Oracle® Application Integration Architecture 3.1: Process Integration Pack Utilities Guide

Release 3.1

Part No. E20495-02

March 2011

ORACLE

Oracle Application Integration Architecture 3.1: Process Integration Pack Utilities Guide

Part No. E20495-02

Contents

Preface

Welcome to the *Oracle Application Integration Architecture 3.1: Process Integration Pack Utilities Guide*.

Oracle Application Integration Architecture (AIA) provides the following guides and resources for this release:

Oracle AIA Guides

- Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.4.0)
- Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.4.0)
- Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.4.0)
- Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.4.0)
- Oracle Fusion Middleware Migration Guide for Oracle Application Integration Architecture 11*g* Release 1 (11.1.1.4.0)
- Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.4.0)

Additional Resources

The following resources are also available:

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

Chapter 1: Session Pool Manager

This chapter introduces Session Pool Manager (SPM) and discusses:

- Working with SPM
- Setting SPM configuration properties
- Administering the AIASessionPoolManager Service on the SOA Server
- Developing integrations to use the AIASessionPoolManager Service

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 (multithreaded) BPEL flows
- SPM supports multiple application Web server instances (hosts).
- Contains the Sentinel, which periodically checks session tokens, removes expired sessions, and replaces them with new ones

Working with SPM

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

This section discusses the following topics:

- Understanding Session Pool Manager Configuration
- Understanding the Initialization Process
- Understanding the Get() Operation
- Understanding the Sentinel Process
- Understanding Statistics
- Understanding Trace Logging to a CSV File

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 Setting SPM Configuration Properties.

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.

Once 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 will fail. This prevents infinite loops or queuing up BPEL processes requesting a session token that will fail. To reset this ***STOPPED*** state, the administrator should call the Terminate() or Start() operation after fixing the problem.

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.

Once SPM has been initialized, it proceeds to get a session token from the pool table by way of one or more of the following paths.

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. This 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 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.

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

When no session token available in the pool table, SPM waits a number of milliseconds, determined by the ***GetFromPoolTable_WaitInterval*** property, in anticipation that a session token will become 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 Predicting an Expired Session Token.

If SPM is not able to get a session token from the pool table, then it proceeds to create a new session token, as long as 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 new session token is denied, and the Get() operation throws a fault to the caller.

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 will renew it according 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 new one.

  For more information about how SPM creates a new session token, see Creating a New 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 new session token is not successful, SPM throws a fault to the caller.

Creating a New 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 a number of 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.

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 need to 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.

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 a new one for replacement by calling the application Web server again.

The Sentinel terminates (removes) session tokens that have not been reused for some time. This is controlled by the ***Sentinel_Renew_Max*** configuration property.

SPM keeps a tally on the number of consecutive times the Sentinel has renewed a session token that has not been used between renewals. Once the value of the ***Sentinel_Renew_Max*** property has been 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 Sentinel_Renew_Max.

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() are operations 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 will try 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.

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 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 multithread-safe. These statistics are solely meant to provide bulk data to help you tune SPM.

SPM tallies statistics for the following operations:

Operation	Statistic Description
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. **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:

Operation	Statistic Description
	1. The application Web server was slow. 2. Some SOA server threads used too much bandwidth, causing other threads "starve" and timeout. In either case, the BPEL server sent a timeout to the client. The client knows that a session token will eventually be assigned by SPM, so the client calls the release operation to prevent SPM from assigning a session token. **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.
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 will eventually be 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. **Tuning tip:** See getSession_ReleaseAhead.

Operation	Statistic Description
Sessions_Validated	Count of session tokens that were predicted to be expired, thus confirming that the session validation occurred. The validation outcome is unknown. **Tuning tip:** Reducing the number of validations can help to free up the application Web server and will prevent 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. **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. **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. **Tuning tip:** See Sessions_Discarded.

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 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 will cause the closure of the current CSV file.

A trace log CSV file captures the following data:

Column	Description
Date-Time	Date and time when the operation was implemented.
Client Instance Id	Instance ID passed in Get or Release operations. 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.

SPM logs the following operations in the trace log CSV file:

Operation	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.

Operation	Description
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.

Setting SPM Configuration Properties

SPM is automatically configured when a PIP utilizing SPM is installed. SPM will be installed with a default configuration. When implementing or administering the PIP, these configurations can be changed to tune the configurations to better suit your integration environment and the needs of the implemented PIP.

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) will be used. If no all_hosts default property is defined, the caller will receive 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 a Process Integration Pack (PIP) for Siebel may use SEBL_01 as the system ID for the Siebel application Web server. They will 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 PIP 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.

Here is an example of an 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.
         CSV file will be located at the 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?SW
EExtSource=SecureWebService&amp;SWEExtCmd=Execute&amp;WSSOAP=1</Prop
erty>
   <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.CRMSiebelS
ession</Property>
   <Property name="SEBL_01.InvalidSessionErrorCodes">10944642|SBL-
BPR-00162|SBL-DAT-00175</Property>
   <!-- === -->
   <!-- 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/Integrati
on</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.CRMOnDema
ndSession</Property>
   <!-- === -->
   <!-- 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.NoSvrCRMS
iebelSession</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 11g Release 1*, "Building AIA Integration Flows," How to Set Up AIA Workstation.

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 will have its own pool.

This property is not prefixed with a HostId value.

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 "Maximum active dispatcher threads" (dspMaxThreads) value set for the BPEL server.

In addition, we recommend that you set the BPEL server dspMaxThreads value to be the same or lower than the maximum number of tasks that the application Web server allows. For example, for Siebel, this is the value of the App Max Tasks parameter.

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.

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 will make to connect. The minimum value should be ***1***.
- ***ConnectServer_WaitInterval***: Determines the amount of time in milliseconds that SPM will wait between the failed attempt and its next attempt. The value should be the same as the value you set for the ServerSessionRequestRate property, which will help ensure that the host does not mistake the connection attempts for a server attack.

For more information, see ServerSessionRequestRate.

ServerSessionRequestRate

It determines the amount of time in milliseconds that SPM will wait 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.

GetFromPoolTable_MaxAttempts and GetFromPoolTable_WaitInterval

If no session tokens are available in the pool for the Get() operation, SPM waits a number of milliseconds in anticipation that a process will release 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 new session token.

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

- ***GetFromPoolTable_MaxAttempts***: Determines the maximum number of attempts SPM will make 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 will wait between attempting to obtain a session token from the pool.

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 will be a number of 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.

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 11g Release 1*, "Viewing Trace and Error Logs in EM."

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 Sentinel_PoolSize_Min.

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.

Sentinel_PoolSize_Min

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

This functionality is activated only when Sentinel is activated.

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 TRACE.LOG.ENABLED.

The Sentinel_LogLevel value should be one of the following java.util.logging.Level values. 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 11g Release 1*, "Viewing Trace and Error Logs in EM."

UserId

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

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.pass
word</Property>.
```

For CRM Siebel:

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

For AECM Siebel

```
<Property
name="SEBL_01.Password">participatingapplications.aecm.server.eai.pa
ssword</Property>
```

For more information about how to update the password, see *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1*, "Post Install Configurations," Modifying Passwords Used for AIA Deployments.

EndpointURI

It determines the endpoint URI that will be used 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;SWEExtCmd=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.

ClassName

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

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

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.

Regular expressions can be used 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</Property>
```

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 Understanding Trace Logging to a CSV File.

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.

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***.

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***.

Describing Recommended Configuration Settings for Siebel Web Server and SPM

To help optimize the performance of SPM with your Siebel Web server, we recommend balancing your Siebel Web server and SPM configuration settings according to the recommendations discussed in this section.

Siebel Web server sessions are controlled by the following parameters. Corresponding SPM configuration properties are also listed.

Siebel Web Server Parameter	SPM Configuration Property
SessionTimeout	N/A
SessionTokenTimeout	PredictExpiration_Idle

Siebel Web Server Parameter	SPM Configuration Property
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 PredictExpiration_Idle and PredictExpiration_Age.

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.

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.

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.

SWSENoSessInPref

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

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.

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***

SPM

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

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 will require manual intervention.

Running AIASessionPoolManager Administrator Operations

To run the administrator operations, call the AIASessionPoolManager Web service. You can use the test Web service functionality provided by Enterprise Manager (EM).

To call the AIASessionPoolManager Web service through the test Web service functionality provided by EM:

1. Log in to EM.
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.

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.

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 will be 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 will be aborted and a fault will be 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 Use*** or ***Available*** state.

Statistic values and the pool are cleared.

Once terminated, the next Get() or Start() operation will initialize SPM.

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.

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.

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 new daemon thread on which to run Sentinel.

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, we recommended that you deactivate statistics functionality once 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 multithread safe. Therefore, some tallies may occasionally miss one or two counts. The reason for not making the logic multithread safe is to avoid decreasing performance of SPM.

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.

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, we recommended deactivating the statistics functionality once they are not 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.

Describing Scenarios Requiring Manual Intervention

When any of these scenarios arises, an administrator must manually terminate SPM for the host IDs in place. This will clean up invalid session pool tokens, as well as prevent 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 will invalidate 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 will be 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 will be left open on the application Web server.

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

Developing Integrations to Use the AIASessionPoolManager Service

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 describes the following client operations:

- Get(string HostId, string InstanceId)
- Release(string HostId, string InstanceId, string UpdatedSessionToken, string ErrorCode)

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.

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

HostId Required.

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 Required.

The caller must pass a unique ID for the current caller instance. The BPEL XPath function `ora:getInstanceId()` can be used.

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.

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.

An empty value can be passed in the UpdatedSessionToken element.

Implementation details:

If SPM has not been started, the Get() operation will start 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 will fail. This prevents infinite loops.

To reset this STOPPED state, the administrator should 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 will not be available for reuse until a Release() operation completes.

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 will keep the session idle time synchronized with the application Web server.

HostId and InstanceID Required.

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.

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.

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.

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 one of the error codes listed in the ***InvalidSessionErrorCodes*** configuration property for that HostId, then the session token will be removed from the pool and the application Web server will be called to request termination of the session. Otherwise, the session token will be marked as available for reuse and the idle time counter will be restarted.

Using Session Pool Manager in a Cluster Environment

In a SOA cluster environment, the AIA SPM solution creates an SPM pool for each node in the cluster. The SPM pool is instantiated as a singleton (one per Java Virtual Machine (JVM)) therefore, when SPM is deployed in a SOA cluster environment, each node instantiates its own SPM pool. There is no communication or interaction between the different SPM instances.

The PIP flows are coded in a way that ensures the execution of the Get and Release operation calls to the AIASessionPoolManager composite in the same node as the PIP flow. See Development Considerations for details.

SPM configuration in a cluster environment is almost the same as in a non-cluster environment except for the pool size management.

Configuring SPM

The SPM configuration settings are applied to all SPM instances in the cluster. In other words, all SPM instances will have the same maximum pool size, the same Sentinel configurations, and so on.

Most of the SPM configuration properties are set up the same way in a SOA cluster vs. a non-cluster environment, except for two properties that need to be understood from a different perspective:

- ***PoolSize_Max***
- ***Sentinel_PoolSize_Min***

As indicated below, these two property settings have a different implication in a cluster environment.

PoolSize_Max

In a cluster environment, what is different is that that each node in the SOA cluster has its own pool. Therefore the sum of session tokens across all of the pools should not exceed the maximum number of open session token allowed by the Application Web Server (Siebel or CRM on Demand).

The recommendation for a non-cluster environment still applies: the ***PoolSize_Max*** value should match the ***Dispatcher Invoke Threads*** property of the ***BPEL Service Engine Properties***.

Therefore it is important that the value for ***Dispatcher Invoke Threads*** multiplied by the number of nodes in the SOA cluster does not exceed the maximum number of open sessions (or concurrent tasks) in the Application Web Server.

The recommended way to determine the maximum value for ***Dispatcher Invoke Threads*** is as follows:

Dispatcher Invoke Threads = PoolSize_Max = Truncate ((Max Open Sessions Allowed in Appl Web Server) ÷ (Number of nodes in SOA Cluster Env))

For example, if your edge application web server allows 100 open sessions and your cluster environment has 3 nodes, then the ***Dispatcher Invoke Threads*** and ***PoolSize_Max*** should be 33 (i.e Truncate(100 ÷ 3)).

Sentinel_PoolSize_Min

This parameter is applicable at the pool level, so in a cluster environment, it applies at the node level.

For example, setting this property to 5 in a 3 node cluster environment will result in a minimum of 15 session tokens at any given time; each node will hold at least 5 session tokens.

Running AIASessionPoolManager Administrator Operations

To run the administrator operations in a cluster environment, you should call the AIASessionPoolManager web service using the specific address for the node where you want the operation to happen.

If you use the cluster address, the Load Balancer will decide where the operation will be executed; so your call may not get assigned to the desired node or it will be very difficult to determine which node the response comes from.

Avoid using the testing functionality provided by Oracle Enterprise Manager. Also avoid using your browser to invoke the AIASessionPoolManager. This since both applications will end-up calling the cluster address, even if you enter the node specific URL in your browser.

You can use any third party tool for calling the provided web services. Provide the specific node URL to the tool to call the AIASessionPoolManager web service.

Here is an example for calling AIASessionPoolManager in a cluster environment:

Cluster address: http://myhost:7001

Soa_server1: http://myhost1:8001

Soa_server2: http://myhost2:8001

The AIASessionPoolManager web service is protected with wss security. The user/password combination to access it is: weblogic/welcome

The URL for calling AIASessionPoolManager that BPEL Console will give you would look like:

http://**myhost**:**7001**/soa-infra/services/default/AIASessionPoolManager/client

This URL points to the load balancer: http>//myhost:7001

Therefore the URL for calling AIASessionPoolManager on soa_server1 would look like:

http://**myhost1**:**8001**/soa-infra/services/default/AIASessionPoolManager/client

And the URL for calling AIASessionPoolManager on soa_server2 would look like:

http://**myhost2**:**8001**/soa-infra/services/default/AIASessionPoolManager/client

Calling SPM using these URLs allows you to execute the admin operations on each corresponding node.

The payload for “Status” the SPM Status for SEBL_01 on any of the nodes should look like this:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:aias="http://xmlns.oracle.com/AIASessionPoolManager" >
   <soapenv:Header>
      <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
         <wsse:UsernameToken>
            <wsse:Username>weblogic</wsse:Username>
            <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">welcome</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <aias:GenericAIASessionPoolManagerRequest>
         <aias:HostId>SEBL_01</aias:HostId>
      </aias:GenericAIASessionPoolManagerRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The payload for calling Terminate operation for SEBL_01 should look like this:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:aias="http://xmlns.oracle.com/AIASessionPoolManager">
   <soapenv:Header>
      <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
         <wsse:UsernameToken>
            <wsse:Username>weblogic</wsse:Username>
            <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordText">welcome</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body>
      <aias:TerminateAIASessionPoolManagerRequest>
         <aias:HostId>SEBL_01</aias:HostId>
      </aias:TerminateAIASessionPoolManagerRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Development Considerations

Developers must avoid coding patterns that can cause a Get operation to be executed in one node and the Release operation to be executed on another node. Otherwise, when Get and Release operations are executed on different nodes, the session token will not be released back to the pool in the node where it got assigned to the flow; and the Release operation call will return a fault indicating that SPM could not locate the record.

To avoid Get/Release pairs from being executed on different nodes do not override the default value of “oracle.webservices.local.optimization” property on the references to AIASessionPoolManager in the composite where SPM is being called. The default value for “oracle.webservices.local.optimization” property is “true”. One part of the Optimization functionality is to instantiate the referenced composite (AIASessionPoolManager) in the same thread as the caller is executing.

Chapter 2: AIACompositeScheduler

This chapter introduces AIACompositeScheduler and discusses how to:

- Deploy AIACompositeScheduler
- Configure AIACompositeScheduler
- Modify AIACompositeScheduler properties
- Start and stop AIACompositeScheduler

Introduction to AIACompositeScheduler

AIACompositeScheduler is a utility component that is used by Process Integration Packs (PIP) 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 a 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 PIPs use this utility component to invoke the CRMOnDemandActivationAgent composite every 30 seconds.

Deploying AIACompositeScheduler

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

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 PIP deployment plan. However, in the Weblogic cluster, the following additional configurations are required for scheduler to work properly:

- Weblogic job scheduler table and datasource

  During installation, these are created if ***the Create default Job Scheduler table and datasource if one does not exist*** option is selected when the config wizard is run. This option is selected by default and it creates a default table and data source if these are not created already. The default table is created in the AIA database with the name weblogic_timers and 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 needs to 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. Either high-availability database leasing or non-database consensus leasing can be used.

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, Datasource, and Leasing, see *Timer and Work Manager API (CommonJ) Programmer's Guide*, "Using the Job Scheduler."

Configuring AIACompositeScheduler

For CRM On Demand - JD Edwards EnterpriseOne and CRM On Demand - Oracle E-Business Suite PIPs, 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 the Creating Direct Binding and Modifying AIACompositeScheduler Properties sections.

Creating Direct Binding

AIACompositeScheduler uses direct binding to invoke a composite. Composites that need to 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, you can see this service uses the direct binding.
7. Deploy the composite on the SOA server to be invoked or scheduled by AIACompositeScheduler.

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 PIPs, these properties are pre-configured to invoke CRMOnDemandActivationAgent every 30 seconds. Administrator can change the pre-configured properties, if required, by completing the steps mentioned in this section.

This table lists the properties required by the AIACompositeScheduler:

Property Name	Description
timeInterval	Scheduling time in seconds For example, If it is set to 30 seconds, composite will be invoked every 30 seconds. For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured to 30 seconds.
compositeName	Name of the composite to be invoked For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as CRMOnDemandActivationAgent.
compositeDomain	SOA domain on which the composite to be invoked is deployed For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as default.
compositeVersion	Version of the composite to be invoked For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as 1.0.
compositeDirectBindingName	Direct binding name of the composite For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as client.
bpelOperationName	Name of the operation that needs to be invoked For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as process.
bpelOperationPartName	Operation part name For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as command.

bpelOperationMessageValue	XML input for the operation
	For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, the following script is preconfigured:
	`CDATA[<?xml version="1.0" encoding="windows-1252" ?>`
	`<ns1:CRMOnDemandActivationAgentRequestMessage xmlns:ns1="http://xmlns.oracle.com/CRMOnDemandActivationAgent">`
	`<ns1:input>Test</ns1:input>`
	`</ns1:CRMOnDemandActivationAgentRequestMessage>]]`

Property Name	Description
bpelOperationType	Operation type This can be either ***one-way*** or ***request-response***. The ***one-way*** option signifies one-way operation and the ***request-response*** option signifies request response operation. For CRM OD – JD Edwards EnterpriseOne and CRM OD – Oracle E-Business Suite PIPs, this property is preconfigured as request-response.

AIACompositeScheduler can invoke more than one composite. To specify more than one composite, first composite is named as compositeName and subsequent ones as compositeName1, compositeName2, and so on. You may need to append the suffix 1, 2, and so on for all properties except timeInterval. Therefore, it will be compositeName1, compositeDomain1, compositeVersion1, compositeDirectBindingName1, bpelOperationName1, bpelOperationPartName1, bpelOperationMessageValue1, and bpelOperationType1.

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

To View Properties

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

1. Log in to Enterprise Manager (EM).
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

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 you want to modify.
9. Modify the value.

10. Return to Weblogic Console.
11. Select AIACompositeScheduler
12. Redeploy AIACompositeScheduler with the updated deployment plan.

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.

Starting and Stopping AIACompositeScheduler

This section discusses how to:

- Start and stop AIACompositeScheduler on Weblogic server and EM.
- Start and stop AIACompositeScheduler on the cluster server.

Starting and Stopping AIACompositeScheduler on Weblogic Server

This section discusses how to:

- To Start AIACompositeScheduler
- To Stop AIACompositeScheduler from Weblogic server
- To Stop AIACompositeScheduler from EM
- To Restart AIACompositeScheduler from Weblogic server
- To Restart AIACompositeScheduler from EM

To Start AIACompositeScheduler

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

To Stop AIACompositeScheduler from Weblogic Server

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

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

To Stop AIACompositeScheduler from EM

1. Log in to EM.
2. Open Application Deployments.
3. Select AIACompositeScheduler.
4. Right click and select > Control > Shutdown.

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.

To Restart AIACompositeScheduler from Weblogic Server

1. Log in to EM.
2. Open Application Deployments.
3. Select AIACompositeScheduler.
4. Right click and select Control > Start up.

Starting and Stopping AIACompositeScheduler on the 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.