Java CAPS Management and Monitoring APIs

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–3388–10
June 2008

081030@21288

Contents

Java CAPS Management and Monitoring APIs

This topic provides information on common management and monitoring tasks. The management and monitoring APIs are not installed during the Java™ Composite Application Platform Suite (Java CAPS) installation; they are packaged as `EM_API_KIT.zip`. This ZIP file is on `ESB_API_Kit.zip`. You need to extract `EM_API_KIT.zip` to your root Java CAPS 6 installation directory. If you extract the file to another directory you need to set the JAVA_HOME, JAVA_CAPS, and the ENGINE_HOME environment variables in `env.bat`. For information on how to do this, see "To Modify the Environment Variables in env.bat" on page 51. Also included with the ZIP file are JavaDocs, Java samples, and Groovy scripting samples that demonstrate how to use the APIs.

If you have any questions or problems, see the Java CAPS Support web site at `http://goldstar.stc.com/support`.

Tip – To access all the Java CAPS documentation in HTML format on the Sun Developer Network (SDN) web site, go to `http://developers.sun.com/docs/javacaps/`.

You can also access the Java CAPS documentation on the SDN web site by clicking the HELP tab in Enterprise Manager.

The HELP tab in the Java CAPS Uploader opens the "Installing Java CAPS Components Using the Java CAPS Uploader" topic.

What You Need to Know

- "Using Oracle and Other Databases for Alert Persistence" on page 45
- "Setting Up Scripting Engines" on page 50

You expose management and monitoring using:

- A common API

  The client API kit is installable from the Java Composite Application Platform Suite (Java CAPS) Installer and is packaged along with the Groovy scripting engine, Groovy libraries, and samples.

- JSR 223 Scripting

  All management capabilities are exercisable from any language that supports JSR 223 scripting for the Java platform. Currently this includes 25 different scripting languages, including Groovy, JRuby, Jython, JACL, etc.

- Sun Java System Application Server Admin Console

  All JBI administration use-cases are built into the Sun Java System Application Server Admin Console. This enables administrators to administer their domain runtimes and components remotely using a browser.

- Command-line interface

  The AS Admin command-line interface that comes with the Sun Java System Application Server enables you to manager your application server environment. All JBI administration use-cases have been integrated into the AS Admin command-line interface. In addition, the command-line interface that used to manage non-JBI components is also available for you to use.

- Ant

  asant enables you to run ant scripts on the Sun Java System Application Server. All JBI administration use-cases have been integrated into the Sun Java System Application Server's asant, and are available to exercise though any ant scripting that is available to you.

- Java CAPS Enterprise Manager

  Use Enterprise Manager to remotely to administer your non-JBI based components.

- NetBeans Administration plugins

  The JBI Manager that comes with NetBeans IDE enables developers to manage the JBI runtimes component containers and composite applications. The Composite Application Project System (CASA) enables developers to deploy and manage lifecycle operations during development.

Java CAPS Management Client

This topic introduces you to the Java CAPS Management Client and the clients that use the API set with JBI based Java CAPS runtime and Non-JBI based Java CAPS runtime.

- Scripting client — Use any JSR 223 scripting client, such as Groovy, JRuby, Jython, or JACL. See "Setting Up Scripting Engines" on page 50 for additional information.

  Note – Groovy samples are included with the Java CAPS Release 6 delivery.

- CLI client — Exercise the APIs using your own Command-line Interface client that you built using predefined functioning code, preferably Java.
- Other Web/GUI clients — Exercise the APIs using your custom Web/GUI client.

Examples of Typical Client Usage

```
try {
  // Get the Management Client
  ManagementClient client = ManagementClientFactory.getInstance
  ("localhost", 4848, "admin", "adminadmin");

  // Get the Administration Service
  AdministrationService administrationService =
  client.getService(AdministrationService.class);
  // ... Use the Administration Service ...

  // Get the Configuration Service
  ConfigurationService configurationService =
  client.getService(ConfigurationService.class);
  // ... Use the Configuration Service ...

  // Get the Deployment Service
  DeploymentService deploymentService =
  client.getService(DeploymentService.class);
  // ... Use the Deployment Service ...

  // Get the Installation Service
  InstallationService installationService =
  client.getService(InstallationService.class);
  // ... Use the Installation Service ...

  // Get the Runtime Management Service
  RuntimeManagementService runtimeManagementService =
  client.getService(RuntimeManagementService.class);
  // ... Use the Runtime Management Service ...
```

```
    // Get the JMS Management Service
    JmsManagementService jmsManagementService =
    client.getService(JmsManagementService.class);
    // ... Use the JMS Management Service ...

    // Get the Alert Management Service
    AlertManagementService alertManagementService =
    client.getService(AlertManagementService.class);
    // ... Use the Alert Management Service ...

    // Get the Log Management Service
    LogManagementService logManagementService =
    client.getService(LogManagementService.class);
    // ... Use the Log Management Service ...

  } catch (ManagementRemoteException exception) {
    // Format the exception and print it.
    String formattedExceptionResult=
    ManagementRemoteExceptionProcessor.processTaskException(exception);
    System.out.println(formattedExceptionResult);
  }
```

JavaDocs

The JavaDocs contain a complete list of all the classes in the API. They are included with the Java CAPS delivery as zipped files.

Targets

A target provides the scope of an administrative operation. Directing a command at multiple targets effectively increases the scope of that command. If multiple targets are specified, the success or failure of each target is reported separately. In other words the result of the operation on all targets is not "rolled up" into a summary status. The following table describes the scope of each target type.

Note – Two of the target option names are constant: "domain" and "server." They represent an instance of an operator and are replaceable with a name specific to the current template.

TABLE 1 Target Operations

Target Name	Scope
domain	Command is executed against the domain itself. For JBI purposes, this is equivalent to add-to-repository activity.
server	Command is executed against embedded DAS server instance.
cluster name	Command is executed against all instances in a cluster.
instance name	Command is executed against a single standalone instance.
cluster instance name	Command is executed against the specific instance in a cluster.

Note – The samples that are pertinent to this topic, such as `AdministrationServiceSample.groovy`, are included with the delivery as zipped files.

Getting Started Using APIs

The Java Composite Application Platform Suite (Java CAPS) APIs are available for users and developers to create applications and web pages.

▼ To Start Using APIs to Create Applications

The following task includes everything a you need to create an application using the Java CAPS APIs. If you are connecting using the Sun Java System Application Server (SJSAS) that you installed during the Java CAPS Installation you do not need any additional JAR files in your working directory. However, if you are connecting remotely, you need the following JAR files in your working directory:

- When connecting through just the RMI protocol using the JSR-160 MX URL you need these JAR files:

```
%CAPS_MANAGEMENT_HOME%\api\caps.management.client.jar;
%SJSAS_HOME%\jbi\lib\jbi-admin-common.jar;
%SJSAS_HOME%\lib\javaee.jar;
```

- When connecting through the HTTP/HTTPS protocols, you need these JAR files:

```
%CAPS_MANAGEMENT_HOME%\api\caps.management.client.jar;
%SJSAS_HOME%\jbi\lib\jbi-admin-common.jar;
%SJSAS_HOME%\lib\javaee.jar;
%SJSAS_HOME%\lib\appserv-deployment-client.jar;
%SJSAS_HOME%\lib\appserv-ext.jar;
```

```
%SJSAS_HOME%\lib\appserv-rt.jar;
%SJSAS_HOME%\lib\jmxremote_optional.jar
```

1. **Use this example to connect to the JMX URL**

```
String hostName = "localhost";
int jrmpPortNumber = 8686;
String userName = "admin", password = "adminadmin";
boolean isRemoteConnection = true;
String jrmpURLString = "service:jmx:rmi:///jndi/rmi://" + hostName
     + ":" + jrmpPortNumber + "/jmxrmi";
CAPSManagementClient managementClient =
     CAPSManagementClientFactory.getInstance(jrmpURLString,
     userName, password, isRemoteConnection);
```

2. **Get the service you need to write your application, for example the AdministrationService.**

```
// get services
          CAPSAdministrationService administrationService =
          managementClient.getService(CAPSAdministrationService.class);
```

3. **After creating the application, for example JBIRuntime, invoke the application.**

```
// use the service,
          System.out.println("The JBI Runtime is
          "+(administrationService.isJBIRuntimeEnabled()?
          "Enabled." : "NOT Enabled."));
```

Connecting to the Server Through APIs

Java CAPS currently provides seven options for you to connect to the Sun Java System Application Server using APIs.

CAPSManagementClientFactory Client Usage

Option 1: host, port, userName, password

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                                  4848,
                                                                  "admin",
                                                                  "adminadmin");
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 2: host, port, userName, password, connectionType

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                                  4848,
```

```
                                                        "admin",
                                                        "adminadmin",
                                                        ConnectionType.HTTP);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 3: url, userName, password, isRemoteConnection

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance(
       "service:jmx:rmi:///jndi/rmi://localhost:22287/management/rmi-jmx-  connector",
       "admin", "adminadmin", false);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 4: MBeanServerConnection

```
/** Only relevant piece of code is shown */
MBeanServerConnection connection = ... // Get the MBeanServerConnection
ManagementClient client = CAPSManagementClientFactory.getInstance(connection);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 5: MBeanServerConnection, isRemoteConnection (true/false)

```
/** Only relevant piece of code is shown */
MBeanServerConnection connection = ... // Get the MBeanServerConnection
ManagementClient client = CAPSManagementClientFactory.getInstance(connection, true);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 6: host, port, userName, password, connectionType, promtUserForMasterPassword(true/false)

```
/** Only relevant piece of code is shown */
ManagementClient client =
CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                        8686,
                                                        "admin",
                                                        "adminadmin",
                                                        ConnectionType.JRMP,



                                                 false);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Option 7: hostName, portNumber, userName, password, connectionType, keyStoreFileLocation, masterPassword, promptForMasterPassword (true/false)

```
/** Only relevant piece of code is shown */
ManagementClient client =
CAPSManagementClientFactory.getInstance
```

```
                                                    ("127.0.0.1",
                                                    8686,
                                                    "admin",
                                                    "adminadmin",
                                                    ConnectionType.JRMP,
                                                    "C:/CAPS6/Glassfish/
                                                    domains/domain1/
                                                    config/keystore.jks",
                                                    "changeit",
                                                    true);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

Connection Type Definition

```
public enum ConnectionType {
    HTTP("s1ashttp"),
    HTTPS("s1ashttps"),
    JRMP("jmxrmi"),
    IIOP("iiop");

    // ... Implementation ...
    /** @return the protocol */
    public String getProtocol();
    /** @return the protocol description */
    public String getDescription();
}
```

CAPSManagementClientFactory Definition

```
/** Only relevant piece of code is shown */
public class CAPSManagementClientFactory {

    // Option 1 - host, port, userName, password
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
                String userName, String password) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 2 - host, port, userName, password, connectionType
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
                String userName, String password, ConnectionType connectionType)
                throws ManagementRemoteException {

        // ... Implementation ...
```

```
    }

    // Option 3 - url, userName, password, isRemoteConnection
    public static CAPSManagementClient getInstance(String url, String userName,
                String password, boolean isRemoteConnection) throws
ManagementRemoteException {

        // ... Implementation ...
    }


    // Option 4 - MBeanServerConnection
    public static CAPSManagementClient getInstance(MBeanServerConnection connection)
            throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 5 - MBeanServerConnection, isRemoteConnection
    public static CAPSManagementClient getInstance(MBeanServerConnection connection,
                boolean isRemoteConnection) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 6 - host, port, userName, password, connectionType,
    promtUserForMasterPassword(true/false)
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
            String userName, String password, ConnectionType connectionType,
            boolean promptForPasswordFlag) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 7 - hostName, portNumber, userName, password, connectionType,
    keyStoreFileLocation,
    //            masterPassword, promptForMasterPassword (true/false)
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
            String userName, String password, ConnectionType connectionType,
            String trustStoreFilePath, String trustStorePassword,
            boolean promptForPasswordFlag) throws ManagementRemoteException {

        // ... Implementation ...
    }
}
```

Note – The Java CAPS Management API samples that are pertinent to this topic, such as `AdministrationServiceSample.groovy`, are included with the delivery as zipped files.

The Alert Management API

With releases 6 the capabilities of the Alert Management Service have been increased to provide more control for the user. The service is now divided into three services:

- Alert Administration Service enables monitoring and control of the alerts stored in the event database in Enterprise Manager
- Alert Configuration Service on the Sun Java System Application Server (SJSAS) provides more control over the persistence of alerts generated in the event management component.
- Alert Notification Service on SJSAS provides more control over the type of alerts that are delivered to a scripting client from the event management component

Caution – Persistence on and journaling off means that the removal of events are completed upon delivery to all current clients.

Persistence on and journaling on means there is no removal of the events. Remember, if the removable policy is enabled, the events will removed.

Support for Databases

The Alert Management API supports five databases:

- Derby, which is delivered with the Sun Java System Application Server installation
- Oracle
- Sybase
- DB2
- PointBase

Note – You must start your database prior to using the Java CAPS APIs.

AlertConfigurationService

The AlertConfigurationService enables the Enterpriser Manager and other management clients to manage and control the delivery reliability of alerts from applications or JBI based

components to their clients. Enterprise Manager always receives alerts when it is started, while all other clients are required to register in order to receive alerts. The configuration to manage includes the enabling and disabling of the persistence of alerts in the database as well as the policy to manage the deletion of alerts.

Note – Users of earlier releases of Java CAPS must enable persistence and journaling to obtain the functionality to which they are accustomed.

Persisted Alerts Removal Policy

The composite removal policy is defined by controlling three items:

- Alerts Count

  The maximum number of alerts that can be persisted. When this policy is enforced, the oldest-generated alerts are removed.

- Alerts Age

  The maximum amount of time, that is, "age," that an alert can be persisted.

- Alerts Level

  Every generated alert has one priority level, from low to high, associated with it:

  - INFO
  - WARNING
  - MINOR
  - MAJOR
  - CRITICAL
  - FATAL

  All the alerts that have a lower priority up to the defined level are removed.

The implementation is based upon a "first come first served" policy, as described in the Alert Configuration Service API. The policy is a combination of all the defined policy items, an "and" condition, and is applied even if the result of the executed policy item nullifies the execution of the next policy item.

By default, persistence is set to "not enabled." If you change the persistence default to "enabled" but journaling is not "on," you do not need to set the policy. However, if journaling is set to "on," you can set the policy for Alerts Count, Alerts Age, and Alerts Level.

Alert Configuration Service API

```
* Enable alerts persistence in the alerts database.
     * enabling it allow for reliable alerts delivery in
     * case of delivery channel failure or Application
     * server restart.
```

```
 *
 * @param enableJournaling
 *                    true - will prevent the system from
 *                           removing alerts after they
 *                           are delivered. The alert stay
 *                           in the database until the
 *                           user removes them.
 *                    false - The system will remove the alert
 *                            upon acknowledgment from
 *                            the reliable client in case one was
 *                            setup or upon send the alert to
 *                            all the non reliable client/s.
 * @throws ManagementRemoteException if JMX related exception is
 *         thrown or the list of target name is null or empty.
 */
 public void enableAlertsPersistence(Boolean enableJournaling) throws
 *        ManagementRemoteException;

 /**
 * Disable alerts persistence in the alerts database.
 *
 * @throws ManagementRemoteException if JMX related exception is
 *         thrown or the list of target name is null or empty.
 */
 public void disableAlertsPersistence() throws ManagementRemoteException;


 /**
 * @return the last setting of alert persistence enabling operation.
 *         true if enable otherwise false.
 * @throws ManagementRemoteException if JMX related exception.
 */
 public  Boolean isAlertsPersistenceEnabled()throws
 * ManagementRemoteException;

/**
 * @return the last setting of alert journal enabling operation.
 *         true if enable otherwise false.
 * @throws ManagementRemoteException if JMX related exception.
 */
public Boolean isAlertsJournalEnabled()throws ManagementRemoteException;

/**
 * set the JNDI name of the data source database to be used
 * for persistence. if not provided at least once the persistence
 * will be disabled even if enableAlertsPersistence was set to true.
 *
 * @param jndiName - of the data source to be used in conjunction with
```

```
 *   enableAlertsPersistence set to true

 * @throws ManagementRemoteException if JMX related exception is
 *         thrown orjndiName parameter is null or empty.
 */
public void setPersistenceDataSourceJndiName(String jndiName) throws
* ManagementRemoteException;
/**

 * @return the last set JNDI name for the alert persistence data source.
 * @throws ManagementRemoteException
 */
public String  getPersistenceDataSourceJndiName() throws ManagementRemoteException;

/**
 * set the database type to be used for persistence.
 * Derby is the assumed default database. If different database is
 * used this method should be called prior to enabling the persistence.
 *

 * @param dbtype - one of the predefined typed defined in {@link
 *             com.sun.caps.management.api.alerts.AlertPersistenceDBType}
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistenceDataBaseType(AlertPersistenceDBType dbtype) throws
 *             ManagementRemoteException;

/**
 * @return The return value represent the last set DB type
 *             {@link com.sun.caps.management.api.alerts.AlertPersistenceDBType}
 *             for each.
 * @throws ManagementRemoteException if JMX related exception is thrown.     */
public AlertPersistenceDBType  getPersistenceDataBaseType() throws
ManagementRemoteException;

/**
 * Set the maximum time a persisted alert will be stored in the alert database
 * before it will be deleted as part of the removal policy
 * @param timeToLive - maximum time in millisecond.
 *
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsMaxAge(Long timeToLive) throws
ManagementRemoteException;

/**
 * return the last setting for the allowed persisted alert age.
 * A value of 0 current time which could cause all persisted alerts to be deleted.
```

```
 * A negative value this policy element is ignored.
 * @return  the returned value  representing as time in milliseconds set for each.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsMaxAge() throws ManagementRemoteException;

/**
 * set the maximum number of  alerts allowed to be persisted  before it will be
 * deleted as part of the removal policy
 * @param size - Maximum number of alerts.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsMaxCount(Long size) throws ManagementRemoteException;

/**
 * return the last setting for the  maximum of  alerts allowed to be persisted.
 * A value of 0 mean no alerts persisted. It behave as if the user
 *  set enableAlertsPersistence to false
 *
 * A negative value this policy element is ignored.
 * @return the returned value represent the maximum number of alerts allowed to be
 *         persisted on each target.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsMaxCount() throws ManagementRemoteException;

/**
 *  The priority based alert level that will be part of the removal policy.
 *  the priorities are as follows (from low to high):
 *  INFO,WARNING,MINOR,MAJOR,CRITICAL,FATAL.
 *  all alerts from the provided level and below will be candidates for removal.
 *
 * @param level - an AlertLevelType representing the level.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsLevel(AlertLevelType level) throws
ManagementRemoteException;
/**
 * @return  the returned value represent the last setting for the level of alerts
 *          that allowed to be removed from persistence for each target.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public AlertLevelType getPersistedAlertsLevel() throws ManagementRemoteException;


/**
 * set the effective policy for the removal of persisted alerts.
 * @param policyList - an array of AlertRemovalPolicyType where the priority
```

```
 *                    of the given policy is defined by its position in the list.
 *                    i.e the lower the index that policy will be applied first.
 *                    possible values are:
 *                    ALERTS_AGE,ALERTS_COUNT,ALERTS_LEVEL.
 *
 *                    null value or empty array indicate no policy will be
 *                    enforced.

 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsRemovelPolicy(AlertRemovalPolicyType[] policyList)
*          throws ManagementRemoteException;

/**
 * @return the return value representing an array the last setting the policy used
 *         when persisted alerts are to be removed. An empty array mean no policy
 *         is enforced.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public String[] getPersistedAlertsRemovalPolicy() throws ManagementRemoteException;


/**
 * enable or disable the ability to use removal policy.
 * @param enableExecuation -  true the current setting is enforced. False the
 *           current policy is ignored.

 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void enablePersistedAlertsPolicyExecution(Boolean enableExecution) throws
*            ManagementRemoteException;

/**
 * @return the returned value represent the last setting that enable/disable the
 *         removal policy
 * @throws ManagementRemoteException
 */
public Boolean isPersistedAlertsPolicyExecutionEnabled()throws
*          ManagementRemoteException;

/**
 * set how often the persisted alerts removal policy will be executed.
 * @param excutionInterval - The interval is defined in milliseconds.

 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsRemovelPolicyExecInterval(Long excutionInterval)
*          throws ManagementRemoteException;
```

```
/**
 * @return the returned value representing The last interval setting of
 *         the persisted alerts removal policy will be executed.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsRemovelPolicyExecInterval() throws
 *         ManagementRemoteException;

/**
 * Set the persisted alerts table name.
 * Note: if the same database is used across the whole enterprise. it
 *       must be unique for each domain used.
 * @param tableName - the table name to be used for the stored alerts
 * @throws ManagementRemoteException
 */
public void setAlertTableName(String tableName) throws ManagementRemoteException;


/**
 * @return The current assign persisted alerts table name.
 * @throws ManagementRemoteException
 */
public String getAlertTableName() throws ManagementRemoteException;
/**
 *
 * @return return the total number of alerts currently persisted. This value
 * is volatile and may V between two sequential calls to this method.
 * @throws ManagementRemoteException
 */
public  Integer getPersistedAlertsCount() throws ManagementRemoteException;

/**
 * the API allows the caller to set all the parameters defined in the other API in
 * this interface. All the setting are applied prior to enabling the persistence.
 * @param enableJournaling
 *                    true - will prevent the system from removing alerts after
 *                           they are delivered. The alert stay in the database
 *                           until the user remove them.
 *                    false - The system will remove the alert upon acknowledgment
 *                            from the reliable client in case one was setup or
 *                            upon send the alert to all the non reliable client/s.
 * @param jndiName - JNDI name of the data source database to be used for
 * persistence.
 * @param dbtype - one of the predefined typed defined in
 *                    {@link com.sun.caps.management.api.alerts.
 *                     AlertPersistenceDBType}
 * @param timeToLive -  maximum time in millisecond.
```

```
     * @param maxCount - Maximum number of alerts.
     * @param level -  an AlertLevelType representing the level.
     * @param policyList - an array of AlertRemovalPolicyType where the priority
     *                     of the given policy is defined by its position in the list.
     *                     i.e the lower the index that policy will be applied first.
     *                     possible values are:
     *                     ALERTS_AGE,ALERTS_COUNT,ALERTS_LEVEL.
     * @param enablePolicyExecution - true the current setting is enforced. False the
     *         current policy is ignored.
     * @param interval The interval the policy will be executed (is defined in
     *         milliseconds).
     * @param inMemoryCacheSize - The interval is defined in milliseconds.

     * @throws ManagementRemoteException if JMX related exception is thrown or
     *         the list of target names is null or empty.
     */
     public void enableAlertsPersistence(Boolean enableJournaling,String jndiName,
                                         AlertPersistenceDBType dbtype,Long timeToLive,
                                         Long maxCount,AlertLevelType level,
                                         AlertRemovalPolicyType[] policyList,
                                         Boolean enablePolicyExecution,Long interval,
                                         Integer inMemoryCacheSize) throws
     *                                   ManagementRemoteException;
```

AlertNotificationService

The Alert Notification Service notifies the client, such as Groovy, of an event. However, for the client to receive notifications it must subscribe using the Alert Notification Service API. There is a noticeable change between prior releases and Release 6 regarding reliable and non-reliable clients. With Release 5.1.x, the client was reliable, meaning that no events could be removed until their delivery was confirmed. With Release 6, a client can be non-reliable, which means there is no guarantee the client will receive every event. In Release 6, the last client set as "reliable" makes all clients before it unreliable.

Alerts Notification Service API

```
/**
     * request to the event management system to get an events that satisfy the
     * filter provided. The method will validate the call-back object for the
     * call-back method name and parameter (see below for more information).
     * This method allow the caller to register multiple time with diffrent
     * filtering,target and call-back parameters.
     *
     * @param filter - the filter that will be applied to the events prior to
     *                 forwarding them to this client. the valid keys for the map
     *                 are defined in  {@link com.sun.caps.management.api.alerts.
```

```
 *                AlertNotificationFilterElementType}.
 *                For the ALERTSEVERITY type the valid value are define in
 *                {@link com.sun.caps.management.api.alerts.AlertLevelType#}.
 *
 * @param targetNames - the server instances that is subscription will initially be
 *                      filtered on. if targetNames is/are defined it/they have
 *                      precedence over the servername element in the filter
 *                      mentioned above.
 *
 * @param CallbackObject an instance of the client object that contain the
 *                       call back method to be called when event received
 *                       from the server.
 * @param methodName the method name to be invoke when event received from the
 *                       server.
 *       IMPORTANT: THE METHOD MUST HAVE ONE PARAMETER OF TYPE
 *                      {@link com.sun.caps.management.api.alerts.Alert"} ALERT.
 *
 * @param requireReliableDelivery - true mean this client request that all events
 *                       should be delivered to him reliably otherwise the client
 *                       may miss event.
 *       IMPORTANT: THE SYSTEM ALLOW ONLY ONE RELIABLE CLIENT. THE  LAST CLIENT
 *                       TO SET IT TO TRUE TAKES OVER THE RELIABLE DELIVERY. IT
 *                       WILL AFFECT ALL THE SUBSCRIPTIONS DONE BY THE CLIENT IN
 *                       THE CURRENT APPLICATION SESSION.
 *
 * @param exceptionCallBack an instance of the client object that contain the call
 *                       back method to be called when an connectivity exception
 *                       is generated by this service.
 *
 * @param exceptionMethodName the method name to be invoke when an exception is
 *                       generated by this service.
 *       IMPORTANT: THE METHOD MUST HAVE ONE PARAMETER OF TYPE
 *                       {@link java.lang.Exception} EXCEPTION.
 *
 *       NOTE: the exception call be should be the same for all subscriptions
 *                       calls otherwise the last the exception call back
 *                       defined by the last subscription will be used.
 *
 * @return Unique identification string that need to be used in the un-subscribe
 *                       operation.
 *
 * @throws ManagementRemoteException
 *         1. if fail to communicate with the event management system
 *         2. unable to invoke call back method because of invalid parameter.
 */
public String subscribe(Map filter,String[] targetName,Object CallbackObject,String
 *          methodName, Boolean requireReliableDelivery,Object exceptionCallBack,
            String exceptionMethodName) throws ManagementRemoteException;
```

```
/**
 * request the event management system to stop forwarding events to this client
 * based on the subscription the caller made using the subscribe method.
 * once all the caller unsubscribe all the IDs any events that are waiting
 * to be delivered to this client will be discarded.
 *
 * @param - subscriptionIDs A list of IDs return by the subscribe call/s  that the
 *          caller wish to unsubscribe from.
 *
 * @throws ManagementRemoteException if fail to communicate with the Domain server.
 */
public void unsubscribe(String[] subscriptionIDs) throws ManagementRemoteException;

/**
 * utility method that returns the parameters the client used to subscribe for
 * alerts for the given subscription ID.
 *
 * @param - subscriptionIDs list return by the subscribe call/s.
 *
 * return a map keyed IDs on the provided as a parameter and values as
 *        SubcriptionInformationinstances.
 *        @see com.sun.caps.management.api.alerts.SubcriptionInformation
 * @throws ManagementRemoteException if fail to communicate with the Domain server.
 */
public Map getSubscriptionInformation(String[] subscriptionIDs);
```

Services — JavaCAPSManagementAPI

Currently there are ten Java CAPS Management Client API services:

- Administration
- Runtime Management
- Configuration
- Deployment
- Installation
- Performance Measurement
- Alert Management
- Alert Administration
- Alert Notification
- Alert Configuration
- JMS Management
- Log Management
- BPEL Management
- Data Management (MDM)
- Sun Adapters Management

- Target Option Behavior for the Management Client

There is also a Target option. It's behavior for the Java CAPS Management Client is also part of Services. Depending upon the values you specify, this option causes the install and deploy commands to behave differently. For details, see "Target Option Behavior for the Management Client" on page 42.

Note – The sample files for the services (for example: `AdministrationServiceSample.groovy`) and the Target option behavior are zipped, and are included with the Java CAPS 6 delivery.

Administration Service

The Administration Service enables

- Descriptors of component containers, shared libraries, service assemblies, and service units
- Consuming and provisioning endpoints exposed by all component containers
- Operations to retrieve the WSDL and XSD resources associated with each endpoint

TABLE 2 Administration Service Method Names and Descriptions

API Method Name	Description
getComponentInstallationDescriptor	Retrieves the jbi.xml deployment descriptor for the component.
getSharedLibraryInstallationDescriptor	Retrieves the jbi.xml deployment descriptor for the shared library.
getServiceAssemblyDeploymentDescriptor	Retrieves the jbi.xml deployment descriptor for the service assembly.
getServiceUnitDeploymentDescriptor	Retrieves the jbi.xml deployment descriptor for the service unit.
isJBIRuntimeEnabled	Checks to see if the JBI Runtime is enabled.
isServiceEngine	Checks for the ServiceEngine.
isBindingComponent	Checks for the BindingComponent.
getConsumingEndpoints	Retrieves the list of consuming endpoints for a component.
getProvisioningEndpoints	Retrieves the list of provisioning endpoints for a component.
getWSDLDefinition	Retrieves the primary WSDL associated with the specified endpoint.

TABLE 2 Administration Service Method Names and Descriptions *(Continued)*

API Method Name	Description
getWSDLImportedResource	Retrieves the WSDL or XSD associated with the specified endpoint and targetNamespace.
isClassicEnterpirseManagerUp	Checks to see if the Enterprise Manager server for non JBI based components is running.

Runtime Management Service

The Runtime Management Service:

- Enables lifecycle operations, such as start, stop, and shutdown
- Enables lifecycle operations, such as enable and disable, for Java EE applications
- Provides operations to list component containers, shared libraries, and applications for both JBI and Java EE

TABLE 3 Runtime Management Service Method Names and Descriptions

API Method Name	Description
isTargetUp	Checks to see if the target (server, cluster) is up or down.
listServiceEngines	Lists the service engines.
listBindingComponents	Lists the binding components.
listSharedLibraries	Lists the shared libraries.
listServiceAssemblies	Lists the service assemblies.
showServiceEngine	Shows the service engine conforming to various options.
showBindingComponent	Shows the binding component conforming to various options.
showSharedLibrary	Shows the shared library conforming to various options.
showServiceAssembly	Shows the service assembly conforming to various options.
listComponents	Retrieves a list of components.
getState	Retrieves the state of the runtime component (such as UP/DOWN/UNKNOWN/etc.).

TABLE 3 Runtime Management Service Method Names and Descriptions *(Continued)*

API Method Name	Description
getProperties	Retrieves a list of the component's properties.
startComponent	Starts the component.
stopComponent	Stops the component.
restartComponent	Restarts the component.
shutdownComponent	Shuts down the component.
startServiceAssembly	Starts service assembly.
stopServiceAssembly	Stops the service assembly.
shutdownServiceAssembly	Shuts down the service assembly.
listCompositeApplications	Retrieves a list of composite applications.
getRuntimeComponentProperties	Obtains a list of the runtime unit's properties.
getRuntimeComponentStatus	Obtains the state of the runtime component, that is, UP/DOWN/UNKNOWN/etc.
startRuntimeComponent	Starts the runtime component.
restartRuntimeComponent	Restarts the runtime component.
stopRuntimecomponent	Stops the runtime component.
listTargets	Returns all deployable targets in this domain. All groups and all servers (servers that are not part of any groups).
listLifecycleModules	Lists the lifecycle modules.
listExtensionModules	Lists the extension modules.
listSystemConnectors	Lists the system connectors modules.
listAppclientModules	Returns all deployed modules of specified type and on specified target.
listConnectorModules	Returns an array of deployed connectors.
listEjbModules	Returns all deployed modules of specified type and on specified target.
listWebModules	Returns all deployed modules of specified type and on specified target.
listJavaEEApplications	Returns a list of deployed JavaEE Applications. These are the applications that are deployed to a domain, and are registered under.

TABLE 3 Runtime Management Service Method Names and Descriptions *(Continued)*

API Method Name	Description
enableJavaEEApplication	Enables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also enabled.
enableAppclientModule	Enables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also enabled.
enableConnectorModule	Enables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also enabled.
enableEjbModule	Enables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also enabled.
enableWebModule	Enables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also enabled.
disableJavaEEApplication	Disables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also disabled.
disableAppclientModule	Disables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also disabled.
disableConnectorModule	Disables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also disabled.
disableEjbModule	Disables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also disabled.
disableWebModule	Disables an application or module on the specified target. In case of a cluster, the application references of the server instances in that cluster are also disabled.
isJavaEEComponentEnabled	Returns the status of the application as in configuration. If the specified target is null/blank/"domain", only the enabled flag of the actual application is used. Otherwise, the enabled flag of the application reference is used to determine the enabled status.

Configuration Service

The Configuration Service enables you:

- To configure component containers, the JBI runtime, etc.
- To configure logging for component containers and the JBI runtime
- To manage application configurations and manage application variables
- To verify application configurations before deployment

TABLE 4 Configuration Service Method Names and Descriptions

API Method Name	Description
getRuntimeLoggerLevels	Retrieves all the runtime loggers and their levels.
getRuntimeLoggerLevel	Looks up the level of one runtime logger.
setRuntimeLoggerLevel	Sets the log level for a given runtime logger.
getComponentLoggerLevels	Retrieves the component custom loggers and their levels.
setComponentLoggerLevel	Sets the component log level for a given custom logger
getComponentExtensionMBeanObjectNames	Retrieves the extension MBean object names.
getComponentConfiguration	Retrieves the component configuration.
setComponentConfiguration	Sets the component configuration.
getRuntimeConfigurationMetaData	Returns the runtime configuration metadata associated with the specified property. The metadata contain name-value pairs like default, descriptionID, descriptorType, displayName, displayNameId, isStatic, name, resourceBundleName, tooltip, tooltipId, etc.
setRuntimeConfiguration	Sets the runtime configuration.
getRuntimeConfiguration	Retrieves the runtime configuration.
getDefaultRuntimeConfiguration	Retrieves the default runtime configuration.
isServerRestartRequired	Checks if the server needs to be restarted to apply the changes made to some of the configuration parameters.

Deployment Service

The Deployment Service enables you to deploy and un-deploy:

- JBI Service Assemblies

- Java EE artifacts, such as enterprise archives (EAR files), EJB modules, web archives (WAR files), and Java EE connectors, application clients, etc.

TABLE 5 Deployment Service Method Names and Descriptions

API Method Name	Description
deployServiceAssembly	Deploys the service assembly.
deployServiceAssemblyFromDomain	Deploys the service assembly from the domain target.
undeployServiceAssembly	Undeploys the service assembly.
deployJavaEEComponent	Deploys a component to the given array of targets which can be domains, clusters, or standalone instances. Since there are restrictions around how clusters and standalone instances share deployments, the component bits are deployed only to the first target in the list and then application references are created for the rest of the targets in the array specified.
undeployJavaEEComponent	Undeploys a component to the given array of targets which can be domains, clusters, or standalone instances. Since there are restrictions around how clusters and standalone instances share deployments, the application references are removed for all the targets except for the first one in the array. After this, the component bits are undeployed from the first target in the array.

Installation Service

The Installation Service enables you to:

- To install and uninstall JBI component containers and shared libraries
- To upgrade component containers without undeploying existing artifacts

TABLE 6 Installation Service Method Names and Descriptions

API Method Name	Description
installComponent	Installs the component (service engine, binding component).
uninstallComponent	Un-installs the component (service engine, binding component).
installSharedLibrary	Installs the shared library.
uninstallSharedLibrary	Un-installs the shared library.

TABLE 6 Installation Service Method Names and Descriptions *(Continued)*

API Method Name	Description
installComponentFromDomain	Installs the component (service engine, binding component) from the domain target.
installSharedLibraryFromDomain	Installs the shared library from domain target.
upgradeComponent	Upgrades the component (service engine, binding component). Upgrades a component in a way that actually involves the component. During the upgrade processing, the component's implementation of the new upgrade SPI is invoked to give the component the opportunity to perform any special processing necessary to complete the upgrade. Components which do not provide an implementation of the upgrade SPI can still be updated using the updateComponent API. Also, in the upgrade implementation, changes in the component's installation descriptor are allowed, with the exception of the component name (for obvious reasons). This allows new shared library dependencies, changes to the class names of the component's SPI implementations, and changes to the component's class loading preferences (class path and class loading order). These changes are allowed regardless of whether or not the component provides an implementation of the new upgrade SPI.

Performance Measurement Service

The Performance Measurement Service attempts to address performance characterization for developers and administrators.

- JBI Framework
  - StartTime
  - StartupTime
  - Component count
  - Endpoint count
  - ServiceAssembly count
- Normalized Message Router (NMR)
  - Time spent in each component
  - Time spent in DeliveryChannel
  - Time spent in NMR
  - Per Component/Endpoint
  - Queried by Component/Endpoint

- Component Endpoints
  - UpTime
  - Total sent/received requests, such as “in” messages
  - Total sent/received replies, such as “out” messages
  - Total sent/received faults
  - Total completed exchanges, such as consumed/provided
  - Total active exchanges, such as consumed/provided
  - Total error exchanges, such as consumed/provided
  - Response Time
  - Status Time
  - Active MessageIds
  - Waiting MessageIds

TABLE 7 Performance Measurement Service Method Names and Descriptions

API Method Name	Description
clearPeformaceInstrumentationMeasurement	Resets the performance measurements on the endpoint.
getPerformanceInstrumentationEnabled	Retrieves the performance measurement enabling flag.
getPerformanceInstrumentationMeasurement	Retrieves the performance measurement data for the specified endpoint.
getPerformanceMeasurementCategories	Retrieves the performance statistics categories. Each item in the array is the key to the composite performance data, which also indicates the type of measurement, such as normalization.
setPerformanceInstrumentationEnabled	Sets the performance measurement enabling flag.
getFrameworkStatisticsAsTabularData	This method is used to provide JBIFramework statistics in the given target. Data is displayed in tabular format.
getFrameworkStatistics	This method is used to provide JBIFramework statistics in the given target.
getComponentStatistics	This method is used to provide statistics for the given component in the given target.
getComponentStatisticsAsTabularData	This method is used to provide statistics for the given component in the given target. Data is displayed in tabular format.
getEndpointStatisticsAsTabularData	This method is used to provide statistic information about the given endpoint in the given target. Data is displayed in tabular format.

TABLE 7 Performance Measurement Service Method Names and Descriptions *(Continued)*

API Method Name	Description
getEndpointStatistics	This method is used to provide statistic information about the given endpoint in the given target.
getNMRStatisticsAsTabularData	This method is used to provide statistics about the message service in the given target. Data is displayed in tabular format.
getNMRStatistics	This method is used to provide statistics about the message service in the given target.
getServiceAssemblyStatisticsAsTabularData	This method is used to provide statistics about a Service Assembly in the given target. Data is displayed in tabular format.
getServiceAssemblyStatistics	This method is used to provide statistics about a Service Assembly in the given target.
enableMessageExchangeMonitoring	This method is used to enable monitoring of timing information about message exchanges.
disableMessageExchangeMonitoring	This method is used to disable monitoring of timing information about message exchanges.

Alert Management Service

TABLE 8 Alert Management Service Method Names and Descriptions

API Method Name	Description
getAlertAdministrationService	Creates and returns Alert Administration Service.
getAlertConfigurationService	Creates and returns Alert Configuration Service.
getAlertNotificationService	Creates and returns Alert Notification Service.

Alert Administration Service

TABLE 9 Alert Administration Service Method Names and Descriptions

API Method Name	Description
getAlerts	Retrieves all alerts.
getAlertFieldNames	Retrieves a list of AlertInstance DB fieldNames.

TABLE 9 Alert Administration Service Method Names and Descriptions *(Continued)*

API Method Name	Description
update	Updates the field's value based on criteria.
delete	Deletes the alert object from persistence storage.
observe	Sets the list of AlertInstance's observationState to observed.
resolved	Sets AlertInstance's observationState to resolved based on a criteria.
reset	Sets the list of AlertInstance's observationState to its initial state, which is unobserved.
UpdateComment	Updates the comment field of the alert.
resetAll	Sets the observationState to its initial state for all the table entries.
resolveAll	Sets Alert observationState to resolved state for all the table entries.
observeAll	Sets Alert observationState to observed state for all the table entries.

Alert Notification Service

TABLE 10 Alert Notification Service Method Names and Descriptions

API Method Name	Description
subscribe	Requests the event management system to get an event that satisfies the filter provided. The method will validate the call-back object for the call-back method name and parameter (see below for more information). This method enables the caller to register multiple times with different filtering, target, and call-back parameters.
unsubscribe	Requests the event management system to stop forwarding events to this client based on the subscription the caller made using the subscribe method. Once all the callers unsubscribe all the IDs, any events that are waiting to be delivered to this client are discarded.

TABLE 10 Alert Notification Service Method Names and Descriptions *(Continued)*

API Method Name	Description
getSubscriptionInformation	Utility method that returns the parameters the client used to subscribe for alerts for the given subscription ID.

Alert Configuration Service

TABLE 11 Alert Configuration Service Method Names and Descriptions

API Method Name	Descriptions
enableAlertsPersistence	Enables alerts persistence in the alerts database. Enabling provides reliable alerts delivery in case of delivery channel failure or application server restart.
disableAlertsPersistence	Disables alerts persistence in the alerts database.
isAlertsPersistenceEnabled	Returns the last setting of the alert persistence enabling operation. True if enabled, otherwise false.
isAlertsJournalEnabled	Returns the last setting of the alert journal enabling operation. True if enabled, otherwise false.
setPersistenceDataSourceJndiName	Sets the JNDI name of the data source database to be used for persistence. If not provided at least once the persistence will be disabled even if enableAlertsPersistence is set to true.
getPersistenceDataSourceJndiName	Returns the last set JNDI name for the alert persistence data source.
setPersistenceDataBaseType	Sets the database type to be used for persistence. Derby is the assumed default database. If a different database is used, this method should be called prior to enabling the persistence.
getPersistenceDataBaseType	The return value represents the last set DB type.
setPersistedAlertsMaxAge	Sets the maximum time a persisted alert is stored in the alert database before it is deleted as part of the removal policy.
getPersistedAlertsMaxAge	Returns the last setting for the allowed persisted alert age. A value of 0 current time could cause all persisted alerts to be deleted. When a negative value is used this policy element is ignored.

TABLE 11 Alert Configuration Service Method Names and Descriptions *(Continued)*

API Method Name	Descriptions
setPersistedAlertsMaxCount	Sets the maximum number of alerts allowed to be persisted before an alert is deleted as part of the removal policy in effect. If persistence is on and the count set to zero, it negates journaling being on.
getPersistedAlertsMaxCount	Returns the last setting for the maximum of alerts allowed to be persisted. A value of 0 means no alerts are persisted.
setPersistedAlertsLevel	The priority based alert level that is part of the removal policy. The priorities are as follows (from low to high): INFO, WARNING, MINOR, MAJOR, CRITICAL, and FATAL. All alerts from the provided level and below will be candidates for removal.
getPersistedAlertsLevel	The returned value represents the last setting for the level of alerts that are allowed to be removed from persistence for each target.
setPersistedAlertsRemovelPolicy	Sets the effective policy for the removal of persisted alerts.
getPersistedAlertsRemovalPolicy	Returns the return value representing an array of the last setting the policy used when persisted alerts are to be removed. An empty array means no policy is enforced.
enablePersistedAlertsPolicyExecution	Enables or disables the ability to use the removal policy.
isPersistedAlertsPolicyExecutionEnabled	The returned value represents the last setting that enables/disables the persisted alerts removal policy.
setPersistedAlertsRemovelPolicyExecInterval	Sets how often the persisted alerts removal policy is executed.
getPersistedAlertsRemovelPolicyExecInterval	Returns the returned value representing The last interval setting of the persisted alerts removal policy is executed.
setInMemoryAlertsCacheMaxSize	Sets the maximum number of alerts that can be cached in memory prior to being delivered to the registered listeners.
getInMemoryAlertsCacheMaxSize	Returns the returned value representing the last setting of the maximum size of memory in the alerts cache.
setAlertTableName	Sets the persisted alerts table name.

TABLE 11 Alert Configuration Service Method Names and Descriptions *(Continued)*

API Method Name	Descriptions
getPersistedAlertsCount	Returns the total number of alerts currently persisted. This value is volatile and could change.
enableAlertsPersistence	This API enables the caller to set all the parameters defined in the other APIs in this interface. All the settings are applied prior to enabling the persistence.

JMS Management Service

TABLE 12 JMS Management Service Method Names and Descriptions

API Method Name	Description
getServerProperties	Returns the server properties.
isServerReady	Checks if the message server is ready.
getXids	Returns a list of transactions on the message server.
rollbackXid	Rolls back a specified transaction on the message server.
getTopicProperties	Retrieves the specified topic properties.
getTopics	Retrieves the list of topics on the message server.
getTopicsWithHeaders	Returns a list of topics with header properties on the message server.
getTemporaryTopics	Retrieves a list of temporary topics.
createTopic	Creates a new specified topic with the specified name on the message server.
deleteTopic	Deletes a specified topic on the message server.
getTopicMessage	Returns a message body of a specified message of a specified topic.
getTopicMsgProperties	Returns the specified topic message properties.
getTopicMsgPropertiesList	Returns a list of messages with the properties for the given start message index.
getSubscribers	Returns a list of subscribers for a specified topic.
changeTopicTextMessage	Changes the content of a specified text message of a specified topic.

TABLE 12 JMS Management Service Method Names and Descriptions *(Continued)*

API Method Name	Description
changeTopicBytesMessage	Changes the content of a specified bytes message of a specified topic.
deleteTopicMessage	Deletes a specified message from a specified topic.
getTopicMessageType	Returns the message type of a specified message from a specified topic.
suspendTopic	Suspends a specified topic.
resumeTopic	Resumes the suspended topic.
submitNewMessage	Submits a new message to a specified topic or queue on the message server.
createTopicDurableSubscriber	Creates a new specified topic durable subscriber for a specified topic on the message server.
unsubscribeDurableSubscriber	Unsubscribes a specified durable subscription on the message server.
republishTopicMessage	Resends a specified journalled message to a specified queue on the message server.
getTopicStatistics	Returns the statistics of a specified topic on the message server.
getQeueueProperties	Retrieves the queue properties.
getQueues	Retrieves the list of queues on the message server.
getQueuesWithHeaders	Returns a list of queues with header properties on the message server.
getTemporaryQueues	Retrieves the list of temporary queues.
createQueue	Creates a new queue with the specified name on the message server.
deleteQueue	Deletes a specified queue on the message server.
getQueueMsgProperties	Returns the specified topic message properties.
getQueueMsgPropertiesList	Returns a list of messages with its properties for the given start message index.
getQueueMessage	Returns a message body of a specified message of a queue.
getReceivers	Returns a list of receivers for a specified queue.

TABLE 12 JMS Management Service Method Names and Descriptions *(Continued)*

API Method Name	Description
changeQueueTextMessage	Changes the content of a specified text message of a queue.
changeQueueBytesMessage	Changes the content of a specified bytes message of a specified queue.
deleteQueueMessage	Deletes a specified message from a specified queue.
getQueueMessageType	Returns the message type of a specified message of the queue.
suspendQueue	Suspends a specified queue.
resumeQueue	Resumes the suspended queue.
resendQueueMessage	Republishes a specified journalled message to a specified topic on the message server.
getQueueStatistics	Returns the statistics of a specified queue on the message server.

Log Management Service

TABLE 13 Log Management Service Method Names and Descriptions

API Method Name	Description
getLogString	Reads the page, filter, and search log. Returns only the lines prepared as a single String without any other metadata.
getLog	Reads, filters, and searches a page of lines from the log; the lines are returned in a Map.
getLogAsString	Logs, filters, and searches a page of lines from the log; the lines are prepared as one string.
listLoggerNames	Lists the names of loggers registered with the target.
listLoggerObjectNames	List the names of loggers registered with the target.
isLoggerRegistered	Asks if the logger instance is registered with the target server instance.
registerLogger	Registers a logger instance with the target server instance.
unregisterLogger	Unregisters a logger instance with the target server instance.

TABLE 13 Log Management Service Method Names and Descriptions *(Continued)*

API Method Name	Description
setLogFile	Set the logger file name.
getLogFile	Obtains the logger file name.

BPEL Management Service

The BPEL Management Service supports two types of API method names:

- Java CAPS (non-JBI based components)
- Java CAPS (JBI based components)

Note – These APIs are in addition to the APIs exposed for BPEL from the configuration service.

TABLE 14 BPEL Management Service API Method Names and Descriptions (non-JBI Based Components)

API Method Name	Description
getBPELInstances	Obtains BPEL instances given optional BPEL process QName and/or an optional BPStatus or an optional instance ID. If instanceId is present, the BPEL process QName and BPStatus are ignored. The maximum instances to return is 1000, user specifies a lower number for maxRecords to limit the number of the instances returned. If the BPInstanceQueryResult.overflow is true, it indicates the number of qualifying instances is larger than 1000, no instances are returned in the result list, user should specify a maxRecords (<= 1000) and sortColumn and order.
getBPELInstanceActivityStatus	Obtains the list of ActivityStatus of a BPEL instance.
getBPELProcessIds	Obtains the list of BPEL process QName as String within a service unit.
getBPELInstanceFault	Obtains the fault detail of a faulted BPEL instance.
getInvokerInstance	Obtains the list of an invoker (parent) BPEL instance(s) that invoked a specific bpel instance
getInvokeeInstance	Obtains the list of invokee (sub) BPEL instance(s) that a specific BPEL instance invoked.
isMonitoringEnabled	Asks if Monitoring is enabled.
isPersistenceEnabled	Asks if Persistence is enabled.

TABLE 14 BPEL Management Service API Method Names and Descriptions (non-JBI Based Components) *(Continued)*

API Method Name	Description
resumeInstance	Resumes a Business Process Instance for a given Business Process Instance.
terminateInstance	Terminates a Business Process Instance for a given Business Process Instance
suspendInstance	Suspends a Business Process Instance for a given Business Process Instance
getBPELInstanceFault	Obtains the fault detail of a faulted BPEL instance.
suspendAllInstance	Suspends all instances of a BPEL process.
resumeAllInstance	Resumes all instances of a BPEL process.
terminateAllInstance	Terminates all instances of a BPEL process.
changeVariableValue	Changes the BPEL variable value. Note that, only the leaf node can be changed.
getVariableValue	Obtains the BPEL variable value. The content of the BPEL variable is returned.
listBPELVaraibles	Obtains the BPEL variable information for a BPEL instance.

TABLE 15 BPEL Management Service API Method Names and Descriptions (JBI Based Components)

API Method Name	Description
setBusinessProcessInstanceVariableValue	Sets the value for the property using XPath on a Part given a business process instance, container, part, the XPath expression, and value.

Master Data Management (MDM) Service

This API service provides the capability to integrate and manage data and applications in a complex and distributed enterprise business environment, including the following data management products:

- Master Index Studio (formerly eView Studio and Single Patient View)
- Data Integrator (formerly eTL Integrator)
- Data Quality
- Data Services
- Data Migrator

TABLE 16 Master Index Management (MDM) Service API Method Names and Descriptions

API Method Name	Description
listApplicationNames	Returns a list of MDM Applications that are currently deployed.
getApplicationStatus	Returns the status of MDM Applications.
getDatabaseStatus	Returns the status of the Database connection.
getWebModuleStatus	Returns the status of the MDM Web Application Module.

Sun Adapters Management Service

TABLE 17 Sun Adapters Management Service API Method Names and Descriptions

API Method Name	Description
start	Starts the component; the semantics of this operation is left to the implementation.
restart	Restarts the component; the semantics of this operation is left to the implementation.
stop	Stops the component; the semantics of this operation is left to the implementation.
getStatus	Returns the status.
getProperties	Returns the properties.
isStartable	Determines whether a "start" button would be presented to the user.
isRestartable	Determines whether a "restart" button would be presented to the user.
isStoppable	Determines whether a "stop" button would be presented to the user.
getType	Returns the type of the adapter—monitored component.
getConfiguration	Returns the adapter—monitored component's configuration.
getState	Returns the state of the adapter—monitored component.

TABLE 17 Sun Adapters Management Service API Method Names and Descriptions *(Continued)*

API Method Name	Description
getTargetState	Returns the target state of the adapter—monitored component. This is the state that the adapter component is or was changing to. This status may be different from that returned by getState().
getRAConfiguration	Returns the adapter's metadata details.

Target Option Behavior for the Management Client

The --target option causes the install and deploy commands to behave differently. Depending upon the option value specified, these differences can be radical.

Note – Two of the target option names are constant: "server" and "domain." They represent an instance of an operator and are replaceable with a name specific to the current template.

TABLE 18 Target Option Value and Behavior

Option Value	Behavior
server	The command is executed against the embedded domain administration server (DAS) instance.
domain	When the target option is the literal string *domain*, the component is executed against the domain itself, but not to any instances or clusters running on the domain. This option value is only applicable for install or deploy commands.
cluster name	When a cluster name is specified, the command is executed against all instances in the specified cluster.
instance name	When an instance name is specified, the command is executed against the specific instance specified.

Writing Java Code to Access APIs Using Java Code Samples

Use the Java code samples to write your own Java Code to access the APIs. The samples are included with the software delivery as zipped files.

When the Alert Management Subsystem receives alerts it sends them to subscribers, such as Enterprise Manager or Groovy. There are two ways the process works.

- When persistence is not turned on alerts are sent directly to all subscribers that use the client API. If there are no subscribers the alerts are dropped.

- When persistence is turned on alerts are sent to a database. If there are no subscribers the alerts are persisted in the database until subscribers are ready to consume them. Optionally, you can enable journaling, which ensures that the alerts are never deleted.

Setting Up Databases

Derby is the database that ships with Java CAPS. However, you can set up and use another database. But remember, you are limited to using a database that Java CAPS supports.

- Derby
- Oracle
- Sybase
- DB2
- PointBase

As an option to the following procedure, you could replace steps 4-6 by using the Alert Configuration Management API to write a groovy script or small Java utility. For an example of this see the *JavaCAPS6*`/ESB_API_KIT/samples` directory, if *JavaCAPS6* is the directory where you extracted `EM_API_KIT.zip`.

Caution – Remember that the last command you script or program and then execute should be enabling persistence in the specified database. Examples of methods from the Alert Configuration Service API are setPersistenceDataSourceJndiName, getPersistenceDataSourceJndiName, setPersistenceDataBaseType, getPersistenceDataBaseType, and optionally setAlertTableName.

In case the database has a limit to the table name, for example Oracle is limited to 30 characters, the auto-generated table may exceed that limit. Use the Alert Configuration API (SetAlertTableName) to set the table name. Keep in mind that each domain must have unique table name to prevent events from one domain appearing in another domain.

▼ To Set Up a Database Using Enterprise Manager

1 **Start the domain you want to use.**

2 **Use the Sun Java System Application Server Admin Console or AS Admin command-line utility to set up a connection pool and resource.**

Note – Record the name you assign to the resource name.

For detailed instructions on how to perform this task see Admin Console online help.

3 **Start the selected database.**

4 **Start Enterprise Manager and add the domain you started in step 1.**

5 **Launch the Alert Configuration screen (for information see *Monitoring Java EE Components*).**

   a. **Select the database type that matches the database you selected.**

   b. **Enter the JNDI name.**

   Note – This is the resource name you created in step 2.

6 **Enable persistence and journaling, and then click Save to commit the changes.**

Note – When you enable persistence you do not have to enable journaling; that is, journaling is optional when persistence is enabled.

▼ To Set Up a Database Using a Scripting Utility

1 **Start the domain you want to use.**

2 **Use the Sun Java System Application Server Admin Console or AS Admin command-line utility to set up a connection pool and resource.**

Note – Record the name you assign to the resource name.

For detailed instructions on how to perform this task see Admin Console online help.

3 **Start the selected database.**

4 **Write a scripting utility to call the appropriate APIs.**

```
setPersistenceDataSourceJndiName

setPersistenceDataBaseType(AlertPersistenceDBType dbtype)

(Optional) setAlertTableName(String tableName)

enableAlertsPersistence(Boolean enableJournaling)
```

Or, optionally, to execute all of the above APIs, use:

```
enableAlertsPersistence(Boolean enableJournaling,String jndiName,
                                    AlertPersistenceDBType dbtype,Long timeToLive,
                                    Long maxCount,AlertLevelType level,
```

```
                                    AlertRemovalPolicyType[] policyList,
                                    Boolean enablePolicyExecution,Long interval,
                                    Integer inMemoryCacheSize) throws
*                                   ManagementRemoteException;
```

Note – The order here is not set and can be altered except for enabling persistence, which must be last.

Using Oracle and Other Databases for Alert Persistence

Oracle, and the other supported databases besides Derby, are also capable performing alert persistence. However, there a number of changes you must perform to get it to work.

Note – If you plan to use Oracle instead of Derby for alert persistence, make sure you read this topic.

▼ To Set Up an Oracle Database for Alert Persistence

1 **Modify the** `eventmanagement.properties` **file under** `appserver/domains/domain1/jbi/config`**.**

   a. **Modify the DatabaseType to** `ORACLE`**.**

   b. **Change the AlertTablename to** `EMHOSTNAMEVistastccom8080`**.**

      Note – This step is required as the default name, `EVENTMANAGEMENTHOSTNAMEistastccom8080`, exceeded Oracle's 30-character limit for table names.

      ---

   c. **Change the DBJndiName to** `OracleXPDB`**.**

      Note – You create this in the Admin Console; this is noted in step 2.

      ---

   d. **Change PersistenceEnbled to** `true`**.**

   e. **Using Enterprise Manager, set the values for DatabaseType, DBJndiName, and PersistenceEnabled in the normal Alert Management Config Agent.**

Note – The database scripts should probably be run before enabling persistence. The table name must be changed in the file manually.

---

2 Create the JDBC connection pool and resource in the Sun Java System Application Server Admin Console.

a. Add the location of `classes12.zip` to the classpath, JVM Settings→Path Settings→Classpath Suffix, and then restart the domain.

Note – This is needed to get the datasource for Oracle.

---

b. Create the Connection Pool for Oracle.

i. Enter a name, such as OracleXPPool, but this name can be your choice.

ii. Select `javax.sql.DataSource` for the Resource Type.

iii. Select Oracle for the Database Vendor.

iv. Set the appropriate properties:

User: `eventdb_user`

DatabaseName: `orcl`

Note – This and other database specific configurations may change depending on how you configured the Oracle database.

Password: `eventdb_user`

ServerName: `hostname`

Note – This is the server where the database is running.

PortNumber: `1521`

URL: `jdbc:oracle:thin:@hostname:1521:orcl`

Caution – This URL may actually override the other settings; it should match the other settings.

v. **Create the JDBC Resource.**

Enter a JNDI Name, for example `OracleXPDB`.

Note – This should match what is set in the `Alert Management Config Agent/eventmanagement.properties` file mentioned above.

Select the appropriate Pool Name; in our example we used OracleXPPool.

3 **Create the user, tables, etc. needed for alert persistence and journaling manually.**

Note – This is automatically done for Derby.

There are database scripts that are packaged in the `jbi_rt.jar` file under `appserver/jbi/lib`. However there are some errors, so you need to be correct these scripts manually.

- The example scripts have been modified to work with an Oracle 10 GB database; see "Oracle Script Examples" on page 48.
- Run the `create_event_store_user.sql` and `create_event_store_schema.sql` scripts, in that order, with the system (admin) user.
- Corrections made in this example:
  - Set the absolute path to tablespace data files (database installation dependent)
  - Modify the command to match Business Process persistence
  - Comment out the second data file
  - Move the comments
  - Fix the table name references to match user schema and table name, which were set above in `eventmanagement.properties`
  - Fix the column reference; that is, change the second column name from event_timestamp to timestamp
  - Change the datatype from timestamp to decimal
  - Fix the reference for sequence

Oracle Script Examples

truncate_event_store_schema.sql

```
TRUNCATE TABLE eventdb_user.EMHostNameVistastccom8080;
```

create_event_store_schema.sql

```
create table eventdb_user.EMHostNameVistastccom8080(
      id NUMBER CONSTRAINT ID_PK PRIMARY KEY,
            timeStamp decimal,
         physicalHostName varchar(256),
            environmentName varchar(256),
            logicalHostName varchar(256),
            serverType varchar(256),
            serverName varchar(256),
            componentType varchar(256),
            componentProjectPathName varchar(1024),
            componentName varchar(256),
            eventType varchar(256),
            severity integer,
            operationalState int,
            messageCode varchar(256),
            messageDetail varchar(4000),
            observationalState int,
            deploymentName varchar(256));
);
-- INSERT statement need to use it to insure autoincrement functionality
CREATE SEQUENCE eventdb_user.autoincrement_id;
create index  eventTime on eventdb_user.EMHostNameVistastccom8080(timeStamp);
```

create_event_store_user.sql

```
--Create a tablespace named EVENTDB_USER_DB. Change this value if a different
name is desired.
--Specify the name and the location of the file where the data related to
the tablespace
--created above will be stored. The location is by default the location determined by
--the database server/instance on which this script is run
--For example, for Windows c:\MyDatafiles\EVENTDB_USER_DB.dat, for Unix
/dev/home1/EVENTDB_USER_DB.dat
--Note that the name of the actual file need not be EVENTDB_USER_DB.dat
--Specify the size constraints

    -- Window and Oracle 9i there is a limitation on file size, it is 2 GB.
This by default creats 4GB, add more files if you need more than 4 GB.
    --- provide abosolute path if you preference is not default location
'C:\OracleDirectory\EVENTDB_USER_DB.dat' SIZE 2000M,
```

```
CREATE TABLESPACE EM_EVENTSTORE_DB
 DATAFILE
    'C:\oracle\product\10.2.0\oradata\orcl\EVENTDB_USER_DB.dat' SIZE 512M REUSE
AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED;

    -- 'C:\oracle\product\10.2.0\oradata\orcl\EVENTDB_USER_DB1.dat' SIZE 512M
REUSE AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED --- provide abosolute path
if you preference is not defaultlocation 'C:\OracleDirectory\EVENTDB_USER_DB1.dat'
SIZE 2000M
        -- when TABLESPACE is created with these options performance is degrading
gradually as more and more records added to schema  EXTENT MANAGEMENT LOCAL SEGMENT
SPACE MANAGEMENT AUTO


--Create a new user EVENTDB_USER. Change the name if so desired. Password will
be same as
--the user name by default. This username and password will be used to create the
--connection pool on the application server. Also specify the tablespace
and the quota on
--the tablespace the user has. Note that if you used a different tablespace
name above,
--you will have to specify that tablespace name here.

CREATE USER EVENTDB_USER IDENTIFIED BY EVENTDB_USER
DEFAULT TABLESPACE EM_EVENTSTORE_DB
QUOTA UNLIMITED ON EM_EVENTSTORE_DB
TEMPORARY TABLESPACE temp
QUOTA 0M ON system;

--Modify the user name if the default user name was changed

GRANT CREATE session to EVENTDB_USER;
GRANT CREATE table to EVENTDB_USER;
GRANT CREATE procedure to EVENTDB_USER;
```

drop_event_store_schema.sql

```
DROP TABLE eventdb_user.EMHostNameVistastccom8080;
drop sequence eventdb_user.autoincrement_id;
```

drop_event_store_user.sql

```
--Drop the user that was created earlier. Note that if you chose a
different name for the
--user while creating the user, you will have to specify that name here.
DROP USER EVENTDB_USER CASCADE;
```

```
--Drop the tablespace that was created earlier. Note that if you chose a
different name for
--the tablespace while creating the user, you will have to specify that name here.
DROP TABLESPACE EM_EVENTSTORE_DB INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;

--Manually delete the datafiles that were created. If you used the defaults
while creating
--the datafiles, the names would be EVENTDB_USER_DB1.dat'and 'EVENTDB_USER_DB2.dat'
```

Setting Up Scripting Engines

The Java CAPS Management and Monitoring API should be callable from any JSR-223 Scripting Environment. This section documents how to set up these scripting engines like Groovy, JRuby, Jython (Java Python), JACL (Java TCL) or one of the 25 other JSR-223 scripting engines capable of calling the Java CAPS Management and Monitoring APIs that are currently available . You can use these instructions to setup your own environment to invoke these APIs from the JSR-223 Scripting Environment of your choice.

Downloading, Installing, and Setting Up A Scripting Environment

For the purposes of this topic, we will focus on four scripting engines:

- Groovy: The Java CAPS Management and Monitoring API has been tested and works with Groovy 1.0, Groovy 1.1 beta 1, and with the latest Groovy 1.1 beta 2. Use the installation instructions that are available to install the engine, and ensure that it runs properly. Download Groovy from `http://groovy.codehaus.org/`.
- JRuby: The Java CAPS Management and Monitoring API has been tested and works with JRuby 1.0 and with the latest JRuby 1.0.1. Use the installation instructions that are available to install the engine, and ensure that it runs properly. Download JRuby from `http://jruby.codehaus.org/`.
- Jython (Java Python): The Java CAPS Management and Monitoring API has been tested and works with Jython 2.2 RC1 and with the latest Jython 2.2. Use the installation instructions that are available to install the engine, and ensure that it runs properly. Download Jython from `http://www.jython.org/Project/index.html`.
- JACL (Java TCL): The Java CAPS Management and Monitoring API has been tested and works with the latest JACL 1.4.0. Use the installation instructions that are available to install the engine, and ensure that it runs properly. You may have to modify `jaclsh.bat` in the `bin` folder as described in the `readme.txt` file that is included with the download. My `bin/jaclsh.bat` looks like this after modification. Download JACL from `http://tcljava.sourceforge.net/docs/website/index.html`.

Note – The sample files for the services (for example: `AdministrationServiceSample.groovy`) and the Target option behavior are zipped, and are included with the Java CAPS 6 delivery.

Setting Up a Scripting Environment to Invoke Java CAPS Management and Monitoring APIs

There are two files that are necessary before you set up your scripting environment to invoke Java CAPS Management and Monitoring APIs:

- `env.bat`
- `caps.management.client.jar`

Note – These files are included with the Java CAPS delivery.

After downloading `env.bat` and `caps.management.client.jar`, modify the environment variables in `env.bat`.

To modify environment variables, click "To Modify the Environment Variables in env.bat" on page 51.

▼ To Modify the Environment Variables in env.bat

1. **Set the JAVA_HOME variable to the JDK/JRE Home folder where your JDK or JRE is installed.**

   For example: `set JAVA_HOME=C:\java\jdk1.6.0`.

2. **Set the SJSAS_HOME variable to the SJSAS Home folder where your Sun Java System Application Server is installed.**

   For example: set `SJSAS_HOME=C:\CAPS6\SJSAS`.

3. **Set the CAPS_MANAGEMENT_HOME variable to the folder where you saved the Java CAPS Management and Monitoring API Java archive file locally.**

   For example: `set CAPS_MANAGEMENT_HOME=C:\scripting\engines\common`.

4. **Set the ENGINE_HOME variable to your Scripting Engine Home folder where your scripting engine is installed.**

   For example: `set ENGINE_HOME=C:\scripting\engines\groovy\groovy-1.1-beta-2`.

Note – You are now ready to run the scripting samples.

Using a Scripting Language to Exercise the Java CAPS Management and Monitoring APIs

Before running the scripting samples, it is assumed that you installed one of the scripting engines.

- Groovy (located at `C:\scripting\engines\groovy\groovy-1.1-beta-2`)
- JRuby (located at `C:\scripting\engines\jruby\jruby-1.0.1`)
- Jython (located at `C:\scripting\engines\jython\jython2.2`)
- JACL (located at `C:\scripting\engines\jacl\jacl140`)

It is also assumed that the Sun Java System Application Server is running with the following settings:

Host	HTTP Administration Port	Admin UserName	Admin Password
`localhost`	`4848`	`admin`	`adminadmin`

Tip – If your environment is different, change the following steps or the scripts according to your environment or installation.

Currently Sun supports these services:

- Administration Service
- Configuration Service
- Deployment Service
- Installation Service
- Runtime Management Service

Additional services will be supported in the future.

Exercising the Administration Service

Before you can exercise the Administration Service, you must have the following installed, and/or deployed, in your environment:

- A binding component named sun-http-binding
- A shared library named sun-wsdl-library

- A service assembly named SynchronousSampleApplication, which contains a service unit named SynchronousSampleApplication-SynchronousSample

Tip – You must create the SynchronousSampleApplication from NetBeans, deploy it, and start it before you run the Administration Service scripts.

Note – Change the script as you see fit to run in your environment.

Exercising the Administration Service in Groovy

The Groovy script for executing the Administration Service is attached in `AdministrationServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\AdministrationServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the Configuration Service

Before you can exercise the Configuration Service, you must have the following installed in your environment:

- A binding component named sun-http-binding
- A service engine named sun-bpel-engine

Note – Change the script as you see fit to run in your environment.

Exercising the Configuration Service in Groovy

The Groovy script for executing the Configuration Service is attached in `AdministrationServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\ConfigurationServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the Deployment Service

The deployment service tries to deploy a Service Assembly named SynchronousSampleApplication. Before you run the Deployment Service scripts, from NetBeans you must first create a SynchronousSampleApplication, deploy it, and start it. Change the script as you see fit to run in your environment.

Exercising the Deployment Service in Groovy

The Groovy script for executing the Deployment Service is attached in `DeploymentServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\DeploymentServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the Installation Service

Before you can exercise the Installation Service, you must have `aspectserviceengine.jar` installed in your environment.

Caution – Read the following carefully.

1. A service engine named sun-aspect-engine attempts to stop, shutdown, and uninstall itself.
2. The service engine then attempts to install sun-aspect-engine from the attached `aspectserviceengine.jar`.
3. Before you run the Installation Service sample scripts, install `aspectserviceengine.jar` in your environment and start it.
4. Change the script as you see fit to run in your environment.

Exercising the Installation Service in Groovy

The Groovy script for executing the Installation Service is attached in `InstallationServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\InstallationServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the Runtime Management Service

Before you can exercise the Runtime Management Service, you must have the following installed and deployed in your environment:

- A service engine named sun-bpel-engine
- A service assembly named SynchronousSampleApplication, which you created from NetBeans.

Note – Change the script as you see fit to run in your environment.

Exercising the Runtime Management Service in Groovy

The Groovy script for executing the Runtime Management Service is attached in `RuntimeManagmentServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\RuntimeManagementServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the JMS Management Service

Before you can exercise the JMS Management Service, you must have first installed the Sun JMS IQ Manager from the Java CAPS 6 Installer and deployed it in your environment:

Note – The Java CAPS Installer, by default, allows you to install Sun JMS IQ Manager during the installation process. However, you can install Sun JMS IQ Manager at any time after you have installed Java CAPS 6.

Exercising the JMS Management Service in Groovy

The Groovy script for executing the JMS Management Service is attached in `JMSManagmentServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\JMSManagementServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the BPEL Management Service

Before you can exercise the BPEL Management Service, you must ensure:

- That monitoring is enabled for your BPEL Service Engine
- That a service engine named sun-bpel-engine has been installed and deployed in your environment

---

Note – Change the script as you see fit to run in your environment.

Exercising the BPEL Management Service in Groovy

The Groovy script for executing the BPEL Management Service is attached in `BPELManagmentServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\BPELManagementServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the HTTP Administration Service

Before you can exercise the HTTP Administration Service, you must have the following installed and deployed in your environment:

- A binding component named sun-http-binding
- A Service Assembly with some HTTP consuming endpoint(s)

Note – Change the script as you see fit to run in your environment.

Exercising the HTTP Administration Service in Groovy

The Groovy script for executing the HTTP Administration Service is attached in `HTTPAdiministrationServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\HTTPAdministrationServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

Exercising the Notification Service

For Notification Service test, there are no prerequisites except that the AppServer is up and running. When any lifecycle changes happen in the JBI runtime, the Notification Service sends these notifications to all subscribed clients.

Note – Change the script as you see fit to run in your environment.

Exercising the Notification Service in Groovy

The Groovy script for executing the Notification Service is attached in `NotificationServiceTest.groovy`. Modify the script to suit your needs before executing it in your environment/installation.

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\NotificationServiceTest.groovy
```

If you are comfortable with the Swing-based groovyConsole, use it to load and execute the script file.

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

JRuby Integrated into NetBeans IDE

If you have the latest version of NetBeans IDE 6.0, you can create your own JRuby Project, and execute the JRuby samples (they have “rb” extensions) from within the NetBeans IDE.

Note – Java CAPS 6 delivered with NetBeans IDE 6.1 ML.

▼ To Create a JRuby Project

1. **Create a new JRuby project.**
2. **Copy the JRuby scripting files into the project.**
3. **Right-click and open the Project Properties tab and provide the appropriate jar files for the project to use.**
4. **Start the Sun Java System Application Server.**
5. **Open a script file in NetBeans, right-click on the source, and then click Run File to run the script.**