ORACLE
INSURANCE

Oracle® Insurance Policy Administration

Cycle

Version 10.0.1.0

Documentation Part Number: E52368-01
February, 2014

Table of Contents

Introduction

Oracle Insurance Policy Administration (OIPA) provides a subsystem for batch processing of 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. By using concurrency techniques, multiple threads, automatic failover, automatic scaling and real time configuration changes, OIPA provides a robust batch solution. Various available commands allow you to run and view Cycle(s) according to your business needs.

The Cycle Agent is also used to process scheduled valuation and scheduled computation work. Scheduled valuation calculates the value of a group of policies at a specific point in time and stores that value for subsequent use. Scheduled computation is used to calculate values on policies that do not have a variable component. The time interval for running scheduled valuation or scheduled computation, and the frequency at which policies (or policies and segments for scheduled computation) are selected is configurable. Typically, these intervals are quarterly, semi-annually or annually. In order for scheduled valuation or scheduled computation to function, two additional options must be included in the Agent's configuration file.

In addition, Cycle is used to advance the system date stored in the `AsSystemDate` table.

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

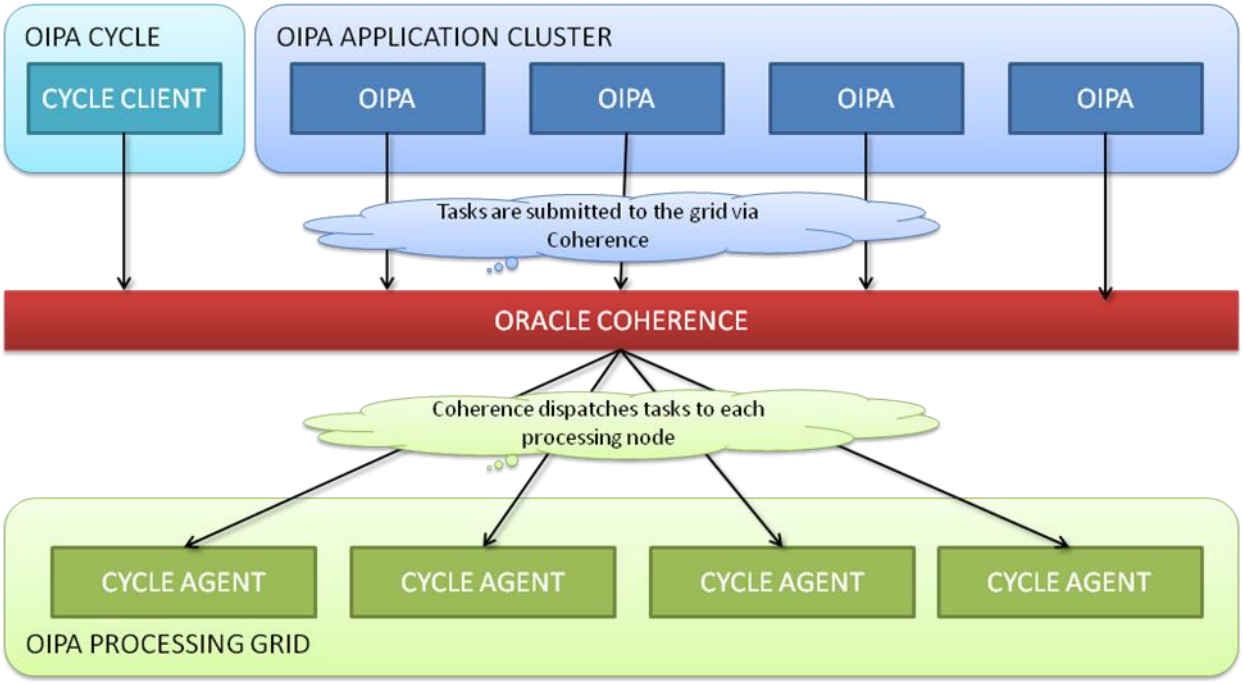
Cycle System Levels

When using OIPA Cycle for pending activity processing, activities may be configured to process according to OIPA system levels. Applicable levels for processing batch activities are:

- Company
- Client
- Plan
- Policy

Architecture

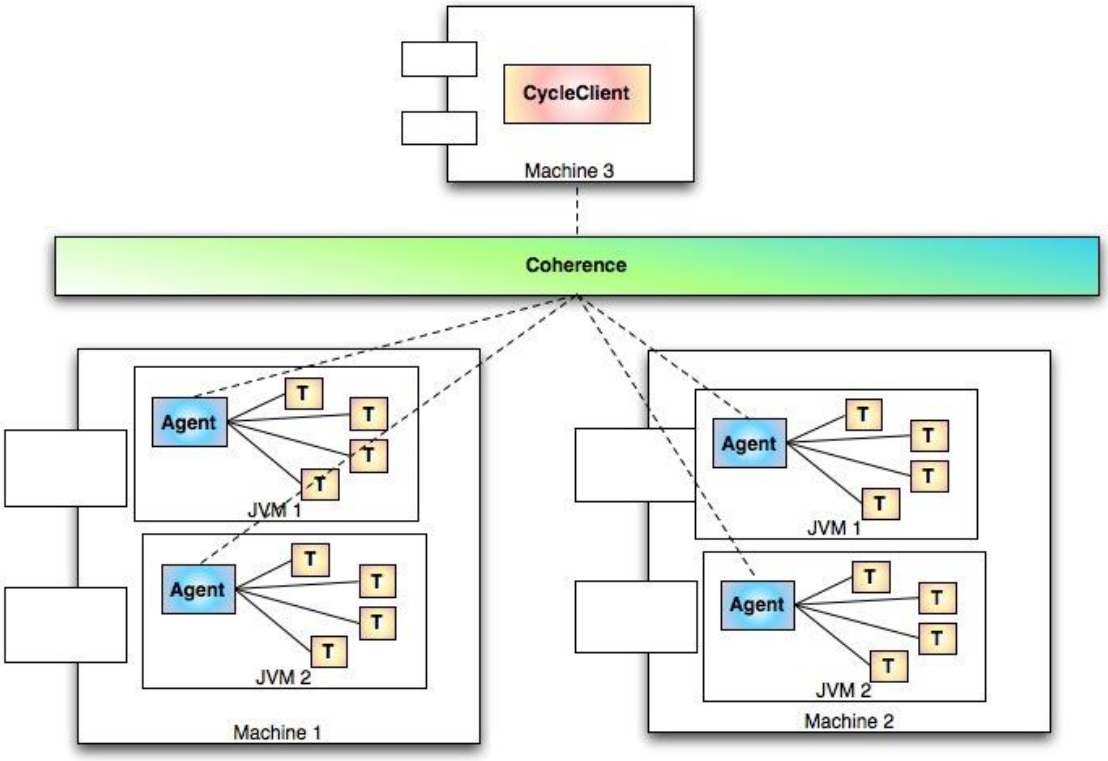
The Cycle subsystem drives processing of transactions through a Cycle Grid, which is comprised of a set of Cycle Agents. Each Cycle Agent is a separate JVM instance, which acts as a processing node in the Grid. A special application called a Cycle Client submits tasks to the Cycle Grid to direct what type of Cycle processing the group is to work on. The following is a high-level diagram showing how the different collaborators work together to start processing a Cycle run.



Cycle Agents

Cycle Agents are the programs that execute the tasks that comprise the Cycle batch process. Each Cycle Agent runs in its own Java Virtual Machine (JVM). Depending on the number of tasks that must be executed during Cycle processing, any number of JVMs may be used to run the desired number of Cycle

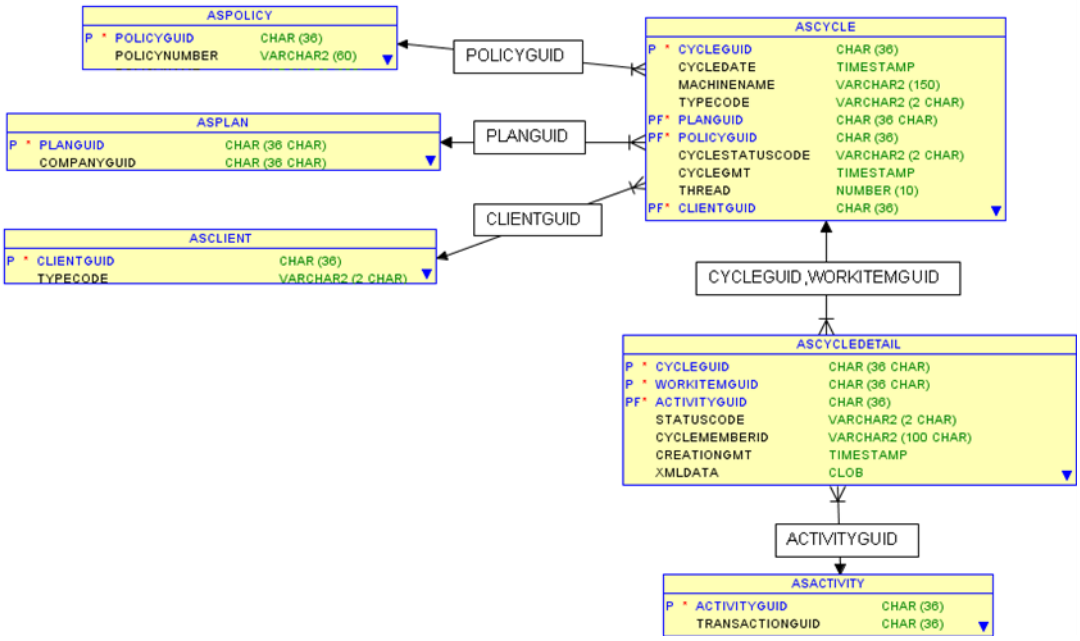
Agents. The JVMs may be started on one or more physical machines. If more than one machine is used, they will be effectively clustered via the Coherence clustered caching and messaging system.
The following conceptual deployment diagram shows how the Cycle Agents may be distributed between multiple machines and JVMs. The boxes labeled “T” represent the threads controlled by the Agents to perform the units of work.



Cycle Client

Cycle Client is a standalone command line application used to control the Cycle process. It allows you to commands to all the Cycle Agents in order to start and stop the process

Main Database Tables



Deployment

Cycle Agent operates inside a web container, like WebLogic or WebSphere. The Cycle Client still operates as a standalone application.

Configuration Files

`Cycle.properties` – Refer to the Client Cycle.properties section below for the properties related to the Cycle Client. Refer to the Agent Cycle.properties section below for the properties related to the Cycle Agent.

`cycle-coherence-config.xml` – This file controls the coherence cluster. Do not alter this file unless you have an in-depth understanding of coherence. The information in this file is based on how the installation was performed. The well-known-address is how you specify what machines are allowed to join the Cycle cluster. If you try to access a machine/IP address which is not in the cluster, you will get an error..

`cycle-coherence-cache-config.xml` – This is a default file that defines how coherence caches are configured in terms of best practices.

`log4j.xml –` This file controls the log files' level of detail. Do not alter this file unless you have an in-depth understanding of this technology.

`client-appContext.xml` – Only used for Cycle Client. This is used to auto-wire data access layer and business logic layer classes. This file should not be modified.

`cycle-spring-beans.xml` – Only used for Cycle Web/Agent. This is used to auto-wire data access layer and business logic layer classes. This file should not be modified.

Client Cycle.properties

cycle.period is the number of seconds that the Cycle Agent will wait before checking for additional work.

- Set it higher in order to avoid pinging the database too frequently for work. Frequent trips to the database, especially across multiple Cycle Agents, can impact performance.
- Set it low enough that you do not starve the Cycle Agent. If the threads in the Cycle Agent run out of work, they will not do anything until cycle.period expires and work is checked again.

cycle.batchSize is the maximum number of Cycle tasks that will process in the Cycle Grid. Consider the following:

- The batchSize should minimally be the sum total of all of the threads available across all Cycle Agents in the grid. For example, if there are four Cycle Agents, each with 10 threads, the minimum batch size should be 40 threads.
- For performance reasons, you may want to make sure that the threads are always running. To ensure that the threads are always running, set the batchSize to twice the number of threads in the Cycle Grid. For example, if there are four Cycle Agents running 10 threads each, set the batchSize to 80.
- If you have a cycle.groupSize greater than 1, multiply the previous result by the groupSize to determine the batchSize. For example, if there are four Cycle Agents running 10 threads each, and the groupSize is 10, set the batchSize to 800.
- The limitation on the number of tasks running in the cluster is dependent on the available memory in the cluster, not the thread pool sizes. The number of threads in the grid is only a theoretical estimate of how many tasks will be running at one time in the grid. Increasing the batchSize has the added benefit of reducing locking in the database when retrieving new tasks, which will increase performance. Setting the batchSize to 1000, 10000 or more is possible, depending on the memory in the cluster.

cycle.groupSize determines the number of Cycle work items to group together and execute on a single thread in the cluster.

- The groupSize has an impact on the number of tasks that are submitted to the grid for processing. If the batchSize is set to 10000, and the groupSize is set to 10, then 1000 actual tasks will be submitted to the grid for processing, each task containing 10 work items to be completed. Consider the groupSize value when determining the batchSize.
- Increase this value in order to group together tasks and speed up task submission and processing. The grid will process 10000 work items grouped in sets of 10 faster than 10000 individual work items.

grid.taskSubmissionThreadPoolSize is the number of threads dedicated to submitting tasks to the grid for processing. Increasing this number can speed up how quickly tasks are distributed to the grid.

- Increase this value in relation to the batchSize. The larger the batchSize, the more threads that should be dedicated to task submission.

updatestats.run specifies whether statistics on the AsCycle table should be updated during policy level of cycle processing. It defaults to No if not specified.

updatestats.degree determines the degree of parallelism.

- It only applies if updatestats.run is set to Yes and an Oracle database is being used. If set, the value should be an integer greater than or equal to 1.
- If not set and updatestats.run=Yes, then the table default value specified by the DEGREE clause in the CREATE TABLE or ALTER TABLE statement for AsCycle is used.

Web/Agent Cycle.properties

Refer to the OIPA System Properties document on OTN for an explanation of other properties related to Cycle Agent.

NOTE: In order for the Cycle Agent to be used in conjunction with Scheduled Valuation, two other options must be set in the Agent's Cycle.properties file.

- **scheduledValuation.batchSize** – This is the batch size for processing scheduled valuation or scheduled computation.
- **scheduledValuation.period** – This is the number of seconds that the scheduled valuation or scheduled computation monitor task will sleep before waking up and checking on the status of queued tasks.

Coherence

The Coherence cluster configuration in `cycle-coherence-config.xml` must be modified to include each Cycle Client and Agent that will participate in the Cycle group.

Since each Cycle member exists in its own JVM process and each member may be spread across multiple machines, the Cycle members in the same Cycle group must communicate. The messaging functionality of Coherence is used to accomplish this communication.

It is required that all Cycle members in the same Cycle group be configured in the same Coherence cluster.

The top of the file contains a section that allows for the addition of each machine that will participate in the Cycle group. Every Cycle Client and Agent that participates in a given Cycle group must be contained in this section. Also, each OIPA instance, if used for cycle processing, should also be listed in this section, as OIPA has the ability to submit work to the Cycle group. In the case that a machine is not included in the Coherence configuration, the machine will not be able to join the Cycle group and will encounter issues starting up.

When configuring the well-known-address list, keep the following in mind:

- Every Cycle Client, Cycle Agent, and OIPA instance (if participating in cycle) must be listed in the well-known-address list
- The well-known-address list must be *exactly the same across all members* (Cycle Clients, Cycle Agents, and OIPA instances)
- If all members do not have the same well-known-address list, some may not start up properly or run at all.
- Incorrect configuration of the well-known-address list is the most common source of Cycle and OIPA problems. Ensure the list is configured correctly and shared by all members.
- If you use a member-identity section like the example below, make sure that the member-name is unique in the cluster, and that the cluster-name is the same for all members of the cluster. If the cluster-name is not the same, members may not be allowed to join the Coherence cluster, causing startup issues for the OIPA instance, Cycle Agent, or Cycle Client.

```
<coherence>
    <cluster-config>
        <unicast-listener>
            <well-known-addresses>
                <socket-address id="1">
                    <address>localhost</address>
                    <port>8088</port>
                </socket-address>
            </well-known-addresses>
        </unicast-listener>
```

A sample configuration:

```
<cluster-config>
      <member-identity>
            <cluster-name>PRODCLUSTER</cluster-name>
            <member-name>CYCLEAGENT1</member-name>
      </member-identity>
      <unicast-listener>
                  <address>123.456.78.10</address>
                  <port>42222</port>
                  <port-auto-adjust>false</port-auto-adjust>
            <well-known-addresses>
                  <socket-address id="1">
                              <address>123.456.78.10</address>
                              <port>42222</port>
                  </socket-address>
                  <socket-address id="2">
                              <address>123.456.78.11</address>
                              <port>42222</port>
                  </socket-address>
                  <socket-address id="3">
                              <address>123.456.78.12</address>
                              <port>42222</port>
                  </socket-address>

            </well-known-addresses>
      </unicast-listener>
</cluster-config>
```

The Coherence cache configuration in `cycle-coherence-cache-config.xml` configures the processing configuration on each Cycle Agent. This file must be modified only for each Cycle Agent, in order to uniquely identify the task processors that will be working in the Cycle Cluster. If the identifiers are not unique across all Agents in the cluster, the Agents with duplicate identifiers will not process any work.

The cache configuration only needs to be configured for Cycle Agents. It does not need to be modified for OIPA instances or Cycle Clients.

The following section highlights the most significant change for the cache config file. This is an example of the cache config file for a Grid Processing node (a Cycle Agent).

```
<cache-config


xmlns:processing="class:com.oracle.coherence.patterns.processing.configuratio
n.ProcessingPatternNamespaceHandler">

    <processing:cluster-config pof="true">

        <processing:dispatchers>
```

```
            <processing:task-dispatcher displayname="Task Dispatcher"
priority="1">

                <processing:composite-policy>

                    <processing:attribute-match-policy />

                    <processing:round-robin-policy />

                </processing:composite-policy>

            </processing:task-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="CycleRunnableTaskProcessor" displayname="Cycle Runnable Task Processor"
type="SINGLE" taskpattern="SingleTask">

                <processing:default-taskprocessor
id="CycleRunnableTaskProcessor" threadpoolsize="10"></processing:default-
taskprocessor>

                <processing:attribute
name="type">runnable</processing:attribute>

            </processing:taskprocessordefinition>

                    <processing:taskprocessordefinition
id="CycleResumableTaskProcessor" displayname="Cycle Resumable Task Processor"
type="SINGLE" taskpattern="SingleTask">

                <processing:default-taskprocessor
id="CycleResumableTaskProcessor" threadpoolsize="5"></processing:default-
taskprocessor>

                <processing:attribute
name="type">resumable</processing:attribute>

            </processing:taskprocessordefinition>

        </processing:taskprocessors>

    </processing:cluster-config>
```

1. Make sure that each Cycle Agent/Cycle Web instance is using its own cycle-coherence-cache-config.xml file. You cannot share a single cycle-`coherence-cache-config.xml` file.
2. Make sure the ID attributes in the `processing:taskprocessors` XML section are unique across the entire cluster. In the example above, you must have a unique identifier for all items that are underlined and highlighted. There are six identifiers in total.

NOTE: If your identifiers are not unique across ALL Cycle Agents in the cluster, those Agents with duplicate identifiers will not process work. There will be no obvious exception in this instance.

The following are possible changes to this file:

1. Change the `threadpoolsize` attribute on the `processing:default-taskprocessor` element with the ID "Cycle`RunnableTaskProcessor`". This includes processing activities on a policy or a plan, or performing scheduled valuation/scheduled computation on a single policy (or segment for scheduled computation). Since runnable tasks tend to be more processing intensive, it is recommended that more threads are given to this thread pool.

2. Change the `threadpoolsize` attribute on the `processing:default-taskprocessor` element with the ID "CycleResumableTaskProcessor". This thread pool executes resumeable tasks, which are long running tasks in the grid that maintain intermediate state and report progress. Since resumable tasks tend to be less processing intensive, it is recommended that fewer threads are given to this thread pool.

Installation

Agent

The steps for setting up Cycle Agent are very similar to the steps for setting up OIPA. Follow the OTN documents “WebLogic Deployment Installation Instructions” for WebLogic and the “WebSphere Deployment Installation Instructions” for WebSphere. Only the differences in the setup are given below.

1. Modify any necessary configuration files as outlined in the Configuration section.
    - **`Cycle.properties`** – Refer to the Agent Cycle.properties section above for the properties related to the Cycle Agent.
    - **`cycle-coherence-config.xml`** – This file controls the coherence cluster. Do not alter this file unless you have an in-depth understanding of coherence. The information in this file is based on how the installation was performed. The well-known-address is how you specify what machines are allowed to join the Cycle cluster. If you try to access a machine/IP address which is not in the cluster, you will get an error..
    - **`cycle-coherence-cache-config.xml`** – This is a default file that defines how coherence caches are configured in terms of best practices.

    Copy these property files into the conf directory.
2. Instead of creating an OIPA Server, create a Cycle Server.
3. Use the Cycle Agent war instead of the PAS Java war. For WebLogic use CycleWeb-weblogic.war instead of PASJava-weblogic.war and CycleWeb-websphere.war instead of PASJava-websphere.war.
4. The coherence file names in the JVM arguments for cycle agent are different from that in OIPA. The arguments must contain the path to the config files. The following is an example of the arguments:

    ```
    -Dtangosol.coherence.override=./conf/cycle-coherence-config.xml -
    Dtangosol.coherence.cacheconfig=./conf/cycle-coherence-cache-config.xml
    ```
5. The application Portable Object config file must be specified as a JVM argument. The Portable Object config file is packaged with the application, and describes the portable object setup. The argument should not include a path to the Portable Object config file, as it is packaged in an application library and available on the class path. The following is an example of the Portable Object config argument for a Cycle Agent:

    ```
    -Dtangosol.pof.config=com-adminserver-cycle-agent-pof-config.xml
    ```
6. The complete JVM startup arguments for the container might look like:

```
-javaagent:/opt/Environments/spring-instrument-3.1.0.RELEASE.jar -
Dtangosol.coherence.override=/opt/Environments/cycle-coherence-config.xml
-Dtangosol.coherence.cacheconfig=/opt/Environments/cycle-coherence-cache-
config.xml -Dtangosol.pof.config=com-adminserver-cycle-agent-pof-
config.xml -Dtangosol.coherence.override=/opt/Environments/cycle-
coherence-config.xml
```

Client

1. Ensure that the `JAVA_HOME` environmental variable is set to a valid JDK 1.7.x installation. Alternatively, you can modify the startup scripts to use a specific JVM.

2. Extract the Cycle.client archive file into the desired location on your file system. The following directories are included:
    - **`bin`** – Contains the executables for the Cycle Client to run.
    - **`conf`** – Contains the configuration files for the Cycle Client.
    - **`lib`** – Contains the Java libraries required for the Cycle Client.
        - Open aspectj-1.6.11.jar with an unzipping software and retrieve aspectjrt.jar and aspectjweaver.jar from the lib folder and copy into the **lib** directory.

3. Modify the necessary configuration files as outlined in the configuration section. In most cases, the following files will need to be configured:
    - **`Cycle.properties`** – Refer to the Client Cycle.properties section above for the properties related to the Cycle Client.

    - **`cycle-coherence-config.xml`** – This file controls the coherence cluster. Do not alter this file unless you have an in-depth understanding of coherence. The information in this file is based on how the installation was performed. The well-known-address is how you specify what machines are allowed to join the Cycle cluster. You will receive an error if you try to access a machine which is not in the cluster.

    - **`cycle-coherence-cache-config.xml`** – This is a default file that defines how coherence caches are configured in terms of best practices.

4. If you are using a database other than Oracle, the correct database drivers must be copied to the `lib` directory, as outlined in the data source section of this document.

5. If you are using a database other than Oracle, the correct database drivers must be copied to the `lib` directory, as outlined in the data source section of this document.

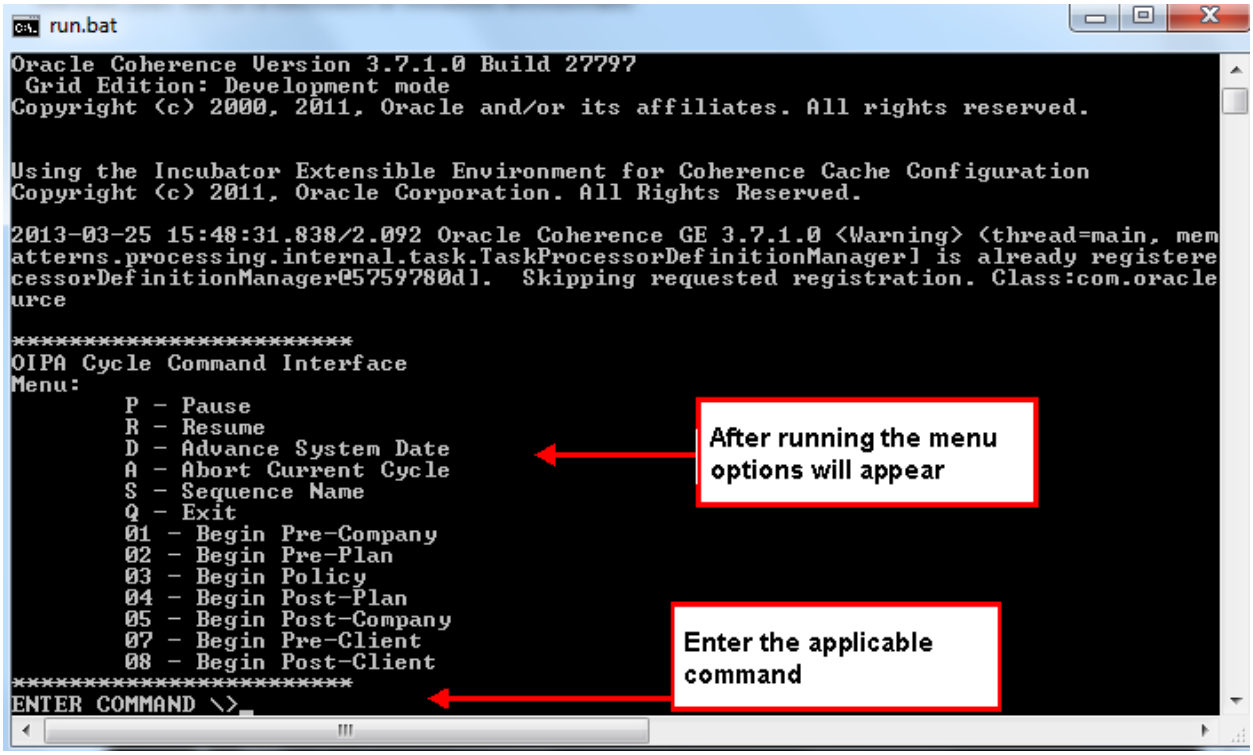
Running Cycle

Client

Windows

Running Cycle Client allows you to set how you want a set of pending activities to process. You must use the `run.bat` file for a Microsoft ® Windows environment.

1. Open a command line.
2. Navigate to the path where the application is installed and then to the Cycle.client folder. Type '`run`'.
3. Type in the letter or number associated with the command you want to run. Explanations of the commands are below.
4. Refer to the log files to see messages and errors.



Unix

The shell script file, `run.sh`, used to start the Cycle Client application, is located in the `client/bin` subdirectory of the Cycle Client installation directory. It may be run manually, or scheduled using any standard scheduling utility.

1. Open a shell
2. Navigate to the path where the application is installed and then to the `Cycle.client` folder.
3. Type '`./run.sh`'.
4. Type in the letter or number associated with the command you want to run.
5. Refer to the log files to see messages and errors.

Commands

- **P: Pause** – Cycle will not process, although it will still check to see if activities should be processed. Then it sees that the status is still in pause, and returns to sleep status.

- **R: Resume** – Takes Cycle out of pause status.

- **D: Advance System Date** – Advances date in the `AsSystemDate` table. Should be run after command(s) 01 through 08.

- **A: Abort Current Cycle** – This should only be used when there is a serious error. This is a non-recoverable scenario, and a shutdown will occur. Aborting a cycle will mark all records in the database as 99.

- **S: Sequence Name** – Specifies the name of the Cycle Sequence to run. A Cycle Sequence is configured in the system specifying an order of execution for cycle. Type "S SequenceName" where "SequenceName" is the name of the Cycle Sequence to execute.

    - More information on Cycle Sequences can be found in the XML Configuration Guide under "CycleSequence" system rule.

- **Q: Exit** – Exits Cycle.

- **01: Pre-Company –** Runs only the pre-company portion of Cycle.
    i. Cycle will attempt to process all company-level activities with an effective date less than or equal to the current system date, with one of the pending statuses and a processing order of 1000 or less.
    ii. A non-overridden business error or system error, by default, will abort further processing for the current Company-Level plan and move onto the next Company-Level plan, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

- **02: Pre-Plan –** Runs only the pre-plan portion of Cycle.
    i. Cycle will attempt to process all plan-level activities with an effective date less than or equal to the current system date, with one of the pending statuses and a processing order of 1000 or less.
    ii. Plans are executed in parallel using JMS threading.
    iii. A non-overridden business error or system error, by default, will abort further processing for the current plan and move onto the next plan, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

- **03: Policy** – Runs the policy-level portion of Cycle processing.
    i. All policy-level activities with an effective date less than or equal to the current system date will be processed in the following order: effective date (ascending), processing order (ascending), creation GMT (ascending).
    ii. Policies are processed in parallel using JMS threading.
    iii. A non-overridden business error or system error, by default, will abort further processing for the current policy and move onto the next policy, if one exists.

- **04: Post-Plan** – Runs only the post-plan portion of Cycle.
    i. Cycle will attempt to process all plan-level activities with an effective date less than or equal to the current system date, with one of the pending statuses and a processing order greater than 1000.
    ii. Plans are executed in parallel using JMS threading.
    iii. A non-overridden business error or system error, by default, will abort further processing for the current plan and move onto the next plan, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

- **05: Post-Company** – Runs only the post-company portion of Cycle.
    i. Cycle will attempt to process all company-level activities with an effective date less than or equal to the current system date with one of the pending statuses and a processing order greater than 1000.
    ii. A non-overridden business error or system error, by default, will abort further processing for the current Company-Level plan and move onto the next Company-Level plan, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

- **07: Pre-Client –** Runs only the pre-Client portion of Cycle.
    i. Cycle will attempt to process all Client-level activities with an effective date less than or equal to the current system date, with one of the pending statuses and a processing order of 1000 or less.
    ii. Clients are executed in parallel using JMS threading.
    iii. A non-overridden business error or system error, by default, will abort further processing for the current client and move onto the next client, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

- **08: Post-Client –** Runs only the post-Client portion of Cycle.
    i. Cycle will attempt to process all Client-level activities with an effective date less than or equal to the current system date, with one of the pending statuses and a processing order greater than 1000.
    ii. Clients are executed in parallel using JMS threading.
    iii. A non-overridden business error or system error, by default, will abort further processing for the current client and move onto the next client, if one exists. To alter this behavior, view CycleProcessBehavior business rule's configuration.

Troubleshooting

Check List

1. Make sure that you change both the `jpa.databasePlatform` and `application.databaseType` properties in the `Cycle.properties` file to match **your** database. Often one or both of these can be overlooked, causing exceptions to be thrown.
2. Make sure to change the `datasource.type` property in the `Cycle.properties` file to `jndi` when running in a container.
3. Make sure `Cycle.properties` is on the classpath and loaded by the container. You may see a message like "cannot find bundle Cycle" if it cannot find the properties file.
4. Make sure that the coherence configuration files are on the classpath. Make sure that the well-known-address list in the cycle-coherence-config.xml and the OIPA coherence-config.xml match *exactly* if the OIPA instance is also used for cycle processing.
5. Make sure that the cycle-coherence-cache-config.xml file is not *shared* by Cycle Agents. Each Cycle Agent should have its own cycle-coherence-cache-config.xml
6. Make sure that the cycle-coherence-cache-config.xml file has unique identifiers as mentioned in the Coherence configuration section of this document.

Common Questions Concerning Cycle

The Cycle Agent doesn't do anything, even though I ran the Cycle Client.

The most common problem is that the Cycle Agent is not sharing the same Coherence Cluster as the other members of the Cycle group, or the Cycle Client. If the Cycle Agent is not sharing the same cluster, then it will not be able to receiving messages, and will not do anything. There are a few reasons why the Cycle Agent is not part of the same Coherence Cluster:

- Check the startup parameters for the application server to ensure that the system property is passed in. The system property is **`-Dtangosol.coherence.override=`**, and should point to the cycle-coherence-config.xml file. If the instance cannot find the `cycle-coherence-config.xml file`, it may start its own, and will be outside of the shared cluster.
- Check the well-known-addresses section in the `cycle-coherence-config.xml` file and make sure that the Cycle Agent and/or Cycle Client have exactly the same addresses listed. If you are denied access to the cluster, the instance may start its own, and will therefore be outside of the shared cluster.
- Make sure the `cycle-coherence-cache-config.xml` file is loaded. You can check this in the log file or the console output. Sometimes it will indicate that it could not be loaded, and load the default. If the default cache config file is loaded, it will not be in the shared coherence cluster.