# Oracle® Application Integration Architecture

Pre-Built Integrations 11.1: Utilities Guide

Release 11.1

**E24458-04**

September 2012

Documentation for administrators that describes how to setup Session Pool Manager (SPM) and AIA CompositeScheduler. This guide provides configuration and deployment steps, property configurations and how to develop integrations to use SPM.

ORACLE

Oracle Application Integration Architecture Pre-Built Integrations 11.1: Utilities Guide, Release 11.1

E24458-04



Primary Author: Oracle Corporation

# Contents

## 2 AIA Composite Scheduler

## List of Examples

## List of Tables

# Preface

Welcome to *Oracle Application Integration Architecture Pre-Built Integrations 11.1: Utilities Guide*.

## What's New in this Guide

- The Oracle Application Integration Architecture Installation and Upgrade Guide for Pre-Built Integrations is restructured into a general installation chapter with an individual configuration and deployment chapter for each pre-built integration.
- The term *process integration pack* is replaced with the term *pre-built integrations*.
- The implementation guides are restructured into two parts: design and set up.

  Part I - Design: This part provides functional overviews, activity diagrams, assumptions and constraints, and technical sequence diagrams and steps.

  Part II - Set up: This part provides prerequisites, data requirements, and configuration steps.
- Starting with this release, these integrations are no longer available:
  - Oracle CRM On Demand Integration Pack for JD Edwards EnterpriseOne: Lead to Order
  - Oracle Workforce Administration Integration Pack for PeopleSoft Human Resources

## Common Oracle AIA Pre-Built Integration Guides

Oracle Application Integration Architecture Pre-Built Integrations 11.1 includes the following guides shared by all products delivered with this release:

- Oracle Application Integration Architecture Installation and Upgrade Guide for Pre-Built Integrations Release 11.1

  This guide provides an overview of the installation process, including how to install, configure, and deploy your pre-built integrations. The steps required to upgrade your pre-built integrations to the latest release are also provided.
- Oracle Application Integration Architecture Pre-Built Integrations 11.1: Utilities Guide

  This guide describes:
  - How to work with and configure Session Pool Manager (SPM), which is a service in the Oracle SOA Suite web server whose primary function is to manage a pool of web server session tokens that can be reused by BPEL flows.

  - How to deploy and configure the AIACompositeScheduler. This is a utility component that is used by pre-built integrations to schedule a service-oriented architecture (SOA) composite to be invoked at the specified time interval.

- Oracle Application Integration Architecture Pre-Built Integrations 11.1: Product-to-Guide Index

  The Product-to-Guide index lists the guides that provide information for each product delivered in this release.

# Documentation Accessibility

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

### Access to Oracle Support

Oracle customers 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.

# Additional Resources

The following resources are also available:

- **Oracle Application Integration Architecture Foundation Pack:**

  Oracle AIA Pre-Built integrations require Foundation Pack 11.1.1.5.0 to be installed. Refer to the Foundation Pack documentation library on OTN to download the Foundation Pack guides at http://download.oracle.com/docs/cd/E21764_01/aia.htm.

- **Oracle Application Integration Architecture: Product-to-Guide Index:**

  Oracle Technology Network:
  http://www.oracle.com/technetwork/index.html

- **Known Issues and Workarounds:**

  My Oracle Support: https://support.oracle.com/

- **Release Notes:**

  Oracle Technology Network:
  http://www.oracle.com/technetwork/index.html

- **Documentation updates:**

  Oracle Technology Network:
  http://www.oracle.com/technetwork/index.html

# 1 Session Pool Manager

This chapter introduces Session Pool Manager (SPM) and includes the following sections:

- Section 1.1, "Introduction to SPM"
- Section 1.2, "Working with SPM"
- Section 1.3, "Setting SPM Configuration Properties"
- Section 1.4, "Administering the AIASessionPoolManager Service on the SOA Server"
- Section 1.5, "Developing Integrations to Use the AIASessionPoolManagerService"

## 1.1 Introduction to SPM

SPM is a service in the Oracle SOA Suite web server whose primary function is to manage a pool of web server session tokens that can be reused by BPEL flows.

Reusing session tokens significantly enhances the overall performance of BPEL flows that call web services. This is because the session token creation process is a time-consuming operation in the application web server.

Some features of SPM include:

- SPM is automatically initialized upon the request of a session token.
- The session token pool is stored in memory.
- SPM supports concurrent (multi threaded) BPEL flows.
- SPM supports multiple application web server instances (hosts).
- SPM contains the Sentinel, which periodically checks session tokens, removes expired sessions, and replaces them with new ones.

## 1.2 Working with SPM

Understanding the functionality covered in this section helps you configure and tune SPM.

This section includes the following topics:

- Section 1.2.1, "Understanding SPM Configuration"
- Section 1.2.2, "Understanding the Initialization Process"
- Section 1.2.3, "Understanding the Get() Operation"

- Section 1.2.4, "Understanding the Sentinel Process"
- Section 1.2.5, "Understanding Statistics"
- Section 1.2.6, "Understanding Trace Logging to a CSV File"

### 1.2.1 Understanding SPM Configuration

SPM is configured using the AIAConfigurationProperties.xml file located in <aia.home>/config/. Its properties are located under **Module Configuration: SessionPoolManager.**

For more information about SPM configurations, see Section 1.3, "Setting SPM Configuration Properties".

### 1.2.2 Understanding the Initialization Process

The initialization process is a time-consuming operation because the SPM must perform the following tasks:

- Read settings in the AIAConfigurationProperties.xml file.
- Create pool tables in memory.
- Call the application web server to generate session tokens and store them in the pool, so that they are available for use.

  The number of session tokens that are initially loaded is determined by the PoolInitialLoad property. You can configure the wait interval between session token requests to the application web server by setting the ServerSessionRequestRate property. These properties are set in the SessionPoolManager module configuration in the AIAConfigurationProperties.xml file located in <AIA_HOME>/aia_instances/<aia_instance_name>/AIAMetadata/config/.
- Initialize and start the Sentinel.
- When SPM is initialized, the session token pool is stored in memory, ready to provide session tokens.

> **Note:** If the initialization process fails, SPM is set to a STOPPED state. Any concurrent and subsequent attempts to initialize SPM using the Get() operation fails. This prevents infinite loops or queuing up BPEL processes requesting a session token that fails. To reset this state, the administrator should call the Terminate() or Start() operation after fixing the problem.

### 1.2.3 Understanding the Get() Operation

SPM may follow multiple paths when you invoke the Get() operation. The paths taken depend on the pool state and the session token state.

If SPM has not been initialized, then the initialization process is invoked.

When SPM is initialized, it proceeds to get a session token from the pool table by way of the following paths.

#### 1.2.3.1 Getting an Available Session Token from the Pool Table

This is the first path taken. SPM's ability to take this path is possible when a session token is available in the pool table and SPM predicts that it is not expired. The printout enables SPM to immediately assign and pass the session token to the caller.

For more information about SPM's ability to predict the expiration status of a token, see Section 1.2.3.3, "Predicting an Expired Session Token".

If no session token is available in the pool table, SPM proceeds with the Getting a Session Token When One is Not Available in the Pool Table path.

#### 1.2.3.2 Getting a Session Token When One is Not Available in the Pool Table

When no session token available in the pool table, SPM waits some milliseconds, determined by the GetFromPoolTable_WaitInterval property, in anticipation that a session token becomes available. SPM then checks the pool table again. SPM continues to check and wait until the maximum number of attempts is reached. The maximum number of attempts is defined by the GetFromPoolTable_MaxAttempts property. These properties are set in the SessionPoolManager module configuration in the *AIAConfigurationProperties*.xml file located at <AIA_HOME>/aia_instances/<aia_instance_name>/AIAMetadata/config/.

If SPM successfully gets a session token from the pool table, the session token is checked for expiration, and if it is not expired, it is assigned and passed to the caller.

For more information about SPM's ability to predict the expiration status of a token, see Section 1.2.3.3, "Predicting an Expired Session Token".

When SPM is not able to get a session token from the pool table, then it proceeds to create a session token, if the number of existing session tokens in the pool does not match the maximum pool size. The maximum pool size is defined by the PoolSize_Max property.

If the number of existing session tokens in the pool matches the maximum pool size, the creation of a session token is denied, and the Get() operation throws a fault to the caller.

#### 1.2.3.3 Predicting an Expired Session Token

Session tokens can be expired due to their idle time or age on the application web server. To avoid providing an expired session token to a caller, SPM contains logic that tries to predict whether the session token has expired. The properties that provide the values used to predict session token expiration are PredictExpiration_Idle and PredictExpiration_Age

If SPM predicts that the session token has not expired, then it uses it.

If SPM predicts that the session token has expired, it renews it according to the following logic:

- If SPM predicts that the session token has expired due to its age, it calls the application web server to terminate the session token and calls it again to create a one.

  For more information about how SPM creates a session token, see Section 1.2.3.4, "Creating a Session Token".

- If SPM predicts that the session token has expired due to idle time, it calls the application web server to reset the idle time, and then assigns and passes it to caller.

- If resetting the idle time or creating a session token is not successful, SPM throws a fault to the caller.

#### 1.2.3.4 Creating a Session Token

This is a slow service.

SPM calls the application web server to get a session token. If the call is not successful, SPM waits some milliseconds, determined by the ConnectServer_WaitInterval property, and tries again. SPM repeats this wait-and-try logic until it obtains a session token, or the maximum number of attempts is reached. The maximum number of attempts is defined by the ConnectServer_MaxAttempts property.

Between attempts, SPM checks the pool table for an available session token. If one becomes available, it stops calling the application web server, assigns the available session token, and passes it to caller.

### 1.2.4 Understanding the Sentinel Process

You can optionally configure the initialization process to activate a process that prevents session tokens from expiring, while also keeping a minimum number of sessions in the pool table. This process is called the Sentinel.

Sentinel guards session tokens in the pool table and keeps them from expiring due to idle time or age. It uses the logic behind predicting session expiration to detect the tokens that must be renewed. Sentinel runs in its own low-priority thread, sleeping most of the time. It awakes every *x* milliseconds, determined by the PredictExpiration_Idle property, to check the session tokens in the pool table.

#### 1.2.4.1 Keeping Session Tokens Alive

The Sentinel resets the idle time of session tokens that it has predicted are expired. To reset the idle time, it calls the application web server that is passing the session token.

For those session tokens that it has predicted are expired due to age, the Sentinel terminates the session token, by calling the logoff operation of the application web server, and creates one for replacement by calling the application web server again.

The Sentinel terminates (removes) session tokens that have not been reused for some time. The Sentinel_Renew_Max configuration property controls this function.

SPM keeps a tally on the number of consecutive times the Sentinel renews a session token that has not been used between renewals. When the value of the Sentinel_Renew_Max property is reached for a session token, it is terminated (removed) from the pool table.

The activation of the Sentinel is also controlled by the Sentinel_Renew_Max configuration property.

For more information about the Sentinel_Renew_Max configuration property and activating the Sentinel, see Section 1.3.9, "Sentinel_Renew_Max".

#### 1.2.4.2 Keeping a Minimum Number of Session Tokens in the Pool Table

Each time the Sentinel awakes and finishes renewing expired session tokens, it checks the size of the pool table. If it is smaller than the value set in the Sentinel_PoolSize_Min property, the Sentinel replenishes the pool table with new session tokens up to this minimum value.

Before and during session token creation, the Sentinel checks if any Get() operations are trying to create session tokens. If yes, then the Sentinel stops replenishing session

tokens in the pool table. The next time the Sentinel awakes, it tries to replenish the remaining session tokens again, up to the Sentinel_PoolSize_Min property value. This logic should prevent the Sentinel from competing against business flows for application web server time.

The Sentinel creates session tokens sequentially, not concurrently, checking for active Get() operations between creating session tokens.

### 1.2.5 Understanding Statistics

SPM keeps cumulative tallies of the operations it has implemented and the actions it has taken. These statistics can help administrators optimize their SPM configurations. These statistics are available in the Pool Status report. You generate the report using the Status operation of AIASessionPoolManager.

For more information about generating the Pool Status report, see Section 1.4, "Administering the AIASessionPoolManager Service on the SOA Server".

SPM starts tallying statistics when an administrator calls the ActivateStatistics() operation and stops when the DeactivateStatistics() operation is called. By default, this feature is deactivated.

The tallies are not reset between activation and deactivation. They are reset when the ResetStatistics() operation is called or SPM is terminated.

**Note:** These tallies cannot provide exact counts because this functionality is not multi thread-safe. These statistics are solely meant to provide bulk data to help you tune SPM.

Table 1–1 shows SPM tallies statistics for the following operations:

***Table 1–1 Operations Statistics***

| Operation | Statistic |
|---|---|
| getSession | Count of get() session token operation invocations, regardless of the outcome. |
| getSession_Successful | Count of successful get() session token operations. |
| getSession_Failed | Count of failed get() session token operations. The primary causes for these failures include SPM not being able to get a session token from the application web server or no session tokens were available in the pool table and the pool size had reached the maximum pool size. Severe errors are printed in the logs with causes indicated. SPM uses the AIALogger APIs to write to the Oracle Application Integration Architecture trace log. The trace log can be viewed in Oracle Enterprise Manager.<br>Tuning tip: If the cause of the failure is that the maximum pool size was reached, increase the pool size in the SPM configuration to match or exceed the number of maximum threads in the BPEL server. |
| getSession_ReleaseAhead | Count of get() session token operations that were preceded by the invocation of the release operation. Two primary causes for this scenario include:<br>The application web server was slow.<br>Some SOA server threads used too much bandwidth, causing other threads "starve" and timeout.<br>In either case, the BPEL server sent a timeout to the client. The client knows that a session token is eventually assigned by SPM, so the client calls the release operation to prevent SPM from assigning a session token.<br>Tuning tip: Balance the number of active tasks in application web server with the number of active threads in the BPEL server, with the application web server number being greater than or equal to the BPEL server number. Then set the SPM maximum pool size property value to be at least the same as the number of threads in the BPEL server. |
| releaseSession | Count of invokes to the release() operation regardless of the outcome. |

***Table 1–1 (Cont.) Operations Statistics***

| Operation | Statistic |
|---|---|
| releaseSession_Successful | Count of successful release session token operations with the updated session token passed by the caller. |
| releaseSession_NoUpdST_Successful | Count of successful release session token operations without the updated session token passed by the caller. This does not indicate that a problem exists. |
| releaseSession_Failed | Count of failed release session token operations. This is rare. SPM may be terminated or terminating while a client tried to release a session token. Check BPEL server logs for clues. |
| releaseSession_NoUpdST_Failed | Count of failed release session token operations without an updated session token passed by the caller. Similar to releaseSession_Failed. Check BPEL server logs for clues. |
| releaseSession_Successful_OfaFailedGet | Count of release session token operations for which corresponding get session token operations failed. This is expected to be the same or less than the getSession_Failed value. |
| releaseSession_AheadOfGet | Count of release session token operations for which corresponding Get session token operations have not been completed or implemented. This scenario occurs when SPM takes too long to respond to the Get operation. Therefore, the BPEL server sends a timeout to the client. The client knows a session token is eventually assigned by SPM, so the client calls a Release operation to let SPM know that it should not assign a session token when the Get operation becomes active.<br>Tuning tip: See getSession_ReleaseAhead. |
| Sessions_Validated | Count of session tokens that were predicted to be expired, thus confirming that the session validation occurred. The validation outcome is unknown.<br>Tuning tip: Reducing the number of validations can help free up the application web server and prevents BPEL from sending timeouts to clients. Increase the Max Idle Session Token value in the application web server and set the PredictExpiration_Idle value in SPM to a longer idle time. |
| Sessions_Created | Count of session tokens that were created. Renewed session tokens are not included. |
| Sessions_Discarded | Count of session tokens that were discarded, primarily because they could not be validated. This may be an indication that the application web server may be overloaded. Check BPEL logs for clues.<br>Tuning tip: Reduce the number of threads in the BPEL server and SPM maximum pool size accordingly. |
| Sentinel_SessionIdleTime_Refreshed | Count of session tokens for which idle times were refreshed by the Sentinel.<br>Tuning tip: See Sessions_Validated. |
| Sentinel_SessionAge_Renewed | Count of session tokens that the Sentinel renewed because it predicted that the session was expired. The over-aged session is terminated on the application web server and a new one is created. |
| Sentinel_SessionDiscarded_NotReused | Count of session tokens that were discarded by the Sentinel because they reached the maximum number of consecutive renewals. |
| Sentinel_SessionDiscarded_Error | Count of session tokens that the Sentinel discarded because their idle time could not be reset or because a new session token could not be obtained from the application web server.<br>Tuning tip: See Sessions_Discarded. |

## 1.2.6 Understanding Trace Logging to a CSV File

SPM logs the operations being implemented in the session pool in comma-separated values (CSV) files. You can use the data in these CSV files as a troubleshooting tool.

To enable trace logging in CSV files, set the TraceToAIA-SPM-CSVFile_Enabled property to **TRUE**.

For more information about the TraceToAIA-SPM-CSVFile_Enabled property, see Section 1.3.17, "TraceToAIA-SPM-CSVFile_Enabled".

The trace log CSV files are located in the <home>Middleware/user_projects/domains/soa_domain/servers/soa_server1/logs directory. The CSV file naming convention is *aia-spm-<HostId-Time>*.csv, where **HostId** is the value of the host ID and **Time** is the date and time at which the CSV file was created. For example, the directory may contain the following files:

- aia-spm-CRMOD_01-20110617080010.csv

- aia-spm-CRMOD_01-20110617143000.csv

A new CSV file is created after every 10,000 records logged. Terminating the SPM causes the closure of the current CSV file.

Table 1–2 shows data captured by the trace log CSV file.

***Table 1–2 Data Description of CSV File***

| Column | Description |
|---|---|
| Date-Time | Date and time when the operation was implemented. |
| Client Instance Id | Instance ID passed in Get or Release operations.<br>A value of Sentinel indicates that the operation was implemented by Sentinel functionality. |
| Operation | Operation implemented. |
| Record ID | ID of the session token used for the operation. |
| Session Token | Session token value. |
| Created Date | Date and time when the session token was created. A value of N/A stands for Not Available. |
| Age (milliseconds) | Age of the session token in milliseconds. |
| Assigned Since | Date and time when the session token was assigned to the client. A value of N/A stands for Not Available, meaning the session token is not assigned. |
| Idle Since | Date and time when the session token moved into an idle state. A value of N/A stands for Not Available, meaning the session token is not in an idle state. |
| Renewed by Sentinel | Number of times the Sentinel has renewed the session token. |

Table 1–3 shows the operations SPM logs in the trace log CSV file.

***Table 1–3 Operations in Trace Log CSV File***

| Opertaion | Description |
|---|---|
| getSession() | Session token assigned to requester. |
| releaseSession() | Session token returned to the pool for reuse. |
| releaseInvalidSessionToken() | Session token discarded from the session pool table because the client reported an Invalid Session error code. |
| sessionTerminated() | Session token terminated and removed from the session pool table because the SPM is being terminated. |
| sessionCreated() | Session token added to the pool by the SPM Start operation. |
| SessionIdleTime_Refreshed() | The Sentinel validated the session token against the application server. The outcome of the validation can be that the idle time was reset, or a new session token was created because the previous one had expired. |
| SessionDiscarded_Error() | An unexpected error occurred while the Sentinel was validating the session token against the application server. The primary reason for this error is that the SPM was not able to connect with the application server, so the Sentinel removed the session token from the SPM pool table. |
| SessionDiscarded_NotReused() | Session token reached the renewal limit allowed by the Sentinel so the Sentinel terminated the session token and removed it from the SPM pool table. |
| AgedSession_Terminated() | Session token reached its maximum age so the Sentinel terminated the session in the application server and removed the session token from the SPM pool table. |
| SessionAge_Replaced() | The Sentinel created this new session token to replace a session token terminated due the age. |
| SessionAdded() | The Sentinel added this session to replenish the minimum number of session tokens in the pool threshold. |

## 1.3 Setting SPM Configuration Properties

SPM is automatically configured when a integration utilizing SPM is installed. SPM is installed with a default configuration. When implementing or administering the

integration, these configurations can be changed to tune the configurations to better suit your integration environment and the needs of the implemented integration.

SPM configuration properties are set in the *AIAConfigurationProperties*.xml file as a Module Configuration. The module name is SessionPoolManager.

SPM has the following configurable properties:

- all_hosts
- PoolSize_Max
- PoolInitialLoad
- ConnectServer_MaxAttempts
- ConnectServer_WaitInterval
- ServerSessionRequestRate
- GetFromPoolTable_MaxAttempts
- GetFromPoolTable_WaitInterval
- PredictExpiration_Idle
- PredictExpiration_Age
- TRACE.LOG.ENABLED
- Sentinel_Renew_Max
- Sentinel_PoolSize_Min
- Sentinel_LogLevel
- UserId
- Password
- EndpointURI
- ClassName
- InvalidSessionErrorCodes
- TraceToAIA-SPM-CSVFile_Enabled
- ProxySettings_Enabled
- ProxyHost
- ProxyPort

Defaults can be defined for all application web servers by using the all_hosts prefix value.

For example, the following line defines 40 as the default maximum pool size:

```
<Property name="all_hosts.PoolSize_Max">40</Property>
```

Specific values can be defined for individual application web servers by using the HostId prefix.

For example, the following line defines 50 as the maximum pool size value for the CRM On Demand application web server, where the HostId for CRM On Demand is CRMOD_01:

```
<Property name="CRMOD_01.PoolSize_Max">50</Property>
```

All properties must be defined by application web server or default. If a property is not defined for a specific application web server, then the default property (all_hosts) is used. If no all_hosts default property is defined, the caller receives a fault indicating the missing property.

SPM can work with multiple hosts (application web servers). Therefore, each property can be set as a default for all hosts, and overridden for a specific host. The only exception is the TRACE.LOG.ENABLED property, which cannot be set to be server-specific.

Each property has a prefix that indicates the application web server. For example:

```
<Property name="all_hosts.PoolSize_Max">40</Property>

<Property name="SBL_03.PoolSize_Max">20</Property>
```

The first line defines 40 as the default maximum pool size for all hosts.

The second line overrides the default pool size to 20 for the application web server SBL_03.

The concept of system ID and HostId are synonymous.

For example, a customer installing an integration for Siebel may use SEBL_01 as the system ID for the Siebel application web server. They see SEBL_01 in the *AIAConfigurationProperties*.xml file as the Default.SystemID property for the services connecting to the Siebel application web server. This SEBL_01 value should also be used as the HostId value in SPM to refer to the Siebel application web server.

Another customer installing a integration for CRM On Demand may use CRMOD_01 as the system ID for the CRM On Demand application web server. Likewise, they should use CRMOD_01 as the HostId value in SPM to refer to the CRM On Demand application web server.

Example 1–1 is an example of an SPM module configuration:

***Example 1–1 SPM Module Configuration***

```
  <ModuleConfiguration moduleName="SessionPoolManager">
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    AIA Session Pool Manager
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <!-- === Default values: all_hosts === -->
   <!-- List Host IDs that can have an SessionPoolManager-->
   <Property name="all_hosts">SEBL_01 CRMOD_01</Property>
   <Property name="all_hosts.PoolSize_Max">40</Property>
   <Property name="all_hosts.PoolInitialLoad">1</Property>
   <Property name="all_hosts.GetFromPoolTable_MaxAttempts">5</Property>
   <!--GetFromPoolTable_WaitInterval:  Milliseconds -->
   <Property name="all_hosts.GetFromPoolTable_WaitInterval">250</Property>
   <Property name="all_hosts.ConnectServer_MaxAttempts">2</Property>
   <!--ConnectServer_WaitInterval:  Milliseconds -->
   <Property name="all_hosts.ConnectServer_WaitInterval">500</Property>
   <!-- PredictExpiration_Idle: Milliseconds -->
   <!-- 780000 = 13 Minutes -->
   <Property name="all_hosts.PredictExpiration_Idle">780000</Property>
   <!--PredictExpiration_Age:  Milliseconds-->
   <!-- 82800000 = 23 Hours -->
   <Property name="all_hosts.PredictExpiration_Age">82800000</Property>
   <Property name="all_hosts.Sentinel_PoolSize_Min">4</Property>
   <!--Sentinel_Renew_Max:  -1 means do not start it. 0 means always renew; >0 max
renew. -->
   <Property name="all_hosts.Sentinel_Renew_Max">2</Property>
```

```
   <!-- ServerSessionRequestRate: milliseconds. -->
   <!-- 50 = 1/20th second wait time between requests.-->
   <Property name="all_hosts.ServerSessionRequestRate">50</Property>
   <!-- InvalidSessionErrorCodes: Regular expressions accepted. -->
   <Property name="all_hosts.InvalidSessionErrorCodes">404</Property>
   <!-- Login -->
   <Property name="TRACE.LOG.ENABLED">true</Property>
   <Property name="all_hosts.Sentinel_LogLevel">ERROR</Property>
   <!-- Enable/Disable logging SPM operations into aia-spm-host-time.csv file.
         "true" enables logging.
         The location of the CSV file is SOA_HOME/opmn/logs folder.
         This property is independent of other logging properties-->
   <Property name="all_hosts.TraceToAIA-SPM-CSVFile_Enabled">FALSE</Property>

   <!-- === -->
   <!-- === Specific values for SEBL_01 === -->
   <Property name="SEBL_01.UserId">SiebelUser1</Property>
   <Property name="SEBL_01.Password">TheEncryptedPassword</Property>
   <Property
name="SEBL_01.EndpointURI">http://mySiebel:8080/eai_enu/start.swe?SWEExtSource=Sec
ureWebService&amp;SWEExtCmd=Execute&amp;WSSOAP=1</Property>
   <Property name="SEBL_01.GetFromPoolTable_MaxAttempts">3</Property>
   <Property name="SEBL_01.GetFromPoolTable_WaitInterval">300</Property>
   <Property
name="SEBL_01.ClassName">oracle.apps.aia.core.sessionpool.CRMSiebelSession</Proper
ty>
   <Property
name="SEBL_01.InvalidSessionErrorCodes">10944642|SBL-BPR-00162|SBL-DAT-00175</Prop
erty>
   <!-- === -->
   <!-- Specific values for CRMOD_01 -->
   <Property name="CRMOD_01.UserId">MyCompany/CRMUser1</Property>
   <Property name="CRMOD_01.Password">MustBeEncrypted</Property>
   <!-- CRMOD_01.EndpointURI example: http://xxx.siebel.com/Services/Integration
-->
   <Property
name="CRMOD_01.EndpointURI">http://xxx.siebel.com/Services/Integration</Property>
   <Property name="CRMOD_01.GetFromPoolTable_MaxAttempts">3</Property>
   <Property name="CRMOD_01.GetFromPoolTable_WaitInterval">1000</Property>
   <Property name="CRMOD_01.PredictExpiration_Idle">120000</Property>
   <Property name="CRMOD_01.PredictExpiration_Age">600000</Property>
   <Property name="CRMOD_01.ServerSessionRequestRate">50</Property>
   <Property name="CRMOD_01.InvalidSessionErrorCodes">SBL-ODU-01006</Property>
   <Property
name="CRMOD_01.ClassName">oracle.apps.aia.core.sessionpool.CRMOnDemandSession</Pro
perty>
   <!-- === -->
   <!-- Specific values for NOSERVER.
        Class NoSvrCRMSiebelSession emulates a server with random errors and
successes.
        Use it for simulation/testing. .-->
   <Property name="NOSERVER.UserId">fakeusername</Property>
   <Property name="NOSERVER.Password">26bf4moKoU0=</Property>
   <Property name="NOSERVER.EndpointURI">http://any.fake.value.is/good</Property>
   <Property
name="NOSERVER.ClassName">oracle.apps.aia.core.sessionpool.NoSvrCRMSiebelSession</
Property>
  </ModuleConfiguration>
 </SystemConfiguration>
</AIAServiceConfiguration>
```

Whenever the *AIAConfigurationProperties*.XML file is updated, the file must be reloaded to SOA-MDS for updates to be reflected in the applications or services that use the updated properties.

For more information about how to update the *AIAConfigurationProperties*.xml file, see *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*, "Building AIA Integration Flows" and "How to Set Up AIA Workstation".

### 1.3.1 All_hosts

SPM can work with multiple application web server instances.

In this property, list the hosts for which SPM can create a session token pool. Separate the host names by spaces. Each host has its own pool.

This property is not prefixed with a HostId value.

### 1.3.2 PoolSize_Max

It limits the number of session tokens that the pool can have.

When the Get() operation is invoked, the SPM creates a session token if none is available and adds it to the pool. If the number of session tokens in the pool has reached this property value, the Get() operation returns a fault error indicating that no session tokens are available and that it cannot create a new one because the pool size has reached its maximum value.

We recommend that you set this value to match the *Dispatcher Invoke Threads* value set for the BPEL server.

In addition, setting the BPEL server *Dispatcher Invoke Threads* value to be the same or lower than the maximum number of tasks that the application web server allows is recommended. For example, for Siebel, this is the value of the App Max Tasks parameter.

### 1.3.3 PoolInitialLoad

It indicates the number of session tokens to be initially loaded into the pool and made available when the pool is initialized for the first time.

Subsequently, if additional session tokens are needed, they are added one per Get() operation.

### 1.3.4 ConnectServer_MaxAttempts and ConnectServer_WaitInterval

When creating session tokens, SPM has the logic to retry connecting to the application web server when the first attempt fails. These two properties determine the maximum number of attempts SPM makes to connect and the wait time between the failed attempt and its next attempt.

- ConnectServer_MaxAttempts

  Determines the maximum number of attempts SPM makes to connect. The minimum value should be 1.

- ConnectServer_WaitInterval

  Determines the amount of time in milliseconds that SPM waits between the failed attempt and its next attempt. The value should be the same as the value you set for

the ServerSessionRequestRate property, which helps ensure that the host does not mistake the connection attempts for a server attack.

For more information, see Section 1.3.5, "ServerSessionRequestRate".

### 1.3.5 ServerSessionRequestRate

It determines the amount of time in milliseconds that SPM waits between making calls requesting a session token. This property is used to slow down successful connections requests to help prevent the host from mistaking the request calls for a server attack.

For example, CRM On Demand expects a 50 millisecond wait time between requests.

### 1.3.6 GetFromPoolTable_MaxAttempts and GetFromPoolTable_WaitInterval

If no session tokens are available in the pool for the Get() operation, SPM waits some milliseconds in anticipation that a process releases a session token and then tries to get it. If after *n* number of attempts without a session token being released in the pool, SPM proceeds to call the application web server to create a session token.

These properties determine the maximum number of attempts and wait time SPM uses when getting a session token from the pool before proceeding to call the application web server to create a session token.

- GetFromPoolTable_MaxAttempts

  Determines the maximum number of attempts SPM makes to obtain a session token from the pool. The minimum value should be 1.

- GetFromPoolTable_WaitInterval

  Determines the amount of time in milliseconds that SPM waits between attempting to obtain a session token from the pool.

### 1.3.7 PredictExpiration_Idle and PredictExpiration_Age

Usually, session tokens can expire due to idle time or age on the application web server. To prevent providing an expired session token to a Get() operation, SPM uses logic that tries to predict session token expiration. These properties provide values for predicting session token expiration.

- PredictExpiration_Idle:

  Indicates the maximum time in milliseconds that a session token can be idle before expiring. We recommend that you set this value to a value lower than the actual maximum idle time configured for the application web server. We recommend a value lower than the actual value to compensate for the gap between the time at which the application web server responded and the time at which the BPEL flow called SPM to release the session token.

- PredictExpiration_Age:

  Indicates the maximum age in milliseconds that a session token can reach before expiring. We recommend that you set this value to a value lower than the actual maximum age configured for the application web server. The creation time registered in the application web server is some seconds earlier than the one registered in SPM. A value of 1 or 2 minutes is a good start. For example, if the maximum age configured on the application web server is 15 minutes, set this property to 13 minutes.

### 1.3.8 TRACE.LOG.ENABLED

To enable trace logging for SPM, set this property to **TRUE**.

SPM uses the AIALogger APIs to write to Oracle Application Integration Architecture trace log. The trace log can be viewed in Oracle Enterprise Manager.

For more information, see *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*, "Using Trace and Error Logs."

### 1.3.9 Sentinel_Renew_Max

It determines the maximum consecutive number of times a session token can be consecutively renewed by the Sentinel.

SPM tallies the consecutive number of times the Sentinel has renewed a session token. A consecutive renewal is defined as a renewal of a session token by the Sentinel without any Get() + Release() operations between the previous and latest renew.

This tally is reset to zero each time the session token is acted upon by the Release() operation

When a session token reaches the maximum number of consecutive renews, the session token is terminated in the application web server and it is removed from the pool table.

For more information, see Section 1.3.10, "Sentinel_PoolSize_Min".

Table 1–4 shows sample values.

***Table 1–4 Sample Values***

| Sentinel_Renew_Max Value | Usage |
|---|---|
| -1 | Do not activate the Sentinel. |
| 0 | No maximum number of consecutive renewals. Session tokens can be renewed an infinite number of times. |
| 1, 2, 3 and so on | Maximum number of consecutive renewals. |

### 1.3.10 Sentinel_PoolSize_Min

It determines the minimum number of session tokens the Sentinel keeps in the pool table.

This functionality is activated only when Sentinel is activated.

### 1.3.11 Sentinel_LogLevel

It determines the trace logging level for the Sentinel. By default, only SEVERE messages are written into the log. Sentinel uses the AIALogger APIs for writing to the trace logs. The trace log can be view in Oracle Enterprise Manager. The TRACE.LOG.ENABLED property must be set to **TRUE** for this property to work.

For more information, see Section 1.3.8, "TRACE.LOG.ENABLED".

The Sentinel_LogLevel value should be a java.util.logging.Level value. These are the levels in descending order:

- SEVERE (highest value) (default)
- WARNING
- INFO

- CONFIG
- FINE
- FINER
- FINEST (lowest value)

For more information, see *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*, "Using Trace and Error Logs."

### 1.3.12 UserId

It determines the user ID that is used to connect to the application web server.

### 1.3.13 Password

It determines the password that is used to connect to the application web server. This value should contain the XPATH into AIAInstallProperties.xml. Following are some examples:

- For CRM On Demand:

```
<Property
name="CRMOD_01.Password">participatingapplications.lto.crmod.ws.password</Prope
rty>
```

- For CRM Siebel:

```
<Property
name="SEBL_01.Password">participatingapplications.siebel.server.eai.password</P
roperty>
```

- For AECM Siebel:

```
<Property
name="SEBL_01.Password">participatingapplications.aecm.server.eai.password</Pro
perty>
```

For more information about how to update the password, see *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*, "Post Install Configurations" and "Modifying Passwords Used for AIA Deployments".

### 1.3.14 EndpointURI

It determines the endpoint URI that uses to connect to the application web server. Following are some examples:

- For Siebel application web server:

```
http://mySiebel.com/eai_enu/start.swe?SWEExtSource=SecureWebService&amp;SWEExtC
md=Execute&amp;WSSOAP=1
```

- For CRM On Demand application web server:

```
http://myCRMOD.com/Services/Integration
```

---

**Note:** *AIAConfigurationProperties*.xml content is XML sensitive. Therefore, if the URI contains **&**, use **&** to represent it.

---

### 1.3.15 ClassName

It determines the full class name that SPM uses to get the session tokens from the application server. The class listed in this property implements the oracle.apps.aia.core.sessionpool.PoolableResource interface.

Table 1–5 shows the list of the host types and the class names.

***Table 1–5 Class Names***

| Host Type | Delivered Class Name |
|---|---|
| Siebel On Premises | oracle.apps.aia.core.sessionpool.CRMSiebelSession |
| Host simulator (for testing purposes) | oracle.apps.aia.core.sessionpool.NoSvrCRMSiebelSession |
| CRM On Demand | oracle.apps.aia.core.sessionpool.CRMOnDemandSession |

### 1.3.16 InvalidSessionErrorCodes

It determines the list of error codes that the application web server can return for a fault when the session token is not valid.

You can use regular expressions to set up multiple error codes or patterns. For example, if for application web server *XXX_01* the error codes are *inv-300*, *exp-301* and *dny-303*; the property can be set as follows:

```
<Property
name="XXX_01.InvalidSessionErrorCodes">inv-300|exp-301|dny-303</Property>
```

For more information about Regular Expressions, see http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html.

The following error codes are configured when SPM is installed:

- For CRM On Demand application web server:

  ```
  <Property name="CRMOD_01.InvalidSessionErrorCodes">SBL-ODU-01006</Property>
  ```

- For Siebel application web server:

  ```
  <Property
  name="SEBL_01.InvalidSessionErrorCodes">10944642|SBL-BPR-00162|SBL-DAT-00175</P
  roperty>
  ```

### 1.3.17 TraceToAIA-SPM-CSVFile_Enabled

To enable SPM trace logging in CSV files, set this property to **TRUE**.

Set this property to **FALSE** to disable trace logging in CSV files.

This property works independent of the other trace logging properties.

For property value changes to take effect, ensure that you terminate and restart the SPM for the host ID.

For more information, see Section 1.2.6, "Understanding Trace Logging to a CSV File".

### 1.3.18 ProxySettings_Enabled

To enable SPM use proxy settings while calling the application web server, set this property to **TRUE**.

Set this property to **FALSE** to not use proxy settings.

### 1.3.19 ProxyHost

It determines the server to be set in the system properties for http.proxyHost property.

In the `java.net` API used by SPM, proxies are supported through two system properties: http.proxyHost and http.proxyPort. They must be set to the proxy server and port respectively.

This value is only set when ProxySettings_Enabled is set to **TRUE**.

### 1.3.20 ProxyPort

It determines the port to be set in the system properties for the http.proxyPort property.

In the `java.net` API used by SPM, proxies are supported through two system properties: http.proxyHost and http.proxyPort. They must be set to the proxy server and port respectively.

This value is only set when ProxySettings_Enabled is set to **TRUE**.

### 1.3.21 Describing Recommended Configuration Settings for Siebel Web Server and SPM

To help optimize the performance of SPM with your Siebel web server, balancing your Siebel web server and SPM configuration settings according to the information is this section is recommended.

Table 1–6 shows the Siebel web server parameters and the corresponding SPM configuration properties.

***Table 1–6 Values to Configure Siebel Web Server***

| Siebel Web Server Parameter | SPM Configuration Property |
|---|---|
| SessionTimeout | N/A |
| SessionTokenTimeout | PredictExpiration_Idle |
| SessionTokenMaxAge (session token maximum age) | PredictExpiration_Age |

For more information about setting Siebel web server parameters, see "Session and Session Token Timeout-Related Parameters" in *Integration Platform Technologies: Siebel Enterprise Application Integration* available at `http://download.oracle.com/docs/cd/B40099_02/books/EAI2/EAI2_WebServices32.html`.

For more information about these SPM configuration properties, see Section 1.3.7, "PredictExpiration_Idle and PredictExpiration_Age".

### 1.3.22 SessionTimeout

The Siebel web server SessionTimeout parameter value should be set to a low value.

The value should not be so low that sessions are created too frequently. The value should also not be so high that adapter and database connections drop before sessions expire, and thus creating vulnerabilities.

A SessionTimeout parameter value of 300 seconds (5 minutes) is a good starting point.

### 1.3.23 SessionTokenTimeout and PredictExpiration_Idle

The Siebel web server SessionTokenTimeout parameter value should be set to a value that is greater than the SessionTimeoutLength value.

The parameter value is set to 15 minutes by default and is a good starting point.

In SPM, the PredictExpiration_Idle configuration property value should be set to a value less than the Siebel web server SessionTokenTimeout parameter value. A PredictExpiration_Idle configuration property value of 720,000 milliseconds (12 minutes) is a good starting point for a SessionTokenTimeout parameter value of 15 minutes.

### 1.3.24 SessionTokenMaxAge and PredictExpiration_Age

The Siebel web server SessionTokenMaxAge parameter value should be set to a value that is greater than the SessionTokenTimeout value. The parameter value is set to 2880 minutes (2 days) by default and is a good starting point.

If you choose to lower the value, just ensure that you do not set it to a value that is lower than the SessionTokenTimeout value.

In SPM, the PredictExpiration_Age configuration property should be set to a value less than the Siebel web server SessionTokenMaxAge parameter value. 82,800,000 milliseconds (23 Hours) is a good starting point.

### 1.3.25 SWSENoSessInPref

The Siebel web server SWSENoSessInPref parameter should be set to **TRUE**.

### 1.3.26 Example Configurations for Siebel Web Server and SPM

Based on the recommendations discussed in this section, here are example configurations for the Siebel web server and SPM.

#### 1.3.26.1 Siebel Web Server

- SessionTimeout = *300* (in seconds, equivalent to 5 minutes)
- SessionTokenTimeout = *15* (in minutes)
- SessionTokenMaxAge = *2880* (in minutes, equivalent to 2 days)
- SWSENoSessInPref = *TRUE*

#### 1.3.26.2 SPM

- PredictExpiration_Idle = *720000* (12 minutes)
- PredictExpiration_Age = *82800000* (23 Hours)

## 1.4 Administering the AIASessionPoolManager Service on the SOA Server

This section discusses how to run the following AIASessionPoolManager administrator operations as needed on the SOA server:

- Start(string HostId)
- Terminate(string:HostId)

- Status(string HostId)
- TerminateSentinel(string HostId)
- RestartSentinel(string HostId)
- ActivateStatistics(string HostId)
- ResetStatistics(string HostId)
- DeactivateStatistics(string HostId)

This section also describes scenarios that require manual intervention.

## 1.4.1 Running AIASessionPoolManager Administrator Operations

To run the administrator operations, call the AIASessionPoolManager web service. You can use the test web service functionality provided by Oracle Enterprise Manager Fusion Middleware Control.

To call the AIASessionPoolManager web service through the test web service functionality provided by Oracle Enterprise Manager Fusion Middleware Control:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. Search for **AIASessionPoolManager**.
3. Click **Test**.
4. Select the required operation and provide the relevant value in the **HostID** field.
5. Click **Test Web Service**. The outcome appears.

### 1.4.1.1 Start(string HostId)

- Initializes SPM.
- Intended to be used by: the SOA server administrator can use this operation to manually start SPM.
- Caller responsibilities: none.

  The HostId element is optional. If not specified, an SPM is started each host.
- Implementation details:
  - SPM creates an instance in the SOA server if it has not been previously initialized.
  - SPM reads the configuration values and loads session tokens into the pool.

### 1.4.1.2 Terminate(string:HostId)

- Terminates SPM.

  **Note:** Use care when using this operation.

- Intended to be used by: the SOA server administrator can use this operation to terminate SPM.
- Caller responsibilities: SPM is terminated immediately.

**WARNING: Ensure that no BPEL flows that are requesting or releasing session tokens are currently running. Forced termination of any concurrent Get() or Release() operation is aborted and a fault is thrown to each caller.**

The HostId element is optional. If not specified, SPMs running on all hosts are terminated.

- Implementation details:
  - SPM aborts any Get() or Release() operations currently running.
  - For each session token in the pool, SPM calls the application web server to terminate it regardless of its In UseAvailable or state.
  - Clears statistic values and the pool.
  - After termination, the next Get() or Start() operation initializes SPM.

#### 1.4.1.3 Status(string HostId)

- Provides an SPM text status report.

  The report includes SPM state, configuration values, current pool size, some operation statistics, session token idle times and ages, and Available or In Use state.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: none.

  The HostId element is optional. If not specified, all hosts are reported.
- Implementation details:

  Regardless of the state of the Statistics property (Active or Deactivated), the current statistics are reported.

#### 1.4.1.4 TerminateSentinel(string HostId)

- Terminates the Sentinel daemon thread for the specified HostId.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: none.

  The HostId element is required.
- Implementation details:

  SPM stops the daemon thread in which the Sentinel is running.

#### 1.4.1.5 RestartSentinel(string HostId)

- Restarts the Sentinel daemon thread for the specified HostId.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: none.

  The HostId element is required.
- Implementation details:

  SPM creates a daemon thread on which to run Sentinel.

#### 1.4.1.6 ActivateStatistics(string HostId)

- Activates the functionality that tallies operations performed by SPM.
- Only operations that have statistics with a value of 1 or greater are displayed by the Status() operation.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: to improve SPM performance, deactivate statistics functionality when they are no longer needed.

  The HostId element is optional. If not specified, statistics are activated for SPMs on all hosts.
- Implementation details: by default, this functionality is deactivated.

> **WARNING: The logic for these statistics is not multi thread safe. Therefore, some tallies may occasionally miss one or two counts. The reason for not making the logic multi thread safe is to avoid decreasing performance of SPM.**

#### 1.4.1.7 ResetStatistics(string HostId)

- Resets the statistics counts for SPM.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: none.

  The HostId element is optional. If not specified, statistics are reset for SPMs on all hosts.
- Implementation details: all statistic counts are reset to zero.

#### 1.4.1.8 DeactivateStatistics(string HostId)

- Deactivates the statistics functionality that tallies operations performed by SPM.
- Intended to be used by: SOA server administrator.
- Caller responsibilities: to improve SPM performance, deactivate the statistics functionality when they are no longer needed.

  The HostId element is optional. If not specified, statistics functionality is deactivated for SPMs on all hosts.
- Implementation details:
  - By default, this functionality is deactivated.
  - SPM stops tallying the operations implemented, but does not reset current counts.

### 1.4.2 Describing Scenarios Requiring Manual Intervention

When any of these scenarios arises, an administrator must manually terminate SPM for the host IDs in place. This cleans up invalid session pool tokens and prevents leaving open session tokens on the application web server. While Sentinel can handle this clean-up automatically for most scenarios, it cannot handle the following scenarios.

- Restarting the application web server instance invalidates session tokens in SPM for the restarted application web server instance.

You must terminate the AIASessionPoolManager service for the restarted application web server instance.

- Before shutting down the SOA server, you must terminate the SPM for all host IDs, otherwise existing session tokens in SPM are left open on the application web server.
- Before redeploying or reinstalling any of SPM components, you must terminate SPM for all host IDs, otherwise existing session tokens in SPM are left open on the application web server.

For more information about terminating SPM, see Section 1.4.1.2, "Terminate(string:HostId)".

## 1.5 Developing Integrations to Use the AIASessionPoolManagerService

The AIASessionPoolManager web service is the interface to SPM. If the integration you are developing, interfaces with a Siebel instance and can benefit from using SPM, you can develop your integration to call SPM client operations that get and release session tokens.

This section includes the following topics:



### 1.5.1 Get(string HostId, string InstanceId)

- Provides a session token with a high probability that the session token is not expired. If no session token is available, a fault is thrown.
- Intended to be used by: BPEL flows calling application web services. These BPEL flows are referred as callers.
- Caller responsibilities: the caller shall call the Release() operation immediately after it has finished using the session token.

  Table 1–7 shows a list of the caller responsibilities.

  For more information, see Section 1.5.2, "Release(string HostId, string InstanceId, string UpdatedSessionToken, string ErrorCode)".

***Table 1–7 Caller Responsibilities Details***

| Caller | Required (Yes/No) | Description |
|---|---|---|
| Hostid | Yes | The caller must pass the system ID that identifies the application web service in the SessionPoolManager module configuration in AIAConfigurationProperties.xml. For example, *SEBL_01* or *CRMOD_01*. |
| InstanceId | Yes | The caller must pass a unique ID for the current caller instance. You can use the BPEL XPath function **ora:getInstanceId()**.<br>The caller should implement logic to catch an "invalid/expired session token" fault response from the application web service in the remote case that the session token is expired.<br>If the caller receives a fault when calling the application web service that is using the session token fetched by the Get(), then the caller shall call the Release() operation, pass the HostId, InstanceId, and ErrorCode values, ensuring to pass the fault error code into the ErrorCode element. |

- Implementation details:
  - If SPM has not been started, the Get() operation starts it.

- If the initialization process fails, SPM is set to a STOPPED state.

  Therefore, concurrent and subsequent attempts to initialize SPM using the Get() operation fails. This prevents infinite loops.
- To reset this STOPPED state, the administrator must call the Terminate() or Start() operation.
- SPM flags the session token as In Use associating the HostId and InstanceId to it. Therefore, this session token is not available for reuse until a Release() operation completes.

## 1.5.2 Release(string HostId, string InstanceId, string UpdatedSessionToken, string ErrorCode)

- Makes a session token available for reuse.
- Intended to be used by: BPEL flows calling application web services. These BPEL flows are referred as callers.
- Caller responsibilities: the caller should call this operation as soon as it no longer needs the session token. This keeps the session idle time synchronized with the application web server. Table 1–8 shows a list of the caller responsibilities

***Table 1–8 Caller Responsibilities Description***

| Responsibily | Required (Yes/No) | Description |
|---|---|---|
| HostId and InstanceID | Yes | The caller shall pass the same HostId and InstanceId values used by the Get(). |
| UpdatedSessionToken | Optional | If the application web service does not return a session token, the caller should pass an empty value in the UpdatedSessionToken element.<br>If the application web server returns a session token as its response, the caller should capture it and pass it in the UpdatedSessionToken element. In a case in which the session token was used by the caller multiple times, the last session token captured is expected to be passed in the UpdatedSessionToken element. |
| ErrorCode | Optional | If the caller does not get a fault response from the application web service, the caller shall pass an empty value in the ErrorCode element.<br>If the caller gets a fault when calling the application web service when using the session token fetched by the Get() operation, then the caller shall call the Release() operation, pass HostId, InstanceId, and ErrorCode values, ensuring to pass the fault error code into the ErrorCode element.<br>An empty value can be passed in the UpdatedSessionToken element. If an updated session token was passed back in the fault response, the caller should pass that updated session token to the Release() operation. |

**Note:** Be aware that depending on the application web server being called, the Error Code may not be mapped into the code element of the RuntimeFaultMessage. It may come in the summary or detail element. Therefore, map the element that contains the error code.

For Siebel on Premises web services, the error code is mapped to the summary element of the RuntimeFaultMessage.

For CRM On Demand web services, the error code is mapped to the detail element of the RuntimeFaultMessage.

- Implementation details:
  - When called, SPM flags the session token as Available and disassociates the InstanceId from it.

In addition, the internal idle time counter is restarted. This internal idle time counter is used to predict the session expiration.

- If a value was passed in the UpdatedSessionToken element, the value is updated in the pool.
- If a value was passed in the ErrorCode element and it matches an error code listed in the InvalidSessionErrorCodes configuration property for that HostId, then the session token is removed from the pool and the application web server is called to request termination of the session. Otherwise, the session token is marked as available for reuse and the idle time counter is restarted.

# 2

# AIA Composite Scheduler

This chapter introduces AIACompositeScheduler and includes the following sections:



## 2.1 Introduction to AIACompositeScheduler

AIACompositeScheduler is a utility component that is used by integrations to schedule a Service-Oriented Architecture (SOA) composite to be invoked at the specified time interval. For example, it can schedule a SOA composite to be invoked every 30 seconds. It supports scheduling composites in cluster environment as well.

AIACompositeScheduler is implemented as a J2EE application. It uses Weblogic Timer and Job Scheduler Application Programming Interfaces (API). These APIs are used to schedule a job to run at a specific time interval. When a job is run, it invokes SOA composite using composite's direct binding.

Currently, CRM On Demand - JD Edwards EnterpriseOne and CRM On Demand - Oracle E-Business Suite integrations use this utility component to invoke the CRMOnDemandActivationAgent composite every 30 seconds.

## 2.2 Deploying AIACompositeScheduler

For CRM On Demand - JD Edwards EnterpriseOne and CRM On Demand - Oracle E-Business Suite integrations, the deployment of this utility on the server is done automatically when the supplementary deployment plan of the integration is run. The supplementary deployment plan is run automatically when the integration deployment plan is run. When AIACompositeScheduler is deployed, it invokes CRMOnDemandActivationAgent every 30 seconds.

### 2.2.1 Deploying AIACompositeScheduler on Weblogic Cluster

Similar to the deployment on the Weblogic server, deployment of AIACompositeScheduler on the Weblogic cluster is done automatically by running the integration deployment plan. However, in the Weblogic cluster, the following additional configurations are required for scheduler to work properly:

- Weblogic job scheduler table and data source

During installation, these are created if **the Create default Job Scheduler table and data source if one does not exist** option is selected when the configuration wizard is run. This option is selected by default and it creates a default table and data source, if these are not created. The default table is created in the AIA database with the name weblogic_timers and the name of the data source is JobSchedulerDataSource.

Table and data source are used in the cluster environment to persist scheduled jobs in the database. It supports load balancing and failures in the cluster environment. When a server fails or must load balance, another server can pick up the job. In a non-cluster environment, this is not needed because scheduled jobs are in memory.

- Leasing

  Leasing must be enabled for Job Schedulers. You can use either high-availability database leasing or non-database consensus leasing.

  AIACompositeScheduler uses the Weblogic Job Scheduler to schedule jobs in the cluster environment. It is a requirement from the Weblogic Job Scheduler for the required configurations to be in place.

For more information about Job Scheduler table, Data source, and Leasing, see *Timer and Work Manager API (Common) Programmer's Guide*, "Using the Job Scheduler."

## 2.3 Configuring AIACompositeScheduler

For CRM On Demand - JD Edwards EnterpriseOne and CRM On Demand - Oracle E-Business Suite integrations, all configurations are automatically done to invoke CRMODActivationAgent every 30 seconds.

If you want to modify the preconfigured settings or to schedule more composites, perform the steps mentioned in Section 2.3.1, "Creating Direct Binding" and Section 2.4, "Modifying AIACompositeScheduler Properties".

### 2.3.1 Creating Direct Binding

AIACompositeScheduler uses direct binding to invoke a composite. Composites that must be scheduled by AIACompositeScheduler must have a direct binding service exposed.

To create a direct binding:

1. In Jdeveloper, select **composite**.
2. On the component palette, select **Direct Binding Service**.
3. Add **Direct Binding Service** to the exposed services side of the composite.
4. Provide the Web Service Definition Language (WSDL) for the direct binding. You can use WSDL of other exposed services.
5. Add a wire to the component.
6. In the source view of the composite, see if this service uses the direct binding.
7. Deploy the composite on the SOA server to be invoked or scheduled by AIACompositeScheduler.

## 2.4 Modifying AIACompositeScheduler Properties

Properties of AIACompositeScheduler are specified in *web*.xml. Administrators can change these properties dynamically through a weblogic deployment plan. For

example, administrator can change the time interval to invoke the composite from 30 seconds to one minute.

For CRM On Demand - JD Edwards EnterpriseOne and CRM On Demand - Oracle E-Business Suite integrations, these properties are preconfigured to invoke CRMOnDemandActivationAgent every 30 seconds. Administrator can change the preconfigured properties, if required, by completing the steps mentioned in this section.

Table 2–1 lists the properties required by the AIACompositeScheduler:

***Table 2–1 Properties for AIACompositeScheduler***

| Property Name | Description |
|---|---|
| timeInterval | Scheduling time in seconds<br>For example, If it is set to 30 seconds, composite is invoked every 30 seconds.<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured to 30 seconds. |
| compositeName | Name of the composite to be invoked<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as CRMOnDemandActivationAgent. |
| compositeDomain | SOA domain on which the composite to be invoked is deployed<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as default. |
| compositeVersion | Version of the composite to be invoked<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as 1.0. |
| compositeDirectBindingName | Direct binding name of the composite<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as client. |
| bpelOperationName | Name of the operation that must be invoked<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as process. |
| bpelOperationPartName | Operation part name<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as command. |
| bpelOperationMessageValue | XML input for the operation<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, the following script is preconfigured:<br>`CDATA[<?xml version="1.0" encoding="windows-1252" ?>`<br>`<ns1:CRMOnDemandActivationAgentRequestMessage`<br>`xmlns:ns1="http://xmlns.oracle.com/CRMOnDemandActivationAgent">`<br>`<ns1:input>Test</ns1:input>`<br>`</ns1:CRMOnDemandActivationAgentRequestMessage>]]` |
| bpelOperationType | Operation type<br>This can be either *one-wayrequest-responseone-way* or . The option signifies one-way operation and the *request-response* option signifies request response operation.<br>For CRM OD - JD Edwards EnterpriseOne and CRM OD - Oracle E-Business Suite integrations, this property is preconfigured as request-response. |

AIACompositeScheduler can invoke multiple composites. To specify multiple composites, first composite is named as **compositeName** and subsequent ones as compositeName1, compositeName2, and so on. You might have to append the suffix 1, 2, and so on for all properties except timeInterval. Therefore, it is compositeName1, compositeDomain1, compositeVersion1, compositeDirectBindingName1,

bpelOperationName1, bpelOperationPartName1, bpelOperationMessageValue1, and bpelOperationType1.

In the cluster environment, timeInterval property less than 30 seconds is not valid, because minimum time for recurring implementation of a timer is 30 seconds.

### 2.4.1 To View Properties

You can view the properties in a web browser by completing the following steps:

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. Expand **Application Deployments**.
3. Click **AIACompositeScheduler.**
4. Under the **Web Modules**, find the **Test point URL**.
5. Append **compositeScheduler** to the URL.
6. View the properties.

### 2.4.2 To Modify Properties

After AIACompositeScheduler is deployed, you can specify the properties of the composite to be invoked through a deployment plan.

Complete these steps to modify the properties:

1. Log in to Weblogic Console.
2. Open your_domain and select **deployments**.
3. Select **AIACompositeScheduler**.
4. In the **Overview** tab, find the path for the deployment plan.
5. On the file system, navigate to the path for the deployment plan found in the **Overview** tab.
6. Search for the deployment plan.
7. Open the deployment plan to modify.
8. Search for the property under **variable-definition** that must be modified.
9. Modify the value.
10. Return to Weblogic Console.
11. Select **AIACompositeScheduler.**
12. Redeploy **AIACompositeScheduler** with the updated deployment plan.

### 2.4.3 To View Logs

AIACompositeScheduler writes to the server logs. Depending on the Weblogic server setup, location of the logs may vary. An example of the location of the log files: oracle/Middleware/user_projects/domains/soa_domain/servers/soa_server1/logs.

## 2.5 Starting and Stopping AIACompositeScheduler

This section includes the following topics:

- Section 2.5.1, "Starting and Stopping AIACompositeScheduler on Weblogic Server"

- Section 2.5.2, "Starting and Stopping AIACompositeScheduler on Cluster Server"

## 2.5.1 Starting and Stopping AIACompositeScheduler on Weblogic Server

This section includes the following topics:

- Section 2.5.1.1, "To Start AIACompositeScheduler"
- Section 2.5.1.2, "To Stop AIACompositeScheduler from Weblogic Server"
- Section 2.5.1.3, "To Stop AIACompositeScheduler from Oracle Enterprise Manager Fusion Middleware Control"
- Section 2.5.1.4, "To Restart AIACompositeScheduler from Weblogic"r
- Section 2.5.1.5, "To Restart AIACompositeScheduler from Weblogic Server"

### 2.5.1.1 To Start AIACompositeScheduler

After AIACompositeScheduler is deployed, it automatically starts invoking composites for the time interval specified.

### 2.5.1.2 To Stop AIACompositeScheduler from Weblogic Server

If you stop composites, these are no longer invoked. Complete the following steps to stop it:

1. Log in to Weblogic Console.
2. Open your_domain.
3. Select **deployments**.
4. Search for and select **AIACompositeScheduler**.
5. Select **Stop**.

### 2.5.1.3 To Stop AIACompositeScheduler from Oracle Enterprise Manager Fusion Middleware Control

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.
2. Open **Application Deployments**.
3. Select **AIACompositeScheduler**.
4. Right click and select **Control, Shutdown**.

### 2.5.1.4 To Restart AIACompositeScheduler from Weblogic

After stopping the AIACompositeScheduler, you can restart it again to invoke composites. Complete the following steps to restart it:

1. Log in to Weblogic Console.
2. Open your_domain.
3. Select **deployments**.
4. Search for **AIACompositeScheduler** and select it.
5. Select **Start**.

### 2.5.1.5 To Restart AIACompositeScheduler from Weblogic Server

1. Log in to Oracle Enterprise Manager Fusion Middleware Control.

2. Open **Application Deployments**.
3. Select **AIACompositeScheduler**.
4. Right click and select **Control, Start up.**

### 2.5.2 Starting and Stopping AIACompositeScheduler on Cluster Server

In a cluster environment to stop a composite from being invoked, you must cancel the AIACompositeScheduler job. This job has the description - AIACompositeSchedulerTimerListener.

Complete the following steps to cancel this job:

1. Log in to Weblogic Console.
2. Select **Environment**, **Servers**.
3. Select the server on which the jobs you want to cancel are scheduled.
4. Select **Control**, **Jobs.**

   The list of scheduled jobs for that server appears.
5. Select the **AIACompositeSchedulerTimerListener** job option.
6. Click **Cancel**.