Oracle® Insurance Policy Administration

Cycle Instructions

Version 9.2.0.0.0

Part number: E16287-01

December 2009

Table of Contents

Cycle 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 you with a robust cycle solution. Various available commands allow you to run and view cycle(s) according to your business needs. Cycle is also used to advance the system date stored in the AsSystemDate table.

Cycle Architecture

There are two main parts to Cycle: the Client and the Service, or Agent. A single instance of the Cycle Client is responsible for ensuring the AsCycle table is populated and organizing the cycle. The Cycle Service is responsible for processing each activity.

The Cycle Client provides the user with a selection of possible activities for Cycle to process, such as Policy level. The Cycle Client then populates AsCycle with all eligible pending activities that have an activity effective date less then or equal to the system date. Each activity's status is initially set to pending, or 02.

The Cycle Service uses the same code as OIPA to process each activity in exactly the same way it would be processed if an end-user did the processing. Multiple instances of the Cycle Service, each in a separate JVM process, can be run concurrently. Each instance of the Cycle Service uses a configurable amount of threads to further increase efficiency. Multiple instances can be run on a single machine or distributed across many machines. Each instance of the Cycle Service is referred to as a **Cycle Agent**.

Each Cycle Agent will, when idle and at a specified interval, request work from the target application. The amount of work sent to the Agent depends upon the configuration, as detailed in this document. AsCycle is then updated to show which Agent was assigned to each specific activity, along with the CycleMemberID and the specific thread.

Upon successful completion of a batch, the CycleStatusCode for the activity is updated to success, or 01, and more work, if available, is given to the Agent. In the event of an error during execution of an activity, the CycleStatusCode will be updated to reflect the specific type of error, and the XMLData column will be populated with the stack trace.

Cycle Communication

Each Cycle Agent exists in its own separate JVM process, and Cycle Agents may be spread across multiple machines. Cycle Agents communicate by sending messages to each other through a Message Bus. The default implementation of the message bus is through Oracle Coherence. The following list describes the messages that are sent during cycle processing:

- Cycle Group Command Message – sent by the Cycle Client. Provides an instruction that is picked up by the Cycle Controller for processing. Examples include Start Cycle, Abort, Pause, Resume, and Advance System Date.
- Cycle Command Message – provides the details of a command that is issued to each member in the Cycle Group. These are only issued by the Cycle Client application. Examples include Start, Stop, Pause, and Resume.
- Member Status Message – sent by a member in the Cycle Group when the member's status changed. Examples include Started, Started_NoWork, Stopped, Paused, Joined, Left, and Resumed.
- Group Status Message – sent by the Cycle Controller when the status of all of the members in the group has changed. For example, when all of the members run out of work to do, they will all be in the IDLE state, and the Cycle Controller will send a Group Status Message indicating that the group is no longer actively processing transactions.

Batch Processing for Activity Processing

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

- Company
- Client
- Plan
- Policy

Running Commands of Activities to Process

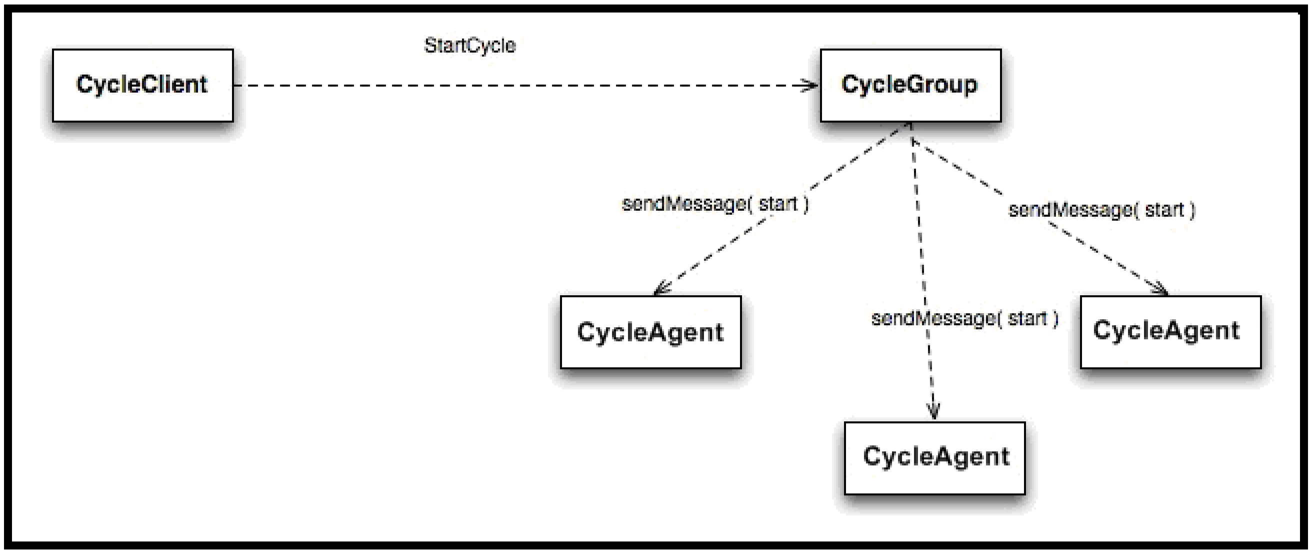
Additionally OIPA can process these activities according to their effective dates, processing order and the current system date. OIPA cycle separates activities to process according to date as follows:

- Pre-Company
- Pre-Client
- Pre-Plan
- Post-Company
- Post-Client
- Post-Plan
- Policy

A full explanation of how processing is handled is explained below in the Cycle Command section.

High Level Parts to Run Cycle

The cycle subsystem drives processing of transactions through a Cycle Group, which is comprised of a set of Cycle Agents. The cycle controller is a role that is assumed by one of the Cycle Agents in the Cycle Group, and is responsible for monitoring the status of the all of the members. A special application called a Cycle Client makes requests to the Cycle Group 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 Overview Diagram

To set-up the Cycle Agent, you will first set-up the cycle properties and then start cycle. The files needed to start cycle are in the cycle.agent and cycle.client folder with the application files.

Note: Batch Screens in OIPA are used for manually processing or spawning the processing of multiple activities in one activity at one time for one policy, where cycle runs all pending activities on various policies at a specified point in time.

Cycle Deployment Options

Cycle can operate in the following deployment configurations:

- **Standalone** - In a Standalone deployment, each Cycle Agent and the Cycle Client operate in separate, standalone JVMs. The Cycle Agents can be started in a background daemon thread or windows service, or started manually.
- **Container** - In a Container deployment, a Cycle Agent operates inside of a web container, like Weblogic or WebSphere. The Cycle Client still operates as a standalone application.

In addition, the Cycle Client can operate in two different modes:

- **Intelligent Client** - In this mode, the Cycle Client actually loads the application runtime (for example, OIPA), and performs many group commands. The group commands include Advance System Date, Load the Cycle Table for a Cycle Run, Abort the current Cycle run, etc. For example, when a cycle run is started, the cycle table is populated first, and then a message is sent to each Agent in the Group to start working on the cycle run. In this scenario, the Intelligent Cycle Client will load the table first, then send the Start message to all of the Agents.
- **Dumb Client** - In this mode, the Cycle Client does not load the application runtime, and sends group commands to be processed by the Cycle Controller. In this mode, the Cycle Client is very dumb, and only communicates via messages. For example, when a cycle run is started, the dumb Cycle Client will send a group command message to the Cycle Controller and then just wait for the cycle run to complete. The Cycle Controller will load the cycle table, and then send the message to the rest of the Cycle Agents in the group to being processing.

Cycle Deployment Considerations

1. **Container Considerations**
    - **Management** - If most of the IT management infrastructure is built on the container platforms they run (like WebSphere), then it may be advantageous to run the Cycle Agents inside of the container so it is easier for the IT organization to monitor and update cycle. The downside is that there are extra environments that need to be created and maintained for cycle, which incurs additional cost.
2. **Intelligent Client Considerations**
    - **Security** - Since the Cycle Client always operates as a standalone java application, database access information is stored in configuration files on the file system, and information is not retrieved via JNDI. These files will need to be updated any time the database information changes.
    - **Start time** - Since the intelligent Cycle Client loads the application run-time, it does take longer for the client application to load up.
3. **Dumb Client Considerations**
    - **Security** - Since the dumb Cycle Client just sends messages to the group, there is no need to store database access information in the client.
    - **Start time** - Since the dumb Cycle Client never loads the application run-time, it is quicker to start up

Setting Up and Starting Up Cycle

The initial set-up of cycle requires in-depth knowledge of the following areas:

- how cycle splits activities to process
- server cluster that will be used
- number of threads that should be designated for optimal performance.

You should not run cycle or change any of the properties unless you have in-depth knowledge. Prior to starting the Cycle Agent, you will want to update properties in various files found in the **conf** folder.

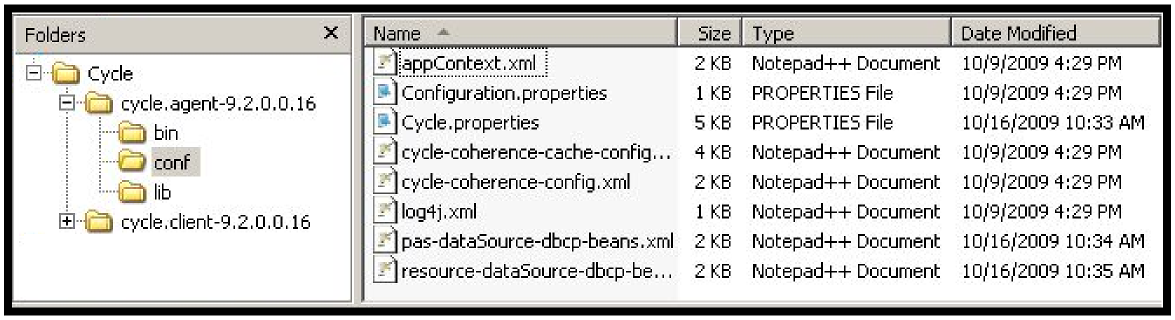
Installing Cycle Agent, Installing Cycle Client and Setting up Cycle Properties

There are various files that need to be updated to reflect preferences, database settings and various locations needed to start-up cycle. These files are found in the **Cycle** folder in the Oracle Insurance Policy Administration application, which can be downloaded from E-delivery. Once you have downloaded the zip file, open the **OIPA_*version number*** folder. You may set-up cycle in a standalone agent or in a container. The directions below outline both methods.

- Standalone Cycle Agent Installation
- Cycle Client for Cycle Agent Standalone Installation
- Container Cycle Agent Installation
- Cycle Client for Cycle Agent Container Installation

Standalone Cycle Agent Installation

1. Open the Cycle folder and extract the cycle.agent zip file to a location on your file system. If you are in a *nix environment, then you will be using the tar.gz file instead of the .zip.
   - **bin** - Contains the executables for the Cycle Agent to run.
   - **conf** - Contains the configuration files for the Cycle Agent. See explanation of agent conf files for more detail on each file.
   - **lib** - Contains the java libraries required for the Cycle Agent.



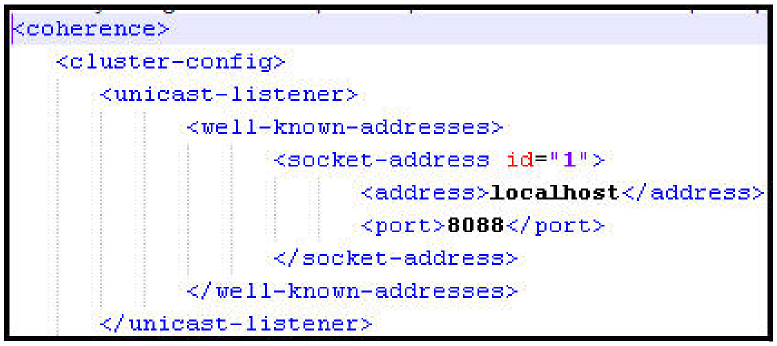
Name	Size	Type	Date Modified
appContext.xml	2 KB	Notepad++ Document	10/9/2009 4:29 PM
Configuration.properties	1 KB	PROPERTIES File	10/9/2009 4:29 PM
Cycle.properties	5 KB	PROPERTIES File	10/16/2009 10:33 AM
cycle-coherence-cache-config...	4 KB	Notepad++ Document	10/9/2009 4:29 PM
cycle-coherence-config.xml	2 KB	Notepad++ Document	10/9/2009 4:29 PM
log4j.xml	1 KB	Notepad++ Document	10/9/2009 4:29 PM
pas-dataSource-dbcp-beans.xml	2 KB	Notepad++ Document	10/16/2009 10:34 AM
resource-dataSource-dbcp-be...	2 KB	Notepad++ Document	10/16/2009 10:35 AM

Cycle.agent conf Folder

2. Make sure that you have your JAVA_HOME set to a valid JDK 1.5 installation. Alternatively, you can modify the startup scripts to use a specific JVM.

3. Modify your coherence configuration. This file is found in conf/cycle-coherence-config.xml. At the top of the file you will see the following section where you can add a list of machines that will participate in the Cycle Group (including the Cycle Client).

Note: Make sure you add all of the machines that may join the group. If you do not, it is possible that a Cycle Agent will start-up in a separate cluster and will not participate in the group.



```
<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>
```

Coherence Configuration

4. Modify your **Cycle.properties** file to fit your configuration. There are three properties at the top of the properties file that affect how cycle processing will run.
   - **cycle.threadCount** is the number of threads that will work on cycle tasks concurrently. Increase this number to improve the total processing time for cycle. Increasing the number of threads beyond a certain point will not necessarily yield improvements. This number is dependent on the processing and memory resources available on the machine.
   - **cycle.period** is the number of seconds that the Cycle Agent will wait before checking for additional work. This is also dependent on the machine, but consider the following:
     - You want to make 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.
     - You want to make it low enough so you do not starve the Cycle Agent. If the threads in the Cycle Agent run out of work, they will not do anything until the cycle.period expires and work again is checked.
   - **cycle.batchSize** is the number of cycle tasks that the Cycle Agent will work on at one time across all threads. The first time the Agent requests work, it will retrieve this number. During the cycle run, the Agent will not exceed this number when requesting additional work. For example, if the batchSize is 100, and the Agent has 50 work items queued, it will request 50 work items to get its batch back to 100 (it will **not** request 100). You need to consider how many Cycle Agents will be running when setting this value. If it is too high, then it is possible that one Cycle Agent can starve another. If it is too low, then the thread pool can run out of work for the current Agent instance, starving this Cycle Agent. The ideal scenario is that each Cycle Agent in the group is always working, and it is a balance between the number of threads, number of Cycle Agents, and number of work items that need to be processed.
5. Modify the **Cycle.properties** file for your database.
   - Set the **datasource.type** = **dbcp** for a standalone Cycle Agent.
   - Set the **jpa.databasePlatform** for your database.
   - Set the **application.databaseType** for your database.

6. Modify the appContext.xml file.

Note: Leave this file as is for a standalone Cycle Agent installation. Modifications for a container Cycle Agent will be detailed later.

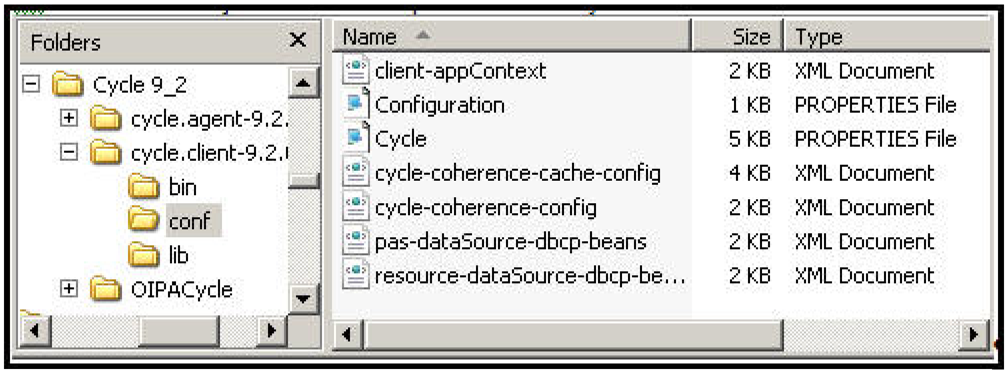
7. Modify the pas-dataSource-dbcp-beans.xml and resource-dataSource-dbcp-beans.xml files for your database.

Note: If you are using a database other than Oracle, you need to copy the driver for your database into the lib folder.

8. Go to the bin folder and run the start-up batch or script file. You will need to wait a few minutes while it starts up, but it should come up without an error. If there are any errors, please review your configuration files.

Cycle Client for Cycle Agent Standalone Installation

1. Unzip the cycle.client zip file into a location on your file system. If you are in a *nix environment, then you will be using the tar.gz distribution. You should notice the following folders:
   - **bin** - Contains the executables for the Cycle Client to run.
   - **conf** - Contains the configuration files for the Cycle Client (more details provided later)
   - **lib** - Contains the java libraries required for the Cycle Client.

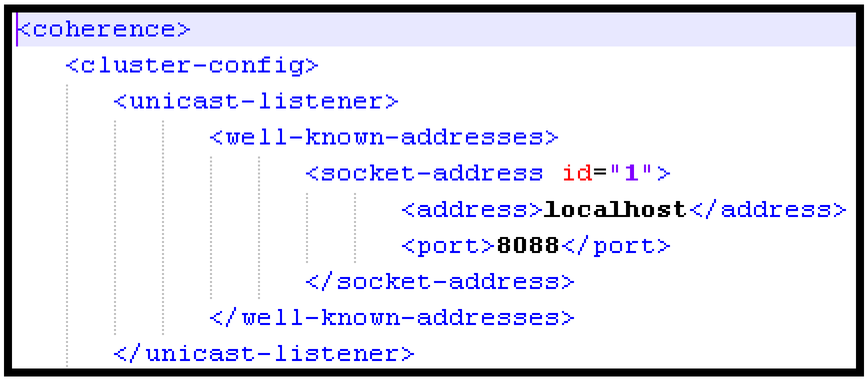


Name	Size	Type
client-appContext	2 KB	XML Document
Configuration	1 KB	PROPERTIES File
Cycle	5 KB	PROPERTIES File
cycle-coherence-cache-config	4 KB	XML Document
cycle-coherence-config	2 KB	XML Document
pas-dataSource-dbcp-beans	2 KB	XML Document
resource-dataSource-dbcp-be...	2 KB	XML Document

Cycle.client conf Folder

2. Make sure that you have your JAVA_HOME set to a valid JDK 1.5 installation. Alternatively, you can modify the start-up scripts to use a specific JVM.
3. Modify your coherence configuration. This file is found in conf/cycle-coherence-config.xml. At the top of the file you will see the following section where you can add a list of machines that will participate in the Cycle Group (including the Cycle Client).

Note: Make sure you add all of the machines that may join the group. If you do not, it is possible that a Cycle Client will start-up in a separate cluster and will not participate in the group.



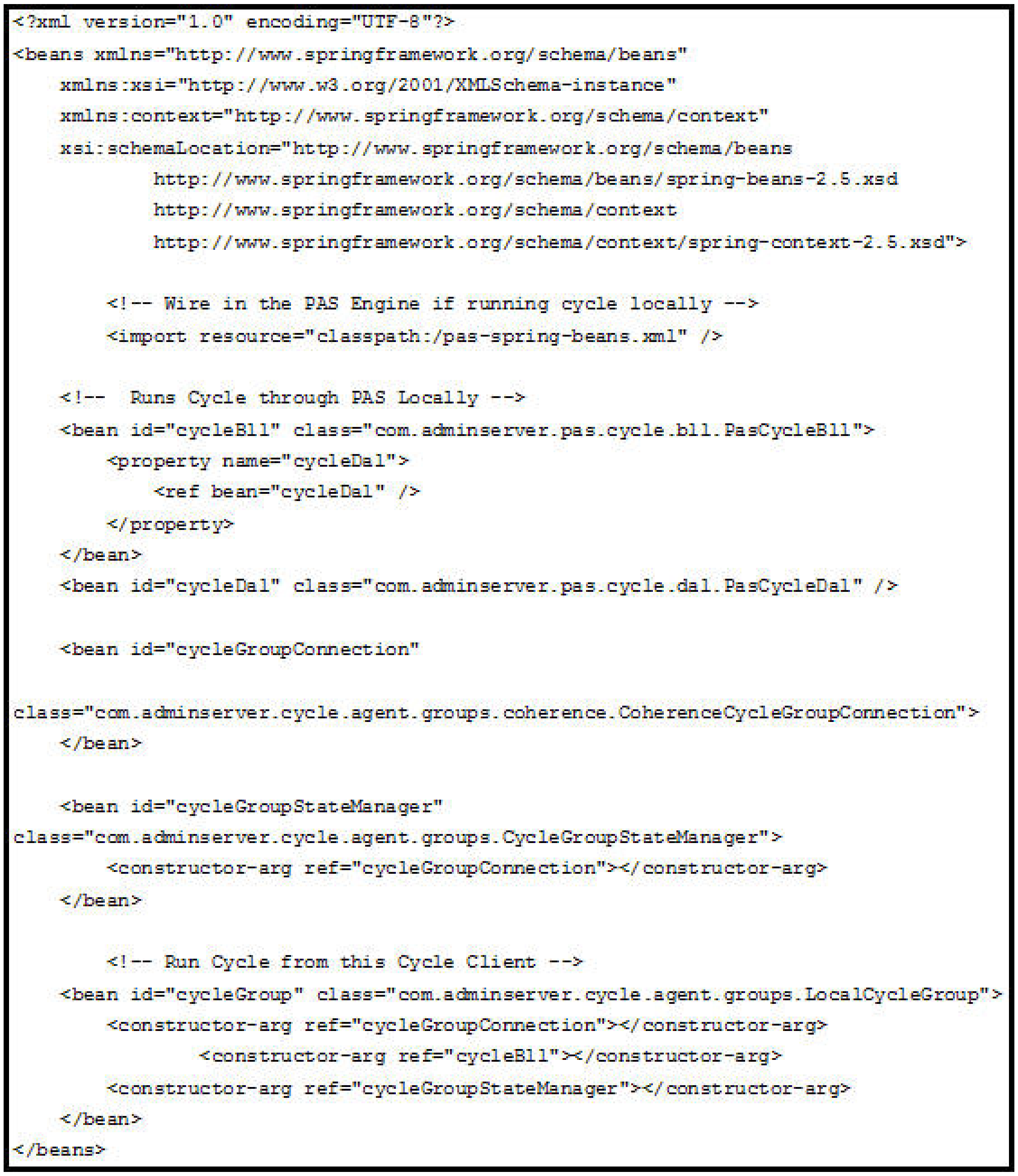
```
<coherence>
   <cluster-config>
      <unicast-listener>
            <well-known-addresses>
                  <socket-address id="1">
                        <address>localhost</address>
                        <port>8088</port>
                  </socket-address>
            </well-known-addresses>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

Coherence Configuration

4. Modify the **Cycle.properties** file for your database.
   - Set the **datasource.type** = **dbcp** for a standalone Cycle Agent.
   - Set the **jpa.databasePlatform** for your database.
   - Set the **application.databaseType** for your database.

5. Modify the client-appContext.xml file.

Note: Leave this file as is for a standalone Cycle Agent installation. Modifications for a container Cycle Agent will be detailed later. The following images shows the default configuration for the Intelligent Standalone Cycle Client.



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-2.5.xsd">

        <!-- Wire in the PAS Engine if running cycle locally -->
        <import resource="classpath:/pas-spring-beans.xml" />

    <!--  Runs Cycle through PAS Locally -->
    <bean id="cycleBll" class="com.adminserver.pas.cycle.bll.PasCycleBll">
        <property name="cycleDal">
            <ref bean="cycleDal" />
        </property>
    </bean>
    <bean id="cycleDal" class="com.adminserver.pas.cycle.dal.PasCycleDal" />

    <bean id="cycleGroupConnection"

class="com.adminserver.cycle.agent.groups.coherence.CoherenceCycleGroupConnection">
    </bean>

    <bean id="cycleGroupStateManager"
class="com.adminserver.cycle.agent.groups.CycleGroupStateManager">
        <constructor-arg ref="cycleGroupConnection"></constructor-arg>
    </bean>

        <!-- Run Cycle from this Cycle Client -->
    <bean id="cycleGroup" class="com.adminserver.cycle.agent.groups.LocalCycleGroup">
        <constructor-arg ref="cycleGroupConnection"></constructor-arg>
                <constructor-arg ref="cycleBll"></constructor-arg>
        <constructor-arg ref="cycleGroupStateManager"></constructor-arg>
    </bean>
</beans>
```

Default configuration for the Intelligent Standalone Cycle Client

6. Modify the pas-dataSource-dbcp-beans.xml and resource-dataSource-dbcp-beans.xml files for your database.

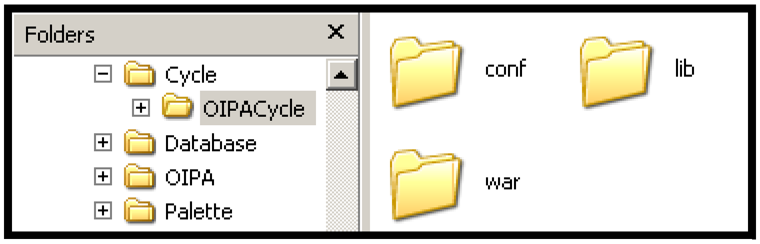
Note: For container installation, these files are not needed.

Note: If you are using a database other than Oracle, you need to copy the driver for your database into the lib folder.

7. Go to the bin folder and run the **run** batch or script file. You will need to wait a few minutes while it starts up, but it should come up without an error. If there are any errors, please review your configuration files.

Container Cycle Agent Installation

1. Unzip the cycle.web zip file. You will notice the following files.

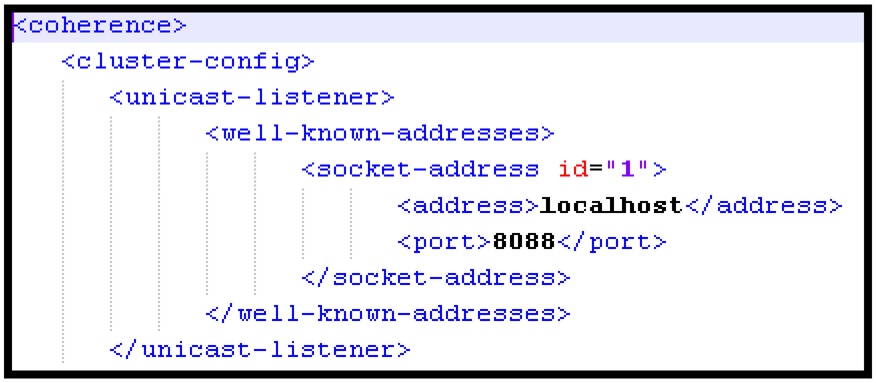


OIPA Cycle Folder

- **OIPACycle.war** – In the war folder is the war file distributable that needs to be installed into your application server.
- **Cycle.properties** – In the conf folder resides the properties file that you need to configure your environment.
- **cycle-spring-beans.xml** - In the conf folder resides the Spring configuration for the Cycle Agent. This can be left alone by default.
- **cycle-coherence-config.xml** - In the conf folder resides the cluster configuration file for Coherence.
- **cycle-coherence-cache-config.xml** - In the conf folder resides the cache configuration file for Coherence.
- **spring-agent.jar** - In the lib folder resides the file that initializes Spring inside the container.

2. Modify your coherence configuration. This file is found is **cycle-coherence-config.xml**. At the top of the file you will see the following section where you can add a list of machines that will participate in the Cycle Group (including the Cycle Client).

Note: Make sure you add all of the machines that may join the group. If you do not, it is possible that a Cycle Agent will start-up in a separate cluster and will not participate in the group.



```
<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>
```

Coherence Configuration

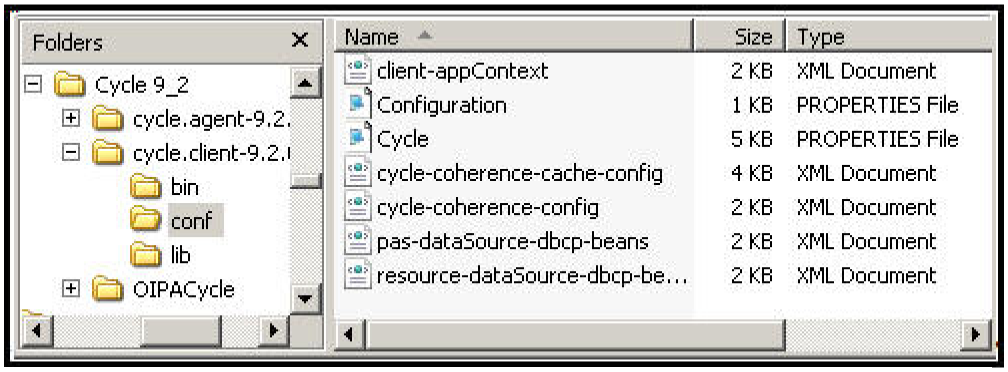
3. Modify your **Cycle.properties** file to fit your configuration. There are three properties at the top of the properties file that affect how cycle processing will run.
   - **cycle.threadCount** is the number of threads that will work on cycle tasks concurrently. Increase this number to improve the total processing time for cycle. Increasing the number of threads beyond a certain point will not necessarily yield improvements. This number is dependent on the processing and memory resources available on the machine.
   - **cycle.period** is the number of seconds that the Cycle Agent will wait before checking for additional work. This is also dependent on the machine, but consider the following:
     - You want to make 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.
     - You want to make it low enough so you do not starve the Cycle Agent. If the threads in the Cycle Agent run out of work, they will not do anything until the cycle.period expires and work again is checked.
   - **cycle.batchSize** is the number of cycle tasks that the Cycle Agent will work on at one time across all threads. The first time the Agent requests work, it will retrieve this number. During the cycle run, the Agent will not exceed this number when requesting additional work. For example, if the batchSize is 100, and the Agent has 50 work items queued, it will request 50 work items to get its batch back to 100 (it will **not** request 100). You need to consider how many Cycle Agents will be running when setting this value. If it is too high, then it is possible that one Cycle Agent can starve another. If it is too low, then the thread pool can run out of work for the current Agent instance, starving this Cycle Agent. The ideal scenario is that each Cycle Agent in the group is always working, and it is a balance between the number of threads, number of Cycle Agents, and number of work items that need to be processed.

4. Modify the **Cycle.properties** file for your database.
   - Set the **datasource.type** = **jndi** for a container Cycle Agent
   - Set the **jpa.databasePlatform** for your database.
   - Set the **application.databaseType** for your database.
5. Upload all of the files **except** the OIPACycle.war file to a location on your server's file system that can be accessed from your application server.
6. Upload the jdbc driver for your database to a location on your server's file system that can be accessed from your application server.
7. Add the location of the files you uploaded to the classpath of the application server so it can find the files. This is different depending on the application server you use.
   - For JBOSS, you can put them into the **conf** folder of the server that you are using
   - For Weblogic 11g (10.3x), you can put them into the root folder of the domain (server) that you are using. For example, in Windows, this might be under **E:\Oracle\Middleware\user_projects\domains\base_domain**
   - For WebSphere, you can upload them onto the application server. Then go to your application server -> Java and Process Management -> Process Definition -> Java Virtual Machine, and you will be able to set the classpath there.
8. Modify the start-up parameters for the application server to use spring as the javaagent, and to override the default coherence configuration. This will appear as follows (paths are examples, you will need to change them)
   -javaagent:"/path/to/your/spring-agent.jar" -Dtangosol.coherence.override=/path/to/your/cycle-coherence-config.xml"
   - For JBOSS, set these in the /jbosshome/bin/run.bat or run.sh file.
   - For WebSphere, go to your application server -> Java and Process Management -> Process Definition -> Java Virtual Machine, and set this in Generic JVM Arguments.
9. Create a Datasource named **ADMINSERVERDS** and another named **ADMINSERVERRESOURCEDS** on the application server. It is important that the data sources are named correctly as they are used by the application.
   - **ADMINSERVERDS** points to your OIPA application database.
   - **ADMINSERVERRESOURCEDS** should by default point to the same OIPA application database. If you choose to store your multi-language definitions in a different database, you would point to it here.

Cycle Client for Cycle Agent Container Installation

1. Unzip the cycle.client zip file into a location on your file system. If you are in a *nix environment, then you will be using the tar.gz distribution. You should notice the following folders:
   - bin - Contains the executables for the Cycle Client to run.
   - conf - Contains the configuration files for the Cycle Client (more details provided later).
   - lib - Contains the java libraries required for the Cycle Client.
2. Make sure that you have your JAVA_HOME set to a valid JDK 1.5 installation. Alternatively, you can modify the start-up scripts to use a specific JVM.
3. Modify your coherence configuration. This file is found in conf/cycle-coherence-config.xml. At the top of the file you will see the following section where you can add a list of machines that will participate in the Cycle Group (including the Cycle Client).

Note: Make sure you add all of the machines that may join the group. If you do not, it is possible that a Cycle Client will start-up in a separate cluster and will not participate in the group.

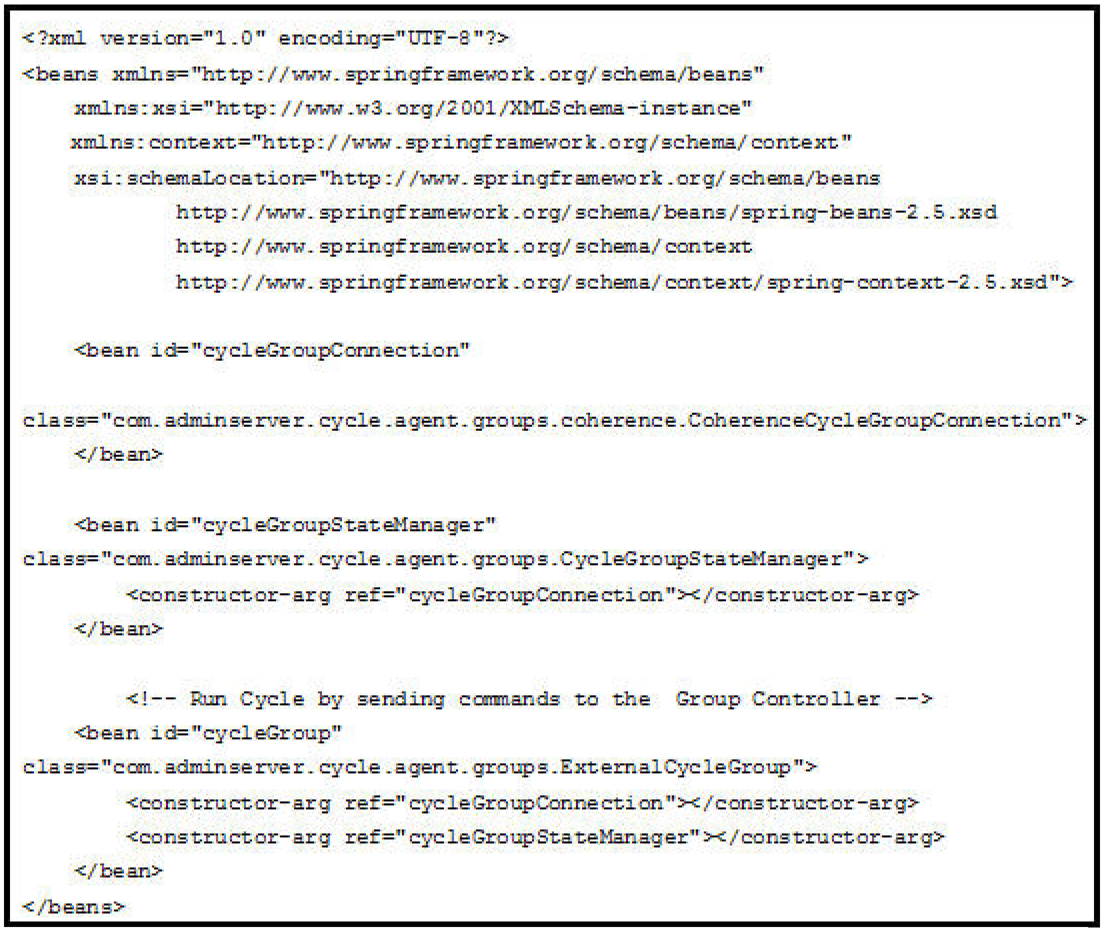


Name	Size	Type
client-appContext	2 KB	XML Document
Configuration	1 KB	PROPERTIES File
Cycle	5 KB	PROPERTIES File
cycle-coherence-cache-config	4 KB	XML Document
cycle-coherence-config	2 KB	XML Document
pas-dataSource-dbcp-beans	2 KB	XML Document
resource-dataSource-dbcp-be...	2 KB	XML Document

Cycle.client conf Folder

4. Modify the **Cycle.properties** file for your database.
   - Set the **datasource.type** = **dbcp** for a standalone Cycle Agent.
   - Set the **jpa.databasePlatform** for your database.
   - Set the **application.databaseType** for your database.
   - You can delete pas-dataSource-dbcp-beans.xml and resource-dataSource-dbcp-beans.xml files. They are not used.

5. Modify the client-appContext.xml file. It will appear as follows:



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <bean id="cycleGroupConnection"

class="com.adminserver.cycle.agent.groups.coherence.CoherenceCycleGroupConnection">
    </bean>

    <bean id="cycleGroupStateManager"
class="com.adminserver.cycle.agent.groups.CycleGroupStateManager">
        <constructor-arg ref="cycleGroupConnection"></constructor-arg>
    </bean>

        <!-- Run Cycle by sending commands to the  Group Controller -->
    <bean id="cycleGroup"
class="com.adminserver.cycle.agent.groups.ExternalCycleGroup">
        <constructor-arg ref="cycleGroupConnection"></constructor-arg>
        <constructor-arg ref="cycleGroupStateManager"></constructor-arg>
    </bean>
</beans>
```

Client-appContext.xml File

Note: If you are using a database other than Oracle, you need to copy the driver for your database into the **lib** folder.

6. Go to the bin folder and run the **run** batch or script file. You will need to wait a few minutes while it starts up, but it should come up without an error. If there are any errors, please review your configuration files.

Explanation of Files

- **appContext.xml** – Do not alter this file.

- **Configuration.PROPERTIES** – By default this file points to the Cycle.PROPERTIES file. You can point to other .PROPERTIES files if necessary by editing this file.

- **Cycle.PROPERTIES** – The first three properties to set in this file relate to cycle and control the number of threads, the time intervals cycle will check for work, and the maximum number of cycle tasks that can be run simultaneously. The rest of the properties in this file should match what are in the OIPA properties file. There are descriptions of each property in the actual Cycle.PROPERTIES file.

- **cycle-coherence-cache-config.xml** – Default file defines how coherence caches were configured in terms of best practices.

- **cycle-coherence-config.xml** – Controls the clusters found in coherence. You should only alter this file if you understand coherence. The information here is based on how your installation was done. The well known address is how you filter out what machines are allowed to join the cycle cluster. You will receive an error if you try to access and you are not on the cluster.

- **log4j.xml** - This file controls the level of detail that reports to the log files. Only alter this file if you have an in-depth understanding of this technology.

- **pas-dataSource-dbcp-beans.xml** – Contains the log-in information to the database you are connecting to. This will need to be changed accordingly. See the next section, Obtaining the Necessary .jar Files into lib folder, which will have specifics on the driver information.

- **resource-dataSource.dbcp-beans.xml** – Contains the log-in information to the database you are connecting to for the international database. This will need to be changed accordingly. See the next section, Obtaining the Necessary .jar Files into lib folder, which will have specifics on the driver information.

Pre-requisites to Running Cycle

Before starting cycle, you must obtain the necessary proprietary and open source executable .jar files, which are required. Then copy them into the lib folder. There are different .jar files that you will need to include depending on whether you are using DB2 or SQL for your database. No .jar files need to be downloaded if you are using an Oracle database.

Also ensure that your Cycle Agent(s) started up. You need at least one active Cycle Agent in the Coherence Cluster to process Cycle. Double-check your configuration. Make sure that the Cycle Client's Coherence configuration is correct and will connect you to the existing Coherence Cluster.

Locating Jar Files

Using DB2 Database

The jar files that need to be copied in the lib folder:

- db2jcc
- db2jcc_license_cisuz
- db2jcc_license_cu

Note: **db2jcc**, **db2jcc_license_cisuz**, and **db2jcc_license_cu** are included with the purchase of the DB2 software. These files are not available for download. Contact your IT department if you need assistance locating these files.

Using SQLServer Database

The jar files that need to be downloaded in the lib folder:

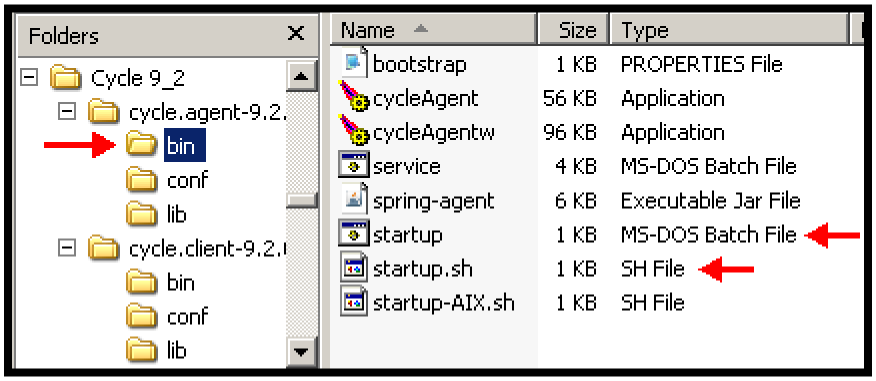
- jtds

Note: Download **jtds** from the following site: http://sourceforge.net/projects/jtds/

1. Select the **Download jtds – SQL Server and Sybase JDBC driver** link.
2. Scroll down to the area that lists the previous File Releases and download the distribution file for version 1.2 (**jtds-1.2-dist.zip**). Save the download .zip file to the temporary directory created to store the .zip files.
3. Open the downloaded .zip file and extract the file **jtds-1.2.jar** from the root of the .zip file.
4. Rename this file **jtds.jar**.

Starting the Cycle Agent Application

To start up the Cycle Agent, select either **startup.bat** or **startup.sh**, depending on whether you are using Microsoft ® Windows or Unix/Linux. See the directions below that are necessary for starting up cycle.

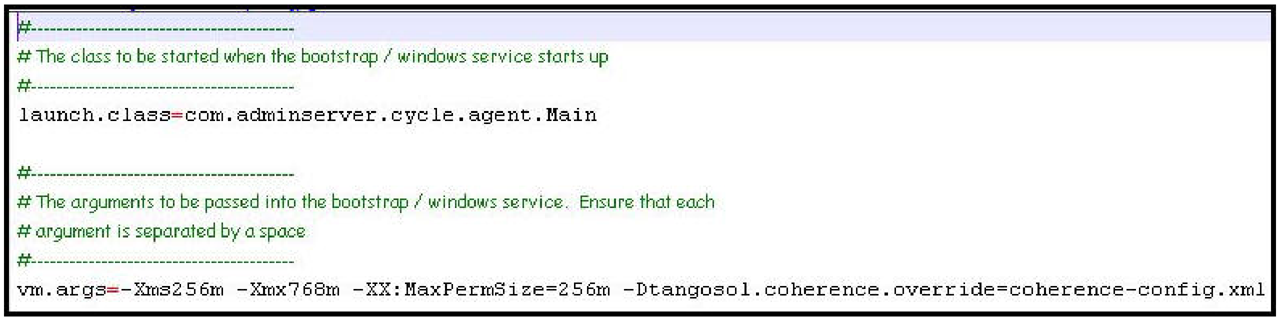


Name	Size	Type
bootstrap	1 KB	PROPERTIES File
cycleAgent	56 KB	Application
cycleAgentw	96 KB	Application
service	4 KB	MS-DOS Batch File
spring-agent	6 KB	Executable Jar File
startup	1 KB	MS-DOS Batch File
startup.sh	1 KB	SH File
startup-AIX.sh	1 KB	SH File

Cycle start-up files

Starting Cycle Using Windows

Prior to starting cycle, you must go in the bootstrap.PROPERTIES file as this is what is loaded when registered as a Microsoft ® Windows service. Modify the arguments such as allotted memory or where the conherence-config.xml file is located.



```
#-----------------------------------------
# The class to be started when the bootstrap / windows service starts up
#-----------------------------------------
launch.class=com.adminserver.cycle.agent.Main

#-----------------------------------------
# The arguments to be passed into the bootstrap / windows service.  Ensure that each
# argument is separated by a space
#-----------------------------------------
vm.args=-Xms256m -Xmx768m -XX:MaxPermSize=256m -Dtangosol.coherence.override=coherence-config.xml
```

Bootstrap.PROPERTIES file

If you are using Windows, you should set-up Cycle Agent as a Windows service. After installing it is important to understand that if you want this to run automatically you must use your Administrative tools to do so. When running cycle as a Windows service, a logs directory is created and the log files are stored in this directory. Log files contain messaging and errors that occurred.

Steps to Install

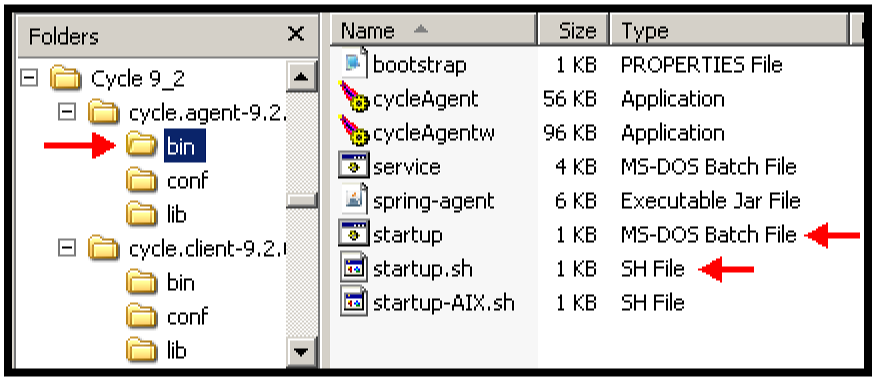
1. Open a command line.
2. Navigate to the path where the application is installed and to the cycle.agent\bin folder and type '`agent install`'.

Steps to Remove

1. Open a command line.
2. Navigate to the path where the application is installed and to the cycle.agent\bin folder and type '`agent remove`'.

Starting Cycle Using Linux/Solaris

The shell script file, startup.sh, which is used to start the Cycle Agent application is located in the *agent/bin* subdirectory of the Cycle service installation directory. It may be run manually or scheduled using any standard scheduling utility. The configuration file *bootstrap.PROPERTIES* is not necessary for this environment.



Name	Size	Type
bootstrap	1 KB	PROPERTIES File
cycleAgent	56 KB	Application
cycleAgentw	96 KB	Application
service	4 KB	MS-DOS Batch File
spring-agent	6 KB	Executable Jar File
startup	1 KB	MS-DOS Batch File
startup.sh	1 KB	SH File
startup-AIX.sh	1 KB	SH File

Cycle start-up files

Running Cycle

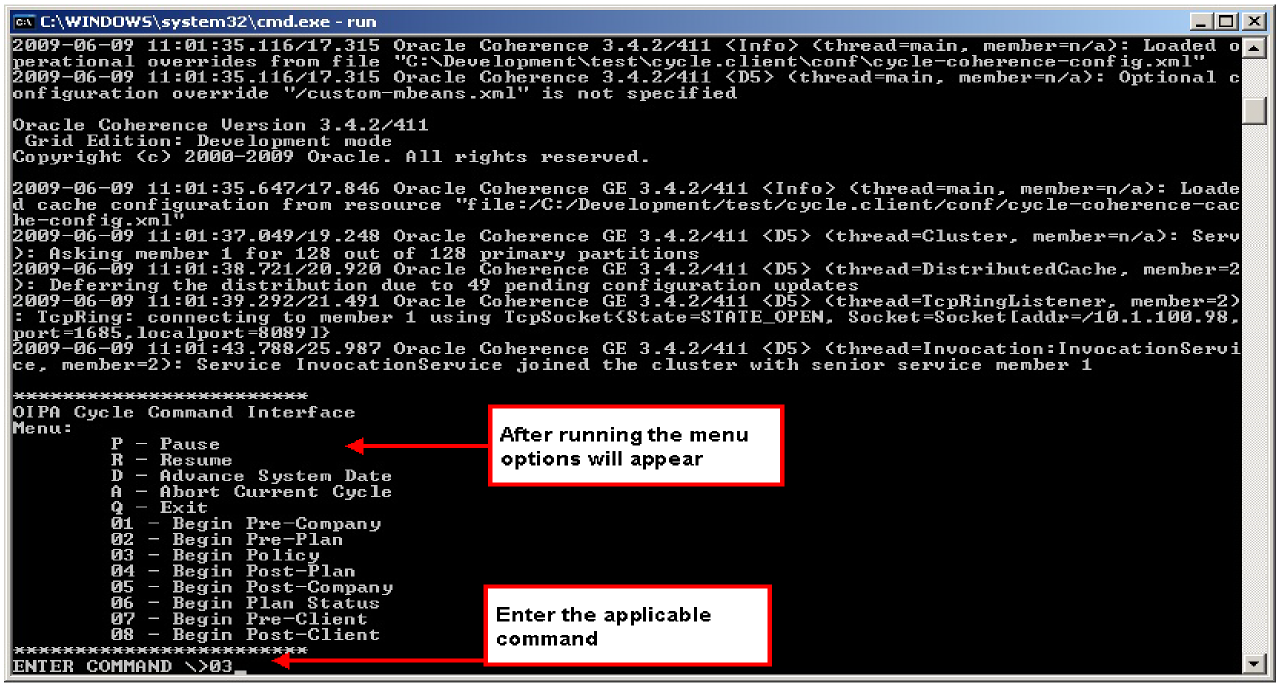
Running cycle allows you to set how you want a set of pending activities to process. You must use the run.bat file for a Microsoft ® Windows environment or the run.sh for a Unix/Linux environment to run the client.

Steps to Run Cycle Commands

1. Open a command line.
2. Navigate to the path where the application is installed and then to the cycle.client folder. Type '`run`'. This will run either the run.sh or run.bat, depending on your environment.
3. Type in the letter or number associated with the command you want to run. Explanations of the commands are below.

Note: In the event of a critical system error such as a database failure or system wide network failure that results in all Cycle Agents in the group failing, the Cycle Client will exit with a system error code.

4. Refer to the log files to see messages and errors.



```
C:\WINDOWS\system32\cmd.exe - run
2009-06-09 11:01:35.116/17.315 Oracle Coherence 3.4.2/411 <Info> (thread=main, member=n/a): Loaded o
perational overrides from file "C:\Development\test\cycle.client\conf\cycle-coherence-config.xml"
2009-06-09 11:01:35.116/17.315 Oracle Coherence 3.4.2/411 <D5> (thread=main, member=n/a): Optional c
onfiguration override "/custom-mbeans.xml" is not specified

Oracle Coherence Version 3.4.2/411
 Grid Edition: Development mode
Copyright (c) 2000-2009 Oracle. All rights reserved.

2009-06-09 11:01:35.647/17.846 Oracle Coherence GE 3.4.2/411 <Info> (thread=main, member=n/a): Loade
d cache configuration from resource "file:/C:/Development/test/cycle.client/conf/cycle-coherence-cac
he-config.xml"
2009-06-09 11:01:37.049/19.248 Oracle Coherence GE 3.4.2/411 <D5> (thread=Cluster, member=n/a): Serv
): Asking member 1 for 128 out of 128 primary partitions
2009-06-09 11:01:38.721/20.920 Oracle Coherence GE 3.4.2/411 <D5> (thread=DistributedCache, member=2
): Deferring the distribution due to 49 pending configuration updates
2009-06-09 11:01:39.292/21.491 Oracle Coherence GE 3.4.2/411 <D5> (thread=TcpRingListener, member=2)
: TcpRing: connecting to member 1 using TcpSocket{State=STATE_OPEN, Socket=Socket[addr=/10.1.100.98,
port=1685,localport=8089]}
2009-06-09 11:01:43.788/25.987 Oracle Coherence GE 3.4.2/411 <D5> (thread=Invocation:InvocationServi
ce, member=2): Service InvocationService joined the cluster with senior service member 1

*************************
OIPA Cycle Command Interface
Menu:
        P - Pause
        R - Resume
        D - Advance System Date
        A - Abort Current Cycle
        Q - Exit
        01 - Begin Pre-Company
        02 - Begin Pre-Plan
        03 - Begin Policy
        04 - Begin Post-Plan
        05 - Begin Post-Company
        06 - Begin Plan Status
        07 - Begin Pre-Client
        08 - Begin Post-Client
*************************
ENTER COMMAND \>03_
```

After running the menu options will appear
Enter the applicable command

Command line with Cycle command

Cycle Commands

- **`P Paused`** – Cycle will not process, although it will still check to see if activities should be processed, but 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 thru 08 are run.

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

- **`Q Exit`** – Exits cycle.

- **`01 Pre-Company`** - Runs only the pre-company portion of cycle.
  - 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 one-thousand or less.
  - A non-automatically overridden business error or a system error will cause cycle to abort further processing at this level and move onto the next task.

- **`02 Pre-Plan -`** Runs only the pre-plan portion of cycle.
  - 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 one-thousand or less.
  - Plans are executed in parallel using JMS threading.
  - A non-automatically overridden business error or a system error will cause cycle to move onto the next plan in the queue or the next processing level task if any.

- **`03 Policy`** - Runs the policy level portion of cycle processing.
  - 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), entry gmt (ascending).
  - Policies are processed in parallel using JMS threading.
  - A non-automatically overridden business error or a system error will cause cycle to abort further processing on the policy and move onto the next policy or processing task (if any).

- **`04 Post-Plan`** - Runs only the post-plan portion of cycle.
  - 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 one-thousand.
  - Plans are executed in parallel using JMS threading.
  - A non-automatically overridden business error or a system error will cause cycle to move onto the next plan in the queue or the next processing level task if any.

- **`05 Post-Company`** - Runs only the post-company portion of cycle.
  - 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 one-thousand.
  - A non-automatically overridden business error or a system error will cause cycle to abort further processing at this level and move onto the next task.

- **`06 Plan Status`** – Only used for V7 cycle.

- **`07 Pre-Client`** - Runs only the pre-client portion of cycle.
  - 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 one-thousand or **less**.
  - Clients are executed in parallel using JMS threading.
  - A non-automatically overridden business error or a system error will cause cycle to move onto the next client in the queue or the next processing level task if any.

- **`08 Post-Client`** - Runs only the post-client portion of cycle.
  - 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 one-thousand.
  - Clients are executed in parallel using JMS threading.
  - A non-automatically overridden business error or a system error will cause cycle to move onto the next client in the queue or the next processing level task if any.

Troubleshooting

Check List

1. Make sure that you change both the **jpa.databasePlatform** and **application.databaseType** properties in your Cycle.properties file to match **your** database. Often one or both of these can be overlooked, causing exceptions to be thrown.
2. Make sure you change the **datasource.type** property in your Cycle.properties file to **jndi** when running in a container. Otherwise, it will attempt to load the data source configuration from the pas-dataSource-dbcp-beans.xml and resource-dataSource-dbcp-beans.xml files.
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-config files are on the classpath. It is possible that the coherence-cache-config file that is referenced in the coherence-config file has to be fully qualified. This is the case with WebSphere, where you need to put the full path to the cycle-coherence-cache-config.xml. Other containers will find it fine on the classpath.

Common Errors

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 receive messages, and will not do anything. There are a few reasons why the Cycle Agent is not part of the same Coherence Cluster:

1. The cycle-coherence-config.xml file was not loaded.
   - Check the start-up parameters for the application server to ensure that the system property is passed in, it 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 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 therefore exist 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. You may need to update the cycle-coherence-config.xml file to include the fully qualified path to the cycle-coherence-cache-config.xml file. This change needs to be made to run inside of WebSphere on Linux.

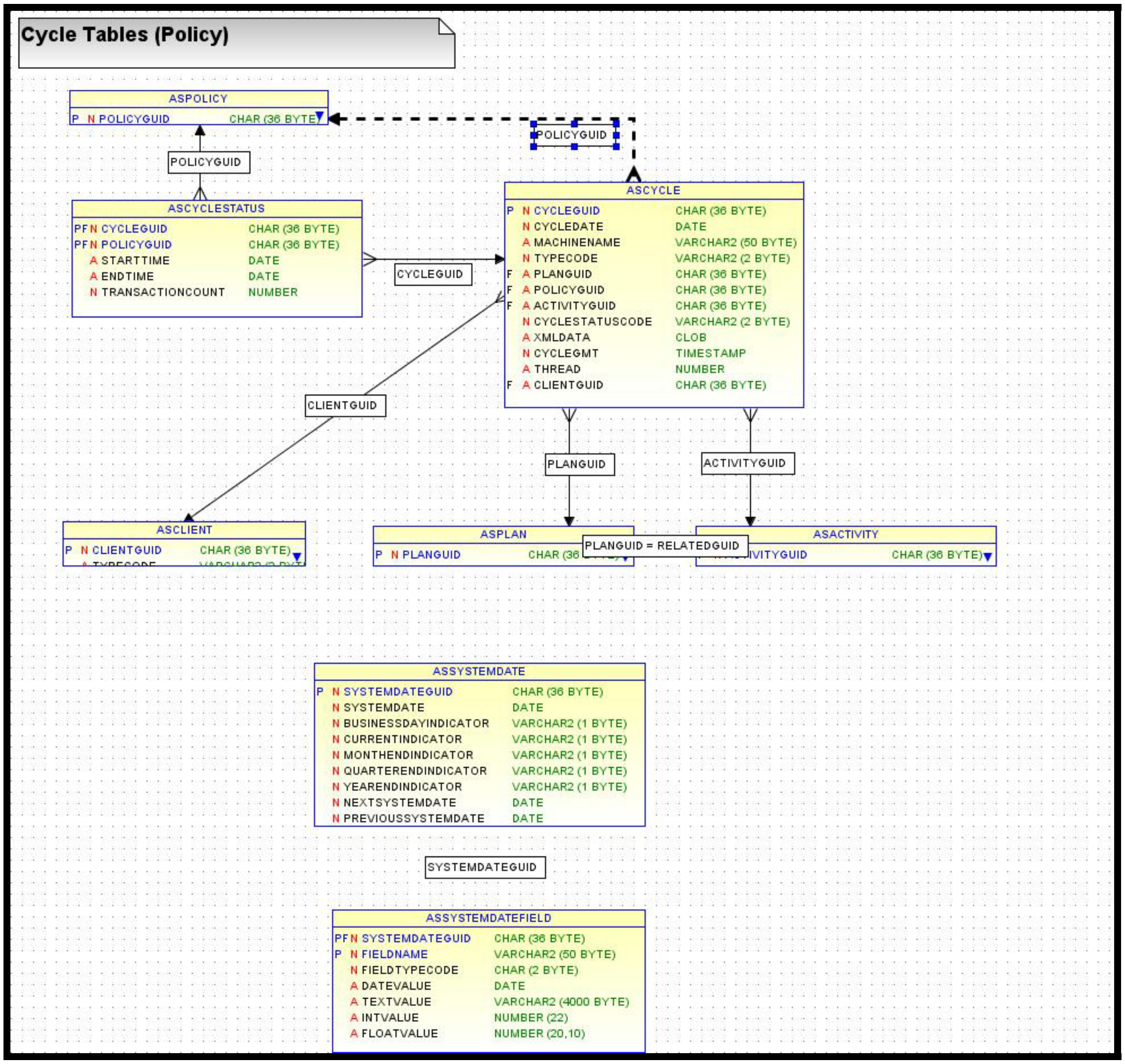
Cycle Tables

AsCycle

Name	Data type	Comment
CycleGUID	uniqueidentifier	Primary key
CycleDate	datetime	Current system date
MachineName	varchar(50)	Name of the machine running the activity
TypeCode	varchar(2)	From AsCode.CodeValue where CodeName = AsCodeCycleType 01:Company Level activities (<= 1000) 02:Plan Level activities (<= 1000) 03:Policy Level activities 04:Plan Level activities (> 1000) 05:Company Level activities (> 1000) 06:Plan level activity that ran and its detailed status
PlanGUID	uniqueidentifier	Link to AsPlan
PolicyGUID	uniqueidentifier	Link to AsPolicy
ActivityGUID	uniqueidentifier	Link to AsActivity
CycleStatusCode	varchar(2)	From AsCode.CodeValue where CodeName = AsCodeCycleStatus 01:Success 02:Pending 03:Business Error 04:System Error 05:Executing
XMLData	text	Holds stack dumps from errored activities
CycleGMT	datetime	Date/Time Cycle was executed
Thread	int	Identifies thread number on agent
ClientGUID	uniqueidentifier	Link to AsClient
CycleMemberID	Varchar(100)	Identifies assigned agent

AsCycleStatus

Name	Datatype	Comment
CycleGUID	uniqueidentifier	Link to AsCycle
PolicyGUID	uniqueidentifier	Link to AsPolicy
StartTime	datetime	Start time of the cycle
EndTime	datetime	End time of the cycle
TransactionCount	int	Number of transactions processed



ASPOLICY

P N	POLICYGUID	CHAR (36 BYTE)

ASCYCLESTATUS

PF N	CYCLEGUID	CHAR (36 BYTE)
PF N	POLICYGUID	CHAR (36 BYTE)
A	STARTTIME	DATE
A	ENDTIME	DATE
N	TRANSACTIONCOUNT	NUMBER

ASCYCLE

P N	CYCLEGUID	CHAR (36 BYTE)
N	CYCLEDATE	DATE
A	MACHINENAME	VARCHAR2 (50 BYTE)
N	TYPECODE	VARCHAR2 (2 BYTE)
F A	PLANGUID	CHAR (36 BYTE)
F A	POLICYGUID	CHAR (36 BYTE)
F A	ACTIVITYGUID	CHAR (36 BYTE)
N	CYCLESTATUSCODE	VARCHAR2 (2 BYTE)
A	XMLDATA	CLOB
N	CYCLEGMT	TIMESTAMP
A	THREAD	NUMBER
F A	CLIENTGUID	CHAR (36 BYTE)

ASCLIENT

P N	CLIENTGUID	CHAR (36 BYTE)
A	TYPECODE	VARCHAR2 (2 BYT

ASPLAN

P N	PLANGUID	CHAR (36

ASACTIVITY

	IVITYGUID	CHAR (36 BYTE)

ASSYSTEMDATE

P N	SYSTEMDATEGUID	CHAR (36 BYTE)
N	SYSTEMDATE	DATE
N	BUSINESSDAYINDICATOR	VARCHAR2 (1 BYTE)
N	CURRENTINDICATOR	VARCHAR2 (1 BYTE)
N	MONTHENDINDICATOR	VARCHAR2 (1 BYTE)
N	QUARTERENDINDICATOR	VARCHAR2 (1 BYTE)
N	YEARENDINDICATOR	VARCHAR2 (1 BYTE)
N	NEXTSYSTEMDATE	DATE
N	PREVIOUSSYSTEMDATE	DATE

ASSYSTEMDATEFIELD

PF N	SYSTEMDATEGUID	CHAR (36 BYTE)
P N	FIELDNAME	VARCHAR2 (50 BYTE)
N	FIELDTYPECODE	CHAR (2 BYTE)
A	DATEVALUE	DATE
A	TEXTVALUE	VARCHAR2 (4000 BYTE)
A	INTVALUE	NUMBER (22)
A	FLOATVALUE	NUMBER (20,10)

Cycle Tables