Legal Notices

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

License Restrictions, Warranty/Consequential Damages Disclaimer

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.

Warranty Disclaimer

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.

Restricted Rights Notice

If this is software or related documentation that 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 END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Hazardous Applications Notice

This software or hardware 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 or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Third Party Content, Products, and Services Disclaimer

This software or hardware 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.

.

Introduction

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. The purpose of this document is to provide a comprehensive explanation of coherence and how it relates to OIPA.

Customer Support

If you have any questions about the installation or use of our products, please visit the My Oracle Support website: https://support.oracle.com, or call (800) 223-1711.

Oracle customers 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.

Overview

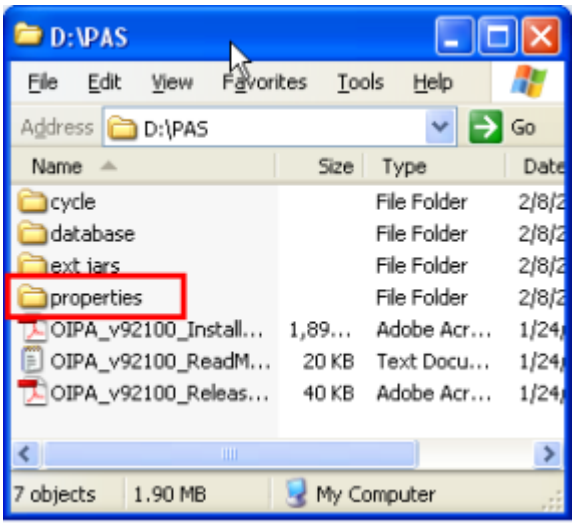
Coherence has no single point of failure. It automatically and transparently fails over and redistributes its clustered data when a server becomes inoperative or is disconnected from the network. When a new server is added, or when a failed server is restarted, it automatically joins the cluster, and Coherence transparently redistributes the cluster load. Coherence includes network-level fault tolerance features and a transparent soft restart capability, enabling servers to self-heal.

There are two different scenarios for which OIPA employs Oracle Coherence:

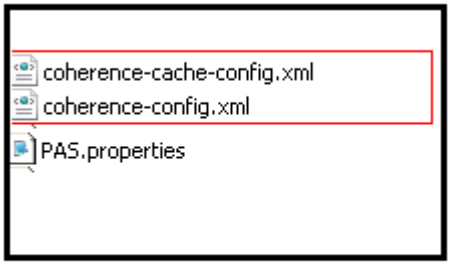
1. Caching
2. Cycle Messaging

Locating Configurable Coherence Files

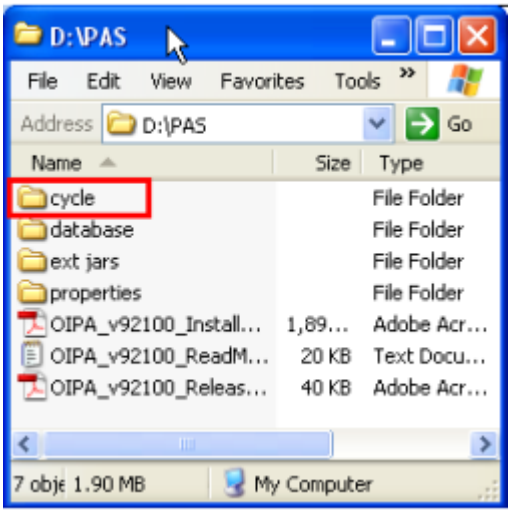
To locate the Coherence configuration files for caching, open the properties folder in the OIPA Media Pack from the Oracle Software Delivery Cloud. For caching in cycle, the files are located in each one of the cycle sub folders. The Coherence configuration files for the web application are called coherence-cache-config.xml and coherence-config.xml. These files will be explained later in the document. The Coherence configuration files for the web application are called cycle-coherence-cache-config.xml and cycle-coherence-config.xml. These files are explained in the "OIPA Cycle" document.



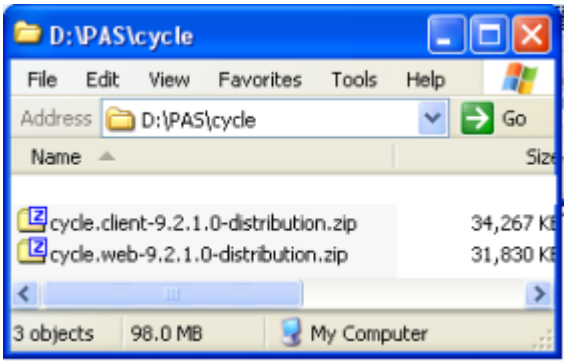
OIPA deployable folder structure



properties folder content



OIPA deployable folder structure

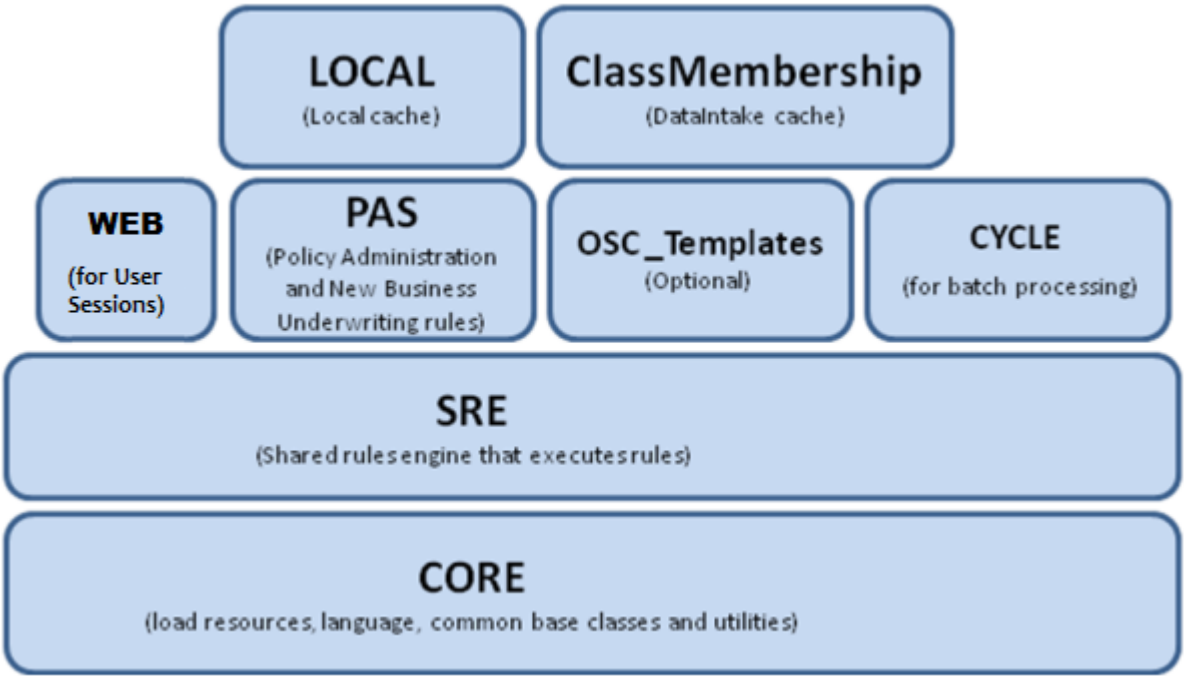


Each cycle zip folder has its own set of coherence files

Caching

OIPA caches static configuration data and other data that rarely change. There are eight cache regions in the system, each for a different module. Every cache region is configured separately, allowing each one to be configured differently. The cache regions are:

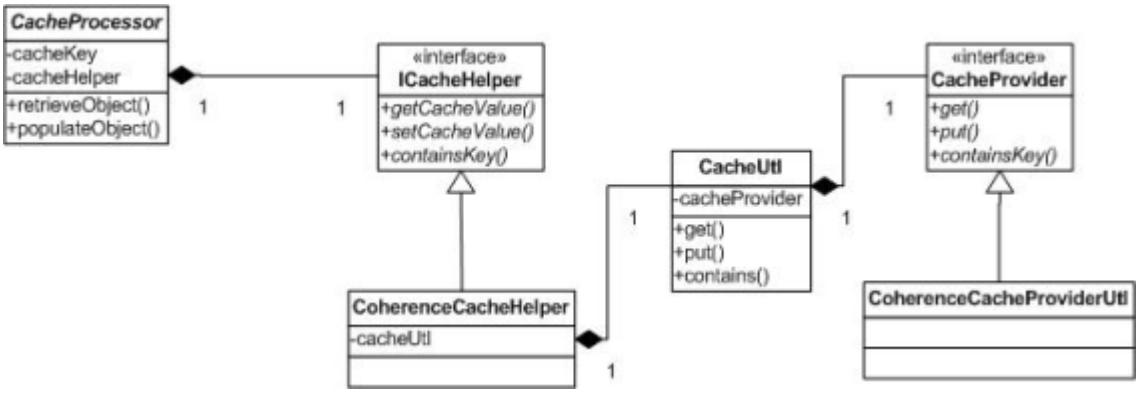
1. Region CORE for shared libraries.
2. Region SRE for shared rules engine components.
3. Region PAS for OIPA specific data.
4. Region LOCAL for providing caching services on the local node.
5. Region OSC_Templates for OSC specific caching.
6. Region CYCLE for batch processing.
7. Region ClassMembership for Data Intake processing.
8. Region WEB for User Session Data



Cache Regions

Above are the Region OSC_Templates for OSC-specific caching. This only needs to be configured if OSC is being used—see the document "OSC Installation Instructions" for further details. The regions CORE, SRE, PAS, WEB, OSC_Templates, CYCLE and ClassMembership are distributed across all nodes in a cluster. The LOCAL region is not distributed. Each JVM has its own LOCAL cache.

The following figure demonstrates how caching works in the system:



Cache Diagram

CacheProvider and ICacheHelper Interfaces

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

The 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 the 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 specify what value object is 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() method 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.

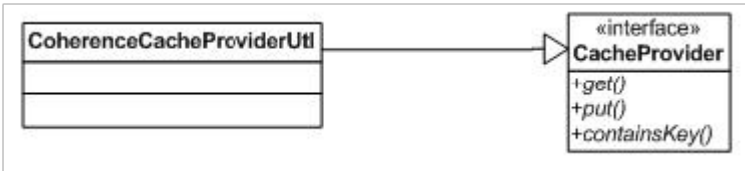
The following is an example of the code used for caching companyDcl for every companyGuid.

```
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 that only one CacheProvider exists per cache region per cache provider type.

CacheHelper 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

Additionally, the LocalCacheHelper class also implements ICacheHelper and is used for local node caching. It is in the com.adminserver.sre.helper package.

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

CoherenceCacheHelper

These four CacheHelper classes are almost identical, except that each declares its own unique cache region name (i.e., "CORE", "SRE", "PAS","LOCAL") and each registers with Spring using a different bean name.

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

Coherence Cluster Configuration

Coherence cluster configuration defines all the nodes in the cluster, along with a pointer to the cache factory configuration file. This is necessary in order for Coherence to run. During the configuring of the application server, an argument must be passed to the JVM to define the location of this Coherence configuration file, coherence-config.xml. The argument to be passed is "tangosol.coherence.over", along with the absolute path to the configuration file.

Example: "-Dtangosol.coherence.override=/opt/oracle/oipa/properties/coherence-config.xml

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

Example of Coherence Cluster Configuration

Coherence-config.xml can be found in the properties folder in the OIPA deployable from E-Delivery.

```
<coherence xml-override="/tangosol-coherence-override.xml">
   <cluster-config>
       <member-identity>
       <cluster-name>OIPA_CLUSTER</cluster-name>
       <member-name>OIPA_MEMBERNAME</member-name>
       </member-identity>
       <unicast-listener>
       <address>localhost</address>
       <port>42222</port>
       <port-auto-adjust>false</port-auto-adjust>
       <well-known-addresses>
       <socket-address id="1">
       <address>localhost</address>
       <port>42222</port>
       </socket-address>
       </well-known-addresses>
       </unicast-listener>
   </cluster-config>
   <logging-config>
   <destination>stdout</destination>
<!--
```

```
0 - only output without a logging severity level specified will be logged
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 must be accounted for in the well-known-addresses list.
2. Every member of the OIPA installation 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.

> **Note**: Refer to the "OIPA Cycle" document in the 11.0.0.0 release documentation library on OTN for an explanation of the coherence configuration of cycle.

Coherence Configuration for Caching

There are eight cache mappings defined in the Coherence caching scheme mapping configuration for the cache names "**CORE**", "**SRE**", "**PAS**", "**CYCLE**", "**WEB**", "**LOCAL**", "**OSC_Templates**", "**ClassMembership**". 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:

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

```
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config xmlns:processing="class://com.oracle.coherence.patterns.processing.
config.xml.ProcessingPatternNamespaceHandler"
xmlns:element="class://com.oracle.coherence.common.namespace.preprocessing.
XmlPreprocessingNamespaceHandler"
element:introduce-cache-config="coherence-processingpattern-cache-config.xml">
```

```
   <processing:cluster-config pof="true">
      <processing:dispatchers>
         <processing:task-dispatcher displayname="Task Dispatcher" priority="1"
         />
         <processing:logging-dispatcher displayname="Logging Dispatcher" />
         <processing:local-executor-dispatcher displayname="Local Dispatcher" />
      </processing:dispatchers>
<!-- MAKE SURE THAT ALL IDs ARE UNIQUE ACROSS THE CLUSTER, OR THE MEMBER WILL NOT
PARTICIPATE IN GRID PROCESSING -->
   <processing:taskprocessors>
      <processing:taskprocessordefinition id="PASTaskProcessor" displayname="PAS
      Task Processor" type="SINGLE" taskpattern="SingleTask">
      <processing:default-taskprocessor id="PASTaskProcessor" thread-
      poolsize="15"></processing:default-taskprocessor>
      </processing:taskprocessordefinition>
   </processing:taskprocessors>
   </processing:cluster-config>
<!-- ============================ -->
<!-- Map Caches to the NearScheme -->
<!-- ============================ -->
   <caching-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>ClassMembership</cache-name>
    <scheme-name>SampleMemoryScheme</scheme-name>
</cache-mapping>
<cache-mapping>
    <cache-name>LOCAL</cache-name>
    <scheme-name>SampleMemoryScheme</scheme-name>
</cache-mapping>
<cache-mapping>
    <cache-name>WEB</cache-name>
    <scheme-name>SampleMemoryScheme</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>10</low-units>
<high-units>32000</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>
    <!-- ======================== -->
    <!-- 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>
</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. Refer to the "OIPA Cycle" document in the 10.2.2.0 documentation library on OTN for a detailed explanation of how cycle works.

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, as well as 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 tasks 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 cycle-coherence-cache-config file of the cycle agent.

Processing tasks have short life cycles, where they execute and are then completed. Monitoring tasks have an extended life cycle, and can potentially have an 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.