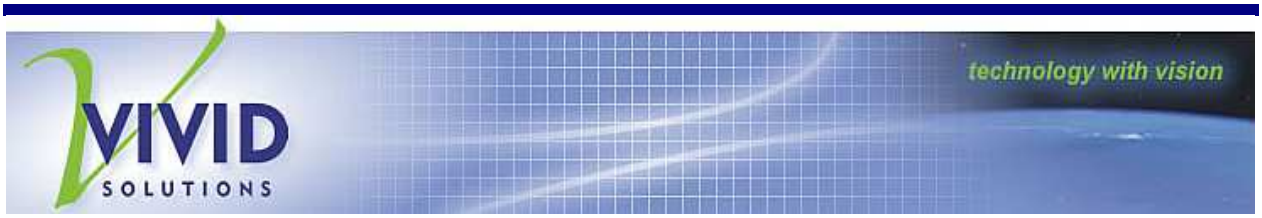

WEBADE TASK MANAGER EXTENSION

Developer's Guide

Client:	Ministry of Forests Information Management Group
Date:	December 10, 2007
Revision:	6

Vivid Solutions Inc.
Suite #1A, 2328 Government St.
Victoria, BC V8T 5G5
Phone: (250) 385-6040
Fax: (250) 385-6046
Website: www.vividsolutions.com

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	DESCRIPTION
1	January 31, 2005	Jason Ross	Original draft
2	March 2, 2005	Jason Ross	Updated to reflect version 1.0.0
3	April 21, 2005	Jason Ross	Updated to reflect version 1.1.0
4	September 21, 2005	Jason Ross	Updated sample start-time value to conform to XML schema spec.
5	January 27, 2006	Jason Ross	Updated to WebADE 4.
6	December 10, 2007	Jason Ross	Updated to include instructions for the optional configuration WebADE preferences.

Table of Contents

1. INTRODUCTION

The Task Manager Extension allows a WebADE application to create and manage tasks that need to be run autonomously on a fixed schedule. This document describes how initialize the Task Manager Extension and create and deploy custom tasks for use in your application.

The Task Manager extension currently relies on the JMX API for most of it's functionality. However, you will notice that the exposed API does not reference any JMX classes. This is to allow future portability. Also, as the JMX API's scope goes well outside of this extension, the Task Manager API simplifies the JMX API to simplify coding for the task developer

2. INITIALIZING THE TASK MANAGER EXTENSION

The Task Manager is a WebADE Extension, and, as such, is initialized by inserting WebADE preferences into the WebADE PREFERENCE table (See the WebADE User's Guide for information on configuring WebADE extensions). This will create the TaskManagerExtension and load it in your WebADE Application singleton, where you can retrieve the extension by calling:

```
application.getWebADEExtension("ca.bc.gov.webade.extension.taskmanager.TaskManagerExte
nsion");
```

2.1 DEFINING THE TASKS CONFIGURATION FILE LOCATION

The XML configuration file defining the tasks for an application, by default, should be placed in the web application at "WEB-INF/classes/tasks.xml".

As default, there are two things to note, the file is read-only, and the file must be included in the WAR file, making it troublesome to modify.

If you wish to include the task file separately, or you wish to enable task properties as "securable" (See below), you should change the task manager extension's WebADE preferences as desired. See Appendix B for the SQL required to set these preferences.

The "load-file-as-resource" preference indicates whether the file will be loaded as a resource or as a file location. This defaults to true, but should be set to false if you wish the file to be located somewhere other than WEB-INF/classes/tasks.xml

The "tasks-file-rewritable" defaults to false, but should be set to true, if your wish the task manager to obscure securable properties in the file.

The "tasks-file-location" is used when the "load-file-as-resource" is false. This will define the location of the task.xml file, usually outside of the web application's directory structure, in a local directory.

2.2 TASK MANAGER EXTENSION REQUIREMENTS

Before the WebADE loads the Task Manager Extension, the following steps must be completed:

1) Make sure the "webade-taskmanager-01_00_00.jar" is included in the web application's WEB-INF/lib directory, or at least is configured at the same class loader level as the WEB-INF/classes directory, as the classes in the "webade-taskmanager.jar" need to be able to load the "tasks.xml" file from this directory.
2) Create a properly configured "tasks.xml" file. See section 3 for instructions on configuring this file.
3) Make sure the Task Manager Extension is loaded by WebADE (See above).

3. CONFIGURING TASKS

In order for the Task Manager Extension to be able to load and configure your application's scheduled tasks, you must:

1) Make sure the TaskManagerExtension class in the "webade-taskmanager.jar" can locate the class files of your task and all classes that your task depends on.
2) Make sure your tasks are properly configured in the "tasks.xml" file.

3.1 SAMPLE "TASKS.XML" FILE

Below is a sample "task.xml" file:

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type=”minutes”>5</period>
            </include>
        </schedule>
    </task>
    <task name="TestTask2" class-name="test.tasks.TestPropertySetTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <property-set name="set2">
            <property name="prop1" value="val4"/>
            <property name="prop3" value="val3"/>
        </property-set>
        <schedule>
            <include>
                <number-of-occurrences>10</number-of-occurrences>
                <period>10000</period>
            </include>
        </schedule>
    </task>
    <task name="TestTask3" class-name="test.tasks.TestTask">
        <schedule>
            <include>
                <start-time>2005-01-25T14:03:00.00000-08:00</start-time>
                <number-of-occurrences>10</number-of- occurrences >
                <period type="hours">24</period>
                <fixed-rate>true</fixed-rate>
            </include>
        </schedule>
    </task>

</task-manager-settings>
```

3.2 CONFIGURING A TASK

There are three steps to configuring a task in the "tasks.xml" file:

1) Declare the task by adding a new task tag in the task-manager base tag.
2) Configure the task with a set of property-set and property tags (optional).
3) Define the schedule for the task.

3.2.1 WRITING A TASK TAG

The "task-manager" tag contains any number of "task" tags that define the tasks to be loaded by the Task Manager Extension. Each "task" tag has two mandatory attributes, "name" and "class-name". The name should be set to a unique name among all "task"tags. The "class-name" value should be the fully qualified Java class that implements ScheduledTask.

3.2.2 CONFIGURING THE TASK

If your task requires configuration properties, you can initialize them in two ways; define a set of "property" tags, or define properties in sets of "property-set" tags. **NOTE:** these options are mutually exclusive.

If your settings can be defined by name/value pairs, with each pair having a unique name, then simply define a separate "property" tag with name and value attributes set to your expected values.

Alternatively, you can divide task properties into separate property set groups. Wrap each set of property tags with "property-set" tags, assigning each "property-set" tag with a unique name.

3.2.2.1 SECURABLE PROPERTIES

Because tasks are configured in a plain text file, it is desirable to have sensitive settings, like passwords, to be obscured. This is possible, if the tasks.xml is loaded as a file (See above), by setting the optional property attribute "securable" to "true". See below:

```
<property name="database.password" value="password" securable="true"/>
```

3.2.3 SCHEDULING THE TASK

Each task will be scheduled by the Task Manager Extension, as specified in the "tasks.xml" file. To define the schedule for a task, you must add a schedule tag after any property or property-set tags defined for the task. Each "schedule" tag should contain an "include" tag. The "include" tag can contain the following set of tags:

- start-time
- end-time
- number-of-occurrences
- period
- fixed-rate

START-TIME TAG

The start-time tag defines the starting time that the task will receive notifications and being processing.

The start-time tag's value must be set using the following pattern:

YYYY-MM-DDThh:mm:ss.00000-hh:mm

As an example, "2005-01-25T14:03:00.00000-08:00" would be read as "January 25, 2005 2:03PM" at –8H off GMT, or Pacific Time. **NOTE:** Ensure that all times defined for your tasks have a GMT offset defined, or else you may get unexpected results.

NOTE: If a "start-time" tag is not defined, the current time is used.

END-TIME TAG

The end-time tag defines the time that the task will stop receiving notifications and halt processing. Only one of "end-time" or "number-of-occurrences" can be defined for each task schedule include definition.

The end-time tag's value must be set using the following pattern:

YYYY-MM-DDThh:mm:ss.00000-hh:mm

As an example, "2005-01-25T14:03:00.00000-08:00" would be read as "January 25, 2005 2:03PM" at –8H off GMT, or Pacific Time. **NOTE:** Ensure that all times defined for your tasks have a GMT offset defined, or else you may get unexpected results.

NOTE: If neither an "end-time" or "number-of-occurrences" tag is not defined, the task will continue to receive notifications indefinitely.

NUMBER-OF-OCCURRENCES TAG

The "number-of-occurrences" tag defines the total number of notifications that the task will receive before halting processing. Only one of "end-time" or "number-of-occurrences" can be defined for each task schedule include definition.

NOTE: If neither an "end-time" or "number-of-occurrences" tag is not defined, the task will continue to receive notifications indefinitely.

PERIOD TAG

The "period" tag defines the amount of time between notifications that are sent to your task. There is one optional attribute, "period-type", that allows you to define the granularity of the period value. By default, this is set to "milliseconds", but can be set to any of the following values: "milliseconds", "seconds", "minutes", "hours", or "days".

FIXED-RATE TAG

The "fixed-rate" tag defines a flag that indicates whether or not the Task Manager Extension will wait for your task to finish responding to a notification before it starts measuring the wait period between notifications. If not defined, by default this flag is set to false.

4. CREATING A TASK

To create a scheduled task, simply implement the ca.bc.gov.mof.taskmanager.Task interface.

For most Tasks, it is sufficient enough to extend the AbstractTask or AbstractBroadcastingTask in the ca.bc.gov.mof.taskmanager package, implementing the onNotification() method.

Each time a notification is raised, according to the schedule for your task as defined in the "tasks.xml" file, this method is called. This means that any and all processing for your task should be performed in here.

4.1 ONNOTIFICATION() METHOD

The onNotification() method of the AbstractTask class is called whenever a notification is raised that needs to be handled by the task. This is most commonly a scheduled notification, as defined by your task's schedule in the tasks.xml file. However, there are other possible notifications, such as notifications from other tasks. See the section on listening to other tasks for information about these types of notifications.

The onNotification() method has one parameter, a TaskNotification object. If your task only operates on a schedule, as defined in the tasks.xml file, then you will have little interest in the values held in this TaskNotification object. This object has much more value when listening to other tasks.

4.2 TASK NOTIFICATION BROADCASTER

If a task is to send custom notifications, it must implement the TaskNotificationBroadcaster interface. This interface allows the task to add listeners, define what notifications are being broadcast, and send notifications to all listeners. Again, we recommend simply extending the abstract class "AbstractBroadcastingTask", as this class implements all of the basic plumbing of this interface, leaving only a single method for you to implement that defines the metadata of your notifications. See listening to other tasks for more information on notifications.

4.3 GETTING PROPERTY SETTINGS

To retrieve the property settings configured in the tasks.xml file for your task, call the getPropertySet() method, passing the desired property set name (as configured in the property-set tag's name attribute), which returns a Properties object containing all properties contained by that set. You can also call getPropertySetNames() to return a list of all sets defined for this task.

4.4 USING THE WEBADE WITHIN YOUR TASK

If your task requires access to the WebADE Application singleton, implement the interface "ca.bc.gov.mof.webade.extension.taskmanager.ApplicationInitialization". When your task is

initialized, the application singleton will be passed via the setApplication() method defined by this interface.

4.5 GETTING A CONNECTION TO THE WEBADE DATABASE

If your task requires access directly to the WebADE database, implement the interface "ca.bc.gov.mof.webade.extension.taskmanager.ADEConnectionProxyInitialization". When your task is initialized, an ADEConnectionProxy instance will be passed via the setADEProxy() method defined by this interface.

ADEConnectionProxy has two methods, "getADEConnection()" and "releaseADEConnection(Connection)", that open and close connections to the WebADE database. **NOTE:** You should always use the releaseConnection() method, as opposed to closing the connection class yourself, by calling close() on the Connection instance.

5. LISTENING TO OTHER TASKS

Some applications have a need to create tasks that, either instead of or in addition to responding to a schedule, need to listen to notifications generated by other tasks. To configure a task to respond to another tasks notifications, you configure these tasks like in the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type=”minutes”>5</period>
            </include>
        </schedule>
    </task>
    <task name="TestTask2" class-name="test.tasks.TestTask">
        <notification-listener>
            <task>TestTask1</task>
        </notification-listener>
    </task>
</task-manager-settings>
```

In this example, "TestTask2" is listening to "TestTask1". When "TestTask1" sends out notifications, "TestTask2" will receive them and have it's onNotification() method called, with the notification object passed in as a TaskNotification instance.

The TaskNotification object contains various information about the raised notification, such as notification type, notification source, a message string, and possible custom user data.

5.1 TASK NOTIFICATION TYPES FOR ALL TASKS

There is one default notification type: Task.NOTIFICATION_TYPE_STATUS_CHANGE. Notifications of this type are generated by the AbstractTask and AbstractBroadcastingTask classes. It is suggested that if you implement your own Task class from scratch, that you also support these notification types. NOTIFICATION_TYPE_STATUS_CHANGE notifications are sent to all listeners when the state of the task changes. These states are defined in the TaskStatus class, and are as follows:

TASK STATUS	DESCRIPTION
CREATED	Code indicating the Task was created.
INITIALIZING	Code indicating the Task is being initialized.
STOPPED	Code indicating the Task is currently stopped, and not responding to scheduled events.
RUNNING	Code indicating the Task is currently running, and is waiting for scheduled events.
PROCESSING_TASK	Code indicating the Task is currently running, and is responding to a scheduled event.

SHUTTING_DOWN	Code indicating the Task is currently shutting down.
HALT_ON_ERROR	Code indicating the Task has encountered a fatal error, as has stopped running

5.2 CUSTOM NOTIFICATIONS

If a task implements the TaskNotificationBroadcaster interface, it can broadcast notifications to classes implementing the TaskNotificationListener interface. **NOTE:** Again, you are encouraged to extend the AbstractBroadcastingTask, instead of creating a completely new implementation, as much of the difficult work is done for you.

There are two steps to implementing a task that broadcasts notifications: Defining the supported notifications and broadcasting these notifications.

5.2.1 DEFINING A TASK NOTIFICATION

The TaskNotificationBroadcaster interface has one method, called getSupportedTaskNotifications(), that allows an external process to discover what notifications are broadcast by this task. This method returns an array of TaskNotificationMetadata objects, with each TaskNotificationMetadata instance defining a set of notification types. This set of notification types should be related to each other, and often a TaskNotificationMetadata instance defines only one type. This metadata instance also specifies the fully-qualified class name of the TaskNotification implementation and a text description of the purpose for this set of notifications.

NOTE: When your task broadcasts a notification of a type specified in a given TaskNotificationMetadata instance, this notification must be able to be cast to the class as defined in the same TaskNotificationMetadata instance.

The AbstractBroadcastingTask class has a slightly different method, called getTaskNotificationMetadata(). This is because the AbstractBroadcastingTask supports the Task.NOTIFICATION_TYPE_STATUS_CHANGE, and appends this type to the set of notification metadata objects your custom task supports.

5.2.2 SENDING A TASK NOTIFICATION

All that is needed to broadcast a notification task is to call the broadcastTaskNotification() method, passing in your initialized TaskNotification instance. This notification will be handed to all listeners, allowing them to respond to the notification.

6. CONFIGURING THE DEFAULT DOMAIN (OPTIONAL)

Oracle's application server, IAS, does not allow web applications to use just any domain for their JMX beans. Instead, IAS requires the domain be set to the same name as the application name component of the application URL (For example, an application with the URL "http://localhost/oss" should use the domain "oss"). Because of this issue, the WebADE Task Manager Extension cannot use the default "task-manager" name as the JMX domain that will host the application's tasks. Instead you will need to define the default domain within the tasks.xml file, as described in the section below.

6.1 CONFIGURING THE DEFAULT DOMAIN

To define the JMX default domain for task JMX beans, you must add a default-domain tag to your tasks.xml file, as in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <default-domain>oss</default-domain>
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type=”minutes”>5</period>
            </include>
        </schedule>
    </task>
</task-manager-settings>
```

6.1.1 THE DEFAULT-DOMAIN TAG

The <default-domain> tag has a single tag value. This value should be set to the application name component of the application URL (For example, an application with the URL "http://localhost/oss" should use the domain "oss").

7. CONFIGURING ADAPTORS (OPTIONAL)

Often it is required to remotely monitor and control tasks deployed in a web application. Sometimes this is to perform related administrative or application tasks in response to events that occur within these tasks. Another reason to remotely communicate with these task is to mange them, stopping, starting, and performing other operations on them as needed.

This section describes how to configure a web application with internal tasks to expose an adaptor that will allow external systems to communicate with the tasks. The next section gives an overview on how to create these remote connections and control and monitor these tasks.

7.1 CONFIGURING THE RMI ADAPTOR

The task scheduler's RMI adaptor allows other applications to connect to the application's tasks using RMI (Java's Remote Method Invocation API). To enable the RMI adaptor, you must add an RMI-configured adaptor tag to your tasks.xml file, as in the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <adaptor>
        <rmi port="1100" server="localhost"/>
    </adaptor>
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type=”minutes”>5</period>
            </include>
        </schedule>
    </task>
</task-manager-settings>
```

7.1.1 THE RMI TAG

The <rmi> tag inside the <adaptor> tag has three optional attributes.

ATTRIBUTE NAME	DEFAULT	DESCRIPTION
server	localhost	The name is the server hosting the application.
port	1099	The port on the server that is opened for the RMI adaptor to listen to remote communication requests.
url	jmx	The relative url of the adaptor. This attribute does not usually need to be set, as the default is sufficient.

7.2 ADDING AUTHENTICATION TO THE ADAPTOR.

Often, when your application is configured to use the RMI adaptor, you want to place some form of authentication, to ensure that no unauthorized access to the tasks is given.

There are 2 supported forms of authentication: username/password and signature-based authentication.

7.2.1 USERNAME/PASSWORD AUTHENTICATION

For simple deployment situations, it is often sufficient to configure several sets of usernames/passwords in the tasks.xml file, allowing access to the tasks if a request contains a matching pair of username/password. This is configured in the tasks.xml file as in the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <authenticator>
        <username-password-module>
            <username-password-pair username="guest" password="guest" />
            <username-password-pair username="admin" password="admin" />
        </username-password-module>
    </authenticator>
    <adaptor>
        <rmi port="1100" server="localhost"/>
    </adaptor>
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type="minutes">5</period>
            </include>
        </schedule>
    </task>
</task-manager-settings>
```

Each <username-password-pair> tag contains a username and password combination that can be passed in the request's credentials. If a request comes without any credentials, or without a matching pair of username/password, the request will be refused. See the later section on creating remote connections for more information on how to send these credentials.

7.2.2 TRUSTED APPLICATION SIGNATURE AUTHENTICATION

If another application needs to access the tasks in your application, there can be a certain level of trust between the two applications. As other applications usually perform some sort of authentication on their users, it's often redundant to re-poll the user for their credentials a second time to give them access to control your tasks.

If you trust another application, that any requests coming from that application are secure, then a second method of authentication is available, trusted application signature authentication. In this method, the requesting application signs each request, and your

task's adaptor validates the signature with a public key. If the signature is valid, the request is allowed through, otherwise it is refused.

To configure the trusted application signature authenticator, you will need to add an <authenticator> tag to your tasks.xml as in the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<task-manager-settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="task-manager.xsd">
    <authenticator>
        <signature-module>
            <signature-key key-source="ESF" public-
key="308201b83082012c06072a8648ce3804013082011f02818100fd7f53811d75122952df4a9c2eece4e
7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6cb9b556cd7813b801d3
46ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c6a8a6150f04fb83f6d3c51ec
3023554135a169132f675f3ae2b61d72aeff22203199dd14801c70215009760508f15230bccb292b982a2e
b840bf0581cf502818100f7e1a085d69b3ddecbbcab5c36b857b97994afbbfa3aea82f9574c0b3d0782675
159578ebad4594fe67107108180b449167123e84c281613b7cf09328cc8a6e13c167a8b547c8d28e0a3ae1
e2bb3a675916ea37f0bfa213562f1fb627a01243bcca4f1bea8519089a883dfe15ae59f06928b665e807b5
52564014c3bfecf492a0381850002818100addbb7547b058d283e5f6a69070507b2cb64882ba17d89be147
993abf74546ac894c2e55ca15c1fd2c9cfb2148270d8b6a87bca663ab6ab8e271f9a6e84cd04e6379efdde
d74ee12d2417ec04a3bb1efc170774c006cce0b78147ab00b2577fb0e648c7a781e2bf32d6603036b5bd7a
81b4227f64c624028a089a2ac54890e5e" />
        </signature-module>
    </authenticator>
    <adaptor>
        <rmi port="1100" server="localhost"/>
    </adaptor>
    <task name="TestTask1" class-name="test.tasks.TestPropertyTask">
        <property-set name="set1">
            <property name="prop1" value="val1"/>
            <property name="prop2" value="val2"/>
        </property-set>
        <schedule>
            <include>
                <period type=”minutes”>5</period>
            </include>
        </schedule>
    </task>
</task-manager-settings>
```

The <signature-module> tag can contain any number of <signature-key> tags. Each <signature-key> has the following attributes.

ATTRIBUTE NAME	DESCRIPTION
key-source	The application acronym of the trusted application.
public-key	The public key for this application, which will be used to validate requests originating from this application.

The public key is a very long string that is generated by the KeyPairFactory class in the ca.bc.gov.mof.taskmanager.authentication package of the task manager extension. See the section on generating keys later on in this document for more information on how to create these keys.

8. COMMUNICATING REMOTELY WITH APPLICATION TASKS

If an adaptor is specified for an application's tasks, then other applications can remotely communicate with this application's tasks. This section gives an overview of how to connect with a remote MBeanServer, and how to invoke methods on remote tasks. This section requires knowledge of JMX, as this release of the task manager extension does not provide a simplified method of connecting to tasks remotely.

8.1 CONNECTING TO THE JMX SERVER USING THE RMI ADAPTOR

The first step in connecting to the MBeanServer remotely is to create a properly formatted JMX Servie URL. The JMXUtils class in the ca.bc.gov.webade.management.jmx package is provides a useful helper method to create an instance of this object. Simply pass in the target server name, port, and relative url (We recommend using the ca.bc.gov.webade.management.jmx .MBeanServerFactory.DEFAULT_RELATIVE_URL value for most cases).

```
JMXUtils.createRMIJMXServiceURL(SERVER_NAME, SERVER_PORT,
MBeanServerFactory.DEFAULT_RELATIVE_URL);
```

8.2 SETTING CREDENTIALS

If the target server has authentication turned on, you will need to send credentials with your connection request. Credentials are passed into an "environment" Map object with the static constant value of javax.management.remote.JMXConnector.CREDENTIALS as the map key. These credentials are different for username/password authentications and trusted application signature authentication.

```
JMXServiceURL url = ...
Map environment = new HashMap();
Object credentials = ...
environment.put(JMXConnector.CREDENTIALS, credentials);
// Connect to the server
JMXConnector cntor = JMXConnectorFactory.connect(url, environment);
```

8.2.1 USERNAME/PASSWORD CREDENTIALS

The credentials object for username/password is a size-2 String array, with username as the first value, and password as the second value in the array.

8.2.2 TRUSTED APPLICATION SIGNATURE CREDENTIALS

The credentials object for trusted application signature authentication is an instance of TrustedApplicationCredentials. This credentials object is filled out like in the example below, using the application's generated private key string, the application's acronym, the requesting user id, and the encryption algorithm.

```
String privateKey = ...
TrustedApplicationCredentials credentials = new TrustedApplicationCredentials();
credentials.setApplication("ESF");
```

```
credentials.setUsername("idir\\myuser");
credentials.setEncryptionAlgorithm("DSA");
credentials.sign(privateKey);
```

8.3 CONNECTING TO THE MBEANSERVER

Putting this all together, once you have created the JMX Service URL and loaded the appropriate credentials, if needed, you just need to connect and retrieve an MBeanServerConnection instance, like in the example below.

```
JMXServiceURL url = ...
Map environment = new HashMap();
Object credentials = ...
environment.put(JMXConnector.CREDENTIALS, credentials);
// Connect to the server
JMXConnector cntor = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection connection = cntor.getMBeanServerConnection();
```

With this MBeanServerConnection object, you can perform all normal MBeanServer operations, like search for MBeans, invoke methods on registered MBeans, and register listeners to MBeans on the server. Please see Sun's JMX API documentation for more information.

9. GENERATING PUBLIC/PRIVATE KEY PAIRS

The KeyPairFactory class provides a mechanism that generates a pair of public and private keys, to be used for trusted application signature authentication. The private key should be kept by the application that wishes to communicate with tasks remotely. The public key should be given to any task environment that is to be called by the application. The KeyPairFactory has two key methods for applications that will remotely call tasks, generateKeyPair() and signMessage().

9.1 THE GENERATEKEYPAIR() METHOD

The generateKeyPair() method generates a set of private and public keys. The return value is a size 2 string array, with the first entry containing the private key, and the second, the public key. This method takes two parameters, a number of bits of encryption (Usually 512 or 1024 are acceptable values) and an encryption algorithm (such as "DSA").

9.2 THE SIGNMESSAGE() METHOD

The signMessage() method should be called by the target application when sending a request to a target task. This method returns the encrypted signature, which should be placed in the TrustedApplicationCredentials instance to be passed in the request as credentials for the request (see above).

This method takes three parameters, the encryption type, the private key (as generated above), and the message to sign. This message is the username of the user of the application that is invoking the request to the target task. This username string **must** also be set in the TrustedApplicationCredentials instance, so the task can use it to validate the signature.

10. FUTURE ENHANCEMENTS

Below is a list of future enhancements to the task scheduling extension.

- Add an API that greatly simplifies remote calls to tasks.
- Add functionality that allows authentication to occur against LDAP directories for a username and password
- Add support for tasks that can remotely listen to other tasks.
- Add WebADE authorization for each exposed method in all tasks. (Granular authorization)
- Add database-configuration of tasks, including dynamic configuration and task loading
- Add support for other task configuration schemes.
- Add more dynamic notification handlers, including filters of notifications, so a task only receives certain notifications from other tasks, instead of all notifications.
- Add support for RMI connections using Secure Socket Layer (SSL).
- Add support for Oracle scheduling

11. APPENDIX A: TROUBLESHOOTING

11.1 START TIME AND/OR END TIME ATTRIBUTE NOT RECOGNIZED

If you observe your start time and/or end time attributes for your task's schedule are being ignored, make sure the format is correct. A common error is to exclude the milliseconds in the value (e.g. 2005-01-25T14:03:00-08:00). The current version of Task Manager requires this format to be followed *exactly*, as in the example below.

```
<task name="TestTask3" class-name="test.tasks.TestTask">
    <schedule>
        <include>
            <start-time>2005-01-25T14:03:00.00000-08:00</start-time>
            <number-of-occurrences>10</number-of- occurrences >
            <period type="hours">24</period>
            <fixed-rate>true</fixed-rate>
        </include>
     </schedule>
</task>
```

12. APPENDIX B: PREFERENCES SQL SCRIPT

This is the full set of SQL inserts used to configure the Task Manager Extension to run within a WebADE application:

```
--------------------------------------------------------------------------------
-- Task Manager Preferences script.                                           --
-- See the WebADE Task Manager Extension Developer's Guide                    --
-- section 2. 'INITIALIZING THE TASK MANAGER EXTENSION'                       --
-- for a description of these database row entries.                           --
--------------------------------------------------------------------------------

set feedback off
prompt Loading PREFERENCE...

insert into PREFERENCE (PREFERENCE_ID, PREFERENCE_TYPE_CODE, PREFERENCE_SUB_TYPE,
APPLICATION_ACRONYM, PREFERENCE_SET_NAME, PREFERENCE_NAME, PREFERENCE_VALUE, CREATED_BY,
CREATED_DATE, UPDATED_BY, UPDATED_DATE, REVISION_COUNT, EUSER_ID)
values (preference_seq.NEXTVAL, 'WDE', 'app-config', '&&APPLICATION_ACRONYM', 'extensions',
'extension', 'task-manager', USER, SYSDATE, USER, SYSDATE, 1, null);

insert into PREFERENCE (PREFERENCE_ID, PREFERENCE_TYPE_CODE, PREFERENCE_SUB_TYPE,
APPLICATION_ACRONYM, PREFERENCE_SET_NAME, PREFERENCE_NAME, PREFERENCE_VALUE, CREATED_BY,
CREATED_DATE, UPDATED_BY, UPDATED_DATE, REVISION_COUNT, EUSER_ID)
values (preference_seq.NEXTVAL, 'EXT', 'task-manager', '&&APPLICATION_ACRONYM', null, 'extension-
class-name', 'ca.bc.gov.webade.extension.taskmanager.TaskManagerExtension', USER, SYSDATE, USER,
SYSDATE, 1, null);
insert into PREFERENCE (PREFERENCE_ID, PREFERENCE_TYPE_CODE, PREFERENCE_SUB_TYPE,
APPLICATION_ACRONYM, PREFERENCE_SET_NAME, PREFERENCE_NAME, PREFERENCE_VALUE, CREATED_BY,
CREATED_DATE, UPDATED_BY, UPDATED_DATE, REVISION_COUNT, EUSER_ID)
values (preference_seq.NEXTVAL, 'EXT', 'task-manager', '&&APPLICATION_ACRONYM', null, 'tasks-
file-rewritable', '&TASKS_FILE_REWRITABLE_FLAG', USER, SYSDATE, USER, SYSDATE, 1, null);
insert into PREFERENCE (PREFERENCE_ID, PREFERENCE_TYPE_CODE, PREFERENCE_SUB_TYPE,
APPLICATION_ACRONYM, PREFERENCE_SET_NAME, PREFERENCE_NAME, PREFERENCE_VALUE, CREATED_BY,
CREATED_DATE, UPDATED_BY, UPDATED_DATE, REVISION_COUNT, EUSER_ID)
values (preference_seq.NEXTVAL, 'EXT', 'task-manager', '&&APPLICATION_ACRONYM', null, 'load-file-
as-resource', '&LOAD_FILE_AS_RESOURCE_FLAG', USER, SYSDATE, USER, SYSDATE, 1, null);
insert into PREFERENCE (PREFERENCE_ID, PREFERENCE_TYPE_CODE, PREFERENCE_SUB_TYPE,
APPLICATION_ACRONYM, PREFERENCE_SET_NAME, PREFERENCE_NAME, PREFERENCE_VALUE, CREATED_BY,
CREATED_DATE, UPDATED_BY, UPDATED_DATE, REVISION_COUNT, EUSER_ID)
values (preference_seq.NEXTVAL, 'EXT', 'task-manager', '&&APPLICATION_ACRONYM', null, 'tasks-
file-location', '&TASKS_FILE_LOCATION', USER, SYSDATE, USER, SYSDATE, 1, null);

prompt 5 records loaded. Not Committed.
set feedback on
prompt Done.
```

The last three preference row entries are optional (the preferences for 'tasks-file-rewritable', 'load-file-as-resource', and 'tasks-file-location'), and so do not need to be inserted into the WebADE PREFERENCE table, unless required by your application (remove them from the script before running it if you do not need them). Instructions for configuring these optional preferences can be found in the section below.

12.1 PREFERENCES CONFIGURATION PARAMETERS

When running the preferences script listed above, you will be prompted for the following parameters:

PARAMETER	DESCRIPTION
APPLICATION_ACRONYM	The acronym for the target WebADE Application.

TASKS_FILE_LOCATION	Set to the path and file name of the tasks.xml file used to load the application's tasks and their configurations into memory by the Task Manager Extension. By default, the Task Manager Extension looks for the tasks.xml file at the root of the WEB-INF/classes directory of your application's WAR file. By setting this parameter, you can place the file anywhere in the WEB-INF/classes directory tree. If the LOAD_FILE_AS_RESOURCE_FLAG is set to 'false', you can specify any absolute path on the local hard drive as the location for this file, removing the need to include it in your application EAR.
LOAD_FILE_AS_RESOURCE_FLAG	Valid values are 'true' and 'false'. Flag parameter that indicates whether the tasks.xml file is loaded into memory as a resource or as a file. This allows the task manager to load the tasks.xml file from anywhere on the local drive (value set to false), or only from the classpath (value set to true). This preference must be set to true if the TASKS_FILE_REWRITABLE_FLAG is set to 'true'.
TASKS_FILE_REWRITABLE_FLAG	Valid values are 'true' and 'false'. Flag parameter that indicates whether the tasks.xml file is rewritable. This allows the task manager to save the tasks.xml file back to disk, encrypting any task property tag values with the securable attribute set to 'true'. This option is ignored if the LOAD_FILE_AS_RESOURCE_FLAG is set to 'true'.