registered
        super.registerProviders();
        getSingletons().add(new MyProvider());
        }

    protected void registerContainerRequestFilters()
        {
        // remove if you don't want Coherencedefaults to be registered
        super.registerContainerRequestFilters();
        getContainerRequestFilters().add(new MyRequestFilter());
        }

    protected void registerContainerResponseFilters()
        {
        // remove if you don't want Coherence defaults to be registered
        super.registerContainerResponseFilters();
        getContainerResponseFilters().add(new MyResponseFilter());
        }

    protected void registerResourceFilterFactories()
        {
        // remove if you don't want Coherence defaults to be registered
        super.registerResourceFilterFactories();
        getResourceFilterFactories().add(new MyResourceFilterFactory());
        }
    }
```

Custom resource configuration classes are enabled in the cache configuration file by adding the fully qualified name of the class using the `<resource-config>` element within an HTTP acceptor configuration. The class is mapped to a specific context path or the default context path (`/`) if no context path is defined. Multiple resource configuration class definitions can be added and mapped to different context paths. The following example registers a custom resource called `MyResourceConfig` and maps it to the `/MyApplication` context path.

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         ...
            <resource-config>
               <context-path>/MyApplication</context-path>
               <instance>
                    <class-name>package.MyResourceConfig</class-name>
               </instance>
            </resource-config>
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Changing the Embedded HTTP Server

> **Note:**
>
> For production environments, Coherence REST recommends implementations only for the Netty HTTP server. Support for Simple HTTP server, the Jetty HTTP server, and the Grizzly HTTP server is deprecated.

Coherence REST uses Oracle's lightweight HTTP server by default to handle requests. However, the implementation is not recommended for production environments and is typically used during development and testing.

For production environments, Coherence includes implementations for the Netty HTTP server, the Simple HTTP server, the Jetty HTTP server, and the Grizzly HTTP server.

These servers are supported in Jersey. Refer to the Jersey documentation for instructions on integrating additional HTTP servers, which are beyond the scope of this documentation. http://jersey.java.net/

This section includes the following topics:

- Using Netty HTTP Server
- Using Simple HTTP Server
- Using Jetty HTTP Server
- Using Grizzly HTTP Server

Using Netty HTTP Server

Coherence REST provides a Netty HTTP server implementation (`com.tangosol.coherence.http.netty.NettyHttpServer`) that can be used instead of the default HTTP server. For more information on the Netty HTTP server, see:

https://netty.io/

The Netty server is enabled in the cache configuration file by adding the fully qualified name of the implementation as a value of the `<class-name>` element within an HTTP acceptor configuration. For example:

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         <class-name>com.tangosol.coherence.http.netty.NettyHttpServer</class-
name>
         ...
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Using Simple HTTP Server

Coherence REST provides a Simple HTTP server implementation (`com.tangosol.coherence.http.simple.SimpleHttpServer`) that can be used instead of the default HTTP server. For more information on the Simple framework see:

https://sourceforge.net/projects/simpleweb/

The Simple HTTP server is enabled in the cache configuration file by adding the fully qualified name of the implementation as a value of the `<class-name>` element within an HTTP acceptor configuration. For example:

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
```

```
   <acceptor-config>
      <http-acceptor>
         <class-name>com.tangosol.coherence.http.simple.SimpleHttpServer</class-name>
         ...
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Using Jetty HTTP Server

Coherence REST provides a Jetty HTTP server implementation (`com.tangosol.coherence.http.jetty.JettyHttpServer`) that can be used instead of the default HTTP server. For more information on the Jetty HTTP server, see:

http://www.eclipse.org/jetty/

The Jetty server is enabled in the cache configuration file by adding the fully qualified name of the implementation as a value of the `<class-name>` element within an HTTP acceptor configuration. For example:

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         <class-name>com.tangosol.coherence.http.jetty.JettyHttpServer</class-name>
         ...
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

Using Grizzly HTTP Server

Coherence REST provides a Grizzly 2 HTTP server implementation (`com.tangosol.coherence.http.grizzly.GrizzlyHttpServer`) that can be used instead of the default HTTP server. For more information on the Grizzly HTTP server see:

http://grizzly.java.net/

The Grizzly server is enabled in the cache configuration file by adding the fully qualified name of the implementation as a value of the `<class-name>` element within an HTTP acceptor configuration. For example:

```
<proxy-scheme>
   <service-name>ExtendHttpProxyService</service-name>
   <acceptor-config>
      <http-acceptor>
         <class-name>ccom.tangosol.coherence.http.grizzly.GrizzlyHttpServer</class-name>
         ...
      </http-acceptor>
   </acceptor-config>
   <autostart>true</autostart>
</proxy-scheme>
```

A
REST Configuration Elements

The Coherence REST configuration reference provides a detailed description of the REST configuration deployment descriptor.
This appendix includes the following sections:

REST Configuration File

The Coherence REST configuration deployment descriptor specifies the configuration for the REST implementation. The default name of the descriptor is `coherence-rest-config.xml` and must be found on the classpath. The name can be overridden using the `coherence.rest.config` system property. For example:

```
-Dcoherence.rest.config=MyConfig.xml
```

The REST configuration deployment descriptor schema is defined in the `coherence-rest-config.xsd` file. The XSD file is located in the root of the `coherence.jar` library and at the following Web URL:

http://xmlns.oracle.com/coherence/coherence-rest-config/1.2/coherence-rest-config.xsd

The `<rest>` element is the root element of the configuration file and includes the XSD and namespace declarations. For example:

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

<rest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-rest-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-rest-config
   coherence-rest-config.xsd">
```

> **Note:**
>
> - The schema located in the `coherence.jar` library is always used at run time even if the `xsi:schemaLocation` attribute references the Web URL.
> - The `xsi:schemaLocation` attribute can be omitted to disable schema validation.
> - When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, ensure that the deployment descriptor file is in ASCII format and is deployed into its run-time environment in the binary format.

REST Configuration Element Reference

The Coherence REST configuration element reference includes all non-terminal report file configuration elements. Each section includes instructions on how to use the element and also includes descriptions for all valid subelements.

REST Configuration Element Index

Table A-1 lists all non-terminal REST configuration elements.

Table A-1 REST Configuration Elements

Element	Used In
aggregator	aggregators
aggregators	rest
engine	query-engines
marshaller	resource
processor	processors
processors	rest
query	resource
query-engines	rest
resource	resources
resources	rest
rest	*root element*

aggregator

Used in: aggregators

Description

The `aggregator` element is used to define custom aggregators that are used to aggregate data in a cache. Each `aggregator` element must contain a single binding between a name and an aggregator class or aggregator factory class.

Elements

Table A-2 describes the subelements of the `aggregator` element.

Table A-2 aggregator Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a name to be used in a REST URL that is bound to an aggregator class or aggregator factory class.
`<class>`	Required	Specifies the fully qualified name of a custom aggregator class or custom aggregator factory class that is bound to a name. The class must implement the `com.tangosol.util.EntryAggregator` or `com.tangosol.coherence.rest.util.aggregator.AggregatorFactory` interfaces, respectively.

aggregators

Used in: rest

Description

The `aggregators` element contains any number of custom aggregator definitions.

Elements

Table A-3 describes the subelements of the `aggregators` element.

Table A-3 aggregators Subelements

Element	Required/ Optional	Description
<aggregator>	Required	Specifies a single binding between a name and an aggregator class or aggregator factory class.

engine

Used in: query-engines

Description

The `engines` element contains a single binding between a name and a query engine implementation class. Custom query engines must implement the `com.tangosol.coherence.rest.query.QueryEngine` and `com.tangosol.coherence.rest.query.Query` interfaces. Custom implementations can also extend the `com.tangosol.coherence.rest.query.AbstractQueryEngine` base class which provides useful methods for parsing query expressions and handling parameter bindings.

Elements

Table A-4 describes the subelements of the `engine` element.

Table A-4 engine Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a name for the query engine.
`<class-name>`	Required	Specifies the fully qualified name of the query engine implementation class.

marshaller

Used in: resource

Description

The marshaller element contains bindings between cache entry key/value classes and a marshaller class that is used to marshall and unmarshall instances of those classes.

Elements

Table A-5 describes the subelements of the `marshaller` element.

Table A-5 marshaller Subelements

Element	Required/ Optional	Description
`<media-type>`	Required	Specifies the name of the medium that is used to for marshalling. Coherence provides default implementations for XML and JSON data output.
`<class-name>`	Required	Specifies the fully qualified name of a custom class that implements the `com.tangosol.coherence.rest.io.Marshaller` interface. The implementation is used to marshall/unmarshall cache entry values that are stored in the cache. Marshallers are configured for each object type and media type.

processor

Used in: processors

Description

The `processor` element is used to define custom entry processors that are used to process data in a cache. Each `processor` element must contain a single binding between a name and the processor factory class.

Elements

Table A-6 describes the subelements of the `processor` element.

Table A-6 processor Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a name to be used in a REST URL that is bound to a processor factory class.
`<class-name>`	Required	Specifies the fully qualified name of a custom processor factory class that is bound to a name. The class must implement the `com.tangosol.coherence.rest.util.processor.ProcessorFactory` interface.

processors

Used in: rest

Description

The `processors` element contains any number of custom processor definitions.

Elements

Table A-7 describes the subelements of the `processors` element.

Table A-7 processors Subelements

Element	Required/ Optional	Description
<processor>	Required	Specifies a single binding between a name and a processor factory class.

query

Used in: resources

Description

The `query` element defines a named query for a resource. Named queries allow configured query expressions to be executed by name in the REST URL.

```
GET http://host:port/cacheName/namedQuery?param1=value1,param2=value2...
```

A named query definition consists of a binding between a query name and the query expression to execute. Multiple named queries can be configured for a resource. The query element supports the following attributes:

- `max-results` – Specifies how many results are returned to the client. Note that this attribute does not limit the number of entries that are returned from a cache. This value overrides the `max-results` attribute that is set on the `<resource>` element.
- `engine` – Specifies a query engine implementation that is responsible for executing query expressions against a cache. The default value if the attribute is not specified is `DEFAULT`, which indicates a query expression must be specified as a URL-encoded CohQL expression (the predicate part of CohQL). See query-engines.

Elements

Table A-8 describes the subelements of the `query` element.

Table A-8 query Subelements

Element	Required/ Optional	Description
<name>	Required	Specifies a name for the query.
<expression>	Required	Specifies a query expression that is bound to the query name.

query-engines

Used in: rest

Description

The query-engines element contains any number of custom query engine definitions. A query engine executes query expressions against a cache. Direct queries and named queries rely on an underlying query engine to perform their queries. A default query engine is provided for executing query expression that are specified as a URL-encoded CohQL expression (the predicate part of CohQL). However, custom query engines can be defined as required.

Elements

Table A-9 describes the subelements of the query-engines element.

Table A-9 query-engines Subelements

Element	Required/ Optional	Description
<engine>	Required	Specifies a single binding between a name and a query engine implementation class.

resource

Used in: resources

Description

The resource element provides the metadata that is used to marshall and unmarshall cache entries. The metadata includes a single binding between a cache name and cache entry key and value classes.

The following attributes are available:

- name – Specifies an alias for the <cache-name> element when the name is not ideal for the REST URL path segment. The value defaults to the value of the <cache-name> element if a value is not specified.
- max-results – Specifies how many results are returned to the client. Note that this attribute does not limit the number of entries that are returned from a cache. This value is overridden if a max-results attribute is also defined within the <query> or <direct-query> element.

Elements

Table A-10 describes the subelements of the resource element.

Table A-10 resource Subelements

Element	Required/ Optional	Description
`<cache-name>`	Required	Specifies the name of the cache exposed by this resource. The cache must be defined in the cache configuration file.
`<key-class>`	Optional	Specifies the type of the entry keys stored in this cache.
`<value-class>`	Optional	Specifies the type of the entry values stored in this cache.
`<key-converter>`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.coherence.rest.KeyConverter` interface. The class is used to convert cache entry keys to string and string representations of the keys that are used in the REST URL into an appropriate object instance that can be used to access cache entries. The `com.tangosol.coherence.rest.DefaultKeyConverter` class is used by default if no value is provided. The default class offers reasonable to string and from string conversions for Java primitives, dates, and UUIDs.
<marshaller>	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.coherence.rest.io.Marshaller` interface. The class is used to marshall/unmarshall cache entry values that are stored in a cache. Coherence provides default implementations for XML and JSON data output.
<query>	Optional	Specifies the configuration information for named queries, which allow configured query expressions to be executed by name in the REST URL.
`<direct-query>`	Optional	Specifies the configuration information for direct queries, which allow query expressions to be included in the REST URL as the value of the parameter `q`. `GET http://host:port/cacheName?q=query` The following attributes are available: • `enabled` – Specifies whether a resource supports direct queries. Valid values are `true` and `false`. The default value is `false`. • `max-results` – Specifies how many results are returned to the client. Note that this attribute does not limit the number of entries that are returned from a cache. This value overrides the `<resource>` element's `max-results` attribute. • `engine` – Specifies a query engine implementation that is responsible for executing query expressions against a cache. The default value if the attribute is not specified is `DEFAULT`, which indicates a query expression must be specified as a URL-encoded CohQL expression (the predicate part of CohQL). See query-engines.

resources

Used in: rest

Description

The resources element contains any number of resource definitions. A resource definition provides the metadata that is used to marshall and unmarshall cache entries.

Elements

Table A-11 describes the subelements of the `resources` element.

Table A-11 resources Subelements

Element	Required/ Optional	Description
<resource>	Required	Specifies a single binding between a cache name and cache entry key and value classes.

rest

root element

Description

The rest element is the root element of the `coherence-rest-config.xml` file which is used to configure the Coherence REST implementation. The implementation uses REST Web services to allow remote clients to access data in the cluster over HTTP and does not require the use of POF serialization.

Elements

Table A-12 describes the subelements of each `rest` element.

Table A-12 rest Subelements

Element	Required/ Optional	Description
<resources>	Optional	Specifies any number of resource definitions that provide the metadata that is used to marshall and unmarshall cache entries.
<processors>	Optional	Specifies any number of custom processor definitions that are used to process data in a cache.
<aggregators>	Optional	Specifies any number of custom aggregator definitions that are used to aggregate data in a cache.
<query-engines>	Optional	Specifies any number of custom query engine definitions. A query engine is responsible for executing queries.

B

Integrating with F5 BIG-IP LTM

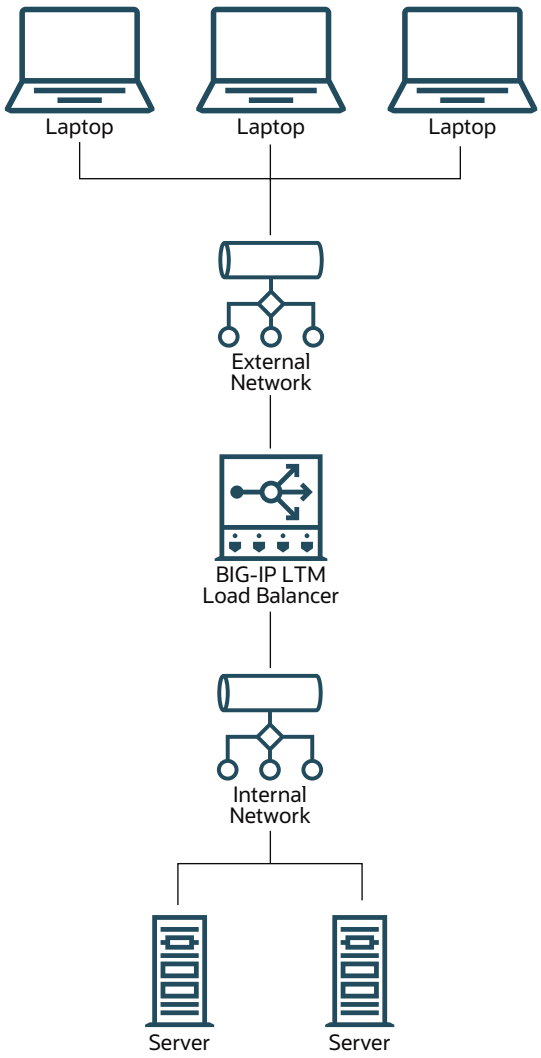
You can use the F5 BIG-IP Local Traffic Manager (LTM) hardware load balancer to balance Coherence*Extend client connections. Instructions are also included to use the BIG-IP system to off load SSL processing.
The instructions are specific to using the BIG-IP Configuration Utility as it pertains to Coherence*Extend setup. Refer to the Help included with the utility for complete usage instructions. In addition, the instructions were created based on BIG-IP LTM 10.2.1 and may not be accurate for future releases of BIG-IP LTM.

This appendix includes the following sections:

Basic Concepts

The F5 BIG-IP LTM is a hardware device that sits between one or more computers running Coherence*Extend clients (client tier) and one or more computers running Coherence*Extend proxy servers (proxy tier). The LTM spreads client connections across multiple clustered proxy servers using a broad range of techniques to secure, optimize, and load balance application traffic.
Figure B-1 shows a conceptual view of the BIG-IP system that is setup between external network clients and internal network servers.

Figure B-1 Conceptual View of F5 BIG-IP LTM



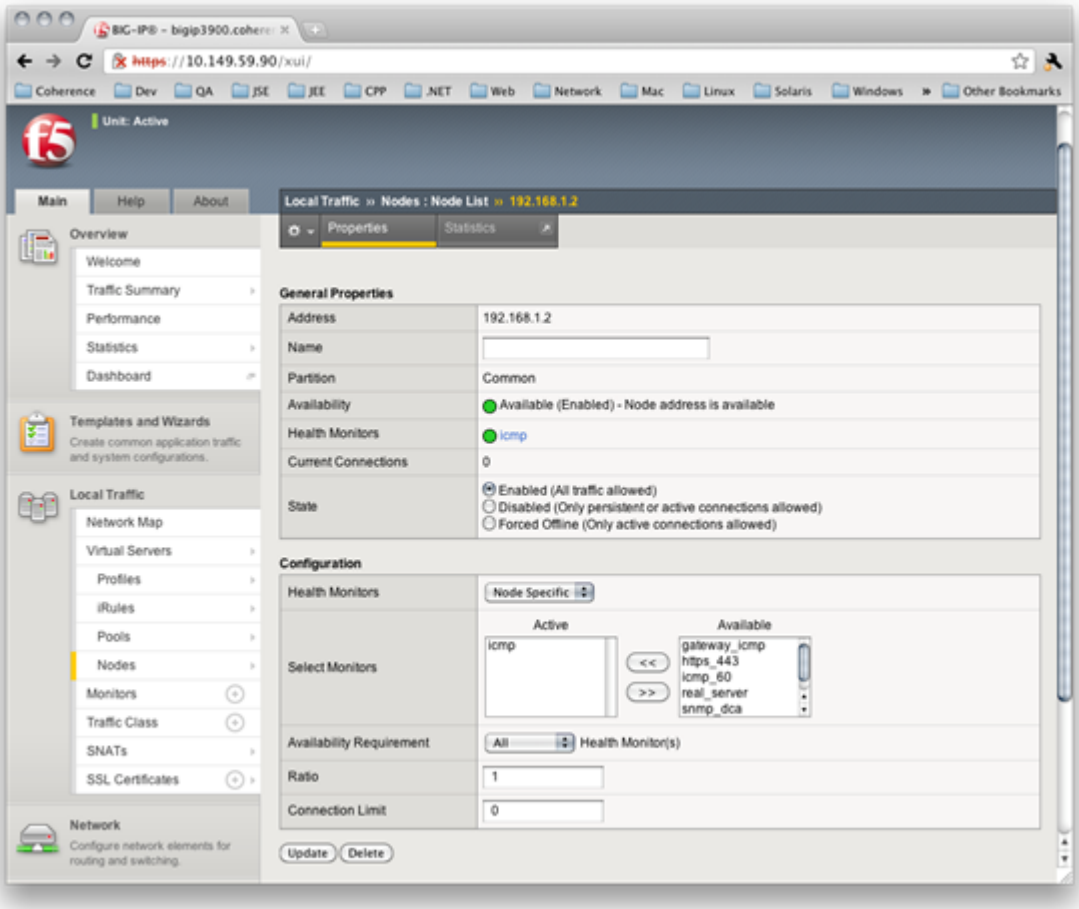
Creating Nodes

A node is a logical object on the BIG-IP system that identifies the IP address of a physical resource on the network. For Coherence*Extend, configure a node for each computer on the internal network that hosts one or more proxy servers.
To create a node:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation pane, expand **Local Traffic** and click **Nodes**.
3. In the upper-right corner of the screen, click **Create**. The New Node screen displays.
4. For the Address setting, type the IP address of the node.
5. Specify, retain, or change each of the other settings.
6. Click **Finished**.

Figure B-2 shows an example node configuration.

Figure B-2 Example Node Configuration



Configuring a Load Balancing Pool

A load balancing pool is a group of logical devices, such as proxy servers, that receive and process traffic. Instead of sending client traffic to the destination IP address specified in the client request, the BIG-IP system sends the request to any of the servers that are members of that pool. This helps efficiently distribute the load on your server resources.
When you create a pool, you assign pool members to the pool. A pool member is a logical object that represents a server endpoint on the network. For Coherence*Extend, create a pool member for each proxy server JVM running on your proxy tier computers.

The specific pool member to which the BIG-IP system chooses to send the request is determined by the load balancing method that you have assigned to that pool. A load balancing method is an algorithm that the BIG-IP system uses to select a pool member for processing a request. For example, the default load balancing method is Round Robin, which causes the BIG-IP system to send each incoming request to the next available member of the pool, thereby distributing requests evenly across the servers in the pool.

This section includes the following topics:

- Creating a Load Balancing Pool
- Adding a Load Balancing Pool Member

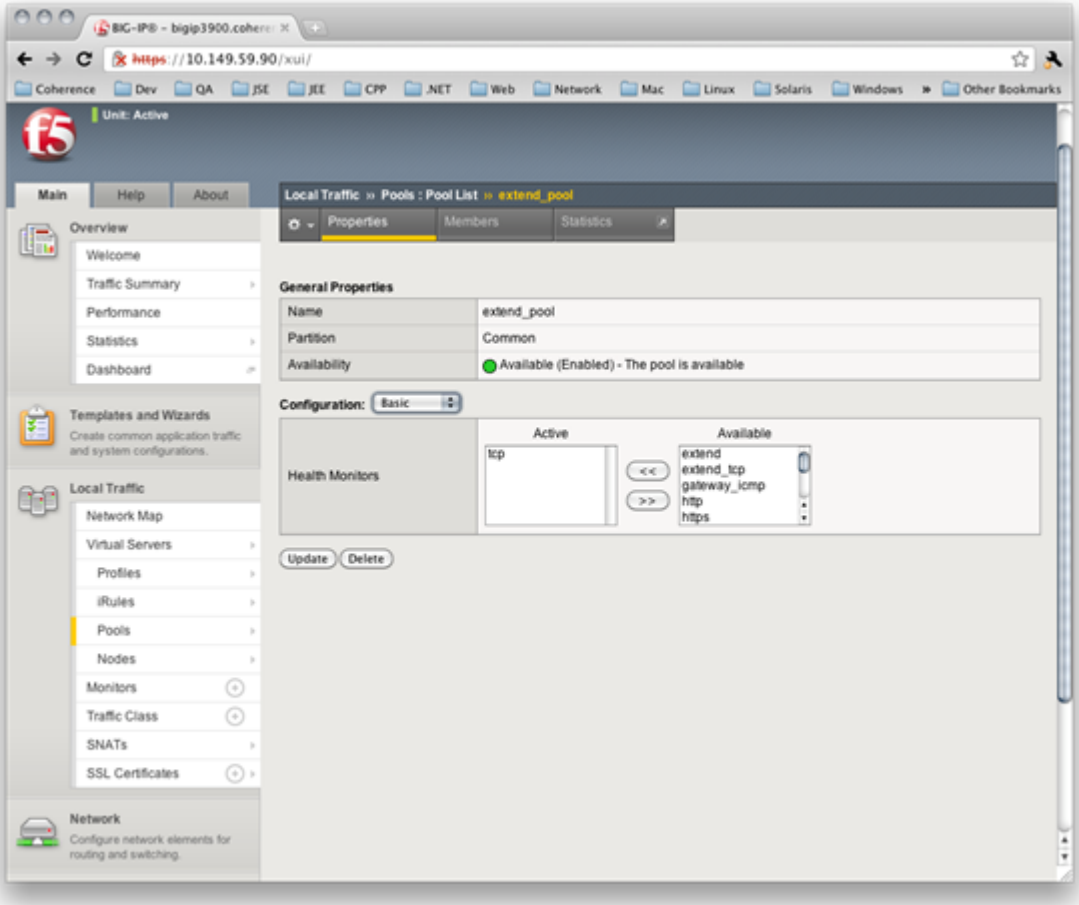
Creating a Load Balancing Pool

To create a load balancing pool:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation pane, expand **Local Traffic** and click **Pools**. The Pools screen displays.
3. In the upper-right corner of the screen, click **Create**. The New Pool screen displays.
4. From the Configuration list, select **Advanced**.
5. For the Name setting, type a name for the pool.
6. Specify, retain, or change each of the other settings.
7. Click **Finished**.

Figure B-3 demonstrates an example pool configuration.

Figure B-3 Example Pool Configuration



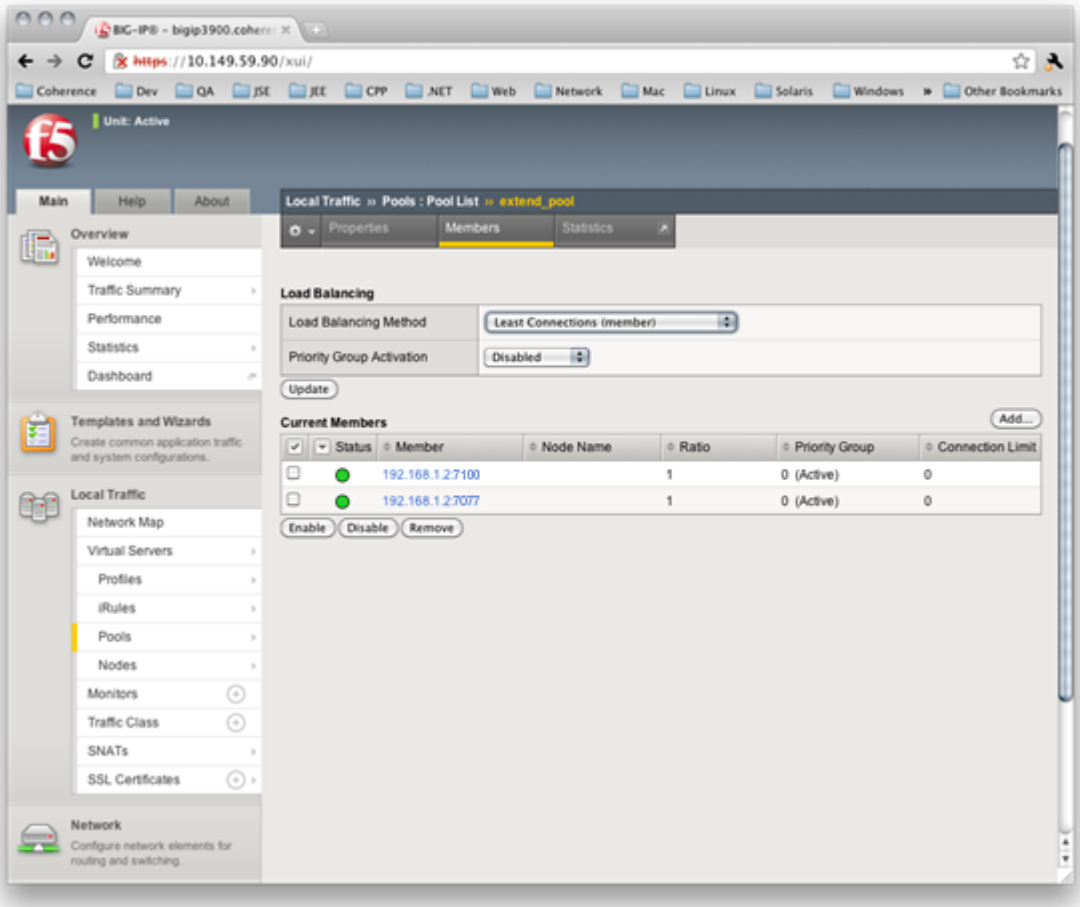
Adding a Load Balancing Pool Member

To add pool members to load balancing pool:

1. From the Members tab, click the number shown. This lists the existing members of the pool.
2. In the right side of the screen, click **Add**. The New Pool Member screen displays.
3. In the Address box, select **Node List** and select an IP address.
4. In the Service Port box, type the port number on which the corresponding proxy server is listening.
5. Retain or change each of the other settings.
6. Click **Finished**.

Figure B-4 shows an example pool configuration. It shows two proxy server pool members running on the previously created node and listening on ports 7100 and 7077, respectively. Additionally, the pool is configured to use a Least Connections load balancing policy.

Figure B-4 Example Pool Members



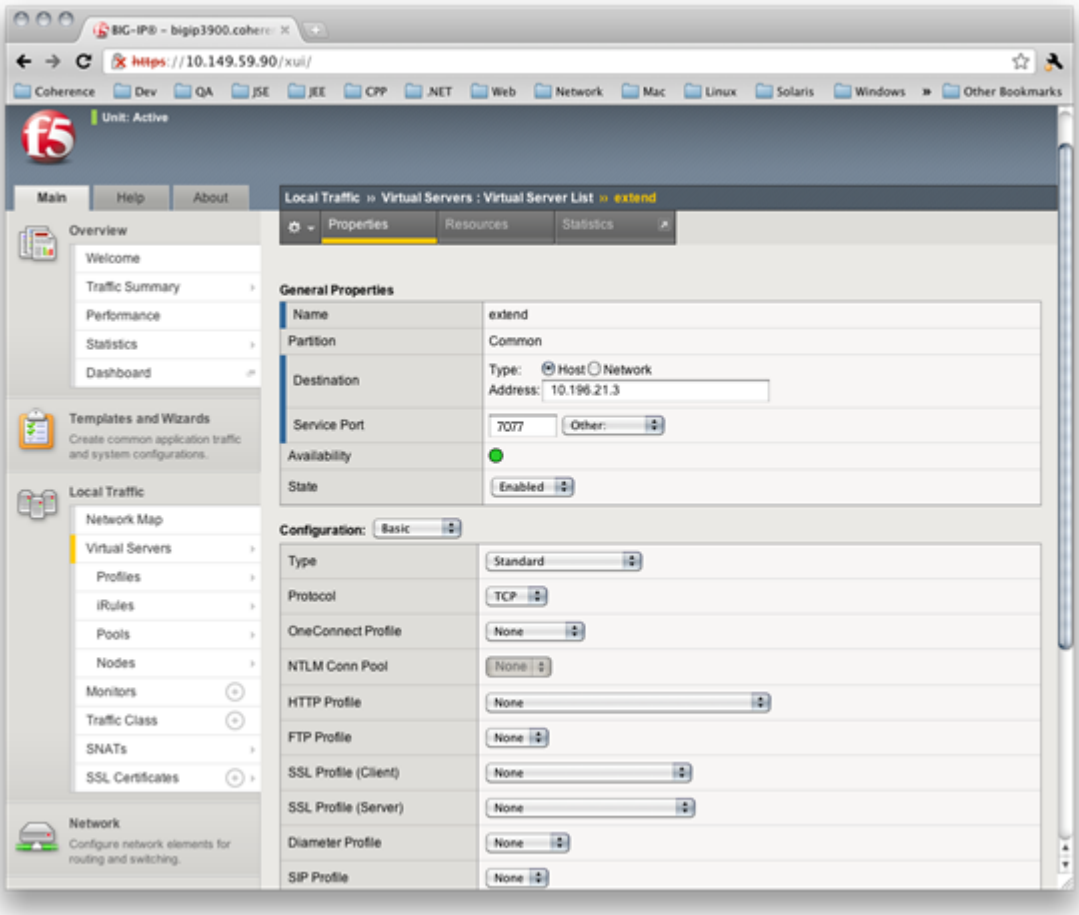
Configuring a Virtual Server

A virtual server is a traffic-management object on the BIG-IP system that is represented by an IP address and port. Clients on an external network can send application traffic to a virtual server, which then directs the traffic according to your configuration instructions. The main purpose of a virtual server is often to balance traffic load across a pool of servers on an internal network. Virtual servers increase the availability of resources for processing client requests. For Coherence*Extend, you should configure a virtual server that directs traffic to the pool of proxy servers that you configured earlier.
To create a virtual server:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation screen, expand **Local Traffic** and click **Virtual Servers**. The Virtual Servers screen displays.
3. From the upper right portion of the screen, click **Create**. The New Virtual Server screen displays.
4. In the Name box, type a name for the virtual server.
5. In the Destination box, assign an external IP address on the BIG-IP device and in the Service Port box, specify a listen port. This is the IP address and port to which Coherence*Extend clients connect.
6. From the SNAT Pool list, select **Automap**.
7. Select the pool created earlier in the Default Pool drop-down box.
8. Retain or change each of the other settings.
9. Click **Finished**.

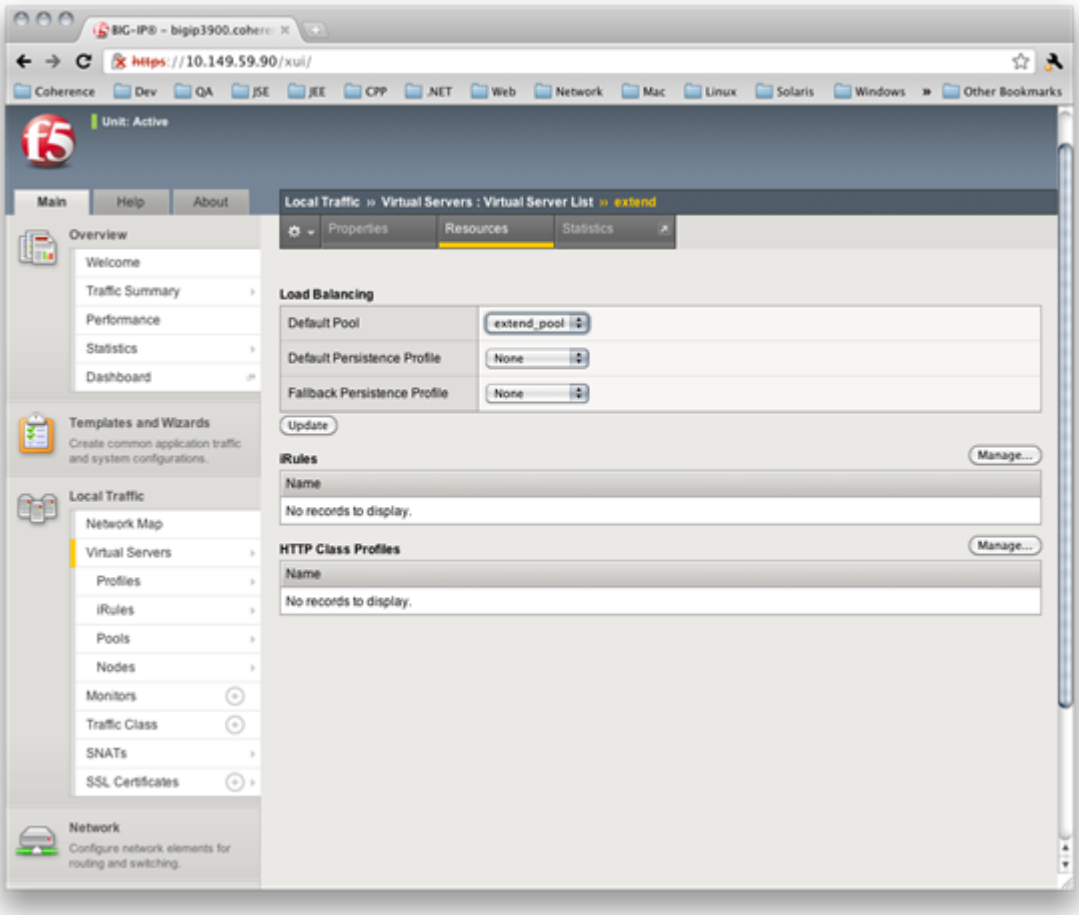
Figure B-5 shows an example virtual configuration that listens for TCP/IP connections on `10.196.21.3:7077`.

Figure B-5 Example Virtual Server



Additionally, this virtual server directs traffic to the configured pool as shown in Figure B-6.

Figure B-6 Example Virtual Server Using a Configured Pool



Configuring Coherence*Extend to Use BIG-IP LTM

Coherence*Extend must be configured to use a BIG-IP LTM virtual server. The configuration must be completed both on the cluster side and the client side cache configuration files.
To configure Coherence*Extend to use BIG-IP LTM:

1. Open the proxy server's cache configuration file.
2. Edit the proxy scheme definition and specify a client load balancing strategy by entering `client` within the `<load-balancer>` element. For example:

```
<proxy-scheme>
   <service-name>ExtendTcpProxyService</service-name>
   <load-balancer>client</load-balancer>
   <autostart>true</autostart>
</proxy-scheme>
```

3. Save and close the proxy server's cache configuration file. Repeat step 2 for additional proxy servers.
4. Open the client's cache configuration file.

5. In the `<remote-cache-scheme>` element, list the IP address and port of the BIG-IP virtual server. See Configuring a Virtual Server. In addition, specify a `<heartbeat-interval>` element within the `<outgoing-message-handler>` element. This causes the client to periodically send a heartbeat message over its TCP/IP connection at the configured time interval. This is required to prevent the BIG-IP device from disconnecting idle clients. For example:

```
<remote-cache-scheme>
   <scheme-name>extend-direct</scheme-name>
   <service-name>ExtendTcpCacheService</service-name>
   <initiator-config>
      <tcp-initiator>
         <remote-addresses>
            <socket-address>
               <address>10.196.21.3</address>
               <port>7077</port>
            </socket-address>
         </remote-addresses>
      </tcp-initiator>
      <outgoing-message-handler>
         <heartbeat-interval>5s</heartbeat-interval>
      </outgoing-message-handler>
   </initiator-config>
</remote-cache-scheme>
```

6. Save and close the client's cache configuration file.

Using Advanced Health Monitoring

A health monitor helps ensure that a server is in an operational state and able to receive traffic. The BIG-IP system contains many different preconfigured health monitors that you can associate with pools, depending on the type of traffic you want to monitor.
For Coherence*Extend, you can use a TCP health monitor to monitor a pool of proxy servers. This type of monitor marks a proxy server up if the BIG-IP device can establish a TCP/IP connection with the proxy server. While this is a fairly decent indication that a proxy server is functional, it does not guarantee that the proxy server can actually process client traffic. For more detailed monitoring, BIG-IP enables you to create custom health monitors that send a Coherence*Extend ping request to a proxy server and validate that an appropriate response is returned. This ensures that the proxy server is up and able to process client traffic.

> **Note:**
>
> BIG-IP LTM monitors do not support SSL over TCP. Health monitoring checks, such as ping, are sent as clear text. To ensure all communication with a proxy server is secure, use SSL offloading. For more information about these settings, see Enabling SSL Offloading. You can also use HTTP/S health checks as described in Enabling HTTP/S Health Monitoring.

This section includes the following topics:

- Creating a Custom Health Monitor to Ping Coherence
- Manually Creating a Custom Health Monitor to Ping Coherence
- Associating a Custom Health Monitor With a Load Balancing Pool
- Enabling HTTP/S Health Monitoring

Creating a Custom Health Monitor to Ping Coherence

To create a custom Coherence*Extend health monitor that sends a Coherence*Extend ping request to a proxy server to ensure that it is operational:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation pane, expand **Local Traffic** and click **Monitors**. The Monitors screen displays.
3. In the upper-right corner of the screen, click **Create**. The New Monitor screen displays.
4. Enter a name for the monitor in the Name box.
5. Select **TCP** in the Type drop-down box.
6. Enter the following in the Send String box:

   ```
   \x07\x00\x03\x00\x00\x42\x00\x40
   ```

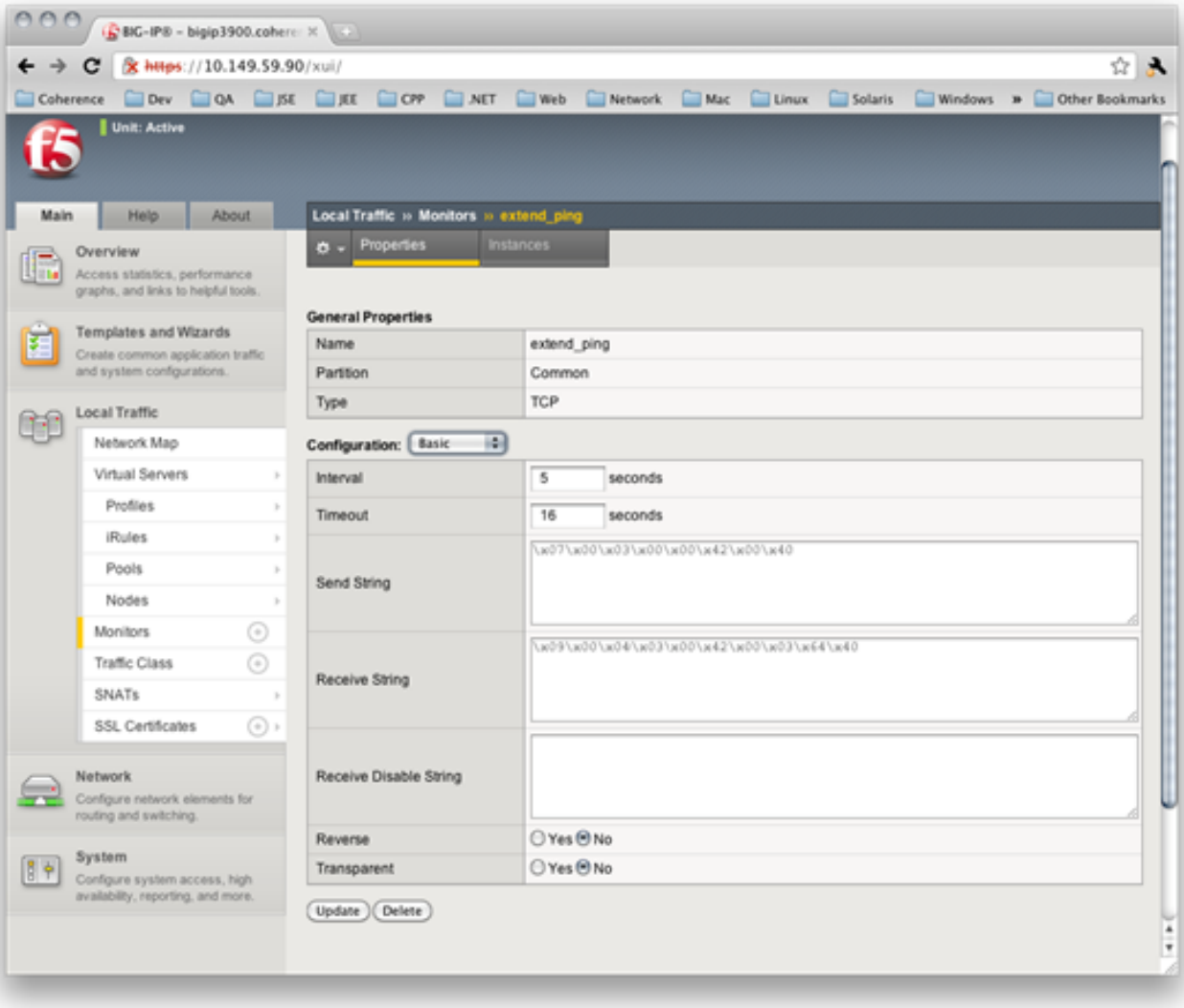
7. Enter the following in the Receive String box:

   ```
   \x09\x00\x04\x03\x00\x42\x00\x03\x64\x40
   ```

8. Click **Finished**.

Figure B-7 shows an example custom Coherence*Extend health monitor configuration.

Figure B-7 Example Coherence*Extend Ping Health Monitor



Manually Creating a Custom Health Monitor to Ping Coherence

Solutions that use BIG-IP versions prior 10.2.1 must manually configure an external health monitor. To do so, create an executable shell script called `extend_ping` in the `/usr/bin/monitors` directory of the BIG-IP device with the following contents:

```
#! /bin/bash
################################################################################
###  EXTERNAL MONITOR FOR COHERENCE*EXTEND
###   INPUTS:
###     $1    The IPV6 formatted IP address of the pool member to test
###     $2    The port number of the pool member to test
###     $3+   White space delimited parms as listed in the monitor "args"
###   OUTPUTS:
###     If null is returned, the member is "down"
###     If any string of one or more characters is returned, the member is "up"
################################################################################
```

```
IP=${1:-"127.0.0.1"}
IP=${IP##*:} # This removes the leading ::ffff:
PORT=${2:-"80"}
TIMEOUT=${3:-1}
SLEEP=${4:-1}

PID_FILE="/var/run/extend_ping.$IP.$PORT.pid"
HEX_REQUEST="0700030000420040"
HEX_RESPONSE="090004030042000036440"

###
### Terminate existing process, if any
###
if [ -f $PID_FILE ]
then
   kill -9 `cat $PID_FILE` > /dev/null 2>&1
fi
echo "$$" > $PID_FILE

###
###  Ping the server and return a user friendly result
###
RESULT=`/bin/echo "$HEX_REQUEST" | /usr/bin/xxd -r -p | /usr/bin/nc -i \
    $SLEEP -w $TIMEOUT $IP $PORT | /usr/bin/xxd -p | /bin/grep \
    "$HEX_RESPONSE" 2> /dev/null`

if [ "$RESULT" != "" ] ; then
   /bin/echo "$IP:$PORT is \"UP\""
fi

rm -f $PID_FILE
```

To configure BIG-IP to use the `extend_ping` script:

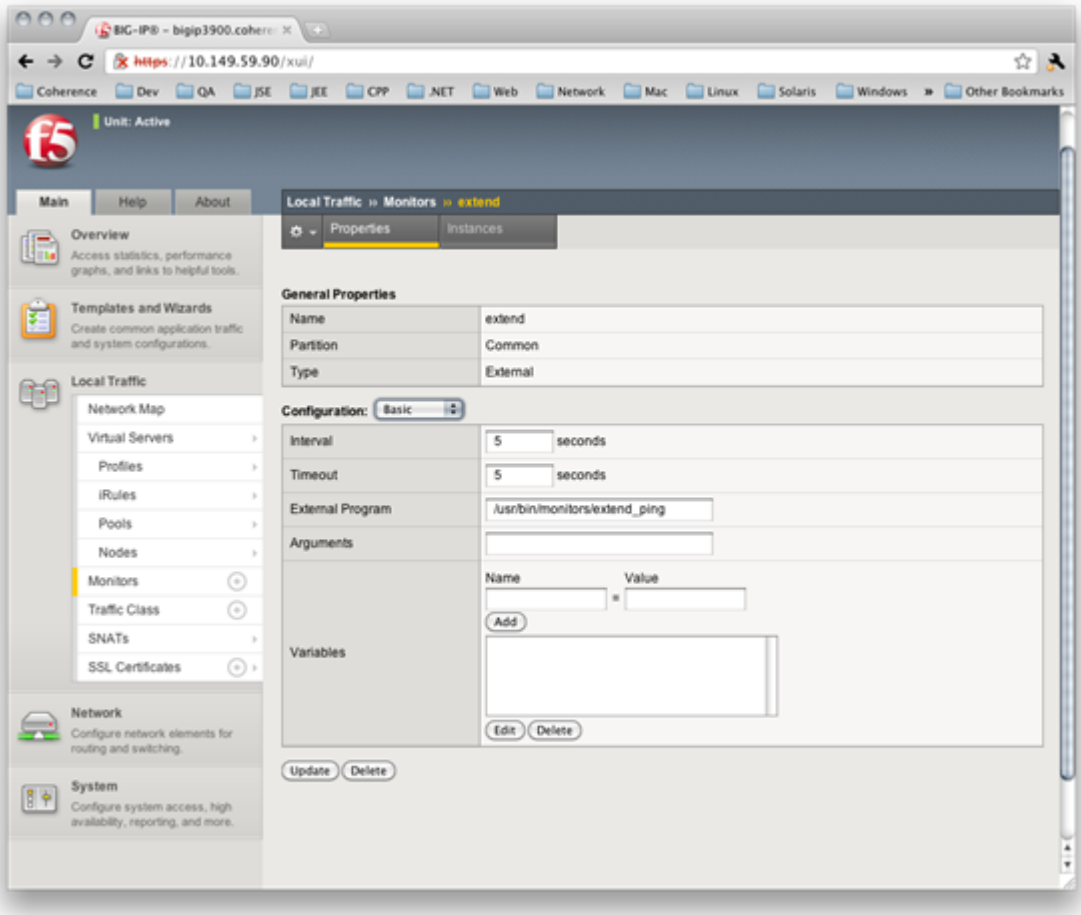
1. From the Main tab of the navigation pane, expand **Local Traffic** and click **Monitors**. The Monitors screen displays.
2. In the upper-right corner of the screen, click **Create**. The New Monitor screen displays.
3. Enter a name for the monitor in the Name box.
4. Select **External** in the Type drop-down box.
5. Enter the following in the External Program box:

   ```
   /usr/bin/monitors/extend_ping
   ```

6. Click **Finished**.

Figure B-8 shows an example external Coherence*Extend health monitor configuration.

Figure B-8 Example Coherence*Extend Health Monitor Implemented in a Shell Script



Associating a Custom Health Monitor With a Load Balancing Pool

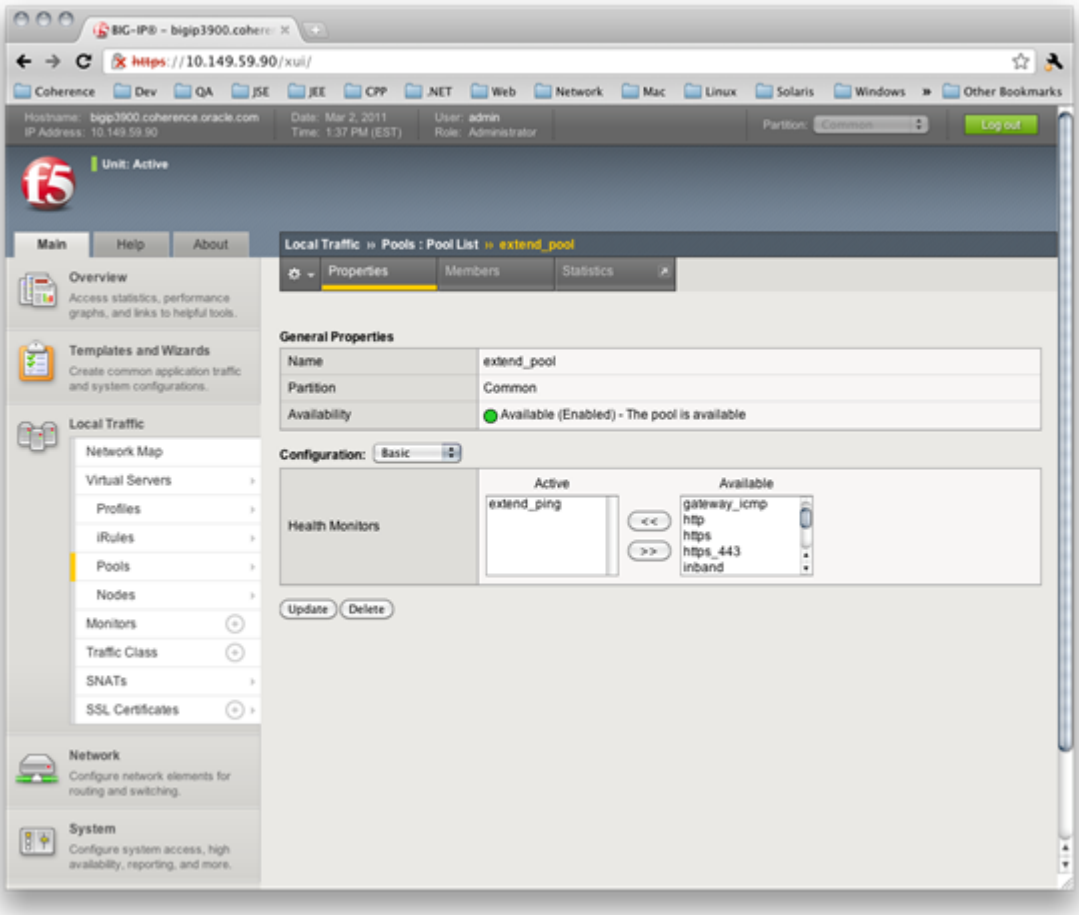
Custom health monitors must be associated with a load balancing pool. After creating a custom Coherence*Extend monitor, associate it with the Coherence*Extend load balancing pool.

To associate a custom health monitor with a load balancing pool:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation pane, expand **Local Traffic** and click **Pools**. The Pools screen displays.
3. Click the name of your Coherence*Extend pool. The Pool screen displays.
4. Select the name of your custom Coherence*Extend health monitor in the Health Monitors box.
5. Click **Update**.

Figure B-9 shows a Coherence*Extend pool that uses a custom health monitor.

Figure B-9 Associating a Coherence*Extend Pool With a Custom Health Monitor



Enabling HTTP/S Health Monitoring

In certain circumstances, such as when using SSL, it is not possible to use the custom health monitoring as described in the sections above. This is because the BIG-IP LTM monitors do not support SSL over TCP. In this case, you can use the HTTP/S health checks as described in Enabling HTTP Health Checks to check the health of a proxy member endpoint.

To enable the HTTP/S health monitoring on your proxy server:

1. Change the start-up class for proxy from `com.tangosol.net.DefaultCacheServer` to `com.tangosol.net.Coherence`.
2. Choose a port for each proxy server to be the health check port. For example, port 6676.
3. Set the system properties to `-Dcoherence.health.http.port=6676` (or chosen port) on each proxy.

   As the health check port does not contain any content, it is safe to use HTTP. However, if you have enabled SSL then you must also set `-Dcoherence.health.http.provider=sslSockerProvider` to point to a SSL socket provider in your override file.
4. Start the proxy server.

A message showing the health proxy status is displayed.

```
<Info> (thread=Proxy:$SYS:HealthHttpProxy:HttpAcceptor, member=1):
HttpAcceptor now listening for connections on 0:0:0:0:0:0:0:0:6676
```

You can then add a Monitor for HTTP/S on your F5 BIG-IP, which will use the URL `http://hostname:6676/live` to check the health of each proxy server. This endpoint will return HTTP code 200 if the endpoint is live and ready or HTTP code 503 if not.

Using SSL Offloading

Coherence*Extend can be configured to use SSL to secure communication between client and proxy server processes. However, this confidentially comes at a price. Specifically, enabling SSL dramatically increases CPU utilization in the proxy tier and increases the latency of each request. BIG-IP SSL Acceleration frees up proxy servers from the difficult task of encrypting and decrypting data secured for privacy reasons. CPU-intensive decryption is migrated onto a high-performance device designed to handle SSL transactions more efficiently. This approach is known as SSL offloading.
This section includes the following topics:

- Enabling SSL Offloading
- Import the Server's SSL Certificate and Key
- Create the Client SSL Profile
- Associate the Client SSL Profile

Enabling SSL Offloading

The following steps are required to enable SSL offloading and should be completed in the order presented:

1. Enable SSL in the Coherence*Extend client cache configuration file. See Using SSL to Secure Extend Client Communication in *Securing Oracle Coherence*.
2. Import the Server's SSL Certificate and Key
3. Create the Client SSL Profile
4. Associate the Client SSL Profile

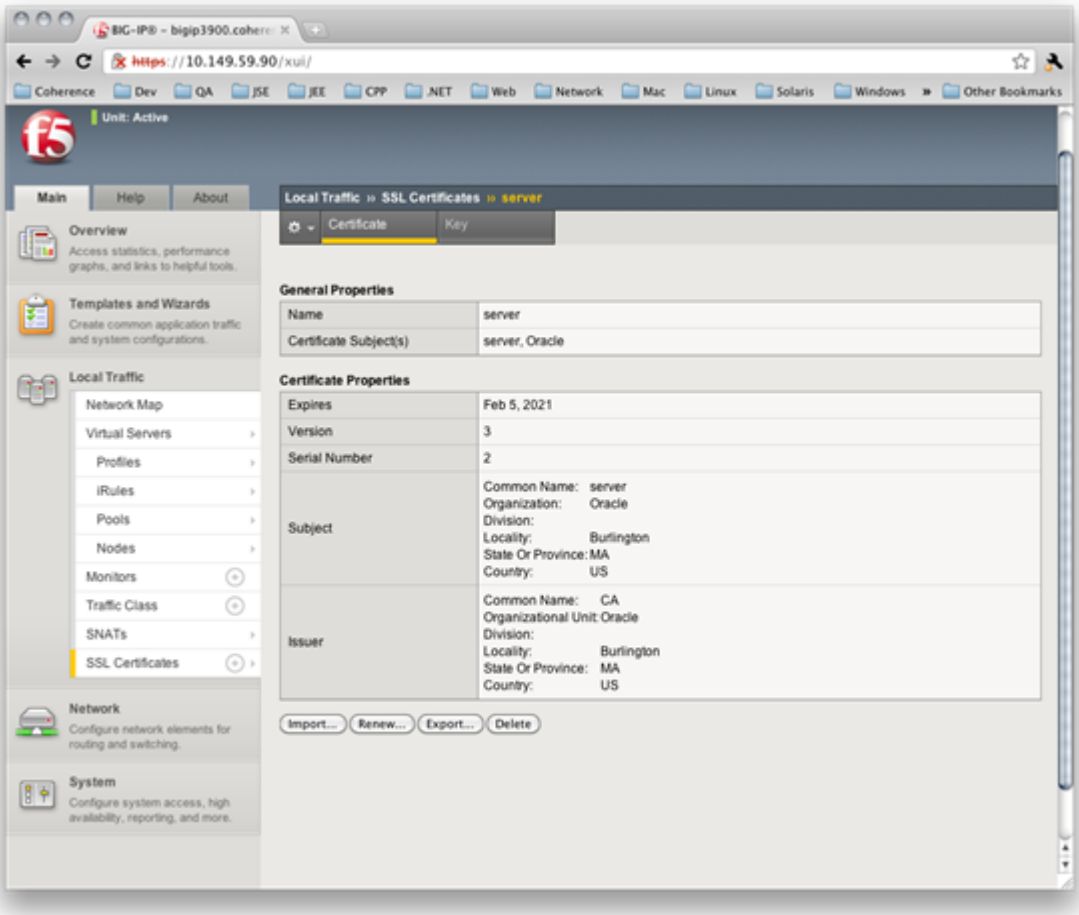
Import the Server's SSL Certificate and Key

To import the server's SSL certificate and key to the BIG-IP system:

1. Log into the BIG-IP Configuration Utility.
2. From the Main tab of the navigation pane, expand **Local Traffic** and hover over SSL Certificates then select **Import**. The SSL Certificate screen displays.
3. From the Import Type drop-down box, select **PKCS12**.
4. Enter a name for the certificate in the Certificate Name box.
5. Click **Choose File** and browse to the server's PKCS12 file.
6. Enter the password for the PKCS12 file.
7. Click **Import**.

Figure B-10 shows an example server SSL certificate configuration:

Figure B-10 Example SSL Certificate Configuration in BIG-IP System



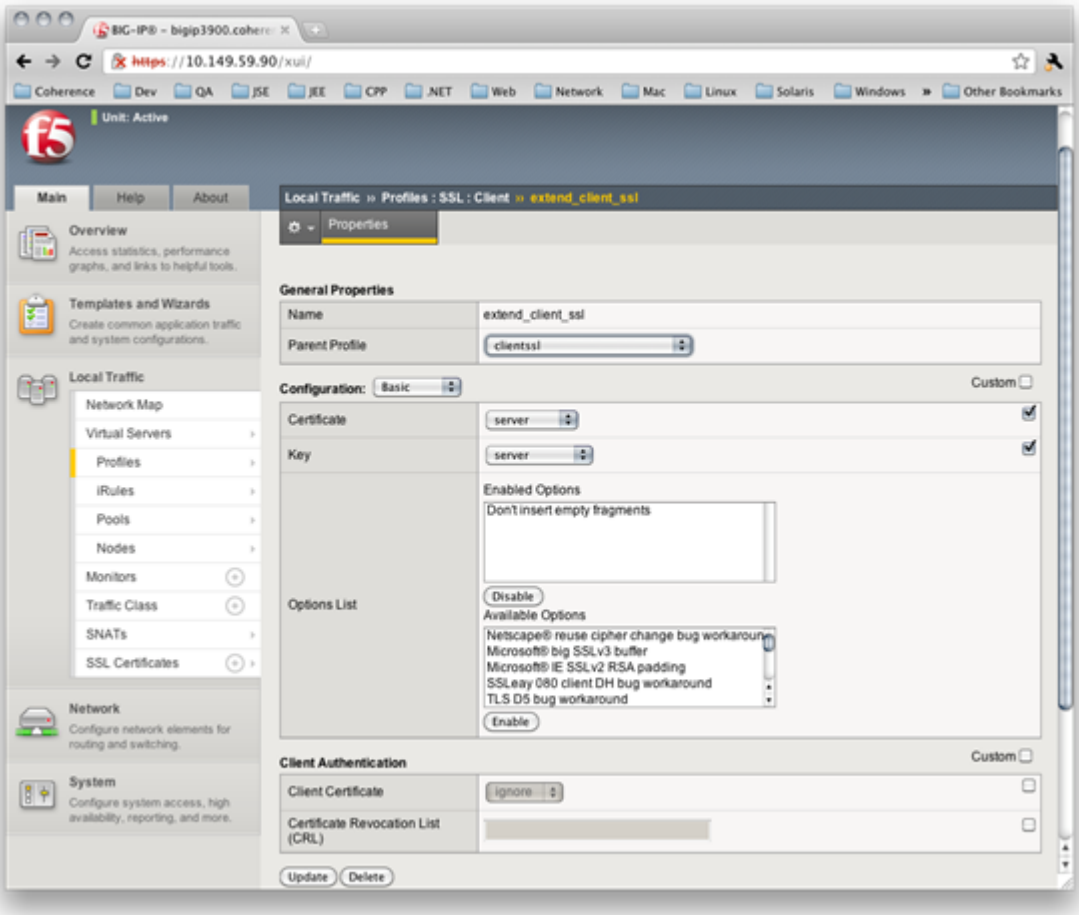
Create the Client SSL Profile

To create the client SSL profile:

1. From the Main tab of the navigation pane, expand **Local Traffic** and hover over Profiles then SSL and select **Client**. The Client SSL Profiles screen displays
2. In the upper-right corner of the screen, click **Create**. The New Client SSL profile screen displays.
3. Enter a name for the client SSL profile in the Name box.
4. Click the **Custom** check box on the right.
5. Select the name of the server certificate that you imported earlier in both the Certificate and Key drop-down boxes.
6. Click **Finished**.

Figure B-11 shows an example client SSL profile configuration:

Figure B-11 Example SSL Profile Configuration



Associate the Client SSL Profile

To modify the Coherence*Extend virtual server configuration to use the client SSL profile:

1. From the Main tab of the navigation screen, expand **Local Traffic** and click **Virtual Servers**. The Virtual Servers screen displays.
2. Click the name of the virtual server.
3. Select the name of the client SSL profile in the SSL Profile (Client) drop-down box.
4. Click **Update**.

Figure B-12 shows an example virtual server configuration that uses a client SSL profile:

Figure B-12 Example Virtual Server Configuration That Includes a Client SSL Profile