

Oracle® Insurance Policy Administration

Coherence

Version 9.4.0.0

Part Number: E18894-01

June 2011

Copyright © 2009, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Where an Oracle offering includes third party content or software, we may be required to include related notices. For information on third party notices and the software and related documentation in connection with which they need to be included, please contact the attorney from the Development and Strategic Initiatives Legal Group that supports the development team for the Oracle offering. Contact information can be found on the Attorney Contact Chart.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Table of Contents

Overview	4
Locating Configurable Coherence Files	5
Caching.....	6
CacheProvider and ICacheHelper Interfaces.....	7
CacheProcessor Abstract Class.....	7
Accessing Cache	7
CoherenceCacheProviderUtl and CacheUtl.....	9
Three CoherenceCacheHelper Classes.....	9
Coherence Cluster Configuration	10
Coherence Configuration for Caching	11
Workload Management	14
Tasks	14

OVERVIEW

Oracle Coherence provides Oracle Insurance Policy Administration (OIPA) with replicated and distributed (partitioned) data management and caching services on top of a reliable, highly scalable peer-to-peer clustering protocol. Coherence has no single points of failure. It automatically and transparently fails over and redistributes its clustered data management services when a server becomes inoperative or is disconnected from the network. When a new server is added, or when a failed server is restarted, it automatically joins the cluster and Coherence fails back services to it; transparently redistributing the cluster load. Coherence includes network-level fault tolerance features and transparent soft re-start capability to enable servers to self-heal.

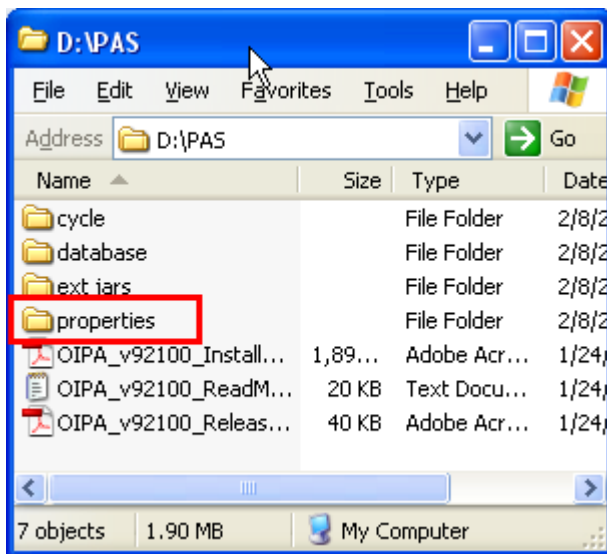
There are two different scenarios where OIPA employs Oracle Coherence. They are as follows:

1. Caching
2. Cycle

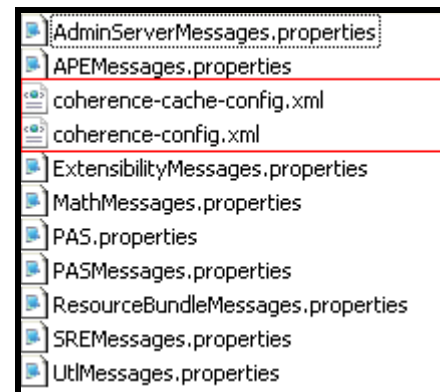
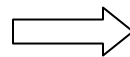
LOCATING CONFIGURABLE COHERENCE FILES

To locate the coherence configuration files for caching and application profiling, open the properties file in the OIPA deployable for E-Delivery. For caching in cycle, the files are located in each one of the cycle sub folders.

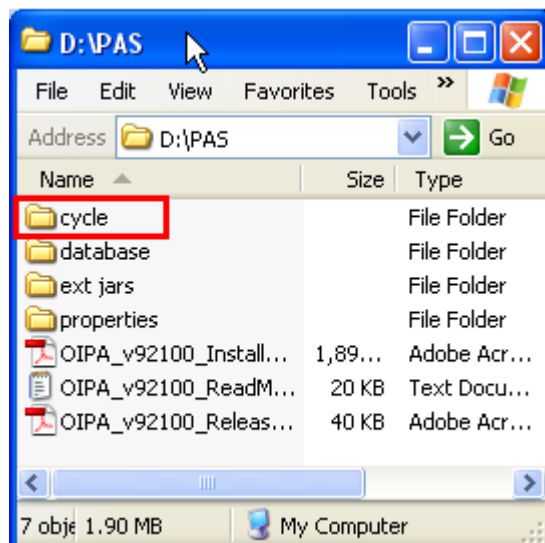
The two coherence configuration files are called coherence-cache-config.xml and coherence-config.xml. These files will be explained later in the document.



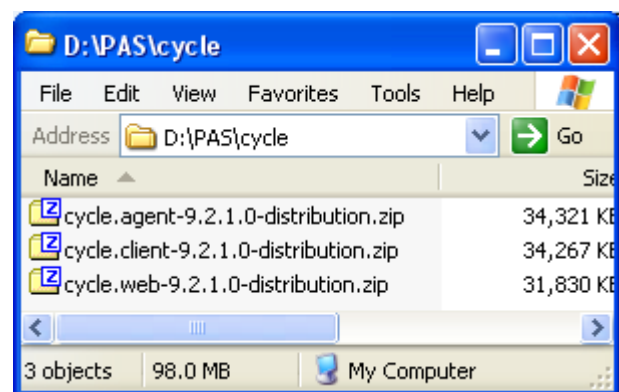
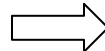
OIPA deployable folder structure



properties folder contents



OIPA deployable folder structure



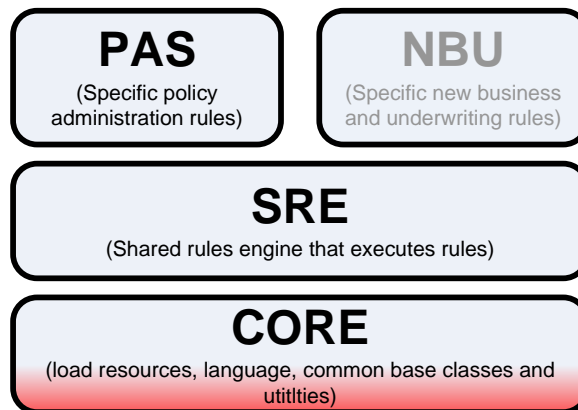
Each cycle zip folder has its own set of coherence files

CACHING

OIPA caches static configuration data and other data that rarely change. There are three cache regions in the system, each for a different module. Every cache region is configured separately, allowing each one to be configured differently. By implementing this model, the integration of new applications is simplified and only one new region must be added. The core and shared functionality regions will remain the same.

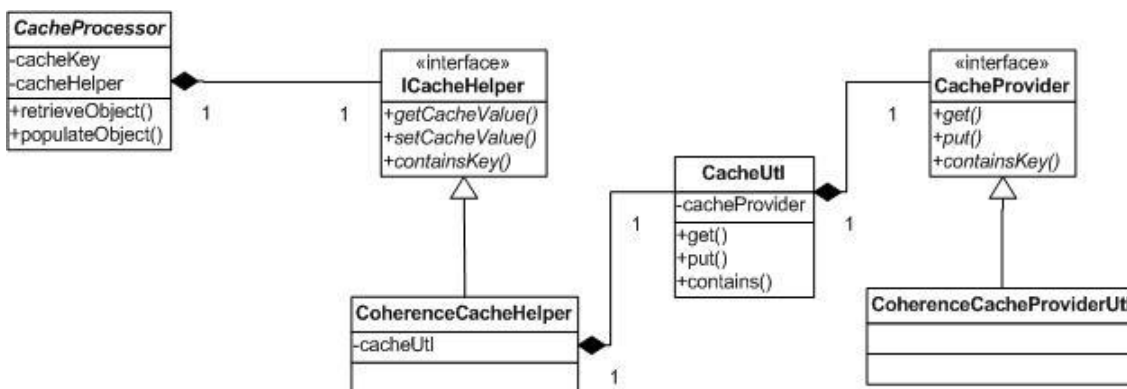
The three cache regions are:

1. region CORE for INSURANCE.SHARED module
2. region SRE for INSURANCE.SRE module
3. region PAS for INSURANCE.PAS module



Cache Regions

The following figure demonstrates how caching works in the system:



Cache Diagram

CACHEPROVIDER AND ICACHEHELPER INTERFACES

CacheProvider interface defines all methods a caching provider must implement in order to manage cached objects in OIPA. Coherence is one of the caching providers employed by OIPA.

ICacheHelper interface defines all methods required by caching access in the system, including getting and setting values for a caching key and method checking if the cache contains a given caching key.

Using CacheProvider and ICacheHelper interfaces allows OIPA to change actual caching technology that is used for caching without impacting any client code.

CACHEPROCESSOR ABSTRACT CLASS

The abstract class, CacheProcessor, retrieves the object for the caching key from the ICacheHelper in every CacheProcessor instance. An abstract method populateObject() is declared for every CacheProcessor subclass to implement what value object to be put into the cache associated with the caching key. An updateCache() method is also available for use when a cached value object needs to be updated for the caching key, such as when the system date is advanced from cycle processing. In that case, a new value is updated in the cache.

The following is the pseudo-code of the retrieveObject() provided by CacheProcessor.

```
IF cacheHelper contains cacheKey THEN
  get cache value of cacheKey from cacheHelper

ELSE

  invoke populateObject() to get cache value
  put cache value into cacheHelper associated with cacheKey
```

ACCESSING CACHE

Wherever a piece of data as cache candidate is requested, one anonymous inner class that extends CacheProcessor is defined and instantiated to determine how the value object is created when it is not yet in the cache. The value returned from CacheProcessor retrieveObject() method will be used by the client code. It could be found from the cache region for that module with the caching key, or, newly created and put into the cache region associated with the caching key. This is transparent so the difference is not visible to the client code.

The following is an example of code used for caching companyDcl for every companyGuid.

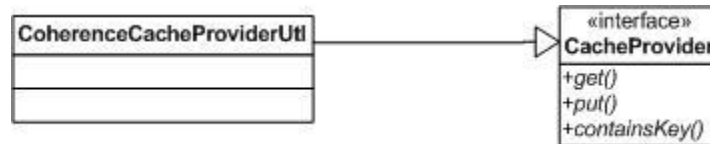
```
String key = "CompanyDcl[CompanyGUID=\"" + companyGuid + "\"]";
CacheProcessor cacheProcessor = new CacheProcessor( cacheHelper, key ) {

    @Override
    protected Object populateObject() {

        CompanyDcl companyDcl = findByPrimaryKey( CompanyDcl.class, companyGuid );
        return companyDcl;
    }
};
return cacheProcessor.retrieveObject();
```


COHERENCECACHEPROVIDERUTL AND CACHEUTL

The utility class `CoherenceCacheProviderUtl` implements the `CacheProvider` interface. It enforces the convention for interacting with Coherence. Every `CoherenceCacheProviderUtl` instance gets a cache region for a given name from the Coherence `CacheFactory`, which creates a clustered `NamedCache` instance when necessary. The `CoherenceCacheProviderUtl` object delegates all caching accesses to its Coherence `NamedCache` object.



Coherence CacheProviderUtl

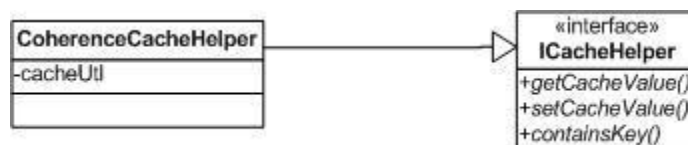
Every instance of utility class `CacheUtl` wraps a `CacheProvider` object. `CacheUtl` also maintains a class level map of mappings from cache region name and cache provider type to a `CacheUtl` instance. This guarantees only one `CacheProvider` per cache region per cache provider type.

THREE COHERENCECACHEHELPER CLASSES

`CoherenceCacheHelper` classes are `ICacheHelper` implementations that use Coherence for caching. There are three `CoherenceCacheHelper` classes in packages. They are as follows:

1. `com.adminserver.dal.helper`
2. `com.adminserver.sre.helper`
3. `com.adminserver.pas.dal.helper`

Each class corresponds to cache region CORE, SRE, and PAS respectively.



CoherenceCacheHelper

These three `CoherenceCacheHelper` classes have almost identical code, except that each declares its own unique cache region name (i.e., "CORE", "SRE", "PAS") and each register at Spring with a different bean name.

Every `CoherenceCacheHelper` has its own unique `CacheUtl` instance, which is mapped from the cache region name for the Coherence cache provider type.

COHERENCE CLUSTER CONFIGURATION

When configuring the Coherence Cluster, it is essential that **every** JVM that will participate in the Cluster is identified in the well-known-address list of the coherence-config.xml file.

Example of Coherence Cluster Configuration in OIPA Version 9

This file can be found in the properties folder in the OIPA deployable from E-Delivery. Open the properties folder and you will see the coherence-config.xml.

```
<coherence xml-override="/tangosol-coherence-override.xml">
  <cluster-config>
    <member-identity>
      <cluster-name>PRODCLUSTER</cluster-name>
      <member-name>OIPAMEMBER1</member-name>
    </member-identity>
    <unicast-listener>
      <address>123.456.78.9</address>
      <port>42222</port>
      <port-auto-adjust>false</port-auto-adjust>
    <well-known-addresses>
      <socket-address id="1">
        <address>123.456.78.9</address>
        <port>42222</port>
      </socket-address>
      <socket-address id="2">
        <address>123.456.78.10</address>
        <port>42222</port>
      </socket-address>
    </well-known-addresses>
  </unicast-listener>
</cluster-config>
<logging-config>
  <destination>stdout</destination>
  <!--
logged      0      - only output without a logging severity level specified will be
              1      - all the above plus errors
              2      - all the above plus warnings
              3      - all the above plus informational messages
              4-9    - all the above plus internal debugging messages (the higher the
number, the more the messages)
              -1     - no messages
-->
  <severity-level>3</severity-level>
</logging-config>
</coherence>
```

The following are guidelines for configuring the Coherence cluster:

1. Every member of the OIPA installation, including Cycle Agents and Cycle Clients, must be accounted for in the well-known-addresses list.
2. Every member of the OIPA installation, including Cycle Agents, Cycle Clients, and OIPA instances, must share the **exact same** well-known-addresses list.
3. If using the member-identity section, make sure that the cluster-name is **exactly the same** across all members of the OIPA installation.
4. If using the member-identity section, make sure that the member-name for each member is **unique across all members of the OIPA installation.**
5. In the logging-config section, the destination is configurable. If not specified, Coherence will log Coherence messages to stderr.

COHERENCE CONFIGURATION FOR CACHING

There are three cache mappings defined in the Coherence caching scheme mapping configuration for the cache name that are "CORE", "SRE", "PAS". The mapped schemes can be different or the same. In the example below each region name is identified along with an associated scheme map. Refer to the Oracle Coherence documentation on the Oracle Technology Network (OTN) for a full list of available configuration parameters. The documentation can be found by typing "coherence" in the Search field on the OTN home page.

Example of Coherence Configuration for Caching in OIPA Version 9

This file can be found in the properties folder and is named coherence-cache-config.xml.

```
<cache-config

xmlns:processing="class:com.oracle.coherence.patterns.processing.configuration.ProcessingPatternNamespaceHandler">
  <processing:cluster-config pof="true"/>
<!-- ===== -->
<!-- Map Caches to the NearScheme -->
<!-- ===== -->
  <coherence-scheme-mapping>
    <cache-mapping>
      <cache-name>CORE</cache-name>
      <scheme-name>SampleNearScheme</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>PAS</cache-name>
      <scheme-name>SampleNearScheme</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>SRE</cache-name>
      <scheme-name>SampleNearScheme</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>CYCLE</cache-name>
      <scheme-name>SampleNearScheme</scheme-name>
    </cache-mapping>
```

```

    <cache-mapping>
      <cache-name>cacheForAsProfiler</cache-name>
      <scheme-name>SampleLimitedPartitionedScheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

<caching-schemes>

  <!-- ===== -->
  <!-- Local In-memory Cache -->
  <!-- ===== -->
    <local-scheme>
      <scheme-name>SampleMemoryScheme</scheme-name>
    </local-scheme>

  <!-- ===== -->
  <!-- Size limited Local In-memory Cache -->
  <!-- ===== -->
    <local-scheme>
      <scheme-name>SampleMemoryLimitedScheme</scheme-name>
      <low-units>1000</low-units>
      <high-units>5000</high-units>
    </local-scheme>

  <!-- ===== -->
  <!-- Distributed In-memory Cache -->
  <!-- ===== -->
    <distributed-scheme>
      <scheme-name>SamplePartitionedScheme</scheme-name>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>SampleMemoryScheme</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
    </distributed-scheme>

  <!-- ===== -->
  <!-- Distributed Limited In-memory Cache -->
  <!-- ===== -->
    <distributed-scheme>
      <scheme-name>SampleLimitedPartitionedScheme</scheme-name>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
    </distributed-scheme>

  <!-- ===== -->
  <!-- Cache Cluster Definition -->
  <!-- ===== -->
    <near-scheme>
      <scheme-name>SampleNearScheme</scheme-name>
      <front-scheme>
        <local-scheme>
          <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
        </local-scheme>
      </front-scheme>
      <back-scheme>
        <distributed-scheme>
          <scheme-ref>SamplePartitionedScheme</scheme-ref>
        </distributed-scheme>
      </back-scheme>
    </near-scheme>
  </caching-schemes>

```

```
</near-scheme>

<!-- ===== -->
<!-- Cycle Invocation -->
<!-- ===== -->
  <invocation-scheme>
    <scheme-name>CycleMessagingService</scheme-name>
    <service-name>InvocationService</service-name>
    <autostart>true</autostart>
  </invocation-scheme>

</caching-schemes>

</cache-config>
```

WORKLOAD MANAGEMENT

OIPA provides a subsystem for batch processing for insurance transactions called Cycle. Cycle is a high performance distributed subsystem designed to process as many pending transactions as possible in the shortest amount of time. The cycle subsystem drives processing of transactions through a cycle group, which is comprised of a set of cycle agents.

Each cycle agent exists in its own JVM process. Cycle agents may be spread across multiple machines. Besides batch processing pending transactions, OIPA also leverages Cycle agents to facilitate long running background processes. This allows OIPA to offload processing to cycle agents, freeing up the OIPA application to service user requests. All of the Cycle agents together form a **processing grid**.

OIPA cycle uses the Coherence Incubator Processing Pattern for workload management of processing across the grid. Using the Processing Pattern and Coherence provides OIPA with the following benefits:

1. **No single point of failure** - coherence manages the failover of data for failed nodes in the cluster. Work that is queued or processing in a Node when it fails will automatically be reassigned to other nodes in the cluster for processing. This ensures that no work is lost (barring catastrophic failure of the entire cluster).
2. **No single bottleneck** - since processing is distributed evenly across the entire cluster, no single node represents a bottleneck for processing.
3. **State Management** - the Processing Pattern leverages Coherence caching for maintaining the state of the work that is processing in the grid. There is no need to track the work that is happening in the grid.
4. **JMX Support** - the Processing Pattern provides JMX MBeans that provide visibility into how processing is performing in the grid.

TASKS

Processing is executed on the grid using tasks. A task represents some kind of processing that needs to take place. Conceptually, there are two different kinds of tasks:

- **Monitoring tasks** – these tasks govern a long running process. They typically wake up on an interval, check some processing, and go back to sleep. Monitoring tasks are not typically processing intensive.
- **Processing tasks** – these tasks execute some unit of work and are typically short lived. Processing tasks typically are processing intensive, examples include:
 - Cycle task, which will process all pending activities on a policy.
 - Scheduled Valuation task, which will value a policy, generate, and store valuation records in the database.

All task processing is accomplished via the Coherence Processing Pattern. A task is submitted by a grid client in different ways, including:

- The cycle client submits a Cycle monitoring task to execute a cycle level.
- An OIPA instance submits a scheduled valuation task to run scheduled valuation on policies on a plan.

When tasks are submitted for processing, they arrive on a queue at one of the cycle agents connected to the grid. Task scheduling is by default achieved using a Round Robin algorithm. When they arrive in the partitioned cache of an agent, they are put into a queue and scheduled for processing in a Thread Pool. The thread pool size is controlled via the coherence-cache-config file of the cycle agent.

Processing tasks have short life cycles, where they will execute and then be complete. Monitoring tasks have an extended life cycle, and potentially intermediate state. When a monitoring task is executed, it processes, and has the ability to **yield** processing. When it yields, it will be re-dispatched to the cycle grid for processing.