Moab Web Services 7.1.1 Reference Guide

Table of Contents

1 Introduction

1.1 Moab® Web Services Overview

Moab Web Services (MWS) is a component of Adaptive Computing Suites that enables programmatic interaction with Moab Workload Manager via a RESTful interface. MWS allows you to create and interact with Moab objects and properties such as jobs, nodes, virtual machines, and reservations. MWS is the preferred method for those wishing to create custom user interfaces for Moab and is the primary method by which Moab Viewpoint communicates with Moab.

MWS communicates with the Moab Workload Manager (MWM) server using the same wire protocol as the Moab command-line interface. By publishing a standard interface into Moab's intelligence, MWS significantly reduces the amount of work required to integrate MWM into your solution.

This documentation is intended for developers performing such integrations. If you are a Moab administrator, and for conceptual information about MWM, see the Moab Administrator's Guide.

1.2 Installation Guide

These instructions describe how to deploy the Moab® Web Services (MWS) to a Tomcat server.

1.2.1 Requirements

Hardware Requirements

- Dual core processor
- At least 4 GB of RAM

Software Requirements

- Moab® Workload Manager 7.1
- Oracle® Java® 6 Runtime Environment
- Apache Tomcat™ 6
- MongoDB® 2.0.1 or later

Oracle Java 6 Runtime Environment is the **only** supported Java environment.

All other versions of Java, including Oracle Java 7, OpenJDK/IcedTea, GNU Compiler for Java, and so on, **cannot** run Moab Web Services.

1.2.2 Quickstart Guide

You must deploy Moab Web Services (MWS) on the same server as Moab Workload Manager (MWM).

- Make sure that you have first installed and configured Moab Workload Manager as desired.
- Generate a secret key to be used for communication between MWM and MWS.

Before you create the key file, make sure Moab Workload Manager is not running. Otherwise, all Moab client commands will stop working. Example: `service moab stop`

- Here is a script you can use to generate the key:

```
dd if=/dev/urandom count=18 bs=1 2>/dev/null | base64 > /opt/moab/etc/.moab.key
chown root /opt/moab/etc/.moab.key
chmod 400 /opt/moab/etc/.moab.key
ln -f /opt/moab/etc/.moab.key /opt/moab/.moab.key
```

- Start MWM. Example: `service moab start`
- Install MongoDB version 2.0.1 or later on the MWM server using packages (recommended) or manual installation.
  - Packages (recommended) are available from the Mongo-provided repositories for CentOS and Fedora or Ubuntu and Debian.
  - After installing the packages, start MongoDB and arrange for it to start automatically during server startup.
    - Here is an example for CentOS and Fedora:
      - `/sbin/chkconfig mongod on`
      - `/etc/init.d/mongod start`
    - For Ubuntu and Debian, see the Configuration section of this page for examples.
  - For manual installation (not recommended), follow these instructions. Be sure to start the Mongo server (`mongod`) after installation and arrange for it to start automatically during server startup (by writing and registering an `/etc/init.d` script, for example).
- Download, install, and enable the 64-bit version of the Oracle Java SE 6 JRE.

Oracle Java 6 Runtime Environment is the **only** supported Java environment.

All other versions of Java, including Oracle Java 7, OpenJDK/IcedTea, GNU Compiler for Java, and so on, **cannot** run Moab Web Services.

- Here is an example for a CentOS system:

```
sh jre-6u33-linux-x64-rpm.bin
rm -f /usr/bin/java
ln -s /etc/alternatives/java /usr/bin/java
/usr/sbin/alternatives --install /usr/bin/java java /usr/java/jre1.6.0_33/bin/java 500
/usr/sbin/alternatives --set java /usr/java/jre1.6.0_33/bin/java
```

The `alternatives` command is called `update-alternatives` on some Linux distributions.

- You can verify the Java installation by running the following command:

```
java -version
```

- The output should look similar to this:

```
java version "1.6.0_33"
Java(TM) SE Runtime Environment (build 1.6.0_33-b03)
Java HotSpot(TM) 64-Bit Server VM (build 20.8-b03, mixed mode)
```

- Create the MWS home directory and its subdirectories `etc`, `hooks`, `plugins`, and `log`.

> ⚠ The default location for the MWS home directory is `/opt/mws`. These instructions assume the default location.

  - Give the Tomcat user read access to these directories and write access to the `plugins` and `log` directories.
  - Here is a sample script for these steps:

```
mkdir -p /opt/mws/etc /opt/mws/hooks /opt/mws/plugins /opt/mws/log
chown -R tomcat /opt/mws # Depending on your OS, the Tomcat username might be tomcat6.
chmod -R 555 /opt/mws
chmod u+w /opt/mws/plugins /opt/mws/log
```

- Extract the contents of the MWS tarball into a temporary directory. Example:

```
mkdir /tmp/mws-install
cd /tmp/mws-install
tar xvzf $HOME/Downloads/mws-<VERSION>.tar.gz
cd /tmp/mws-install/mws-<VERSION>
```

- Set up the MWS configuration file.
  - In the extracted MWS directory are two sample configuration files: `mws-config-cloud.groovy` and `mws-config-hpc.groovy`.
    - `mws-config-cloud.groovy` provides sample configuration for the Moab Cloud Suite.
    - `mws-config-hpc.groovy` provides sample configuration for the Moab HPC Suites.
  - Choose the correct file for your suite, rename it to `mws-config.groovy`, and copy it to `/opt/mws/etc`.
  - Give the Tomcat user read access to `/opt/mws/etc/mws-config.groovy`.
  - In the `/opt/mws/etc/mws-config.groovy` file, change these settings:
    - `moab.secretKey`: needs to match the MWM secret key you generated earlier (contained in `/opt/moab/etc/.moab.key`)
    - `auth.defaultUser.username`: any value you like, or leave as is
    - `auth.defaultUser.password`: any value you like, but choose a good password

```
vi /opt/mws/etc/mws-config.groovy

…
moab.secretKey = "<ENTER-KEY-HERE>"
moab.server = "localhost"
moab.port = 42559

// Change these to be whatever you like.
auth.defaultUser.username = "admin"
auth.defaultUser.password = "adminpw"
```

⚠ If you do not change `auth.defaultUser.password`, then your MWS is not secure, since anyone reading these instructions can log into your MWS. Here are some tips for choosing a good password.

- Set the following parameters in your Tomcat CATALINA_OPTS:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m -Xmx3g -XX:MaxPermSize=384m"
```

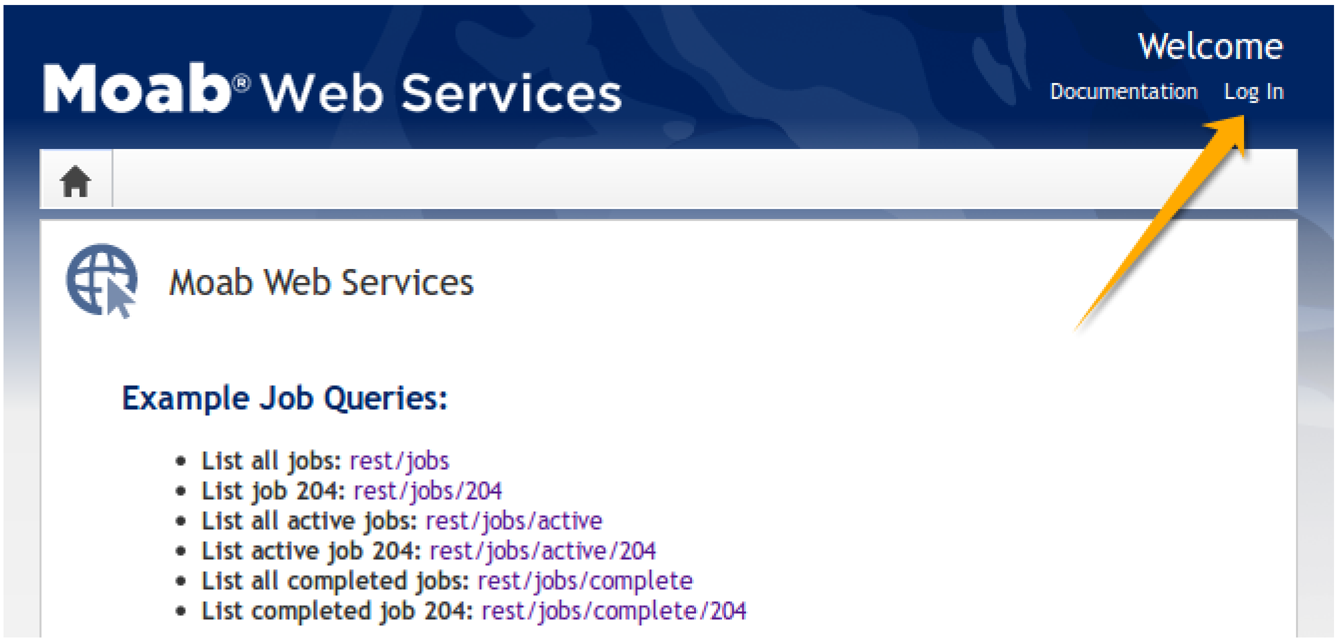
⚠ Where you choose to store `CATALINA_OPTS` depends on various factors, including operating system and sysadmin preference. Here are some suggestions:

- CentOS™ 5 and 6: `/etc/sysconfig/tomcat6`
- Red Hat® Enterprise Linux 5 and 6: `/etc/sysconfig/tomcat6`
- SUSE® Linux Enterprise Server 11: `/etc/tomcat6/tomcat6.conf`
- Ubuntu® 10.04: `/etc/default/tomcat6`

- Start Tomcat and deploy `mws.war`. Example:

```
chkconfig tomcat6 on
service tomcat6 start
cp /tmp/mws-install/mws-<VERSION>/mws.war /var/lib/tomcat6/webapps
```

- Visit http://localhost:8080/mws/ in a web browser to verify that MWS is running. You will see some sample queries and a few other actions.
- Log into MWS to verify that the MWS credentials are working. (The credentials are the values of `auth.defaultUser.username` and `auth.defaultUser.password` that you set above.)



Example Job Queries:

- List all jobs: rest/jobs
- List job 204: rest/jobs/204
- List all active jobs: rest/jobs/active
- List active job 204: rest/jobs/active/204
- List all completed jobs: rest/jobs/complete
- List completed job 204: rest/jobs/complete/204

> ⚠ If you encounter problems, or if MWS does not seem to be running, see the steps below in the Troubleshooting section.

1.2.3 Troubleshooting Installation

If something goes wrong with MWS, look in the following files:

1. The MWS log file. By default this is `/opt/mws/log/mws.log`.
2. The Tomcat `catalina.out` file, usually in `/var/log/tomcat6` or `$CATALINA_HOME/logs`.

> ⚠ If you remove the `log4j` configuration from `mws-config.groovy`, MWS will write its log files to `java.io.tmpdir`. For Tomcat, `java.io.tmpdir` is generally set to `$CATALINA_BASE/temp` or `CATALINA_TMPDIR`.

Here is a list of some errors and their fixes:

MongoDB Errors

If the application fails to start and gives error messages such as these:

```
Error creating bean with name 'mongoDatastore'
can't say something; nested exception is com.mongodb.MongoException
```

MongoDB is most likely not running, or the host and port are mis-configured. Start MongoDB or reconfigure MWS and restart MWS.

java.lang.OutOfMemoryError: Java heap space

Increase the size of the heap using JVM options `-Xms` and `-Xmx`. Here are the suggested values from the Quickstart Guide:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m -Xmx3g -XX:MaxPermSize=384m"
```

- `-Xms`: Set initial Java heap size.
- `-Xmx`: Set maximum Java heap size.

java.lang.OutOfMemoryError: PermGen space

Increase the size of the permanent generation using JVM option `-XX:MaxPermSize`. Here are the suggested values from the Quickstart Guide:

```
CATALINA_OPTS="-DMWS_HOME=/opt/mws -Xms256m -Xmx3g -XX:MaxPermSize=384m"
```

SEVERE: Context [/mws] startup failed due to previous errors

If `catalina.out` contains this error, look in `/opt/mws/log/mws.log` and `/opt/mws/log/stacktrace.log` for more details on the error.

Moab Reached Maximum Number of Concurrent Client Connections

When this error message is encountered, simply add a new line to the `moab.cfg` file:

```
CLIENTMAXCONNECTIONS 256
```

This will change the Moab configuration when Moab is restarted. Run the following command to immediately use the new setting:

```
changeparam CLIENTMAXCONNECTIONS 256
```

⚠ The number `256` above may be substituted for the desired maximum number of MWM client connections.

1.3 Configuration

This section describes the location of the Moab Web Services configuration files. It also shows some examples of how to configure logging.

⚠ To see a full reference to all configuration and logging parameters available in MWS, see the Configuration page under Moab Web Services in the Quick Reference menu.

Home Directory

The MWS home directory contains configuration files, log files, and files that serve features of MWS such as hooks and plugins. You should set the location of the MWS home directory using the `MWS_HOME` property as shown in the Quickstart Guide. If you do not set `MWS_HOME` as a Java property or as an environment variable, then MWS will use `/opt/mws` as the default `MWS_HOME`.

Configuration Files

The primary configuration file is `MWS_HOME/etc/mws-config.groovy`. If this file is missing or contains errors, MWS will not start. If `MWS_HOME/etc/log4j.properties` exists, MWS will load it as well.

Logging Configuration Using `mws-config.groovy`

Shown below is an example that logs all error messages and fatal messages to `/opt/mws/log/mws.log`. It also logs all stack traces to `/opt/mws/log/stacktrace.log`. Note that this example is not configured to log events.

Minimal Logging Configuration

```
log4j = {
    appenders {
        rollingFile name: 'stacktrace',
            file: '/opt/mws/log/stacktrace.log',
            maxFileSize: '1GB'
        rollingFile name: 'rootLog',
            file: '/opt/mws/log/mws.log',
            threshold: org.apache.log4j.Level.ERROR,
            maxFileSize: '1GB'
    }
    root {
        debug 'rootLog'
    }
}
```

Alternatively, you may configure a console appender instead of a rolling file as shown below.

Console Logging Configuration

```
log4j = {
    appenders {
        rollingFile name: 'stacktrace',
            file: '/opt/mws/log/stacktrace.log',
            maxFileSize: '1GB'
        console name: 'consoleLog',
            threshold: org.apache.log4j.Level.ERROR
    }
    root {
        debug 'consoleLog'
    }
}
```

⚠

- You may configure logging using either `MWS_HOME/etc/mws-config.groovy` or `MWS_HOME/etc/log4j.properties`.
- If you do not define any `log4j` configuration, MWS will write its log files to `java.io.tmpdir`. For Tomcat, `java.io.tmpdir` is generally set to `$CATALINA_BASE/temp` or `CATALINA_TMPDIR`.

1.4 Security

When running MWS in production environments, security is a major concern. This section focuses on securing the three kinds of connections with MWS:

1. The connection between MWS and Moab Workload Manager (MWM)
2. The connection between MWS and MongoDB
3. The connections between clients and MWS

Connection with MWM

MWS communicates with MWM via the Moab Wire Protocol, which uses a direct connection between the two applications. The communication over this connection uses a shared secret key, which is discussed in the Quickstart Guide. However, the communication is not encrypted and is therefore susceptible to eavesdropping and replay attacks. For this reason, MWS is supported only when running on the same machine as MWM. This assures that any connections between the two applications occur internally on the server and are not exposed to external users.

Connection with MongoDB

By default, the connection between MWS and MongoDB is not authenticated. To enable authentication between them, see the instructions below.

- **MWS Configuration**: see the Configuration reference guide for information on the `grails.mongo` properties to set in `mws-config.groovy`.
- **MongoDB Configuration**: see the MongoDB Security and Authentication guide. Generally, the following steps are required:
  - Add an administrative user to MongoDB in the `admin` database.
  - Start MongoDB with authentication activated (using the `--auth` command-line option for example).
  - Log in as the administrative user to the `admin` database.
  - Add a user for MWS to use with full read and write access to the database specified in the configuration file (`mws` by default).
  - Change the proper configuration file properties with the created username and password.
  - Restart MWS by restarting the servlet container (Tomcat).

If authentication is activated on MongoDB, but the user was not properly created or configured with MWS, MWS will not start. See the log file(s) for additional information in this case.

Client Connections to MWS

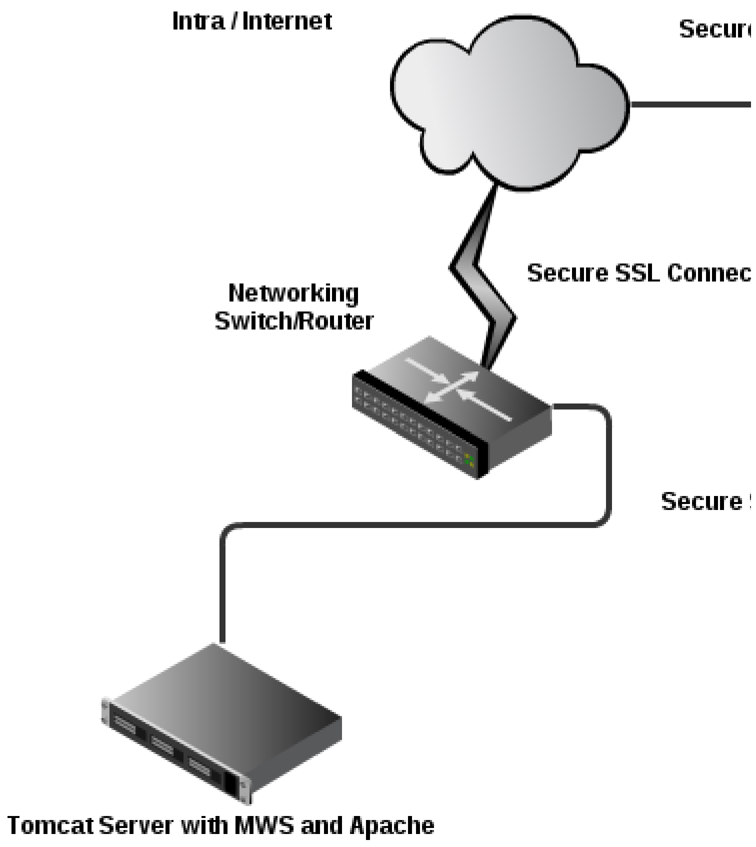
All connections to MWS, except those requesting the documentation or the main page, must be authenticated properly. MWS uses a single-trusted-user authentication model, meaning a single user exists that has access to all aspects of MWS. The username and password for this user are configured with the `auth.defaultUser` properties in the configuration file. See the Configuration reference guide for more information.

When using the MWS user interface in a browser, the user will be prompted for username and password. For information on how to authenticate requests when not using a browser, see the Authentication section in the user guide.

> ⚠ The username and password in the Basic Authentication header are encoded but not encrypted. Therefore, it is **strongly** recommended that MWS be run behind a proxy (like Apache) with SSL enabled. The instructions below provide an example of how to do this.

Encrypting Client Connections using Apache and SSL

This section shows how to encrypt client connections to MWS using Apache and SSL. These instructions have been tested on CentOS™ 6.2 with the "Web Server" software set installed. The same ideas are applicable to other operating systems, but the details might be different. As shown in the diagram below, these instructions assume that Tomcat and Apache are running on the same server.



- **Create a self-signed certificate.** See http://www.openssl.org/docs/HOWTO/certificates.txt for more details if desired.

  > Instead of creating a self-signed certificate, you can buy a certificate from a certificate vendor. If you do, then the vendor will provide instructions on how to configure Apache with your certificate.

  - Run these commands:

```
cd /etc/pki/tls/certs
cp -p make-dummy-cert make-dummy-cert.bak
cp -p localhost.crt localhost.crt.bak
```

  - Edit `make-dummy-cert` and replace the `answers()` function with code similar to this:

```
answers() {
        echo US
        echo Utah
        echo Provo
        echo Adaptive Computing Enterprises, Inc.
        echo Engineering
        echo test1.adaptivecomputing.com
        echo
}
```

  - Run this command:

```
./make-dummy-cert localhost.crt
```

- **Configure Apache to use the new certificate and to redirect MWS requests to Tomcat. To do so, edit `/etc/httpd/conf.d/ssl.conf`.**
  - Comment out this line:

```
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
```

  - Add these lines near the end, just above `</VirtualHost>`:

```
ProxyPass /mws http://127.0.0.1:8080/mws retry=5
ProxyPassReverse /mws http://127.0.0.1:8080/mws
```

- **Configure Apache to use SSL for all MWS requests.**
  - Add these lines to the end of `/etc/httpd/conf/httpd.conf`:

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule (/mws.*) https://%{HTTP_HOST}%{REQUEST_URI}
```

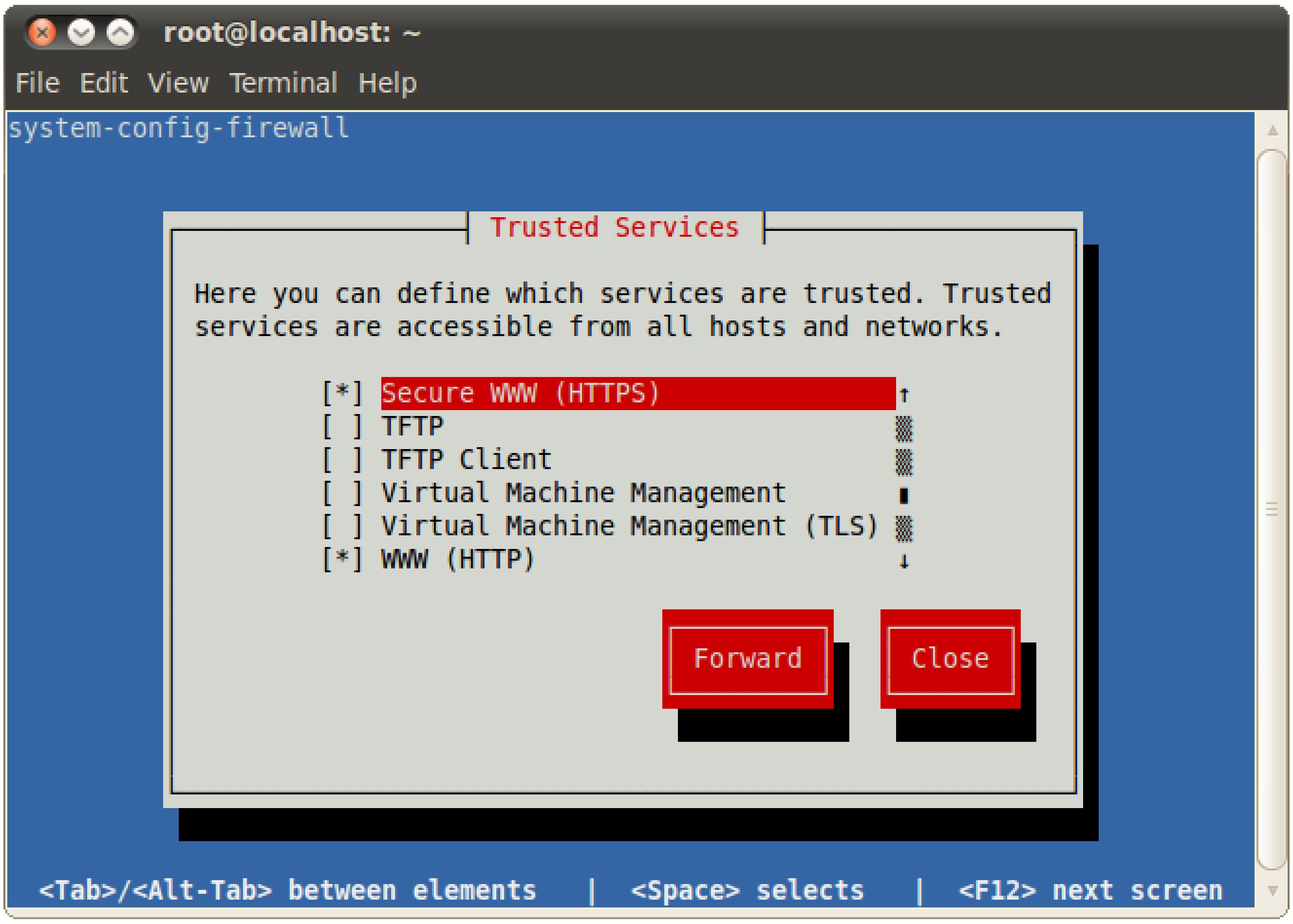
- **Give Apache permission to connect to Tomcat.**

```
setsebool -P httpd_can_network_connect 1
```

- **Turn on Apache.**

```
chkconfig httpd on
service httpd start
```

- **Using `system-config-firewall-tui`, enable "Secure WWW (HTTPS)" and "WWW (HTTP)" as trusted services.**



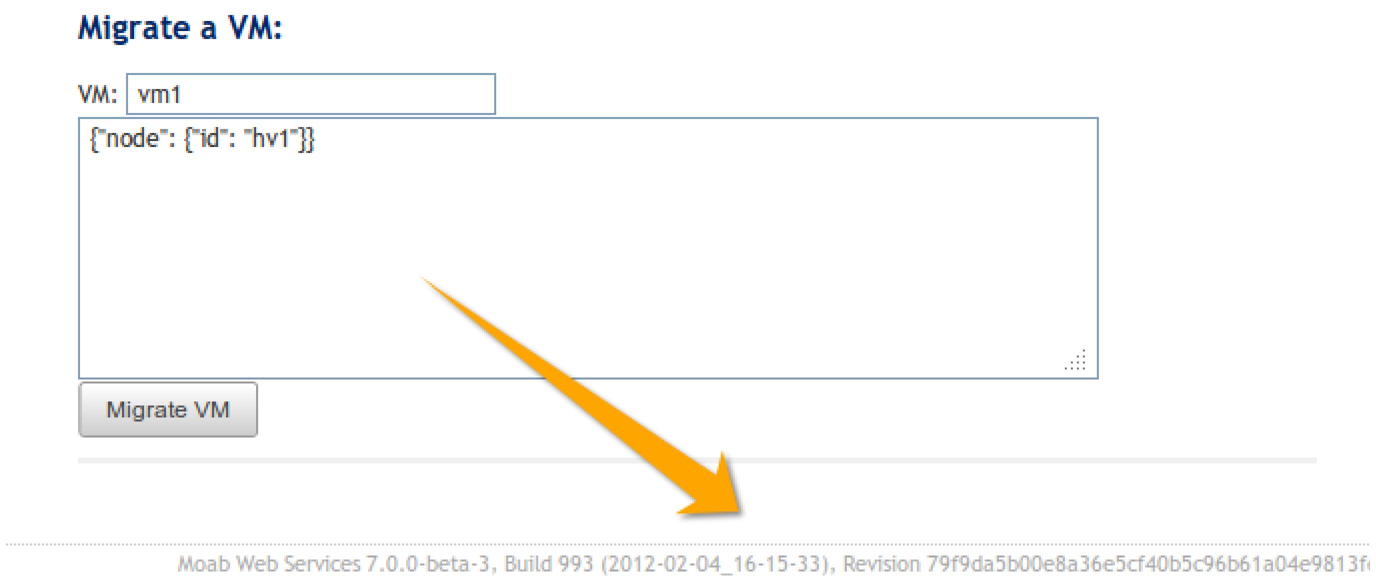
```
system-config-firewall

                    Trusted Services

 Here you can define which services are trusted. Trusted
 services are accessible from all hosts and networks.

          [*] Secure WWW (HTTPS)
          [ ] TFTP
          [ ] TFTP Client
          [ ] Virtual Machine Management
          [ ] Virtual Machine Management (TLS)
          [*] WWW (HTTP)

                              Forward     Close

 <Tab>/<Alt-Tab> between elements  |  <Space> selects  |  <F12> next screen
```

1.5 Version and Build Information

To get detailed version information about MWS, use one of the following three methods:

Browser

Using a browser, visit the MWS home page (for example, http://localhost:8080/mws/). At the bottom of the page is the MWS version information. See the screenshot below:



```
vm1
{"node": {"id": "hv1"}}
```

Moab Web Services 7.0.0-beta-3, Build 993 (2012-02-04_16-15-33), Revision 79f9da5b00e8a36e5cf40b5c96b61a04e9813f

REST Request

Using a REST client or other HTTP client software, send a GET request to the `rest/diag/about` resource. Here is an example:

```
curl -u username:password http://localhost:8080/mws/rest/diag/about
```

This resource is also described under Diagnostics.

MANIFEST.MF File

If MWS fails to start, version and build information can be found in the `META-INF/MANIFEST.MF` file inside the MWS WAR file. The version properties begin with `Implementation`. Below is an excerpt of a `MANIFEST.MF` file:

```
Implementation-Build: 26
Implementation-Build-Date: 2012-06-19_14-18-59
Implementation-Revision: 376079a5e5f552f2fe25e6070fd2e84c646a98fd

Name: Grails Application
Implementation-Title: mws
Implementation-Version: 7.1.0-rc2
Grails-Version: 2.0.3
```

2 Access Control

This section describes how to manage access control in Moab Web Services.

2.1 Application Accounts

Applications are the consumers of MWS. They include Moab Viewpoint and other applications that need the resources provided by MWS. An application account consists of four editable fields and resource specific access control settings:

Field	Required	Default Value	Value Type	Maximum Length	Description
Application Name	Yes	-	String	32	The name of the application. Must start with a letter and may contain letters, digits, underscores, periods, hyphens, apostrophes, and spaces.
Username	Yes	-	String	32	Used for authentication. Must start with a letter and may contain letters, digits, underscores, periods, and hyphens.
Description	No	-	String	1000	The description of the application.
Enabled	-	true	Boolean	-	Controls whether the application is allowed to access MWS.
Access Control Settings	Yes	All Permissions	-	-	The permissions granted to the application. This is controlled by selecting specific check boxes in a grid.

An application account also contains an auto-generated password that is visible only when creating the account or when resetting its password. Whenever an application sends a REST request to MWS, it needs to pass its credentials (username and password) in a Basic Authentication header. See the Authentication section for more information.

The **Application Name** is a human-friendly way to identify an application account, but MWS does not use it during authentication (or at any other time, for that matter).

The **Enabled** field is set to true automatically when an application account is created. To change the value of this field, see Modifying an Application Account.

Here is an example of how you might set the fields when creating an application account:

- **Application Name**: Moab Viewpoint
- **Username**: viewpoint
- **Description**: This application account grants access to Moab Viewpoint for Moab Cloud Suite.

The permissions granted to an application account may be customized while creating or modifying the account. See Creating an Application Account and Modifying an Application Account.

2.1.1 Managing Application Accounts

Application accounts are used to grant access to MWS. Every application with an application account must be granted at least one access control permission to a resource in MWS. To manage application accounts, start with Listing Application Accounts.

2.1.2 Listing Application Accounts

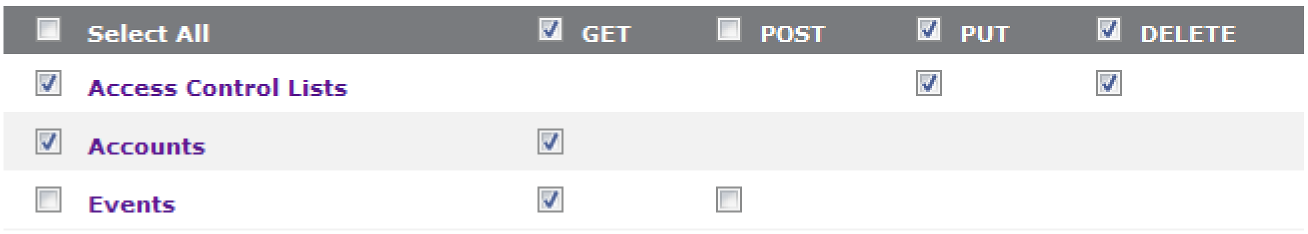
To list all applications accounts, browse to the MWS home page (`https://servername/mws` for example). Log in as the admin user, then click **Admin** and then **Application Accounts**.

Each column (except Password) can be sorted in ascending or descending order by clicking on the column heading.

2.1.3 Creating an Application Account

To create an application account, go to the **Application List** page and click **Add Application**. The **Application Name** and **Username** are required fields. See Application Accounts for more information on the fields.

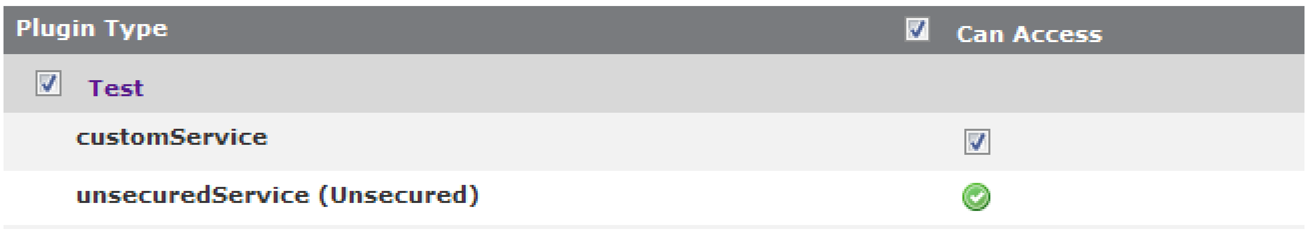
Access to specific resources and plugin custom web services is granted or revoked by checking or unchecking the check boxes in the respective resources or plugin web services access control sections. For each resource, access may be granted to a resource for each method supported by MWS, including GET, POST, PUT, and DELETE. See the figure below for an example.



☐ Select All	☑ GET	☐ POST	☑ PUT	☑ DELETE
☑ Access Control Lists			☑	☑
☑ Accounts	☑			
☐ Events	☑	☐		

In this example, the application has access to all available methods for the Access Control Lists and Accounts resources as well as to retrieve the Events resource through the GET method, but is denied the permission to create new events through the POST method.

Access may also be granted to each plugin type's custom web service(s). When new plugin types or plugin web services are added to MWS, applications must be updated with the new access control settings. See below for an example.



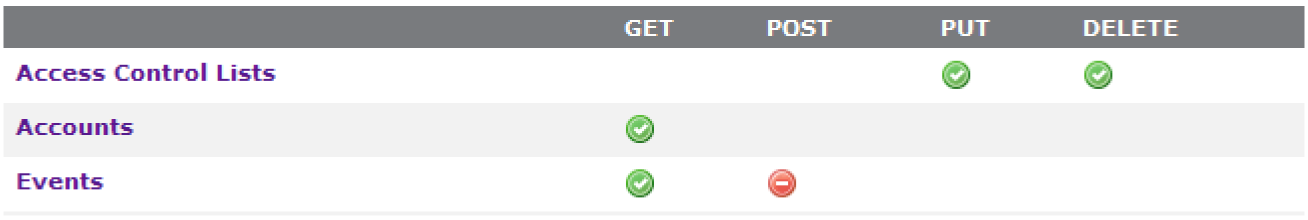
Plugin Type	☑ Can Access
☑ Test	
customService	☑
unsecuredService (Unsecured)	✓

In this example, the application has access to all the custom web services defined for the "Test" plugin type. Note that though Unsecured Web Services are listed, access to them cannot be denied (see Exposing Web Services for more information).

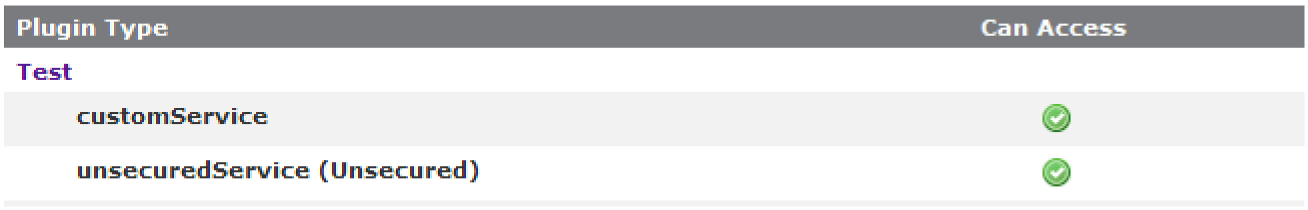
2.1.4 Displaying an Application Account

To show information about an application account, go to the **Application List** page and click the desired application name.

In addition to displaying the values for fields, grids are also displayed which represent the application's access control permissions defined for resources and plugin custom web services. Examples of the resources and the plugin web services access control displays are shown below.



	GET	POST	PUT	DELETE
Access Control Lists			✓	✓
Accounts	✓			
Events	✓	⊖		



Plugin Type	Can Access
Test	
customService	✓
unsecuredService (Unsecured)	✓

2.1.5 Modifying an Application Account

To modify an application account, go to the **Application List** page, click the desired application name, and then click **Edit**. See Creating an Application Account for more information on available fields and access control settings.

2.1.6 Resetting an Application Password

To reset an application password, go to the **Application List** page and click the **Reset** link for the desired application. Alternatively, go to the **Display Application** page for the desired application and click the **Reset** link.

2.1.7 Deleting an Application Account

To delete an application account, go to the **Application List** page, click the desired application name, and then click **Delete**. A confirmation message is shown. If the **OK** button is clicked, the application account is deleted from the system and cannot be recovered.

3 API Documentation

Introduction

The Moab® Web Services (MWS) provide a set of RESTful resources that can be used to create, read, update, and delete various objects in the Moab® Workload Manager.

3.1 RESTful Web Services

In order to understand how to use Moab Web Services, it is first necessary to give a brief introduction to REST. REST (Representational State Transfer) is a set of guidelines which utilizes the full HTTP (Hypertext Transfer Protocol) specification along with endpoint URLs that describe **resources**. The HTTP methods used in REST are comprised of the following:

Method	Description
GET	Query for a list or a single resource.
POST	Creating a resource.
PUT	Modifying a resource.
DELETE	Deleting a resource.

In comparison to other architectures of web services which use a single HTTP method and service endpoint to perform multiple types of operations (such as a POST operation to a URL), REST utilizes all of the available HTTP methods and URLs that directly correlate to resources. For example, RESTful web services for books in a library may expose many URL endpoints and the HTTP methods available for each such as GET, POST, PUT, and DELETE. The list below gives the methods, URLs, and descriptions for a sample set of services. The number `1` represents a unique identifier for books in each case.

Method	URL	Description
GET	/books	Retrieves a list of all books in the library.
POST	/books	Creates a new book.
GET	/books/1	Retrieves a single book.
PUT	/books/1	Modifies a single book.
DELETE	/books/1	Deletes a single book.

⚠ Note that in the cases of the POST and PUT operations, additional information may be needed to describe the resource to be created or the fields that should be modified.

Moab Web Services provides RESTful web services for many resources. The methods and URLs available are documented in the Resources section.

3.2 Data Format

JSON (JavaScript Object Notation) is the data format used for all communication with MWS. This format makes use of two main structures: collections of key/value pairs called *objects* and ordered lists of values called *arrays* . Objects are defined by using curly braces (`{}`), and arrays are defined by using square brackets (`[]`). A JSON object or array may contain several different types of values including numbers, booleans (true/false), strings, objects, arrays, or the keyword 'null' representing no value. For example, a simple JSON object might be defined as:

```
{
  "number": 1,
  "decimalNumber": 1.2,
  "boolean": true,
  "string": "Any string",
  "object": {
    "key": "value"
  },
  "array": [
    "value1",
    "value2"
  ],
  "nullValue": null
}
```

For more information on JSON, see json.org.

The data format of MWS is defined as follows:

- Input for a POST or PUT must be in JSON format. Set the `Content-Type` header to `application/json`.
- Output is in JSON format and always consists of an object with zero or more key/value pairs.
- The output may also be "pretty-printed" or formatted for human viewing by sending a URL parameter. See Global URL Parameters for more information.

3.3 Global URL Parameters

⚠ All URL parameters are optional.

Parameter	Valid Values	Description
pretty	*true*	Controls pretty printing of output
fields	Comma-Separated String	Includes only specified fields in output
exclude-fields	Comma-Separated String	Excludes specified fields from output
max	Integer	The maximum number of items to return
offset	Integer	The index of the first item to return

Pretty (pretty)

By default, the output is easy for a machine to read but difficult for humans to read. The *pretty* parameter formats the output so that it is easier to read.

Field Selection (fields)

The *fields* parameter will include **only** the specified fields in the output. For list queries, the field selection acts on the objects in *results* and not on the *totalCount* or *results* properties themselves.

The format of the *fields* parameter is a comma-separated list of properties that should be included, as in `id,state`. Using periods, sub-objects may also be specified, and fields of these objects may be included as well. This is done with the same syntax for both single sub-objects and lists of sub-objects, as in `id,requirements.requiredNodeCountMinimum,blockReason.message`.

Example for a job query

Request

```
GET
/rest/jobs?fields=id,flags,requirements.requiredProcessorCountMinimum,schedule.offset
```

Response

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "job.1",
    "flags": ["RESTARTABLE"],
    "requirements": [{"requiredProcessorCountMinimum": 4}],
    "schedule": {"offset": 100}
  }]
}
```

Field Exclusion (exclude-fields)

The *exclude-fields* parameter is the opposite of the *fields* parameter. All fields will be included in the output **except** those that are specified. For list queries, the field exclusion acts on the objects in *results* and not on the *totalCount* or *results* properties themselves.

The format of the *exclude-fields* parameter is a comma-separated list of properties that should be excluded from the output, as in `id,state`. Using periods, sub-objects may also be specified, and fields of these objects may be excluded as well. This is done with the same syntax for both single sub-objects and lists of sub-objects, as in `id,requirements.requiredNodeCountMinimum,blockReason.message`.

Example

Suppose a query returns the following JSON:

Request with No Field Exclusion

```
GET /objects
```

Response

```
{
  "id": "1",
  "listOfStrings":   [
    "string1",
    "string2"
  ],
  "listOfObjects": [  {
    "item1": "value1",
    "item2": "value2"
  }],
  "singleObject":   {
    "id": "obj1",
    "field1": "value1"
  }
}
```

The same query with *exclude-fields* would return the following output:

Request with Field Exclusion

```
GET /objects?exclude-fields=id,listOfObjects.item2,singleObject.field1,listOfStrings
```

Response

```
{
  "listOfObjects": [{"item1": "value1"}],
  "singleObject": {"id": "obj1"}
}
```

Sorting (sort)

Services, Service Templates, Images, and Events support sorting based on MongoDB syntax by using the *sort* parameter. To sort in ascending order, specify a `1` for the sorting field. To sort in descending order, specify a `-1`. Objects can also be sorted on nested fields by using dot notation to separate the sub-fields, such as `field.subfield1.subfield2`.

Examples

To sort services in ascending order by account:

```
http://localhost/mws/rest/services?sort={"account":1}
```

To sort services in descending order by account:

```
http://localhost/mws/rest/services?sort={"account":-1}
```

To sort services in descending order by processors:

```
http://localhost/mws/rest/services?sort={"attributes.moab.job.resources.procs":-1}
```

To sort service templates in ascending order by name:

```
http://localhost/mws/rest/service-templates?sort={"name":1}
```

To sort service templates in descending order by name:

```
http://localhost/mws/rest/service-templates?sort={"name":-1}
```

To sort service templates in ascending order by the nested field template:

```
http://localhost/mws/rest/service-templates?sort={"attributes.moab.job.template":1}
```

3.4 Responses and Return Codes

Various HTTP responses and return codes are generated from MWS operations. These are documented below according to the operation that they are associated with.

Listing and Showing Resources

For any successful list or show operation (`GET`), a `200 OK` response code is always returned. No additional headers beyond those typical of a HTTP response are given in the response.

The body of this response consists of the results of the list or show operation. For a list operation, the results are wrapped in metadata giving total and result counts. The result count represents the number of resource records returned in the current request, and the total count represents the number of all records available. These differ when querying or the `max` and `offset` parameters are used. The following is an example of a list operation response:

JSON List Response Body

```
{
    "resultCount":1,
    "totalCount":5,
    "results":[
        {
            "id":"Moab.1",
            …
        }
    ]
}
```

For a show operation, the result is given as a single object:

JSON Show Response Body

```
{
    "id":"Moab.1",
    …
}
```

Creating Resources

A successful creation (`POST`) of a resource has two potential response codes:

- If the resource was created immediately, a `201 Created` response code is returned.
- If the resource is still being created, a `202 Accepted` response code is returned.

In either case, a `Location` header is added to the response with the full URL which can be used to get more information about the newly created resource or the task associated with creating the resource (if a `202` is returned).

Additionally, the body of the response will contain the unique identifier of the newly created resource or the unique identifier for the task associated with creating the resource (if a `202` is returned).

For example, during creation or submission of a job, a `201` response code is returned with the following response headers and body:

Job Creation Response Headers

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: /mws/rest/jobs/Moab.21
X-Moab-Status: Success
X-Moab-Code: 000
Content-Type: application/json;charset=utf-8
Content-Length: 16
Date: Wed, 21 Dec 2011 23:04:47 GMT
```

Job Creation Response Body

```
{"id":"Moab.21"}
```

For another resource that is not immediately created, such as virtual machines, the response headers and body are shown below. In this case, a job is submitted to track the progress of the VM creation. This job contains information pertaining to the VM that is being created.

VM Creation Response Headers

```
HTTP/1.1 202 Accepted
Server: Apache-Coyote/1.1
Location: /mws/rest/jobs/vmcreate-1
X-Moab-Status: Success
X-Moab-Code: 000
Content-Type: application/json;charset=utf-8
Content-Length: 23
Date: Wed, 21 Dec 2011 23:12:50 GMT
```

VM Creation Response Body

```
{"jobId":"vmcreate-1"}
```

As can be seen, the body of the response contains only a job ID and not the ID of the virtual machine.

Modifying Resources

For any successful resource modification operation (`PUT`), a `200 OK` or `202 Accepted` response code is returned. A `200` response code signifies that the modification was immediately completed. No additional headers are returned in this case. A `202` response code is used again to signify that the modification is not yet complete and additional actions are taking place. In this case, a `Location` header is also returned with the full URL of the resource describing the additional actions.

In the case of a `200` response code, the body of this response typically consists of an object with a single `messages` property containing a list of statuses or results of the modification(s). However, a few exceptions to this rule exist as documented in the Resources section. In the case of a `202` response code, the format is the same as for a `202` during a creation operation, in that the body consists of an object with the unique identifier for the task associated with the additional action(s).

For example, when modifying a job, several messages may be returned as follows with the associated `200` response code.

Job Modification Response Headers

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Moab-Status: Success
X-Moab-Code: 000
X-Moab-Message:
Content-Type: application/json;charset=utf-8
Content-Length: …
Date: Thu, 22 Dec 2011 16:49:43 GMT
```

JSON Modify Response Body

```
{
    "messages":[
        "gevent processed",
        "variables successfully modified"
    ]
}
```

When modifying a virtual machine, however, the action sometimes does not occur immediately, such as when migrating the VM to another hypervisor as described in the VM documentation. In this case, the headers and response body are as follows:

VM Modification Response Headers

```
HTTP/1.1 202 Accepted
Server: Apache-Coyote/1.1
Location: /mws/rest/jobs/vmmigrate-1
X-Moab-Status: Success
X-Moab-Code: 000
Content-Type: application/json;charset=utf-8
Content-Length: 22
Date: Wed, 21 Dec 2011 23:12:50 GMT
```

VM Modification Response Body

```
{"jobId":"vmmigrate-1"}
```

Deleting Resources

For any successful resource deletion operation (`DELETE`), a `200 OK` or `202 Accepted` response code is returned. A `200` response code signifies that the deletion was immediately completed. No additional headers are returned in this case. A `202` response code is used again to signify that the deletion is not yet complete and additional actions are taking place. In this case, a `Location` header is also returned with the full URL of the resource describing the additional actions.

In the case of a `200` response code, the body of this response is empty. In the case of a `202` response code, the format is the same as for a `202` during a creation operation, in that the body consists of an object with the unique identifier for the task associated with the additional action(s).

For example, when deleting a job, a `200` response code is returned with an empty body as shown below.

Job Deletion Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Moab-Status: Success
X-Moab-Code: 000
X-Moab-Message:
Content-Type: application/json;charset=utf-8
Content-Length: 0
Date: Thu, 22 Dec 2011 16:49:43 GMT
```

When deleting a virtual machine, however, the action does not occur immediately. In this case, the headers and response body are as follows:

VM Deletion Response Headers

```
HTTP/1.1 202 Accepted
Server: Apache-Coyote/1.1
Location: /mws/rest/jobs/vmdestroy-1
X-Moab-Status: Success
X-Moab-Code: 000
Content-Type: application/json;charset=utf-8
Content-Length: 22
Date: Wed, 21 Dec 2011 23:12:50 GMT
```

VM Deletion Response Body

```
{"jobId":"vmdestroy-1"}
```

Moab Response Headers

In addition to the typical HTTP headers and the `Location` header described above, several headers are returned if the operations directly interact with Moab. These headers are described in the following table:

Name	Description
X-Moab-Status	One of `Success`, `Warning`, or `Failure`. Describes the overall status of the Moab request.
X-Moab-Code	A three digit code specifying the exact error encountered, used only in debugging.
X-Moab-Message	An optional message returned by Moab during the request.

3.5 Error Messages

Below is an explanation of what error message format to expect when an HTTP status code other than 20x is returned. All error codes have a response code of 400 or greater.

400 Bad Request

This response code is returned when the request itself is at fault, such as when trying to modify a resource with an empty PUT request body or when trying to create a new resource with invalid parameters. The response body is as follows:

```
{
    "messages":[
        "Message describing error",
        "Possible prompt to take action"
    ]
}
```

401 Unauthorized

This response code is returned when authentication credentials are not supplied or are invalid. The response body is as follows:

```
{
    "messages":[
        "You must be authenticated to access this area"
    ]
}
```

403 Forbidden

This response code is returned when the credentials supplied are valid, but the permissions granted are insufficient for the operation. This occurs when using Application Accounts with limited access.

```
{
    "messages":[
        "You are not authorized to access this area"
    ]
}
```

404 Not Found

This response code is returned when the request specifies a resource that does not exist. The response body is as follows:

```
{
    "messages":[
        "The resource with id 'uniqueId' was not found"
    ]
}
```

405 Method Not Allowed

This response code is returned when a resource does not support the specified HTTP method as an operation. The response body is as follows:

```
{
    "messages":[
        "The specified HTTP method is not allowed for the requested resource"
    ]
}
```

500 Internal Server Error

This indicates that there was an internal server error while performing the request, or that an operation failed in an unexpected manner. These are the most serious errors returned by MWS. If additional information is needed, the MWS log may contain further error data. The response body is as follows:

```
{
    "messages":[
        "A problem occurred while processing the request",
        "A message describing the error"
    ]
}
```

3.6 Pre and Post-Processing Hooks

MWS provides functionality to intercept and modify data sent to and returned from web services for all available resources. This is done by creating hooks in Groovy files located in a sub-directory of the `MWS_HOME` directory (`/opt/mws/hooks` by default).

> ⚠ The full reference for available hooks and methods available to them can be found on the Hooks page in the reference guide.

Configuring Hooks

The directory of the hooks folder may be changed by providing a value for `mws.hooks.location` in the configuration file. If the directory starts with a path separator (ie `/path/to/hooks`), it will be treated as an absolute path. Otherwise, it will be used relative to the location of the MWS home directory.

For example, if the MWS home directory is set to `/opt/mws`, the hooks directory by default would be in `/opt/mws/hooks`. Changing the `mws.hooks.location` property to `myhooks` would result in the hooks directory being located at `/opt/mws/myhooks`. Due to the default location of the MWS home directory, the default directory of the hooks directory is `/opt/mws/hooks`.

On startup, if the hooks directory does not exist, it will be created with a simple `README.txt` file with instructions on how to create hooks, the objects available, and the hooks available. If the folder or file is unable to be created, a message will be printed on the log with the full location of a README file, copied into a temporary directory.

Defining Hooks for a Resource

Hooks are defined for resources by creating groovy class files in the hooks directory (`MWS_HOME/hooks` by default). Each groovy file must be named by the resource URL it is associated with and end in ".groovy". The following table shows some possible hook files that may be created. Notice that the virtual machines hook file is abbreviated as `vms`, just as the URL for virtual machines is `/rest/vms`. In all cases, the hook file names will match the URLs.

Resource	Hook Filename
Jobs	jobs.groovy
Nodes	nodes.groovy
Virtual Machines	vms.groovy
Pending Actions	pending-actions.groovy
url	*url.groovy*

A complete example of a hook file is as follows:

Complete Hook File

```
// Example before hook
def beforeList = {
    // Perform actions here
    // Return true to allow the API call to execute normally
    return true
}

def beforeShow = {
    // Perform actions here
    // Render messages to the user with a 405 Method Not Allowed
    //    HTTP response code
    renderMessages("Custom message here", 405)
    // Return false to stop normal execution of the API call
    return false
}

// Example after hook
def afterList = { o ->
    if (!isSuccess()) {
        // Handle error here
        return false
    }
    // Perform actions here
    return o
}
```

As the specific format for the hooks for `before` and `after` are different, each will be explained separately.

Before Hooks

As shown above, `before` hooks require no arguments. They can directly act on several properties, objects, and methods as described in the Hooks reference guide. The return value is one of the most important aspects of a `before` hook. If it is `false`, a `renderMessages`, `renderObject`, `renderList`, `render`, or `redirect` method **must** first be called. This signifies that the API call should be interrupted and the render or redirect action specified within the hook is to be completed immediately.

A return value of `true` signifies that the API call should continue normally. Parameters, session variables, request and response variables may all be modified within a `before` hook.

⚠ If no return value is explicitly given, the result of the last statement in the `before` hook to be executed will be returned. This may cause unexpected behavior if the last statement resolves to `false`.

For all methods available to `before` hooks as well as specific examples, see the Hooks page in the reference guide.

After Hooks

`After` hooks are always passed one argument: the object or list that is to be rendered as JSON. This may be modified as desired, but note that the object or list value is either a JSONArray or JSONObject. Therefore, it may not be accessed and modified as a typical groovy Map.

Unlike `before` hooks, `after` hooks should not call the `render*` methods directly. This method will automatically be called on the resulting object or list returned. The `redirect` and `render` methods should also not be called at this point. Instead, if a custom object or list is desired to be used, the `serializeObject` and `serializeList` methods are available to create suitable results to return.

The return value of an `after` hook may be one of two possibilities:

1. The potentially modified object or list passed as the first argument to the hook. In this case, this value will override the output object or list unless it is null.
2. Null or false. In this case, the original, unmodified object or list will be used in the output.

> ⚠ The return value of the `after` hook, if not null or false, **must** be the modified object passed into the hook or an object or list created with the `serialize*` methods.

For all methods available to `after` hooks as well as specific examples, see the Hooks page in the reference guide.

Error Handling

`After` hooks, unlike the `before` hooks, have the possibility of handling errors encountered during the course of the request. Handling errors is as simple as adding a one-line check to the hook as shown above or in the following code:

```
if (!isSuccess()) {
    // Handle error
    return false
}
```

It is recommended that each `after` hook contain at least these lines of code to prevent confusion on what the input object or list represents or should look like.

The `isSuccess()` function is true if and only if the HTTP response code is 400 or higher, such as a 404 Not Found, 400 Bad Request, or 500 Internal Server Error and the cause of the error state was not in the associated `before` hook. In other words, objects and lists rendered in the `before` hook with any HTTP response code will never run the associated `after` hook.

When handling errors, the passed in object will always contain a `messages` property containing a list of Strings describing the error(s) encountered.

Defining Common Hooks

Sometimes it is beneficial to create hooks which are executed for all calls of a certain type, such as a `beforeList` hook that is executed during the course of listing any resource. These are possible using an `all.groovy` file. The format of this file is exactly the same as other hook files. The order of execution is as follows:

1. `Before` common hook executed
2. `Before` resource-specific hook executed
3. Normal API call executed
4. `After` resource-specific hook executed
5. `After` common hook executed

3.7 Authentication

MWS uses Basic Authentication for all REST API requests. This means that a username and password must be provided for each call to resources. There are two types of accounts that can be granted access: **Users** and **Applications**.

- For insructions on how to set the credentials for the default **User** account, see the "Client Connections to MWS" section in Security.
- For insructions on how to manage **Application** accounts, see Application Accounts.

To use Basic Authentication, each client request must contain a header that looks like this:

```
Authorization: Basic YWRhcHRpdmU6YzNVU3R1bkU=
```

The string after the word `Basic` is the base64 encoding of *username* : *password* . In the example above, `YWRhcHRpdmU6YzNVU3R1bkU=` is the base64 encoding of `adaptive:c3UStunE`. See section 2 of RFC 2617 for more details.

⚠ The username and password in the Basic Authentication header are encoded but not encrypted. Therefore, it is **strongly** recommended that MWS be run behind a proxy (like Apache) with SSL enabled. See the section "Encrypting Client Connections using Apache and SSL" under Security.

4 Resources

The sections below show the MWS resources and the HTTP methods defined on them. The prefix for these resources depends on how the `mws.war` file is deployed. A typical prefix would be `http://localhost:8080/mws`. Using this example, one absolute resource URI would be `http://localhost:8080/mws/rest/jobs`.

4.1 Access Control Lists

This section describes behavior of the **ACL Rules** (Access Control List Rules) object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

> ⚠ The ACL API contains the type and description of all fields in the **ACL Rules** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

> ⚠ ACLs are not directly manipulated through a single URL, but with sub-URLs of the other objects such as Virtual Containers and Reservations.

Resource	GET	PUT	POST	DELETE
/rest/reservations/**rsvId**/acl-rules/**aclId**		Create or Update ACLs		Delete ACL
/rest/vcs/**vcId**/acl-rules/**aclId**		Create or Update ACLs		Delete ACL

4.1.1 Getting ACLs

Although **ACL Rules** cannot be retrieved directly using the GET method on any of the `acl-rules` resources, **ACL Rules** are attached to supported objects when querying for them. Each supported object contains a field named `aclRules`, which is a collection of the **ACL Rules** defined on that object.

Supported Objects

The following is a list of objects that will return **ACL Rules** when queried:

- Reservations
- Standing Reservations
- Virtual Containers

4.1.2 Creating or Updating ACLs

The HTTP PUT method is used to create or update **ACL Rules**. The request body can contain one or more **ACL Rules**. If an **ACL Rule** with the same `type` and `value` exists, then it will be overwritten.

Quick Reference

```
PUT http://localhost/mws/rest/reservations/<rsvId>/acl-rules
PUT http://localhost/mws/rest/vcs/<vcId>/acl-rules
```

4.1.2.1 Create or Update ACL

URLs and Parameters

```
PUT http://localhost/mws/rest/reservations/<objectId>/acl-rules
PUT http://localhost/mws/rest/vcs/<objectId>/acl-rules
```

Parameter	Required	Type	Valid Values	Description
objectId	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available for the PUT method, along with some sample values.

JSON Request Body

```
{"aclRules": [{
  "affinity": "POSITIVE",
  "comparator": "LEXIGRAPHIC_EQUAL",
  "type": "USER",
  "value": "ted"
}]}
```

Sample Response

⚠ This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{"messages":["Virtual container 'vc1' successfully modified"]}
```

Samples

Create or update multiple ACLs on a single object:

PUT http://localhost/mws/rest/reservations/system.21/acl-rules

```
{"aclRules": [
    {
    "affinity": "POSITIVE",
    "comparator": "LESS_THAN_OR_EQUAL",
    "type": "DURATION",
    "value": "3600"
  },
    {
    "affinity": "POSITIVE",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "USER",
    "value": "ted"
  }
]}
```

Restrictions

- **ACL Rules** cannot be added to or updated on **Standing Reservations**.
- The **affinity** and **comparator** fields are ignored for **Virtual Containers**.

4.1.3 Deleting ACLs

The HTTP DELETE method is used to remove **ACL Rules**.

Quick Reference

⚠ **ACL Rules** cannot be removed from **Standing Reservations**.

```
DELETE http://localhost/mws/rest/reservations/<rsvId>/acl-rules/<aclId>
DELETE http://localhost/mws/rest/vcs/<vcId>/acl-rules/<aclId>
```

4.1.3.1 Delete ACL

URLs and Parameters

```
DELETE http://localhost/mws/rest/reservations/<objectId>/acl-rules/<aclId>
DELETE http://localhost/mws/rest/vcs/<objectId>/acl-rules/<aclId>
```

Parameter	Required	Type	Valid Values	Description
objectId	Yes	String	-	The unique identifier of the object from which to remove the **ACL Rule**.
aclId	Yes	String	-	A string representing the **ACL Rule**, with the format `type:value`.

See Global URL Parameters for available URL parameters.

Sample Response

> ⚠ This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{"messages":["Successfully modified virtual container 'vc1'"]}
```

Restrictions

- **ACL Rules** cannot be removed from **Standing Reservations**.

4.2 Accounts

This section describes behavior of the **Accounts** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

> ⚠ The Account API contains the type and description of fields that all **Accounts** have in common.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/accounts	Get all accounts			
/rest/accounts/**id**	Get specified account			

4.2.1 Getting Accounts

The HTTP GET method is used to retrieve **Accounts** information.

Quick Reference

```
GET http://localhost/mws/rest/accounts
```

4.2.1.1 Get All Accounts

URLs and Parameters

```
GET http://localhost/mws/rest/accounts?proxy-user=<USER>
[&custom-fields=Department][&query={"deleted":false}][&sort={"requestId":-1}]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
custom-fields	No	Comma-Separated String	Includes custom MAM account attributes.	custom-fields=Department
query	No	JSON	Query for specific results.	query={"deleted":false}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"requestId":-1}

⚠ The `query` parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/accounts?proxy-user=amy&fields=name,description

```
{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
          {
      "id": "biology",
      "description": "Biology Dept."
    },
          {
      "id": "chemistry",
      "description": "Chemistry Dept."
    }
  ]
}
```

4.2.1.2 Get Single Account

URLs and Parameters

```
GET http://localhost/mws/rest/accounts/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Responses

GET http://localhost/mws/rest/accounts/chemistry?proxy-user=amy

```
{
  "id": "chemistry",
  "active": true,
  "organization": "",
  "description": "Chemistry Dept",
  "creationTime": "2012-04-11 06:56:11 MDT",
  "modificationTime": "2012-04-11 06:56:11 MDT",
  "deleted": false,
  "requestId": 94,
  "transactionId": 283,
  "users":   [
          {
      "id": "amy",
      "active": true,
      "admin": false
    },
          {
      "id": "bob",
      "active": true,
      "admin": false
    },
              {
      "id": "dave",
      "active": true,
      "admin": false
    }
  ]
}
```

4.3 Diagnostics

This section describes additional REST calls that are available for performing diagnostics on Moab Web Services.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/diag/about	Get version information			

4.3.1 Version and Build Information

The HTTP GET method is used to retrieve version and build information.

Quick Reference

```
GET http://localhost/mws/rest/diag/about
```

URLs and Parameters

```
GET http://localhost/mws/rest/diag/about
```

Sample Response

The response contains the application version, build number, build date, and revision.

```
{
  "version":"7.0",
  "build":"100",
  "buildDate":"2012-01-01_16-00-00",
  "revision":"1000"
}
```

4.4 Events

This section describes the URLs, request bodies, and responses delivered to and from Moab Web Services for handling events

⚠ The Event API contains the type and description of all fields in the **Event** object. It also contains details regarding which fields are valid during POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/events	Get all events		Create event	
/rest/events/**id**	Get specified event			

Configuration

Logging events to a flat file requires that you make a few changes to the configuration in the log4j section of the mws-config.groovy file so that events will be logged to the events.log file, and all other Moab Web Services logging information will be sent to the mws.log file.

Causing events.log to Roll Based on a Time Window

You can specify how often the events.log file rolls. The following example illustrates the configuration changes you will need make to mws-config.groovy to cause the events.log file to roll based on a time window. (In this example, mws-config.groovy is configured so that events.log rolls daily at midnight.)

Daily rolling events.log configuration in mws-config.groovy

```
log4j = {
  def eventAppender = new org.apache.log4j.rolling.RollingFileAppender(name: 'events',
layout: pattern(conversionPattern: "%m%n"))
  def rollingPolicy = new
org.apache.log4j.rolling.TimeBasedRollingPolicy(fileNamePattern:
'/tmp/events.%d{yyyy-MM-dd}', activeFileName: '/tmp/events.log')
  rollingPolicy.activateOptions()
  eventAppender.setRollingPolicy(rollingPolicy)

appenders {
      appender eventAppender

rollingFile name: 'rootLog',
        file: '/tmp/mws.log',
        maxFileSize: '1GB'
  }

root {
    warn 'rootLog'
  }

trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
}
```

Note the **RollingFileAppender** and the **TimeBasedRollingPolicy** lines. These lines configure Moab Web Services to write the event log to the events.log file.

Rolled log files will have a date appended to their name in this format: 'yyyy-MM-dd' (for example, events.log.2012-02-28).

If you want the event log file to roll at the beginning of each month, change the **TimeBasedRollingPolicy** "fileNamePattern" date format to 'yyyy-MM'. For example:

Monthly event logs

```
def rollingPolicy = new
org.apache.log4j.rolling.TimeBasedRollingPolicy(fileNamePattern:
'/tmp/events.%d{yyyy-MM}', activeFileName: '/tmp/events.log')
```

If you want the event log file to roll at the beginning of each hour, change the date format to 'yyyy-MM-dd_HH:00'. For example:

Hourly event logs

```
def rollingPolicy = new
org.apache.log4j.rolling.TimeBasedRollingPolicy(fileNamePattern:
'/tmp/events.%d{yyyy-MM-dd_HH:00}', activeFileName: '/tmp/events.log')
```

Configuring events.log to Roll Based on File Size Threshold

You can also configure the events.log file to roll when the log size exceeds a specified threshold. The following example illustrates the configuration changes you will need to make to mws-config.groovy to cause the events.log file to roll on a size threshold. (In this example, mws-config.groovy is configured so that events.log rolls when its size exceeds 50 MB.)

mws-config.groovy configuration that rolls events.log based on file size

```
log4j = {
  appenders {
      rollingFile name: 'events',
        file: '/tmp/events.log',
        maxFileSize: '50MB',
        maxBackupIndex:10

rollingFile name: 'rootLog',
        file: '/tmp/mws.log',
        maxFileSize: '1GB'
  }

root {
    warn 'rootLog'
  }

trace additivity:false, events:'com.ace.mws.events.EventFlatFileWriter'
}
```

Note that **maxFileSize** is set to '50MB'. This means that when the events.log file exceeds 50 MB, it will roll.

The name for the rolled log will be "events.log.1". When the new events.log file exceeds 50 MB, it will roll and be named "events.log.1", while the old "events.log.1" file will be renamed "events.log.2". This process will continue until the optional **maxBackupIndex** value is met. In the example above, **maxBackIndex** is set to 10. This means that Moab Web Services will delete all but the ten most recent events.log files. Using this feature helps prevent hard drives from filling up.

Additivity

The **additivity** attribute of the EventFlatFileWriter logger can be either "true" or "false". If you specify "true", events will be logged to the events.log file and the mws.log file. If you specify "false", events will be logged to the events.log file only. (All other Moab Web Services logging information will be logged to the mws.log file, as configured by the rootLog appender.)

To log events to the mws.log file in addition to the events.log file, make this configuration: **additivity:true**. For example:

Logging events to both events.log and mws.log

```
trace additivity:true, events:'com.ace.mws.events.EventFlatFileWriter'
```

For more configuration options, see Apache Extras Companion for log4j.

Deleting Old Events

By default, MWS will hold event data in MongoDB indefinitely. However, if disk space is limited, you may want to regularly delete old, unneeded events from MongoDB. This section contains some examples of how you can do this.

Let's say that you want to delete events that are older than 90 days. You could run this script. (There are 86,400,000 milliseconds in a day, so in this example, `90*86400000` corresponds to 90 days in milliseconds.)

Delete events older than 90 days

```
$ mongo
MongoDB shell version: 2.0.1
connecting to: test
> use mws
> db.event.remove({eventTime:{$lt:new Date(new Date().getTime()-90*86400000)}})
> exit
```

To create a script to perform this task:

deleteOldEvents.sh

```
#!/bin/bash
printf 'use mws_dev\ndb.event.remove({eventTime:{$lt:new Date(new
Date().getTime()-90*86400000)}})\nexit' | mongo
```

Now say that you want to set up a cron job (`$crontab -e`) so that old events are automatically deleted on a certain day of the week (for example, every Sunday at 2:00 a.m.), you would add an entry like this:

cron table entry to delete old events

```
00 02 * * 0 /root/deleteOldEvents.sh
```

4.4.1 Getting Events

The HTTP GET method is used to retrieve **Event** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/events[?query={"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost/mws/rest/events/<id>
```

4.4.1.1 Get All Events

URLs and Parameters

```
GET http://localhost/mws/rest/events[?query={"field":"value"}&sort={"field":<1|-1>}]
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Query for specific results.	query={"status":"failure"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"id":-1}

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost:8080/mws/rest/events

```
{
   "totalCount":3,
   "resultCount":3,
   "results":[
      {
         "details":{

},
         "errorMessage":{
            "errorCode":"542",
            "message":"Cannot communicate with XCAT resource manager",
            "originator":"XCAT"
         },
         "eventCategory":"failure",
         "eventTime":"2012-01-27 15:18:32 MST",
         "eventType":"rmfailure",
         "facility":"rm",
         "primaryObject":{
            "serialization":null,
            "type":"rm",
            "id":"xcat"
         },
         "sourceComponent":"MWM",
         "status":"failure",
         "id":"4faddab8c4aa264506f8ac3d"
      },
      {
         "details":{

},
         "eventCategory":"start",
         "eventTime":"2012-01-27 15:18:31 MST",
         "eventType":"schedcyclestart",
         "facility":"scheduler",
         "sourceComponent":"MWM",
         "status":"success",
         "id":"4faddab8c4aa264506f8ac3c"
      },
      {
         "details":{
            "alpha":"lorem ipsum dolor sit amet",
            "bravo":"2",
            "charlie":"consectetur adipisicing elit, sed do"
         },
         "eventCategory":"start",
         "eventTime":"2012-01-27 15:18:30 MST",
         "eventType":"jobstart",
         "facility":"job",
         "initiatedBy":{
            "proxyUser":"bob",
            "user":"tomcat6"
         },
         "primaryObject":{
            "serialization":"\n         <job JobID="moab.849" ReqAWDuration="100000"
User="bob" />\n      ",
            "type":"job",
            "id":"moab.849"
         },
         "relatedObjects":[
            {
               "type":"vm",
               "id":"vm56"
            },
            {
               "type":"service",
               "id":"lamp.211"
            }
         ],
         "sourceComponent":"MWM",
         "status":"success",
         "id":"4faddab8c4aa264506f8ac3b"
      }
   ]
}
```

Querying Events

It is possible to query events by one or more fields based on MongoDB query syntax. The following contains examples of simple and complex event queries and event queries by date.

Simple Queries

To see only events that are of type "jobsubmit":

```
http://localhost/mws/rest/events?query={"eventType":"jobsubmit"}
```

To see only events of type "jobsubmit" with the status of "failure":

```
http://localhost/mws/rest/events?query={"eventType":"jobsubmit","status":"failure"}
```

To see only events with the "job" facility:

```
http://localhost/mws/rest/events?query={"facility":"job"}
```

To see only events in the "start" event category:

```
http://localhost/mws/rest/events?query={"eventCategory":"start"}
```

More Complex Queries

You can query on embedded JSON objects within the event JSON. For example, to see events associated with proxyUser bob:

```
http://localhost/mws/rest/events?query={"initiatedBy.proxyUser":"bob"}
```

To see only events that are NOT associated with bob:

```
http://localhost/mws/rest/events?query={"initiatedBy.proxyUser":{"$ne":"bob"}}
```

When the field values of the desired events are a finite set, you can use the `$in` operator. For example, to see events that relate to either alice, bob, or charlie:

```
http://localhost/mws/rest/events?query={"initiatedBy.proxyUser":{"$in":["alice","bob",
"charlie"]}}
```

Querying Events by Date

To see events created before January 27, 2012 at 12:08 a.m. MST:

```
http://localhost/mws/rest/events?query={"eventTime":{"$lt":"2012-01-27 12:08:00 MST"}}
```

To see events created before or on January 27, 2012 at 12:08 a.m. MST:

```
http://localhost/mws/rest/events?query={"eventTime":{"$lte":"2012-01-27 12:08:00 MST"}}
```

To see all events created after January 27, 2012 at 12:04 a.m. MST:

```
http://localhost/mws/rest/events?query={"eventTime":{"$gt":"2012-01-27 12:04:00 MST"}}
```

To see all events created after or on January 27, 2012 at 12:04 a.m. MST:

```
http://localhost/mws/rest/events?query={"eventTime":{"$gte":"2012-01-27 12:04:00 MST"}}
```

To see events created between 12:04 a.m. and 12:08 a.m. MST inclusive:

```
http://localhost/mws/rest/events?query={"eventTime":{"$gte":"2012-01-27 12:04:00 MST",
"$lte":"2012-01-27 12:08:00 MST"}}
```

To see events created between 12:04 a.m. and 12:08 a.m. MST inclusive that are associated with proxyUser bob:

```
http://localhost/mws/rest/events?query={"initiatedBy.proxyUser":"bob","eventTime":{
"$gte":"2012-01-27 12:04:00 MST","$lte":"2012-01-27 12:08:00 MST"}}
```

To see events created between 12:04 a.m. and 12:08 a.m. MST inclusive, that are associated with proxyUser bob, and that are of type "jobsubmit":

```
http://localhost/mws/rest/events?query={"initiatedBy.proxyUser":"bob","eventType":
"jobsubmit","eventTime":{"$gte":"2012-01-27 12:04:00 MST","$lte":"2012-01-27 12:08:00
MST"}}
```

Sorting

See the sorting section of Global URL Parameters

Limiting the Number of Results

If you want to limit the number of results of events, you can use the `max` parameter. For example, to see only 10 "vmmigrate" events:

```
http://localhost/mws/rest/events?query={"eventType":"vmmigrate"}&sort={"eventTime"
:1}&max=10
```

To see "vmcreate" events 51-60 when sorted by eventTime in descending order, you can combine `max` with `offset`, as follows:

```
http://localhost/mws/rest/events?query={"eventType":"vmcreate"}&sort={"eventTime"
:-1}&max=51&offset=60
```

Retrieving a Subset of Fields

To cause only certain fields to return for each event, use the `fields` parameter. For example, to show only the eventTime field for each event:

```
http://localhost/mws/rest/events?max=3&fields=eventTime
```

This returns:

```
{
  "totalCount": 187,
  "resultCount": 3,
  "results":   [
    {"eventTime": "2012-03-05 10:01:59 MST"},
    {"eventTime": "2012-03-05 10:02:00 MST"},
    {"eventTime": "2012-03-05 10:02:01 MST"}
  ]
}
```

To show the name, type, and status:

```
http://localhost/mws/rest/events?fields=name,type,status
```

This returns:

```
{
  "totalCount": 187,
  "resultCount": 3,
  "results":   [
        {
      "eventTime": "2012-03-05 10:01:59 MST",
      "eventType": "jobsubmit",
      "status": "failure"
    },
        {
      "eventTime": "2012-03-05 10:02:00 MST",
      "eventType": "jobstart",
      "status": "success"
    },
        {
      "eventTime": "2012-03-05 10:02:01 MST",
      "eventType": "vmmigrate",
      "status": "success"
    }
  ]
}
```

4.4.1.2 Get Single Event

URLs and Parameters

```
GET http://localhost/mws/rest/events/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "details":   {
    "attribute": "walltime",
    "modifier": "increase",
    "value": "200000"
  },
  "eventCategory": "job",
  "eventTime": "2012-01-27 00:07:00 MST",
  "eventType": "jobmodify",
  "facility": "baseline",
  "initiatedBy":   {
    "proxyUser": "bob",
    "user": "viewpoint"
  },
  "primaryObject":   {
    "serialization": null,
    "type": "job",
    "id": "moab.900"
  },
  "sourceComponent": "MWM",
  "status": "success",
  "id": "4fa46081c4aa5b896ac4b757"
}
```

4.4.2 Creating Events

The HTTP POST method is used to create an **Event**.

Quick Reference

```
POST http://localhost/mws/rest/events
```

4.4.2.1 Create Event

URLs and Parameters

```
POST http://localhost/mws/rest/events
```

Request Body

POST http://localhost/mws/rest/events

```
<Events>
    <Event eventTime="2012-01-27T15:18:30.000-07:00"
           eventType="jobstart"
           eventCategory="start"
           sourceComponent="MWM"
           status="success"
           facility="job">
        <InitiatedBy user="tomcat6"
                     proxyUser="bob" />
        <PrimaryObject objectType="job"
                       objectID="moab.849">
            <Serialization>
                <![CDATA[<job JobID="moab.849" ReqAWDuration="100000" User="bob" />]]>
            </Serialization>
        </PrimaryObject>
        <RelatedObjects>
            <RelatedObject objectType="vm"
                           objectID="vm56" />
            <RelatedObject objectType="service"
                           objectID="lamp.211" />
        </RelatedObjects>
        <Details>
            <Detail name="alpha">lorem ipsum dolor sit amet</Detail>
            <Detail name="bravo">2</Detail>
            <Detail name="charlie">consectetur adipisicing elit, sed do</Detail>
        </Details>
    </Event>
    <Event eventTime="2012-01-27T15:18:31.000-07:00"
           eventType="schedcyclestart"
           eventCategory="start"
           sourceComponent="MWM"
           status="success"
           facility="scheduler" />
    <Event eventTime="2012-01-27T15:18:32.000-07:00"
           eventType="rmfailure"
           eventCategory="failure"
           sourceComponent="MWM"
           status="failure"
           facility="rm">
        <PrimaryObject objectType="rm"
                       objectID="xcat" />
        <ErrorMessage originator="XCAT"
                      errorCode="542">Cannot communicate with XCAT resource manager
</ErrorMessage>
    </Event>
</Events>
```

Unlike most other URLs, the events URL accepts XML, instead of JSON.

Sample Response

If the request was successful, the response will be an object with an id property containing the ID of the newly created events. On failure, the response is an error message.

JSON Response

```
[{"id":"4fadd83bc4aa366464599e1a"},{"id":"4fadd83bc4aa366464599e1b"},{"id":
"4fadd83bc4aa366464599e1c"}]
```

Below is an example of events.log output for a successful event request:

```
2012-01-27T15:18:30.000-07:00 type="jobstart" category="start" sourceComponent="MWM"
status="success" facility="job" initiatedBy.user="tomcat6" initiatedBy.proxyUser="bob"
primaryObject.type="job" primaryObject.id="moab.849" relatedObject.0.type="service"
relatedObject.0.id="lamp.211" relatedObject.1.type="vm" relatedObject.1.id="vm56"
detail.alpha="lorem ipsum dolor sit amet" detail.bravo="2" detail.charlie="consectetur
adipisicing elit, sed do" primaryObject.serialization="<job JobID=\"moab.849\"
ReqAWDuration=\"100000\" User=\"bob\" />"
2012-01-27T15:18:31.000-07:00 type="schedcyclestart" category="start" sourceComponent=
"MWM" status="success" facility="scheduler"
2012-01-27T15:18:32.000-07:00 type="rmfailure" category="failure" sourceComponent="MWM"
status="failure" facility="rm" primaryObject.type="rm" primaryObject.id="xcat"
error.originator="XCAT" error.code="542" error.message="Cannot communicate with XCAT
resource manager"
```

Note that " (double quote) characters in the input have been replaced by \" characters in the output. (For other character restrictions, see the "Restrictions" section below.)

Restrictions

Special characters, such as newline, carriage return, and " (double quote) are encoded in the output of events.log to make events.log easy to parse with scripts and third party tools. For example, if the input XML contains:

```
<ErrorMessage>RM says, "Cannot provision vm21"</ErrorMessage>
```

Then the following will be output to events.log:

```
error.message="RM says, \"Cannot provision vm21\""
```

Notice that " has been replaced with \". This table contains the most common encodings.

Character	Escape Sequence
" (double quote)	\"
\ (backslash)	\\
newline	\n
carriage return	\r
tab	\t

(For more information see escape sequences for Java Strings.)

Other restrictions include:

- Detail names can only contain alpha-numeric characters, colons (:), underscores (_), and dashes (-).
- **primaryObject.serialization**, **error.message**, and **detail values** (for example detail.someName="some value") can contain any printable ASCII character. Single quotes (') and double quotes (") cannot be contained in any other field.
- A single event cannot contain more than more than 50 details. If an event has more than 50 details, the excess details will be ignored.

4.5 Funds

This section describes behavior of the **Fund** object and all related objects in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

> ⚠ The Fund API, FundBalance API, FundStatement API, and FundStatementSummary API contain the type and description of all fields in the **Fund** object as well as related objects and reports given in the URLs below.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/funds	Get all funds			
/rest/funds/**id**	Get specified fund			
/rest/funds/balances	Get all fund balances			
/rest/funds/reports/statement	Get fund statement			
/rest/funds/reports/statement/summary	Get fund statement summary			

4.5.1 Getting Funds

The HTTP GET method is used to retrieve **Fund** information.

Quick Reference

```
GET http://localhost/mws/rest/funds?proxy-user=<USER>
[&active=true][&custom-fields=health][&filter={"project"
:"chemistry"}][&filter-type=NonExclusive][&query={"priority":"2"}][&sort={"id":-1}]
GET http://localhost/mws/rest/funds/<id>?proxy-user=<USER>
[&active=true][&custom-fields=health][&filter={"project"
:"chemistry"}][&filter-type=NonExclusive]
GET http://localhost/mws/rest/funds/balances?proxy-user=<USER>
[&custom-fields=health][&filter={"project":"chemistry"}][&filter-type=NonExclusive]
GET http://localhost/mws/rest/funds/reports/statement?proxy-user=<USER>[&filter=
<FILTER>][&filter-type=<FILTER-TYPE>][&start-time=<DATE-STRING>][&end-time=
<DATE-STRING>][&context=<CONTEXT>]
GET http://localhost/mws/rest/funds/reports/statement/summary?proxy-user=<USER>
[&filter=<FILTER>][&filter-type=<FILTER-TYPE>][&start-time=<DATE-STRING>][&end-time=
<DATE-STRING>]
```

4.5.1.1 Get All Funds

URLs and Parameters

```
GET http://localhost/mws/rest/funds?proxy-user=<USER>
[&active=true][&custom-fields=health][&filter={"project"
:"chemistry"}][&filter-type=NonExclusive][&query={"priority":"2"}][&sort={"id":-1}]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
active	No	Boolean	Lists only active or non-active allocations of the fund. The fund amount becomes the sum of the active/inactive allocations.	active=true
custom-fields	No	Comma-Separated String	Includes custom MAM fund attributes.	custom-fields=health
filter	No	JSON	Query funds based on defined MAM filter.	filter={"project":"chemistry"}
filter-type	No	String	Query funds based on defined MAM filter type.	filter-type=NonExclusive
query	No	JSON	Query for specific results.	query={"priority":"2","allocation.active":"fa
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"id":-1}

⚠ The `query` parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.

See Global URL Parameters for available URL parameters.

Sample Response

GET
http://localhost/mws/rest/funds?proxy-user=amy&fields=id,name,allocations,constraints

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": 2,
    "name": 1204,
    "allocations": [    {
      "id": 2,
      "startTime": "-infinity",
      "endTime": "2012-02-02 09:34:42 MST",
      "amount": 9060000,
      "creditLimit": 0,
      "deposited": 9060000,
      "active": true,
      "description": ""
    }],
    "fundConstraints": [    {
      "id": 2,
      "name": "CostCenter",
      "value": 1204
    }]
  }]
}
```

4.5.1.2 Get Single Fund

URLs and Parameters

```
GET http://localhost/mws/rest/funds/<id>?proxy-user=<USER>
[&active=true][&custom-fields=health][&filter={"project"
:"chemistry"}][&filter-type=NonExclusive]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
active	No	Boolean	Lists only active or non-active allocations of the fund. The fund amount becomes the sum of the active/inactive allocations.	active=true
custom-fields	No	Comma-Separated String	Includes custom MAM fund attributes.	custom-fields=health
filter	No	JSON	Query funds based on defined MAM filter.	filter={"project":"chemistry"}
filter-type	No	String	Query funds based on defined MAM filter type.	filter-type=NonExclusive

See Global URL Parameters for available URL parameters.

Sample Responses

Fund sample response

```
{
  "id": 2,
  "name": 1204,
  "priority": 0,
  "description": "R&D for Manufacturing",
  "creationTime": "2012-02-02 09:34:42 MST",
  "amount": 9060000,
  "deposited": 9060000,
  "creditLimit": 0,
  "allocations": [  {
    "id": 2,
    "startTime": "-infinity",
    "endTime": "infinity",
    "amount": 9060000,
    "creditLimit": 0,
    "deposited": 9060000,
    "active": true,
    "description": ""
  }],
  "fundConstraints": [  {
    "id": 2,
    "name": "CostCenter",
    "value": 1204
  }]
}
```

4.5.1.3 Get All Fund Balances

URLs and Parameters

```
GET http://localhost/mws/rest/balances?proxy-user=<USER>
[&custom-fields=health][&filter={"project":"chemistry"}][&filter-type=NonExclusive]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
custom-fields	No	Comma-Separated String	Includes custom MAM fund attributes.	custom-fields=health
filter	No	JSON	Query funds based on defined MAM filter.	filter={"project":"chemistry"}
filter-type	No	String	Query funds based on defined MAM filter type.	filter-type=NonExclusive

See Global URL Parameters for available URL parameters.

Sample Response

The fund balances resources is an aggregation of fund data. See the FundBalance API page for more details.

GET http://localhost/mws/rest/balances?proxy-user=amy

```
{
  "totalCount": 2,
  "resultCount": 2,
  "results": [
    {
      "id": 2,
      "name": 1204,
      "priority": 0,
      "description": "R&D for Manufacturing",
      "creationTime": "2012-02-02 09:34:42 MST",
      "amount": 9060000,
      "deposited": 9060000,
      "creditLimit": 0,
      "reserved": 0,
      "allocations": [
        {
          "id": 2,
          "amount": 9060000,
          "creditLimit": 0,
          "deposited": 9060000
        }
      ],
      "fundConstraints": [
        {
          "id": 2,
          "name": "CostCenter",
          "value": 1204
        }
      ],
      "balance": 9060000,
      "available": 9060000,
      "allocated": 9060000,
      "used": 0,
      "percentRemaining": 100,
      "percentUsed": 0
    },
    {
      "id": 5,
      "name": "",
      "priority": 0,
      "description": "",
      "creationTime": "2012-04-03 09:25:47 MDT",
      "amount": 901290219001,
      "deposited": 901290219021,
      "creditLimit": 30,
      "reserved": 84018308897.68,
      "allocations": [
        {
          "id": 6,
          "amount": 901290219001,
          "creditLimit": 30,
          "deposited": 901290219021
        }
      ],
      "fundConstraints": [],
      "balance": 817271910103.32,
      "available": 817271910133.32,
      "allocated": 901290219051,
      "used": 20,
      "percentRemaining": 100,
      "percentUsed": 0
    }
  ]
}
```

4.5.1.4 Get Fund Statement

URLs and Parameters

```
GET http://localhost/mws/rest/funds/reports/statement?proxy-user=<USER>[&filter=
<FILTER>][&filter-type=<FILTER-TYPE>][&start-time=<DATE-STRING>][&end-time=
<DATE-STRING>][&context=<CONTEXT>]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
filter	No	JSON	Query funds based on defined MAM filter.	filter={"project":"chemistry"}
filter-type	No	String	Query funds based on defined MAM filter type.	filter-type=NonExclusive
start-time	No	Date, -infinity, or now	Filter allocations and transaction after a start time.	start-time=2012-04-03 15:24:39 MDT
end-time	No	Date, -infinity, or now	Filter allocations and transactions before an end time.	end-time=2012-04-03 15:24:39 MDT
context	No	hpc or cloud	The context to use in Moab Accounting Manager.	context=hpc

The `context` parameter overrides the default context set for MAM using the `mam.context` configuration parameter. See the Configuration page for more information on this parameter.

See Global URL Parameters for available URL parameters.

Sample Response

The fund statement report provides a snapshot of the current funds. See the FundStatement API for more details.

```
GET
http://localhost/mws/rest/funds/reports/statement?proxy-user=amy&fields=startBalance,endBalance
```

```
{
  "startBalance":1234.01,
  "endBalance":1000
}
```

4.5.1.5 Get Fund Statement Summary

URLs and Parameters

```
GET http://localhost/mws/rest/funds/reports/statement/summary?proxy-user=<USER>
[&filter=<FILTER>][&filter-type=<FILTER-TYPE>][&start-time=<DATE-STRING>][&end-time=
<DATE-STRING>]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
filter	No	JSON	Query funds based on defined MAM filter.	filter={"project":"chemistry"}
filter-type	No	String	Query funds based on defined MAM filter type.	filter-type=NonExclusive
start-time	No	Date, -infinity, or now	Filter allocations and transaction after a start time.	start-time=2012-04-03 15:24:39 MDT
end-time	No	Date, -infinity, or now	Filter allocations and transactions before an end time.	end-time=2012-04-03 15:24:39 MDT

See Global URL Parameters for available URL parameters.

Sample Response

The fund statement summary is slightly different from the typical fund statement in that the transactions are provided as summaries grouped by `object` and `action`. See the FundStatementSummary API for more details.

```
GET
http://localhost/mws/rest/funds/reports/statement/summary?proxy-user=amy&fields=totalCredits,totalD
```

```
{
  "totalCredits":200.02,
  "totalDebits":-100,
  "transactions":[ {
      "action":"Deposit",
      "amount":200.02,
      "count":2
    }, {
      "action":"Charge",
      "amount":-100,
      "count":1
    }
  ]
}
```

4.6 Images

This section describes behavior of the **Image** resource in Moab Web Services. An image resource is used to track the different types of operating systems and hypervisors available in the data center. It also tracks which virtual machines are available on the hypervisors. This section describes the URLs, request bodies, and responses delivered to and from Moab Web Services.

The Image API contains the type and description of all fields in the **Image** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/images	Get All Images		Create Image	
/rest/images/**id**	Get Specified Image	Modify Image		Delete Image
/rest/images/**name**	Get Specified Image	Modify Image		Delete Image

4.6.1 Getting Images

The HTTP GET method is used to retrieve **Image** information. You can query all objects or a single object.

Quick Reference

```
GET http://localhost/mws/rest/images[?query={"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost/mws/rest/images/<id>
GET http://localhost/mws/rest/images/<name>
```

4.6.1.1 Get All Images

URLs and Parameters

```
GET http://localhost/mws/rest/images[?query={"field":"value"}&sort={"field":<1|-1>}]
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Queries for specific results.	query={"type":"stateful","osType":"linux"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"name":-1}

It is possible to query images by one or more fields based on MongoDB query syntax.

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/images?fields=id,name

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "4fa197e68ca30fc605dd1cf0",
    "name": "centos5-stateful"
  }]
}
```

Sorting and Querying

See the sorting and querying sections of Global URL Parameters.

4.6.1.2 Get Single Image

URLs and Parameters

```
GET http://localhost/mws/rest/images/<id>
GET http://localhost/mws/rest/images/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the Image.
name	Yes	String	-	The name of the Image.

See Global URL Parameters for available URL parameters.

⚠ You must specify either **id** or **name**, but you do not have to specify both.

Sample Response

GET http://localhost/mws/rest/images/centos5-compute-stateful

```
{
  "active":true,
  "extensions":{
    "xcat":{
      "os":"centos",
      "architecture":"x86_64",
      "profile":"compute"
    }
  },
  "features":[],
  "hypervisor":false,
  "id":"4fa197e68ca30fc605dd1cf0",
  "name":"centos5-compute-stateful",
  "osType":"linux",
  "supportsPhysicalMachine":false,
  "supportsVirtualMachine":true,
  "templateName":"",
  "type":"stateful",
  "version":0,
  "virtualizedImages":[]
}
```

The `version` field contains the current version of the database entry and does **not** reflect the version of the operating system. See Modify Image for more information.

4.6.2 Creating Images

The HTTP POST method is used to submit **Images**.

Quick Reference

```
POST http://localhost/mws/rest/images
```

4.6.2.1 Create Single Image

URLs and Parameters

```
POST http://localhost/mws/rest/images
```

See Global URL Parameters for available URL parameters.

Request Body

Three fields are required to submit an image: **name**, **hypervisor**, and **osType**. Each image must also support provisioning to either a physical machine or a virtual machine by using the **supportsPhysicalMachine** or **supportsVirtualMachine** fields.

The **name** field must contain only letters, digits, periods, dashes, and underscores.

The array of virtualized images are themselves objects that contain image IDs or names. For more information on available fields and types, see the Image API.

The following is an example of the most basic image that can be created:

POST http://localhost/mws/rest/images

```
{
  "name": "centos5-stateful",
  "osType": "linux",
  "hypervisor": false,
  "supportsVirtualMachine":true
}
```

Note that this example does not provide any information for a provisioning manager (such as xCAT) to actually provision the machine. In order to provide this, you must add an entry to the **extensions** field that contains provisioning manager-specific information. Each key in the extensions field corresponds to the provisioning manager, and certain properties are required based on this key. For example, the xCAT extension key must be named `xcat` and must contain certain fields. These extension keys are documented in the Image API. See the following examples of creating images with xCAT-specific provisioning information below.

Sample Response

If the request was successful, the response body is the new image that was created exactly as shown in Get Single Image. On failure, the response is an error message.

Samples

The **virtualizedImages** field only accepts input when the image is a hypervisor and expects an array of image IDs **or** names, as shown in the following example:

Example payload of hypervisor with 2 vms

```
{
  "hypervisor":true,
  "name":"esx5-stateful",
  "osType":"linux",
  "supportsPhysicalMachine":true,
  "type":"stateful",
  "virtualizedImages":   [
    {"id": "4fa197e68ca30fc605dd1cf0"},
    {"name": "centos5-stateful"}
  ]
}
```

The following example shows how to create an image that utilizes a cloned template for a virtual machine. (Note that the **type** must be set to `linkedclone` in order to set the **templateName** field.)

VM Utilizing a Cloned Template

```
{
  "active": true,
  "hypervisor": false,
  "name": "centos5-compute-stateful",
  "osType": "linux",
  "type": "linkedclone",
  "supportsVirtualMachine":true,
  "templateName":"centos5-compute"
}
```

The following are samples of a virtual machine and a hypervisor image that can be provisioned with xCAT:

xCAT Virtual Machine Image

```
{
  "active": true,
  "features": [],
  "hypervisor": false,
  "name": "centos5-compute-stateful",
  "osType": "linux",
  "type": "stateful",
  "supportsVirtualMachine":true,
  "extensions": {
     "xcat": {
        "os": "centos",
        "architecture": "x86_64",
        "profile": "compute"
     }
  }
}
```

xCAT Hypervisor Image

```
{
  "active": true,
  "features": [],
  "hypervisor": true,
  "name": "esxi5-base-stateless",
  "osType": "linux",
  "virtualizedImages":   [
    {"name": "centos5-compute-stateless"}
  ],
  "type": "stateless",
  "supportsPhysicalMachine":true,
  "extensions": {
     "xcat": {
        "os": "esxi5",
        "architecture": "x86_64",
        "profile": "base",
        "hvType": "esx",
        "hvGroupName": "esx5hv",
        "vmGroupName": "esx5vm"
     }
  }
}
```

4.6.3 Modifying Images

The HTTP PUT method is used to modify **Images**.

Quick Reference

```
PUT http://localhost/mws/rest/images/<id>
PUT http://localhost/mws/rest/images/<name>
```

4.6.3.1 Modify Single Image

URLs and Parameters

```
PUT http://localhost/mws/rest/image/<id>
PUT http://localhost/mws/rest/image/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the Image.
name	Yes	String	-	The name of the Image.

See Global URL Parameters for available URL parameters.

- You must specify either **id** or **name**, but you do not have to specify both.
- The **name** field must contain only letters, digits, periods, dashes, and underscores.

Example Request

PUT http://locahost/mws/rest/image/centos5-stateful

```
{
  "name": "centos5-stateful",
  "type": "stateful",
  "hypervisor": false,
  "osType": "linux",
  "virtualizedImages": []
}
```

The **version** field contains the current version of the database entry and does **not** reflect the version of the operating system. This field cannot be updated directly. However, if **version** is included in the modify request, it will be used to verify that another client did not update the object in between the time the data was retrieved and the modify request was delivered.

Sample Response

If the request was successful, the response body is the modified image as shown in Get Single Image. On failure, the response is an error message.

4.6.4 Deleting Images

The HTTP DELETE method is used to delete **Images**.

Quick Reference

```
DELETE http://localhost/mws/rest/images/<id>
DELETE http://localhost/mws/rest/images/<name>
```

4.6.4.1 Delete Single Image

URLs and Parameters

```
DELETE http://localhost/mws/rest/image/<id>
DELETE http://localhost/mws/rest/image/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the Image.
name	Yes	String	-	The name of the Image.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

JSON Response

```
{}
```

4.7 Jobs

This section describes behavior of the **Job** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Job API contains the type and description of all fields in the **Job** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/jobs	Get all jobs		Submit new job	
/rest/jobs/active	Get all active jobs			
/rest/jobs/complete	Get all complete jobs			
/rest/jobs/**id**	Get specified job	Modify job		Cancel job
/rest/jobs/active/**id**	Get specified active job			
/rest/jobs/complete/**id**	Get specified complete job			

4.7.1 Getting Job Information

The HTTP GET method is used to retrieve **Job** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/jobs/<id>
```

4.7.1.1 Get All Jobs

URLs and Parameters

```
GET http://localhost/mws/rest/jobs
```

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "...",
    …
  }]
}
```

Samples

GET http://localhost/mws/rest/jobs?fields=id,state,flags

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":    [
        {
      "id": "job.1",
      "state": "IDLE",
      "flags": ["PREEMPTABLE"]
    },
        {
      "id": "job.2",
      "state": "RUNNING",
      "flags": []
    },
        {
      "id": "job.3",
      "state": "REMOVED",
      "flags":       [
        "PREEMPTABLE",
        "RESTARTABLE"
      ]
    }
  ]
}
```

Known Issues

- Some jobs are not returned if `DisplayFlags UseBlocking` is set in the `moab.cfg` file.

4.7.1.2 Get All Active Jobs

URLs and Parameters

```
GET http://localhost/mws/rest/jobs/active
```

See Global URL Parameters for available URL parameters.

Sample Response

Same as Get All.

4.7.1.3 Get All Complete Jobs

URLs and Parameters

```
GET http://localhost/mws/rest/jobs/complete
```

See Global URL Parameters for available URL parameters.

Sample Response

Same as Get All.

Known Issues

This query can take a long time and slow down the Moab Workload Manager, especially on systems with many completed jobs. Avoid this query if possible.

4.7.1.4 Get Single Job

URLs and Parameters

```
GET http://localhost/mws/rest/jobs/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "account": "account",
  "activeDuration": 150,
  "allocatedNodes": [{"id": "node01"}],
  "allocatedVMs": [{"id": "vm1"}],
  "blockReason":   {
    "message": "Check valid user",
    "type": "BADUSER"
  },
  "bypass": 5,
  "commandFile": "/tmp/test.sh",
  "commandLineArguments": "-x -v",
  "completionCode": 0,
  "completionDate": "2011-11-08 13:18:47 MST",
  "dedicatedProcessorSeconds": 1.5,
  "destinationRmJobId": "1000011",
  "earliestStartDate": "2011-11-08 13:18:47 MST",
  "earliestStartDateRequested": "2011-11-08 13:18:47 MST",
  "effectivePartitionAccessList": ["ALL"],
  "effectiveQueueDuration": 600,
  "emailNotifyTypes": ["END"],
  "emailNotifyUsers": ["user@domain.com"],
  "environmentVariables": {"var1": "val1"},
  "expectedState": "IDLE",
  "flags": ["RESTARTABLE"],
  "genericAttributes": ["attr1"],
  "group": "group",
  "holds": ["USER"],
  "hosts": ["host1"],
  "id": "Moab.1",
  "initialWorkingDirectory": "/tmp",
  "latestCompletedDateRequested": "2011-11-08 13:18:47 MST",
  "masterHost": "masterHost",
  "memoryRequested": 1024,
  "messages": [  {
    "creationTime": null,
    "expireTime": null,
    "index": 0,
    "message": "Message one",
    "messageCount": 0,
    "author": "moab",
    "priority": 0
  }],
  "name": "myJob",
  "os": "linux",
  "partitionAccessList": ["ALL"],
  "qos": "QOS1",
  "qosRequested": "QOS1",
  "queue": "BATCH",
  "queueStatus": "ACTIVE",
  "durationRequested": 300,
  "requirements": [  {
    "allocatedNodes": [{"id": "node01"}],
    "allocatedPartition": "",
    "genericResources":     {
      "resource1": 10,
      "resource2": 30
    },
    "nodeAccessPolicy": null,
    "preferredNodeFeatures": [],
    "requiredArchitecture": "",
    "requiredClass": "",
    "requiredDiskPerTask": 0,
    "requiredMemoryPerTask": 0,
    "requiredNetwork": "",
    "requiredNodeCountMinimum": 0,
    "requiredNodeDisk": 0,
    "requiredNodeFeatures": [],
    "requiredNodeMemory": 0,
    "requiredNodeProcessors": 0,
    "requiredNodeSwap": 0,
    "requiredPartition": "",
    "requiredProcessorCountMinimum": 4,
    "requiredProcessorsPerTask": 0,
    "requiredSwapPerTask": 0,
    "tasksPerNode": 0
  }],
  "reservationRequested": "rsv.1",
  "reservationStartDate": "2011-11-08 13:18:47 MST",
  "rmExtension": "x=PROC=4",
  "rmName": "torque",
  "rmStandardErrorFilePath": "/tmp/error.out",
  "rmStandardInputFilePath": "/tmp/input.in",
  "rmStandardOutputFilePath": "/tmp/output.out",
  "runPriority": 5,
  "sourceRmJobId": "1000011",
  "standardErrorFilePath": "/tmp/job.error.out",
  "standardOutputFilePath": "/tmp/job.output.out",
  "startCount": 1,
  "startDate": "2011-11-08 13:18:47 MST",
  "startPriority": 2,
  "state": "COMPLETED",
  "submitDate": "2011-11-08 13:18:47 MST",
  "submitHost": "admin-node",
  "suspendDuration": 60,
  "systemPriority": 6,
  "userPriority": 5,
  "user": "saadmin",
  "variables": {"var1": "val1"},
  "virtualContainers": [{"id":"vc1"}],
  "vmUsagePolicy": "CREATEVM"
}
```

4.7.1.5 Get Single Active Job

URLs and Parameters

```
GET http://localhost/mws/rest/jobs/active/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

Same as Get Single.

4.7.1.6 Get Single Active Job

URLs and Parameters

```
GET http://localhost/mws/rest/jobs/complete/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

Same as Get Single.

4.7.2 Submitting Jobs

The HTTP POST method is used to submit **Jobs**.

Quick Reference

```
POST http://localhost/mws/rest/jobs[?proxy-user=<username>]
```

Restrictions

- The user given in `user` must have read access to the file given in `commandFile`.
- No more than one virtual container can be specified in the request. The virtual container must already exist.
- The `user` and `group` properties are used to submit a job as the specified user belonging to the specified group.
- Job `variables` have the following restrictions:
  - `variable` names cannot contain equals (=), semicolon (;), colon (:), plus (+), question mark (?), caret (^), backslash (\), or white space.
  - `variable` values cannot contain semicolon (;), colon (:), plus (+), or caret (^).
- When submitting jobs, the only supported hold type is `USER`.
- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]          USERS=root,ted ENABLEPROXY=TRUE
```

4.7.2.1 Submit Job with Host List

URLs and Parameters

```
POST http://localhost/mws/rest/jobs[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

To submit a job with a specified host list, only two fields are required: `commandFile` and `hosts`.

The request body below shows all the fields that are available during job submission.

JSON Request Body (specified host list)

```
{
  "account": "project name",
  "commandFile": "/tmp/myscript.sh",
  "commandLineArguments": "-x",
  "earliestStartDateRequested": "2011-09-26 16:28:20 MDT",
  "emailNotifyTypes": ["END"],
  "emailNotifyUsers": ["user@domain.com"],
  "environmentRequested": true,
  "environmentVariables":   {
    "SHELL": "/bin/bash",
    "LC_ALL": "en_US.utf8"
  },
  "flags":   [
    "SUSPENDABLE",
    "BESTEFFORT"
  ],
  "group": "wheel",
  "holds": ["USER"],
  "hosts":   [
    "node2",
    "node3"
  ],
  "initialWorkingDirectory": "/tmp",
  "name": "job name",
  "os": "Ubuntu",
  "qosRequested": "highprio",
  "queue": "priority",
  "durationRequested": 3600,
  "requirements": [  {
    "genericResources":     {
      "resource1": 10,
      "resource2": 30
    },
    "nodeAccessPolicy": "SHARED",
    "requiredArchitecture": "x86_64",
    "requiredDiskPerTask": 500,
    "requiredMemoryPerTask": 1024,
    "requiredNodeFeatures": ["bluray"],
    "requiredPartition": "cs",
    "requiredProcessorsPerTask": 3,
    "requiredSwapPerTask": 600,
    "tasksPerNode": 8
  }],
  "reservationRequested": "grid.3",
  "standardErrorFilePath": "/home/jacob/err",
  "standardOutputFilePath": "/home/jacob/out",
  "submitHost": "admin-node",
  "templateList":   [
    "template1",
    "template2"
  ],
  "user": "jacob",
  "userPriority": 25,
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  },
  "virtualContainers": [{"id": "vc1"}],
  "vmUsagePolicy": "REQUIREPM"
}
```

Sample Response

The response of this task is one of three possibilities:

- An object with a single `messages` property containing a list of error messages on failure

```
{"messages":["Could not create job - invalid requirements"]}
```

- An object with an `id` property containing the ID of the newly created job

```
{"id":"Moab.1"}
```

- An object with an `id` property and a `virtualContainers` list containing the ID of the newly created virtual container

```
{"id":"Moab.1","virtualContainers":[{"id":"vc1"}]}
```

⚠ The virtual container will only be reported when a *new* virtual container has been created by Moab for the job.

4.7.2.2 Submit Job with Node Count

URLs and Parameters

```
POST http://localhost/mws/rest/jobs[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

To submit a job with a specified node count, only two fields are required: `commandFile` and `requiredProcessorCountMinimum` (in the `requirements` array).

The request body below shows all the fields that are available during job submission.

JSON Request Body (specified node count)

```
{
  "account": "project name",
  "commandFile": "/tmp/myscript.sh",
  "commandLineArguments": "-x",
  "earliestStartDateRequested": "2011-09-26 16:28:20 MDT",
  "emailNotifyTypes": ["END"],
  "emailNotifyUsers": ["user@domain.com"],
  "environmentRequested": true,
  "environmentVariables":   {
    "SHELL": "/bin/bash",
    "LC_ALL": "en_US.utf8"
  },
  "flags":   [
    "SUSPENDABLE",
    "BESTEFFORT"
  ],
  "group": "wheel",
  "holds": ["USER"],
  "initialWorkingDirectory": "/tmp",
  "name": "job name",
  "os": "Ubuntu",
  "qosRequested": "highprio",
  "queue": "priority",
  "durationRequested": 3600,
  "requirements": [  {
    "genericResources":     {
      "resource1": 10,
      "resource2": 30
    },
    "nodeAccessPolicy": "SHARED",
    "requiredArchitecture": "x86_64",
    "requiredDiskPerTask": 500,
    "requiredMemoryPerTask": 1024,
    "requiredNodeFeatures": ["bluray"],
    "requiredPartition": "cs",
    "requiredProcessorCountMinimum": 4,
    "requiredProcessorsPerTask": 3,
    "requiredSwapPerTask": 600,
    "tasksPerNode": 8
  }],
  "reservationRequested": "grid.3",
  "standardErrorFilePath": "/home/jacob/err",
  "standardOutputFilePath": "/home/jacob/out",
  "submitHost": "admin-node",
  "templateList":   [
    "template1",
    "template2"
  ],
  "user": "jacob",
  "userPriority": 25,
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  },
  "virtualContainers": [{"id": "vc1"}],
  "vmUsagePolicy": "REQUIREPM"
}
```

Sample Response

The response of this task is the same as submitting a job with a host list.

4.7.2.3 Examples of Job Submission

This section includes some sample job submission requests.

Submit job to run on node2 and node3

```
POST http://localhost/mws/rest/jobs
-----------------------------------
{
  "commandFile": "/tmp/test.sh",
  "group": "adaptive",
  "hosts":   [ "node2", "node3" ]
  "initialWorkingDirectory": "/tmp",
  "user": "adaptive",
}
```

Submit job that requires 20 processors

```
POST http://localhost/mws/rest/jobs
-----------------------------------
{
  "commandFile": "/tmp/test.sh",
  "group": "adaptive",
  "initialWorkingDirectory": "/tmp",
  "requirements": [{"requiredProcessorCountMinimum": "20"}]
  "user": "adaptive",
}
```

Submit job to run after a certain time

```
POST http://localhost/mws/rest/jobs
-----------------------------------
{
  "commandFile": "/tmp/test.sh",
  "earliestStartDateRequested": "2012-08-26 16:28:20 MDT",
  "group": "adaptive",
  "initialWorkingDirectory": "/tmp",
  "requirements": [{"requiredProcessorCountMinimum": "20"}]
  "user": "adaptive",
}
```

Submit job based on `msub` example

Given this `msub` command:

```
msub -l nodes=3:ppn=2,walltime=1:00:00,pmem=100 script2.pbs.cmd
```

Here is an equivalent MWS request:

```
POST http://localhost/mws/rest/jobs
-----------------------------------
{
  "user": "adaptive",
  "group": "adaptive",
  "initialWorkingDirectory": "/home/adaptive",
  "commandFile": "/home/adaptive/script2.pbs.cmd",
  "requirements": [  {
    "requiredProcessorCountMinimum": 6,
    "tasksPerNode": 2,
    "requiredMemoryPerTask": 100
  }],
  "durationRequested": 3600
}
```

- To emulate what `msub` does, make `commandFile` an absolute path, and add `user`, `group`, and `initialWorkingDirectory`.
- As shown above, `nodes=3:ppn=2` is equivalent to setting `requiredProcessorCountMinimum` to 6 and `tasksPerNode` to 2.

4.7.3 Modifying Jobs

The HTTP PUT method is used to modify **Jobs**.

Quick Reference

```
PUT http://localhost/mws/rest/jobs/<id>[/<modifyAction>][?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]           USERS=root,ted ENABLEPROXY=TRUE
```

4.7.3.1 Modify Job Attributes

URLs and Parameters

```
PUT http://localhost/mws/rest/jobs/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when modifying a Job, along with some sample values.

JSON Request Body

```
{
  "account": "engineering",
  "earliestStartDateRequested": "2011-08-24 15:02:00",
  "flags":   [
    "RESTARTABLE",
    "SUSPENDABLE"
  ],
  "holds": ["USER"],
  "messages":   [
    {"message": "First message"},
    {"message": "Second message"}
  ],
  "name": "EngineeringJob",
  "qosRequested": "NORMAL",
  "queue": "BATCH",
  "durationRequested": 600,
  "requirements": [{"requiredPartition": "msm"}],
  "reservationRequested": "rsv.1",
  "trigger": "triggerString",
  "userPriority": 10,
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample Response

⚠ These messages may not match the messages returned from Moab exactly, but are given as an example of the structure of the response.

⚠ Not all messages are shown for the above request body.

JSON Response

```
{
  "messages":[
    "Account modified successfully",
    "Messages modified successfully",
    "Variables modified successfully"
  ]
}
```

Restrictions

- Old messages are not removed from jobs; only new messages are added.
- Job `variables` have the restrictions documented in Submitting Jobs

4.7.3.2 Perform Actions on Job

URLs and Parameters

```
PUT http://localhost/mws/rest/jobs/<id>/<modifyAction>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
modifyAction	Yes	String	cancel	Attempts to cancel the job.
			checkpoint	Attempts to checkpoint the job. Note that the OS must support checkpointing for this to work.
			execute	Executes the job if possible.
			hold	Attempts to hold the job using the holds set in the request body.
			requeue	Attempts to requeue the job.
			resume	Attemps to resume the job.
			suspend	Attempts to suspend the job.
			unhold	Attempts to release the holds set in the request body.
proxy-user	No	String	-	Perform the action as this user.

⚠ Performing a cancel function on a job is equivalent to deleting a job.

See Global URL Parameters for available URL parameters.

Request Body

Request bodies are only required for holding or unholding jobs. All other actions do not require request bodies of any kind.

JSON Request Body to Add Holds to a Job

```
{
  "holds": ["USER"]
}
```

JSON Request Body to Remove Holds from a Job

```
{
  "holds": ["USER"]
}
```

⚠ If no holds are specified when unholding a job, all holds will be removed. This is equivalent to specifying `holds` as a list with a single element of `ALL` .

Sample Response

⚠ This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{
  "messages":[
    "Job modified successfully"
  ]
}
```

4.7.4 Deleting (Canceling) Jobs

The HTTP DELETE method is used to cancel **Jobs**.

Quick Reference

```
DELETE http://localhost/mws/rest/jobs/<id>[?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]          USERS=root,ted ENABLEPROXY=TRUE
```

4.7.4.1 Cancel Job

URLs and Parameters

```
DELETE http://localhost/mws/rest/jobs/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response for successful DELETE

```
{}
```

Additional information about the DELETE can be found in the HTTP response header `X-MWS-Message`.

4.8 Job Templates

This section describes behavior of the **Job Template** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

The Job Template API contains the type and description of all fields in the **Job Template** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/job-templates	Get all job templates			
/rest/job-templates/**id**	Get specified job template			

4.8.1 Getting Job Templates

The HTTP GET method is used to retrieve **Job Template** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/job-templates/<id>
```

4.8.1.1 Get All Job Templates

URLs and Parameters

```
GET http://localhost/mws/rest/job-templates
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/job-templates?fields=id

```
{
  "totalCount": 14,
  "resultCount": 14,
  "results":   [
    {"id": "DEFAULT"},
    {"id": "genericVM"},
    {"id": "genericVM-setup"},
    {"id": "genericVM-destroy"},
    {"id": "genericVM-migrate"},
    {"id": "genericPM"},
    {"id": "genericPM-setup"},
    {"id": "genericPM-destroy"},
    {"id": "OSStorage"},
    {"id": "OSStorage-setup"},
    {"id": "OSStorage-destroy"},
    {"id": "extraStorage"},
    {"id": "extraStorage-setup"},
    {"id": "extraStorage-destroy"}
  ]
}
```

4.8.1.2 Get Single Job Template

URLs and Parameters

```
GET http://localhost/mws/rest/job-templates/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "account": "account",
  "args": "arg1 arg2",
  "commandFile": "/tmp/script",
  "description": "description",
  "genericSystemJob": true,
  "id": "genericVM",
  "inheritResources": false,
  "jobDependencies": [  {
    "name": "genericVM-setup",
    "type": "JOBSUCCESSFULCOMPLETE"
  }],
  "jobFlags": ["VMTRACKING"],
  "jobTemplateFlags": ["SELECT"],
  "jobTemplateRequirements": [  {
    "architecture": "x86_64",
    "diskRequirement": 500,
    "genericResources": {"tape": 3},
    "nodeAccessPolicy": "SINGLEJOB",
    "operatingSystem": "Ubuntu 10.04.3",
    "requiredDiskPerTask": 200,
    "requiredFeatures": ["dvd"],
    "requiredMemoryPerTask": 1024,
    "requiredProcessorsPerTask": 2,
    "requiredSwapPerTask": 512,
    "taskCount": 4
  }],
  "priority": 20,
  "qos": "qos",
  "queue": "queue",
  "durationRequested": 600,
  "select": true,
  "trigger": null,
  "version": 0,
  "vmUsagePolicy": "REQUIREPM"
}
```

4.9 Metric Types

This section describes behavior of the **Metric Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

The MetricType API contains the type and description of all fields in the **Metric Type** object.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/metric-types	Get all metric types			

4.9.1 Getting Metric Types

The HTTP GET method is used to retrieve **Metric Type** information.

Quick Reference

```
GET http://localhost/mws/rest/metric-types
```

4.9.1.1 Get All Metric Types

URLs and Parameters

```
GET http://localhost/mws/rest/metric-types
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/metric-types?fields=id

```
{
  "totalCount": 9,
  "resultCount": 9,
  "results":   [
    {"id": "vmcount"},
    {"id": "watts"},
    {"id": "pwatts"},
    {"id": "temp"},
    {"id": "cpu"},
    {"id": "mem"},
    {"id": "io"},
    {"id": "ccores"},
    {"id": "threads"}
  ]
}
```

4.10 Nodes

This section describes behavior of the **Node** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Node API contains the type and description of all fields in the **Node** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/nodes	Get all nodes			
/rest/nodes/**id**	Get specified node	Modify node		

4.10.1 Getting Nodes

The HTTP GET method is used to retrieve **Node** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/nodes/<id>
```

4.10.1.1 Get All Nodes

URLs and Parameters

```
GET http://localhost/mws/rest/nodes
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/nodes?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "node1"},
    {"id": "node2"},
    {"id": "node3"}
  ]
}
```

4.10.1.2 Get Single Node

URLs and Parameters

```
GET http://localhost/mws/rest/nodes/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "accessPolicy": null,
  "aliases": [],
  "architecture": "",
  "availableClasses": [],
  "availableDisk": -1,
  "availableEndDate": null,
  "availableGenericResources": {},
  "availableMemory": -1,
  "availableProcessors": -1,
  "availableStartDate": null,
  "availableSwap": -1,
  "blockReason": "",
  "comments": "",
  "configuredClasses": [],
  "cpuLoad": 0,
  "dynamic": false,
  "externalLoad": 0,
  "features": [],
  "flags":   [
    "VM_CREATE_ENABLED",
    "RM_DETECTED"
  ],
  "genericEvents": [],
  "genericMetrics": {},
  "genericResources": {},
  "hypervisorType": "",
  "iOLoad": 0,
  "id": "",
  "index": -1,
  "jobs": [{"id": "Moab.1"}],
  "lastStateUpdateDate": null,
  "lastUpdateDate": null,
  "maxIOIn": 0,
  "maxIOLoad": 0,
  "maxIOOut": 0,
  "maxJob": 0,
  "maxJobPerUser": 0,
  "maxLoad": 0,
  "maxPEPerJob": 0,
  "maxPageIn": 0,
  "maxPageOut": 0,
  "maxProc": 0,
  "maxProcPerClass": 0,
  "messages": [],
  "network": "",
  "networkAddress": "",
  "networkLoad": 0,
  "nextOS": "",
  "operations": [],
  "os": "",
  "osList": [],
  "overcommit": null,
  "partition": "",
  "power": null,
  "powerPolicy": null,
  "powerSelected": null,
  "priority": 0,
  "priorityFunction": "",
  "procSpeed": 0,
  "profilingEnabled": false,
  "rack": 0,
  "reservationCount": 0,
  "reservations": [],
  "rmAccessList": "",
  "size": 1,
  "slot": 0,
  "speed": 1,
  "speedWeight": 1,
  "state": null,
  "substate": "",
  "taskCount": -1,
  "totalActiveTime": 0,
  "totalAvailableTime": 0,
  "totalDisk": -1,
  "totalMemory": -1,
  "totalProcessors": -1,
  "totalStatsTime": 0,
  "totalSwap": -1,
  "totalUpTime": 0,
  "type": "",
  "variables": {},
  "version": 0,
  "virtualMachines": [{"id": "vm1"}],
  "vmOsList": []
}
```

4.10.2 Modifying Nodes

The HTTP PUT method is used to modify **Nodes**.

Quick Reference

```
PUT http://localhost/mws/rest/nodes/<id>[?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]          USERS=root,ted ENABLEPROXY=TRUE
```

4.10.2.1 Modify Node

URLs and Parameters

```
PUT http://localhost/mws/rest/nodes/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when modifying a Node, along with some sample values.

Sample JSON Request Body to Modify a Node

```
{
  "genericEvents": [  {
    "name": "event1",
    "message": "Sample message"
  }],
  "genericMetrics":   {
    "metric1": 3,
    "metric2": 5
  },
  "messages":   [
    "message1",
    "message2"
  ],
  "os": "linux",
  "partition": "local",
  "power": "off|on",
  "state": "Busy",
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample Response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{"messages":[
  "Successfully modified os to 'linux'",
  "Successfully powered node off"
]}
```

4.11 Pending Actions

This section describes behavior of the **Pending Action** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

The Pending Action API contains the type and description of all fields in the **Pending Action** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/pending-actions	Get all pending actions			

4.11.1 Getting Pending Actions

The HTTP GET method is used to retrieve **Pending Action** information.

Quick Reference

```
GET http://localhost/mws/rest/pending-actions
```

4.11.1.1 Get All Pending Actions

URLs and Parameters

```
GET http://localhost/mws/rest/pending-actions
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/pending-actions

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "failureDetails": "",
    "hosts": ["hv3"],
    "id": "vmcreate-27",
    "maxDurationInSeconds": 3600,
    "migrationDestination": "",
    "migrationSource": "",
    "motivation": "requested by root",
    "pendingActionState": "RUNNING",
    "pendingActionType": "VMCREATE",
    "requester": "root",
    "serviceId": "Rhel55Vm.200",
    "startTime": "2011-11-15 21:57:55 MST",
    "substate": "installing",
    "targetOS": "",
    "topLevelServiceId": "Lamp.132",
    "vmId": "vm8"
  }]
}
```

Generic vs Non-Generic Types

If generic job templates are used in Moab, MWS may be configured to translate pending actions with the generic type to the proper type such as `VMCREATE`. This is done in the configuration file. The Quickstart Guide provides the default mappings for this feature, as well as an example of adding a custom mapping from a custom template name to the correct type.

The default mappings are shown in the table below. The available pending action types may be seen on the PendingActionType API page.

Template Name	Mapped Type
genericVM-setup	VMCREATE
genericVM-migrate	VMMIGRATE
genericVM-destroy	VMDESTROY
OSStorage-setup	VMSTORAGE
OSStorage-destroy	VMSTORAGEDESTROY
extraStorage-setup	STORAGE
extraStorage-destroy	STORAGEDESTROY
genericPM-setup	OSPROVISION

⚠ When generic mappings are used, MWS will match the first template mapping that the pending action ID ends with. For example, an ID of `Moab.1.genericVM-setup` will map the type to `VMCREATE`.

To enable mapping for a custom template name such as `myCustomVM-setup`, simply add the following line to the MWS configuration file. The value of the pending action type is case insensitive.

```
mws.pendingActions.mappings["myCustomVM-setup"] = "vmcreate"
```

MWS also provides the ability to enable or disable the display of generic pending actions (or those pending actions that are not mapped). This behavior is controlled by the `mws.pendingActions.displayGeneric` setting as shown below. A `false` value will prevent generic pending actions from being displayed, while a `true` value will display all pending actions. By default this value is `true`.

```
mws.pendingActions.displayGeneric = false
```

4.12 Plugins

This section describes behavior of the **Plugin** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The PluginInstance API page contains the type and description of all fields in the **Plugin** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/plugins	Get all plugins		Create new plugin	
/rest/plugins/**id**	Get specified plugin	Modify plugin		Delete plugin
/rest/plugins/**id**/poll			Trigger Plugin Poll	
/rest/plugins/**id**/services/**serviceName**	Access Web Service	Access Web Service	Access Web Service	Access Web Service

4.12.1 Getting Plugins

The HTTP GET method is used to retrieve **Plugin** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/plugins
GET http://localhost/mws/rest/plugins/<id>
```

4.12.1.1 Get All Plugins

URLs and Parameters

```
GET http://localhost/mws/rest/plugins
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/plugins?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "plugin1"},
    {"id": "plugin2"},
    {"id": "plugin3"}
  ]
}
```

4.12.1.2 Get Single Plugin

URLs and Parameters

```
GET http://localhost/mws/rest/plugins/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "id":"plugin1",
  "pluginType":"Native",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  },
  "state":"STARTED",
  "nextPollDate":"2011-12-02 17:28:52 MST",
  "lastPollDate":"2011-12-02 17:28:22 MST"
}
```

4.12.2 Creating Plugins

The HTTP POST method is used to create **Plugins**.

Quick Reference

```
POST http://localhost/mws/rest/plugins
```

4.12.2.1 Create Plugin

URLs and Parameters

```
POST http://localhost/mws/rest/plugins
```

See Global URL Parameters for available URL parameters.

Request Body

When creating a plugin, the `id` and `pluginType` fields are required. The request body below shows all fields that are available when creating a Plugin, along with some sample values.

JSON Request Body

```
{
  "id":"plugin1",
  "pluginType":"Native",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  }
}
```

Sample Response

JSON Response for successful POST

```
{"id": "plugin1"}
```

Restrictions

While it is *possible* to create a plugin with arbitrary nested configuration, such as:

```
…
"config":{
  "nestedObject":{
    "property1":"value1",
    "property2":"value2"
  },
  "nestedList:["listItem1", "listItem2"]
}
```

It is **not** recommended as the user interface does not support editing or viewing any configuration data values other than strings.

4.12.3 Modifying Plugins

The HTTP PUT method is used to modify **Plugins**. Additionally, the POST method may be used to trigger an immediate poll of a **Plugin**.

Quick Reference

```
PUT http://localhost/mws/rest/plugins/<id>
POST http://localhost/mws/rest/plugins/<id>/poll
```

4.12.3.1 Modify Plugin

URLs and Parameters

```
PUT http://localhost/mws/rest/plugins/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when modifying a Plugin, along with some sample values.

JSON Request Body for Plugin Modification

```
{
  "state":"STARTED",
  "pollInterval":30,
  "autoStart":true,
  "config":{
    "getJobs":"exec:///opt/moab/tools/workload.query.pl"
  },
  "state":"STARTED"
}
```

Sample Response

JSON Response

```
{"messages":["Plugin plugin1 updated", "Started Plugin 'plugin1'"]}
```

4.12.3.2 Trigger Plugin Poll

URLs and Parameters

```
POST http://localhost/mws/rest/plugins/<id>/poll
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Trigger Poll

This resource call will trigger an immediate poll of the specified plugin. It is equivalent to the same operation on the Monitoring and Lifecycle Controls page.

Request Body

No request body is required.

Sample Response

JSON Response

```
{"messages":["Polled Plugin with ID 'myPlugin'"]}
```

4.12.4 Deleting Plugins

The HTTP DELETE method is used to delete **Plugins**.

Quick Reference

```
DELETE http://localhost/mws/rest/plugins/<id>
```

4.12.4.1 Delete Plugin

URLs and Parameters

```
DELETE http://localhost/mws/rest/plugins/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response for successful DELETE

```
{}
```

⚠ Additional information about a successful DELETE can be found in the HTTP response header `X-MWS-Message`.

JSON Response for an unsuccessful DELETE

```
{"messages":["Plugin plugin1 could not be deleted", "Error message describing the
problem"]}
```

4.12.5 Accessing Plugin Web Services

All HTTP methods can be used to access **Plugin Web Services**. However, some services only support specific methods. Check the specific plugin type documentation for more information.

Quick Reference

```
GET http://localhost/mws/rest/plugins/<id>/services/<serviceName>
POST http://localhost/mws/rest/plugins/<id>/services/<serviceName>
PUT http://localhost/mws/rest/plugins/<id>/services/<serviceName>
DELETE http://localhost/mws/rest/plugins/<id>/services/<serviceName>
```

4.12.5.1 Access a Plugin Web Service

URLs and Parameters

```
GET http://localhost/mws/rest/plugins/<id>/services/<serviceName>
POST http://localhost/mws/rest/plugins/<id>/services/<serviceName>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
serviceName	Yes	String	-	The name of the web service, either in CamelCase or hyphenated.

See Global URL Parameters for available URL parameters.

Web Service IDs

Translation is done to map CamelCase web service names to hyphenated names in the URL. For example, a web service method named `notifyEvent` on a plugin with a name of `notifications` may be called with the following URLs:

```
// CamelCase
/rest/plugins/notifications/services/notifyEvent

// Hyphenated
/rest/plugins/notifications/services/notify-event
```

HTTP Method and Request Body

Because plugin Custom Web Services do not need to distinguish which HTTP method is used, it is recommended to use GET and POST when making requests to access web services unless documented otherwise. The request body and output may vary for each web service called. See the plugin type documentation for the requested plugin for available web services, request parameters, and expected output.

4.13 Plugin Types

This section describes behavior of the **Plugin Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The PluginType API page contains the type and description of all fields in the **Plugin Type** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/plugin-types	Get all plugin types	Create or update plugin type		
/rest/plugin-types/**id**	Get specified plugin type			

4.13.1 Getting Plugin Types

The HTTP GET method is used to retrieve **Plugin Type** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/plugin-types/<id>
```

4.13.1.1 Get All Plugin Types

URLs and Parameters

```
GET http://localhost/mws/rest/plugin-types
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/plugin-types?fields=id

```
{
  "totalCount": 2,
  "resultCount": 2,
  "results":   [
    {"id": "MSM"},
    {"id": "Native"}
  ]
}
```

4.13.1.2 Get Single Plugin Type

URLs and Parameters

```
GET http://localhost/mws/rest/plugin-types/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "id":"Native",
  "author":"Adaptive Computing",
  "description":"Basic implementation of a native plugin",
  "instances":[
    {"id":"plugin1"}
  ]
}
```

4.13.2 Creating or Updating Plugin Types

The HTTP POST method is used to create or update **Plugin Types**. The `Content-Type` HTTP header is used to determine if the request contains a single class file as plaintext or the binary data of a JAR file. Each request is explained in the following sections.

Quick Reference

```
PUT http://localhost/mws/rest/plugin-types[?reload-plugins=false]
```

⚠ There is a known issue with dynamically updating plugin types with typed field injection. See the Add or Update Plugin Types section for more information.

4.13.2.1 Update Plugin Type (File)

URLs and Parameters

```
PUT http://localhost/mws/rest/plugin-types[?reload-plugins=false]
```

Parameter	Required	Valid Values	Description
reload-plugins	No	true or false	Reloads all plugins of this type on successful update. Defaults to true.

See Global URL Parameters for available URL parameters.

Request Body

This function is idempotent, meaning it will create the Plugin Type if it does not exist or update it if it does. The request body is the actual contents of the class file to upload. This web service is an exception to most as it **requires** a content type of `application/x-groovy` or `text/plain`.

> If the `application/x-groovy` or `text/plain` content types are not used in the request, it will be interpreted as JSON, resulting in a failure.

Plaintext upload

```
package test

import com.ace.mws.plugins.*
import com.ace.mws.plugins.exceptions.*

class UploadPlugin {
    static author = "Adaptive Computing"
    static description = "A sample plugin class"
    String id

public void configure() throws InvalidPluginConfigurationException {
        def myConfig = config
        def errors = []
        if (!myConfig.arbitraryKey)
            errors << "Missing arbitraryKey!"
        if (errors)
            throw new InvalidPluginConfigurationException(errors)
    }

public def customService(Map params) {
        return params
    }
}
```

> If using the curl library to perform plugin type uploading, the equivalent of the command-line option `--data-binary` must be used to send the request body. Otherwise compilation errors may be encountered when uploading the plugin type.

Sample Response

The response of this task is the same as the Get All Plugin Types task. The reason that the return of this task is a list is to accommodate the possibility of uploading multiple plugin types in a single JAR file as explained in the next section.

4.13.2.2 Update Plugin Type (JAR)

URLs and Parameters

```
PUT http://localhost/mws/rest/plugin-types?jar-filename=<filename.jar>
[&reload-plugins=false]
```

Parameter	Required	Valid Values	Description
jar-filename	Yes	String	The filename of the JAR file that is being uploaded.
reload-plugins	No	true or false	Reloads all plugins of this type on successful update. Defaults to true.

See Global URL Parameters for available URL parameters.

Request Body

This function is idempotent, meaning it will create the Plugin Types if they do not exist or update them if they do. The request body is the binary contents of the JAR file to upload. This web service is an exception to most as it **requires** a content type of `application/x-jar`.

If the `application/x-jar` content type is not used in the request, it will be interpreted as JSON, resulting in a failure.

If using the curl library to perform plugin type uploading, the equivalent of the command-line option `--data-binary` must be used to send the request body. Otherwise compilation errors may be encountered when uploading the plugin type.

Sample Response

The response of this task is the same as the Get All Plugin Types task. Note that when using a JAR file, multiple plugin types may be uploaded in the same request.

4.14 Policies

This section describes behavior of the **Policies** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

The Policy API contains the type and description of fields that all **Policies** have in common.

Supported Policies

Name	Id
Auto VM Migration	auto-vm-migration
Hypervisor Allocation Overcommit	hv-allocation-overcommit
Node Allocation	node-allocation
Migration Exclusion List	migration-exclusion-list

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/policies	Get all policies			
/rest/policies/**id**	Get specified policy	Modify specified policy		

4.14.1 Getting Policies

The HTTP GET method is used to retrieve **Policies** information.

Quick Reference

```
GET http://localhost/mws/rest/policies
```

4.14.1.1 Get All Policies

URLs and Parameters

```
GET http://localhost/mws/rest/policies
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Query for specific results.	query={"state":"DISABLED","conflicted":"false"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"id":-1}

It is possible to query policies by one or more fields based on MongoDB query syntax.

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/policies?fields=id,state,conflicted

```
{
  "totalCount": 2,
  "resultCount": 2,
  "results": [  {
    "conflicted": false,
    "state": "DISABLED",
    "id": "auto-vm-migration"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "hv-allocation-overcommit"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "node-allocation"
  },{
    "conflicted": false,
    "state": "DISABLED",
    "id": "migration-exclusion-list"
  }]
}
```

4.14.1.2 Get Single Policy

URLs and Parameters

```
GET http://localhost/mws/rest/policies/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Responses

Auto VM Migration

```
{
  "conflicted": false,
  "description": "Controls how virtual machines are automatically migrated.",
  "id": "auto-vm-migration",
  "name": "Auto VM Migration",
  "potentialConflicts": [],
  "priority": 1,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "genericMetricThresholds":{
      "GMETRIC1":1.3
  },
  "processorUtilizationThreshold":0.5,
  "memoryUtilizationThreshold":0.4
}
```

Hypervisor Allocation Overcommit

```
{
  "conflicted": false,
  "description": "Controls how hypervisors are overallocated with regards to processors
and memory.",
  "id": "hv-allocation-overcommit",
  "name": "Hypervisor Allocation Overcommit",
  "potentialConflicts": [],
  "priority": 2,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "processorAllocationLimit":29.5,
  "memoryAllocationLimit":1.2
}
```

Node Allocation

```
{
  "conflicted": false,
  "description": "Controls how nodes are selected for workload placement.",
  "id": "node-allocation",
  "name": "Node Allocation",
  "potentialConflicts": [],
  "priority": 3,
  "state": "DISABLED",
  "tags": [],
  "types": [],
  "version": 0,
  "nodeAllocationAlgorithm": "CustomPriority",
  "customPriorityFunction": "100*RSVAFFINITY - GMETRIC[numvms]"
}
```

Migration Exclusion List

```
{
    "conflicted": false,
    "description": "Controls which machines are excluded from automatic live migration
operations.",
    "hvExclusionList": ["blade05", "blade02"],
    "name": "Migration Exclusion List",
    "potentialConflicts": [],
    "priority": 100,
    "state": "DISABLED",
    "tags": [],
    "types": [],
    "version": 1,
    "vmExclusionList": ["vm1", "vm5"],
    "id": "migration-exclusion-list"
}
```

4.14.2 Modifying Policies

The HTTP PUT method is used to modify **Policies**.

Quick Reference

```
PUT http://localhost/mws/rest/policies/<id>
```

4.14.2.1 Modify Policy

URLs and Parameters

```
PUT http://localhost/mws/rest/policies/<id>[?change-mode=set]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Additional URL parameters

URL parameters for modifying a Migration Exclusion Lists Policy.

Migration Exclusion Lists Parameter	Required	Valid Values	Description
change-mode	No	set(Default)	Replace the existing exclusion list(s) with the given one.
		add	Add the given VMs/HVs to the existing exclusion list(s).
		remove	Remove the given VMs/HVs from the existing exclusion list(s).

Request Body

In general, the fields shown in the Policy API are **not** available for modification. However, the `state` field may be modified to a valid Policy State. All other fields listed in the specific API pages may be modified unless documented otherwise.

The request body below shows all the fields that are available when modifying a Auto VM Migration Policy, along with some sample values.

JSON Request Body for Auto VM Migration Policy

```
{
    "genericMetricThresholds": {
        "GENERICTHRESHOLD": 5
    },
    "memoryUtilizationThreshold": 0.5,
    "processorUtilizationThreshold": 0.4
}
```

The request body below shows all the fields that are available when modifying a Node Allocation Policy, along with some sample values.

JSON Request Body for Auto VM Migration Policy

```
{
      "nodeAllocationAlgorithm" : "CustomPriority",
      "customPriorityFunction" : "100*RSVAFFINITY - GMETRIC[numvms]"
}
```

The request body below shows all the fields that are available when modifying a Migration Exclusion Lists Policy, along with some sample values.

JSON Request Body for Migration Exclusion Lists Policy

```
{
   "vmExclusionList" : ["vm1","vm3","vm5"],
   "hvExclusionList" : ["hv2","hv3","hv6"]
}
```

Sample Response

JSON Response

```
{
    "messages": ["Policy auto-vm-migration updated"]
}
```

Samples

Enable the Auto VM Migration Policy and set values.

PUT http://localhost/mws/rest/policies/auto-vm-migration

```
{
  "state": "enabled",
  "migrationAlgorithmType": "overcommit",
  "processorUtilizationThreshold": 0.5,
  "memoryUtilizationThreshold": 0.4
}
```

⚠ As noted in the Auto VM Migration API documentation, if the `state` is set to ENABLED, then the `migrationAlgorithmType` must **not** be set to NONE.

Restrictions

All Policies

- Fields cannot be modified while the policy is disabled. Enable the policy to modify the field.

Auto VM Migration

- Arbitrary metrics can be added to **genericMetricThresholds**, but they cannot be removed once added.
- The **migrationAlgorithmType** field cannot be modified while the policy is disabled. Enable the policy to modify the field.
- Moab is configured with a default limit of 10 generic metrics. If this limit is reached, such as when arbitrary metrics are added to **genericMetricThresholds**, the metric will not be reported.
  - To increase this limit, set the `MAXGMETRIC` property in the Moab configuration file.

4.15 Quotes

This section describes behavior of the **Quotes** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Quote API contains the type and description of fields that all **Quotes** have in common.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/quotes/standard/quote			Quoting resources	

4.15.1 Quoting Resources

Quick Reference

```
POST http://localhost/mws/rest/quotes/standard/quote?object-type=<OBJECTTYPE>
&proxy-user=<USER>&charge-duration=<CHARGEDURATION>
```

4.15.1.1 Quote Single Job or Service

URLs and Parameters

```
POST http://localhost/mws/rest/quotes/standard/quote?object-type=<OBJECTTYPE>
&proxy-user=<USER>&charge-duration=<CHARGEDURATION>
```

Parameter	Required	Type	Valid Values	Description
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy

charge-duration	Yes	Integer	The charge duration of the job in seconds.	charge-duration=6400
object-type	Yes	String	The object to quote. It can be job or service.	object-type=job
itemize	No	Boolean	Returns the composite charge information in the response data.	itemize=true
rate	No	JSONArray	Uses the specified charge rates in the quote. The specified rates override the standard and quote rates. If the guarantee field is set to true, these charge rates will be saved and used when this quote is referenced in a charge action.	rate=[{"type":"VBR","name":"Memory","rate":1
guarantee	No	Boolean	Guarantees the quote and returns a quote id to secure the current charge rates. This results in the creation of a quote record and a permanent usage record.	

grace-duration	No	Integer	The guaranteed quote grace period in seconds. If the quote duration is specified but not the quote end time, the quote endtime will be calculated as the quote start time plus the quote duration plus the grace duration.	grace-duration=6400
description	No	String	The guaranteed quote description.	description="ABC Coupon Rate"
start-time	No	Date	The guaranteed quote start time in the format yyyy-MM-dd HH:mm:ss z, -Infinity, Infinity, or Now.	start-time="2012-04-09 13:49:40 MDT"
end-time	No	Date	The guaranteed quote end time in the format yyyy-MM-dd HH:mm:ss z, -Infinity, Infinity, or Now.	end-time="2012-04-09 14:49:40 MDT"

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all of the fields in a Job that could affect the quote.

POST http://localhost/mws/rest/quotes/standard/quote?object-type=job

```
{
  "id": "Moab.1"
  "user": "amy",
  "group": "group",
  "rmName": "machine1",
  "templateList": [
  "genericVm"
  ],
  "account": "biology",
  "qosRequested": "QOS1",
  "variables": {
    "imageName": "centos5.5-stateless",
    "topLevelServiceId": "myService.1",
    "serviceId": "vmService.1",
    "vmid": "VmService.1",
    "pmid": "VmService.1"
  },
  "requirements": [
  {
      "requiredProcessorsPerTask": 2,
      "genericResources": {
           "gold": 100,
           "os": 500
      },
      "requiredNodeCountMinimum": 1,
      "requiredMemoryPerTask": 1024,
      "requiredClass": "batch"
  }
  ]
}
```

The request body below shows all of the fields in a Service that affect the quote in a default MAM installation.

POST http://localhost/mws/rest/quotes/standard/quote?object-type=service

```
{
     "name":"service.1",
     "user": "amy",
     "account": "chemistry"
     "attributes":{
        "moab":{
           "job":{
              "resources":{
                 "procs":1,
                 "mem":2048,
                 "OS":500,
                 "gold":100
              },
              "variables":{
                 "Var1": 1524
              },
              "image":"centos5.5-stateless",
              "template":"genericVM",
           }
        }
     }
  }
```

Sample Responses

If the quote is not guaranteed

JSON Response

```
{
    "instance": "Moab.1",
    "amount": 600
}
```

If the quote is guaranteed

JSON Response

```
{
    "id": 1,
    "usageRecord": 2,
    "instance": "Moab.1",
    "amount": 600
}
```

If the quote is guaranteed and itemized

JSON Response

```
{
  "details":   [
        {
      "name": "Processors",
      "value": "2",
      "duration": 300,
      "rate": 1,
      "scalingFactor": 1,
      "amount": 600,
      "details": "2 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 300 [Duration]"
    },
        {
      "name": "Memory",
      "value": "1024",
      "duration": 300,
      "rate": 1,
      "scalingFactor": 1,
      "amount": 307200,
      "details": "1024 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 300 [Duration]"
    }
  ],
  "id": 20,
  "instance": "Moab.1",
  "usageRecord": 20,
  "amount": 307800
}
```

If the quote is on a service

JSON Response

```
{
  "services":   [
        {
      "details":       [
                {
          "name": "Processors",
          "value": "22",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 660,
          "details": "22 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 30
[Duration]"
        },
                {
          "name": "Memory",
          "value": "32343242",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 970297260,
          "details": "32343242 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 30 [Duration]"
        }
      ],
      "id": 120,
      "instance": "myVmWorkflow",
      "usageRecord": 157,
      "amount": 970297920
    },
        {
      "details": [      {
        "name": "Storage",
        "value": "2500",
        "duration": 30,
        "rate": 1.157E-7,
        "scalingFactor": 1,
        "amount": 0,
        "details": "2500 [Storage] * 1.157e-07 [ChargeRate{VBR}{Storage}] * 30
[Duration]"
      }],
      "id": 122,
      "instance": "myExtraStorageWorkflow",
      "usageRecord": 159,
      "amount": 0
    },
        {
      "details":       [
                {
          "name": "Processors",
          "value": "0",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 0,
          "details": "0 [Processors] * 1 [ChargeRate{VBR}{Processors}] * 30 [Duration]"
        },
                {
          "name": "Memory",
          "value": "0",
          "duration": 30,
          "rate": 1,
          "scalingFactor": 1,
          "amount": 0,
          "details": "0 [Memory] * 1 [ChargeRate{VBR}{Memory}] * 30 [Duration]"
        }
      ],
      "id": 123,
      "instance": "myPmWorkflow",
      "usageRecord": 160,
      "amount": 0
    }
  ],
  "amount": 970297920
}
```

Restrictions

- The details field is only available with MAM version 7.1.0 or later.

4.16 Reports

This section describes behavior of the reporting framework in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Report, Sample, and Datapoint API contains the type and description of all fields in the Report, Sample, and Datapoint objects. They also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/reports	Get all reports		Create Reports	Deleting Reports
/rest/reports/**name**	Get single report with data			
/rest/reports/**id**	Get single report with data			
/rest/reports/**name** /datapoints	Get datapoints for report			
/rest/reports/**id** /datapoints	Get datapoints for report			
/rest/reports/**name** /samples	Get samples for report		Create sample(s) for report	
/rest/reports/**id**/samples	Get samples for report		Create sample(s) for report	

4.16.1 Getting Reports

The HTTP GET method is used to retrieve **Report** information. Queries for all reports with no attached data and a single report with associated data are available.

Quick Reference

```
GET http://localhost/mws/rest/reports[?query={"field":"value"}&sort={"field":<1|-1>}]
GET http://localhost/mws/rest/reports/<id>
GET http://localhost/mws/rest/reports/<name>
```

4.16.1.1 Get All Reports (No Data Included)

URLs and Parameters

```
GET http://localhost/mws/rest/reports[?query={"field":"value"}&sort={"field":<1|-1>}]
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Queries for specific results.	query={"reportSize":4}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"name":-1}

It is possible to query reports by one or more fields based on MongoDB query syntax.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "3efe5c670be86ba8560397ff",
    "name": "cpu-util"
    …
  }]
}
```

⚠ No datapoints are returned when querying for all reports. To view the consolidated datapoints, the Get Single Report API call must be used.

Samples

GET http://localhost/mws/rest/reports?fields=id,name

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
        {
      "id": "3efe5c670be86ba8560397ff",
      "name": "cpu-util"
    },
        {
      "id": "3efe5c670be86ba856039800",
      "name": "cpu-temp"
    },
        {
      "id": "3efe5c670be86ba856039801",
      "name": "cpu-load"
    }
  ]
}
```

4.16.1.2 Get Single Report (Includes Data)

URLs and Parameters

```
GET http://localhost/mws/rest/reports/<id>
GET http://localhost/mws/rest/reports/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the report.
name	Yes	String	-	The name of the report.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

In the example below, the first datapoint has a `null` data element, which means that the `minimumSampleSize` configured for the report was not met when consolidating the datapoint. The second datapoint contains actual data.

JSON Response

```
{
  "consolidationFunction": "average",
  "datapointDuration": 15,
  "datapoints":   [
        {
      "endDate": "2011-12-02 17:28:22 MST",
      "startDate": "2011-12-02 17:28:22 MST",
      "firstSampleDate": null,
      "lastSampleDate": null,
      "data": null
    },
        {
      "endDate": "2011-12-02 17:28:23 MST",
      "startDate": "2011-12-02 17:28:37 MST",
      "firstSampleDate": "2011-12-02 17:28:23 MST",
      "lastSampleDate": "2011-12-02 17:28:30 MST",
      "data":        {
        "utilization": 99.89,
        "time": 27.433333333333337
      }
    }
  ],
  "description": "Example of CPU utilization reporting",
  "id": "3efe5c670be86ba8560397ff",
  "keepSamples": false,
  "minimumSampleSize": 1,
  "name": "cpu-util",
  "reportSize": 2
}
```

4.16.1.3 Get Datapoints For Single Report

URLs and Parameters

```
GET http://localhost/mws/rest/reports/<id>/datapoints[?query={"field":"value"}&sort={
"field":<1|-1>}]
GET http://localhost/mws/rest/reports/<name>/datapoints[?query={"field":"value"}&sort={
"field":<1|-1>}]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the report.
name	Yes	String	-	The name of the report.
query	No	JSON	Queries for specific results.	query={"data.test":true}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"startDate":-1}

It is possible to query datapoints by one or more fields based on MongoDB query syntax.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

This function is exactly the same as requesting a single report with only the datapoints returned. No report metadata (i.e. description, minimumSampleSize, etc.) is returned.

JSON Response

```
{
  "resultCount":1,
  "totalCount":1,
  "results":[
        {
      "endDate": "2011-12-02 17:28:22 MST",
      "startDate": "2011-12-02 17:28:22 MST",
      "firstSampleDate": null,
      "lastSampleDate": null,
      "data": null
    },
        {
      "endDate": "2011-12-02 17:28:37 MST",
      "startDate": "2011-12-02 17:28:37 MST",
      "firstSampleDate": "2011-12-02 17:28:23 MST",
      "lastSampleDate": "2011-12-02 17:28:23 MST",
      "data":        {
        "utilization": 99.89,
        "time": 27.433333333333337
      }
    }
  ]
}
```

4.16.2 Getting Samples For Reports

The HTTP GET method is used to retrieve **Sample** information.

Quick Reference

```
GET http://localhost/mws/rest/reports/<id>/samples[?query={"field":"value"}&sort={
"field":<1|-1>}]
GET http://localhost/mws/rest/reports/<name>/samples[?query={"field":"value"}&sort={
"field":<1|-1>}]
```

4.16.2.1 Get Samples For Report

URLs and Parameters

```
GET http://localhost/mws/rest/reports/<id>/samples[?query={"field":"value"}&sort={
"field":<1|-1>}]
GET http://localhost/mws/rest/reports/<name>/samples[?query={"field":"value"}&sort={
"field":<1|-1>}]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the report.
name	Yes	String	-	The name of the report.
query	No	JSON	Queries for specific results.	query={"agent":"cpu-monitor"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"agent":-1}

It is possible to query samples by one or more fields based on MongoDB query syntax.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

JSON Response

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "timestamp": "2011-12-02 17:28:37 MST"
    "data":{
        "cpu1":2.3,
        "cpu2":1.2,
        "cpu3":0.0,
        "cpu4":12.1
    },
    …
  }]
}
```

4.16.3 Creating Reports

The HTTP POST method is used to create **Reports**. Operations are available to create reports with or without historical datapoints.

Quick Reference

```
POST http://localhost/mws/rest/reports
```

4.16.3.1 Create Report

URLs and Parameters

```
POST http://localhost/mws/rest/reports
```

See Global URL Parameters for available URL parameters.

Request Body

To create a report, several fields are required as documented in the Report API.

The request body below shows all the fields that are available during report creation.

JSON Request Body

```
{
    "name":"cpu-util",
    "description":"An example report on cpu utilization",
    "consolidationFunction":"average",
    "datapointDuration":15,
    "minimumSampleSize":1,
    "reportSize":2,
    "keepSamples":true,
    "datapoints":[
        {
            "startDate":"2011-12-01 19:16:57 MST",
            "endDate":"2011-12-01 19:16:57 MST",
            "data":{
                "time":30,
                "util":99.98
            }
        }
    ]
}
```

Sample Response

```
{
    "messages":["Report cpu-util created"],
    "id":"3efe5c670be86ba8560397ff",
    "name":"cpu-util"
}
```

Samples

POST http://localhost/mws/rest/reports (Minimal report without datapoints)

```
{
    "name":"cpu-util",
    "datapointDuration":15,
    "reportSize":2
}
```

4.16.4 Creating Samples

The HTTP POST method is used to create **Samples** for Reports.

Quick Reference

```
POST http://localhost/mws/rest/reports
```

4.16.4.1 Create Samples For Report

URLs and Parameters

```
GET http://localhost/mws/rest/reports/<id>/samples
GET http://localhost/mws/rest/reports/<name>/samples
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the report.
name	Yes	String	-	The name of the report.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Request Body

To create samples for a report, simply send data and an optional timestamp to the URL above.

The request body below shows all the fields that are available during sample creation. Note that the `data` field can contain arbitrary JSON.

JSON Request Body

```
{
    "timestamp":"2011-12-01 19:16:57 MST",
    "agent":"my agent",
    "data":{
        "cpu1":2.3,
        "cpu2":1.2,
        "cpu3":0.0,
        "cpu4":12.1
    }
}
```

Sample Response

```
{"messages":["1 sample(s) created for report cpu-util"]}
```

4.16.5 Deleting Reports

The HTTP DELETE method is used to delete **Reports**.

Quick Reference

```
DELETE http://localhost/mws/rest/reports/<id>
DELETE http://localhost/mws/rest/reports/<name>
```

4.16.5.1 Delete Report

URLs and Parameters

```
DELETE http://localhost/mws/rest/reports/<id>
DELETE http://localhost/mws/rest/reports/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the report.
name	Yes	String	-	The name of the report.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

JSON Response

```
{"messages":["Report cpu-util deleted"]}
```

4.17 Reservations

This section describes behavior of the **Reservation** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Reservation API contains the type and description of all fields in the **Reservation** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/reservations	Get all reservations		Create reservation	
/rest/reservations/**id**	Get specified reservation	Modify reservation		Release reservation

4.17.1 Getting Reservations

The HTTP GET method is used to retrieve **Reservation** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/reservations/<id>
```

Restrictions

- Only admin or user reservations are returned with this call.

4.17.1.1 Get All Reservations

URLs and Parameters

```
GET http://localhost/mws/rest/reservations
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/reservations?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "system.1"},
    {"id": "system.2"},
    {"id": "system.3"}
  ]
}
```

4.17.1.2 Get Single Reservation

URLs and Parameters

```
GET http://localhost/mws/rest/reservations/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "accountingAccount": "",
  "accountingGroup": "",
  "accountingQOS": "",
  "accountingUser": "root",
  "aclRules": [  {
    "affinity": "NEUTRAL",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "RESERVATION_ID",
    "value": "system.43"
  }],
  "allocatedNodeCount": 1,
  "allocatedProcessorCount": 8,
  "allocatedTaskCount": 1,
  "allocatedNodes": [
      {"id":"node001"}
  ],
  "comments": "",
  "creationDate": null,
  "duration": 200000000,
  "endDate": "2018-03-17 16:49:10 MDT",
  "excludeJobs":   [
    "job1",
    "job2"
  ],
  "expireDate": null,
  "flags":   [
    "REQFULL",
    "ISACTIVE",
    "ISCLOSED"
  ],
  "globalId": "",
  "hostListExpression": "",
  "id": "system.43",
  "idPrefix": "",
  "isActive": true,
  "isTracked": false,
  "label": "",
  "maxTasks": 0,
  "messages": [],
  "owner":   {
    "name": "adaptive",
    "type": "USER"
  },
  "partitionId": "switchB",
  "profile": "",
  "requirements":   {
    "architecture": "",
    "featureList":     [
      "feature1",
      "feature2"
    ],
    "featureMode": "",
    "memory": 0,
    "nodeCount": 0,
    "nodeIds": ["node001:1"],
    "os": "",
    "taskCount": 1
  },
  "reservationGroup": "",
  "resources": {"PROCS": 0},
  "startDate": "2011-11-14 20:15:50 MST",
  "statistics":   {
    "caps": 0,
    "cips": 2659.52,
    "taps": 0,
    "tips": 0
  },
  "subType": "Other",
  "taskCount": 0,
  "trigger": null,
  "triggerIds": [],
  "uniqueIndex": "",
  "variables": {}
}
```

4.17.2 Creating Reservations

The HTTP POST method is used to create **Reservations**.

Quick Reference

```
POST http://localhost/mws/rest/reservations
```

4.17.2.1 Create Reservation

URLs and Parameters

```
POST http://localhost/mws/rest/reservations
```

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when creating a Reservation, along with some sample values.

JSON Request Body

```
{
  "accountingAccount": "",
  "accountingGroup": "",
  "accountingQOS": "",
  "accountingUser": "root",
  "aclRules": [  {
    "affinity": "POSITIVE",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "GROUP",
    "value": "staff"
  }],
  "comments": "",
  "duration": 200000000,
  "endDate": "2018-03-17 16:49:10 MDT",
  "excludeJobs":   [
    "job1",
    "job2"
  ],
  "flags":   [
    "SPACEFLEX",
    "ACLOVERLAP",
    "SINGLEUSE"
  ],
  "hostListExpression": "",
  "idPrefix": "",
  "label": "myreservation",
  "owner":   {
    "name": "adaptive",
    "type": "USER"
  },
  "partitionId": "",
  "profile": "",
  "requirements":   {
    "architecture": "",
    "featureList":    [
      "feature1",
      "feature2"
    ],
    "memory": 0,
    "os": "",
    "taskCount": 1
  },
  "reservationGroup": "",
  "resources":   {
    "PROCS": 2,
    "MEM": 1024,
    "DISK": 1024,
    "SWAP": 1024,
    "other1": 17,
    "other2": 42
  },
  "startDate": "2011-11-14 20:15:50 MST",
  "subType": "Other",
  "trigger": {
    "eventType":"START",
    "actionType":"EXEC",
    "action":"date"
  },
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Create reservation if no conflicting reservations are found.

This is equivalent to mrsvctl -c -h node01 -E.

JSON Request Body

```
{
  "flags":   [
    "DEDICATEDRESOURCE"
  ],
  "hostListExpression": "node01"
}
```

Sample Response

JSON Response for successful POST

```
{"id": "system.44"}
```

4.17.3 Modifying Reservations

The HTTP PUT method is used to modify **Reservations**.

Quick Reference

```
PUT http://localhost/mws/rest/reservations/<id>?change-mode=<add|remove|set>
```

4.17.3.1 Modify Reservation

URLs and Parameters

```
PUT http://localhost/mws/rest/reservations/<id>?change-mode=<add|remove|set>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
change-mode	Yes	String	add	Add the given variables to the variables that already exist.
			remove	Delete the given variables from the variables that already exist.
			set	Replace all existing variables with the given variables.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when modifying a Reservation, along with some sample values.

JSON Request Body for Reservation Modify

```
{
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample Response

⚠ This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{"messages":["reservation 'system.43' attribute 'Variable' changed."]}
```

Restrictions

- You can change the ACL Rules on a reservation, but not using this resource. See Create or Update ACLs.

4.17.4 Releasing Reservations

The HTTP DELETE method is used to release **Reservations**.

Quick Reference

```
DELETE http://localhost/mws/rest/reservations/<id>
```

4.17.4.1 Release Reservation

URLs and Parameters

```
DELETE http://localhost/mws/rest/reservations/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response for successful DELETE

```
{}
```

4.18 Resource Types

This section describes behavior of the **Resource Type** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The ResourceType API contains the type and description of all fields in the **Resource Type** object.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/resource-types	Get all resource types			

4.18.1 Getting Resource Types

The HTTP GET method is used to retrieve **Resource Type** information.

Quick Reference

```
GET http://localhost/mws/rest/resource-types
```

4.18.1.1 Get All Resource Types

URLs and Parameters

```
GET http://localhost/mws/rest/resource-types
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/resource-types?fields=id

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results":    [
    {"id": "throttle_migrate"}
  ]
}
```

4.19 Services

This section describes the behavior of a **Service** (an interdependent collection of workflows). It is possible for a **Service** to be composed of multiple Services. This section describes the URLs, request bodys, and responses delivered to and from Moab Web Services for each approach.

⚠ The Service API contains the type and description of all fields in the **Service** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/services	Get all Services		Create Service	
/rest/services/**id**	Get specified Service	Modify Service		Delete Service

4.19.1 Getting Service Information

The HTTP GET method is used to retrieve **Service** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/services[?query={"field":"value"}&sort={"field"
:<1|-1>}[&[show-recursive-vc|show-vc]=true]]
GET http://localhost/mws/rest/services/<id>[?[show-recursive-vc|show-vc]=true]
GET http://localhost/mws/rest/services/<name>[?[show-recursive-vc|show-vc]=true]
```

4.19.1.1 Get All Services

URLs and Parameters

```
GET http://localhost/mws/rest/services[?query={"field":"value"}&sort={"field"
:<1|-1>}[&[show-recursive-vc|show-vc]=true]]
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Query for specific results.	query={"type":"storage","label":"exlabel"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"account":-1}
show-recursive-vc	No	*true*	Show extended details about the service's virtual container including nested virtual containers and nested jobs.	show-recursive-vc=true
show-vc	No	*true*	Show details about the service's virtual container.	show-vc=true

Sample Response

GET http://localhost:8080/mws/rest/services?query={user:"bob"}

```
{
  "totalCount": 9,
  "resultCount": 3,
  "results":   [
        {
      "dateCreated": "2011-12-07 16:03:40 MST",
      "lastUpdated": "2011-12-07 16:03:40 MST",
      "name": "bobService.1",
      "version": 1,
      "type": "container",
      "label": null,
      "user": "bob",
      "account": "bamboo",
      "status": "A custom status message",
      "statusCode": 0,
      "includedServices":       [
        "machine0.1",
        "OSStoremachine0.1"
      ],
      "parent": null,
      "serviceTemplate":       {
        "id": "4fbd42cfc4aa4c444cc54112",
        "name": "CentosVmPlusStorage"
        },
        "attributes": {"moab":       {
          "vc": {"id": "vc56"},
          "dependencies": [        {
            "service": "machine0.1",
            "dependency": ["OSStoremachine0.1"]
          }]
        }},
        "id": "4edff0cc6852f709fa777826"
      },
          {
        "dateCreated": "2011-12-07 16:03:40 MST",
        "lastUpdated": "2011-12-07 16:03:40 MST",
        "name": "machine0.1",
        "version": 1,
        "type": "vm",
        "label": "bobs machine",
        "user": "bob",
        "account": "bamboo",
        "status": "A custom status message",
        "statusCode": 0,
        "includedServices": [],
        "parent": "bobService.1",
        "serviceTemplate":       {
          "id": "4fbd42cfc4aa4c444cc54113",
          "name": "CentosVm"
        },
        "attributes": {"moab":       {
          "vc": {"id": "vc57"},
          "job":         {
            "id": "Moab.24",
            "template": "genericVM",
            "image": "centos5.5-stateless",
            "features": ["vlan3"],
            "variables": {"QOS": "High"},
            "resources":           {
              "mem": 2,
              "procs": 2,
              "disk": 2
            }
          }
        }},
        "id": "4edff0cc6852f709fa777827"
      },
          {
        "dateCreated": "2011-12-07 16:03:40 MST",
        "lastUpdated": "2011-12-07 16:03:40 MST",
        "name": "OSStoremachine0.1",
        "version": 1,
        "type": "storage",
        "label": null,
        "user": "bob",
        "account": "bamboo",
        "status": "A custom status message",
        "statusCode": 0,
        "includedServices": [],
        "parent": "bobService.1",
        "serviceTemplate":       {
          "id": "4fbd42cfc4aa4c444cc54114",
          "name": "OpSysStorage"
        },
        "attributes": {"moab":       {
          "vc": {"id": "vc58"},
          "job":         {
            "id": "Moab.23",
            "template": "OSStorage",
            "resources": {"OS": 200}
          }
        }},
        "id": "4edff0cc6852f709fa777828"
      }
    ]
  }
```

Querying Services

It is possible to query services by one or more fields based on MongoDB query syntax.

Simple Queries

To see only services that are associated with the user "bob" you can use a query such as the following:

```
http://localhost/mws/rest/services?query={"user":"bob"}
```

To see only services that are of type "vm":

```
http://localhost/mws/rest/services?query={"type":"vm"}
```

To see only bob's vm services:

```
http://localhost/mws/rest/services?query={"user":"bob","type":"vm"}
```

To see only services that are NOT associated with bob:

```
http://localhost/mws/rest/services?query={"user":{"$ne":"bob"}}
```

More Complex Queries

When the field values of the desired services are a finite set, you can use the `$in` operator. For example, to see services that belong to either bob, alice, or charlie, you can do the following:

```
http://localhost/mws/rest/services?query={"user":{"$in":["alice","bob","charlie"]}}
```

You can also query on embedded JSON objects within the service JSON. For example, to see services requesting 3 processors you can use:

```
http://localhost/mws/rest/services?query={"attributes.moab.job.resources.procs":3}
```

Conditional Operators

You can perform <, <=, >, >= comparisons using the `$lt`, `$lte`, `$gt`, `$gte` operators.

Operator	Comparison
$lt	<
$lte	<=
$gt	>
$gte	>=

To see services requesting **<** 2 processors:

```
http://localhost/mws/rest/services?query={"attributes.moab.job.resources.procs":{"$lt"
:2}}
```

To see services requesting **>=** 1024 memory:

```
http://localhost/mws/rest/services?query={"attributes.moab.job.resources.mem":{"$gte"
:1024}}
```

Querying Services by Date

To see all services created after Febuary 8, 2012 at 1:00 PM Mountain Standard Time (MST):

```
http://localhost/mws/rest/services?query={"dateCreated":{"$gt":"2012-02-08 13:00:00
MST"}}
```

To see services created before or on Febuary 8, 2012 at 1:00 PM Pacific Standard Time (PST):

```
http://localhost/mws/rest/services?query={"dateCreated":{"$lte":"2012-02-08 13:00:00
PST"}}
```

To see services created between 12:00 PM and 1:00 PM Eastern Standard Time (EST) on Febuary 8, 2012:

```
http://localhost/mws/rest/services?query={"dateCreated":{"$lte":"2012-02-08 13:00:00
EST","$gte":"2012-02-08 12:00:00 EST"}}
```

Querying Services by Containing Service

Services can contain other services. When a service is contained within another service, you can find out what its container is by looking at the parent field. A service that is not contained in any other service is called a top level service. If you want to see only top level services you need to query for services with a null parent.

In MongoDB syntax you query for services whose parent field have a `$type` of **10** (with 10 representing null). The following query shows all of bob's top level services:

```
http://localhost/mws/rest/services?query={"user":"bob","parent":{"$type":10}}
```

Once you have the top level service, you can find the direct child services:

```
http://localhost/mws/rest/services?query={"user":"bob","parent":"bobService.1"}
```

Once you have the direct children, you can find the children of those children with a similar query.

Sorting

See the sorting section of Global URL Parameters

Limiting the Number of Results

If you want to limit the number of results of services you can use the `max` parameter. For example, to see only 10 of bob's services:

```
http://localhost/mws/rest/services?query={"user":"bob"}&sort={"name":1}&max=10
```

To see bob's services 91-100 when sorted by name in ascending order you can combine `max` with `offset` as follows:

```
http://localhost/mws/rest/services?query={"user":"bob"}&sort={"name"
:1}&max=10&offset=90
```

Retrieving a Subset of Fields

To cause only certain fields to return for each service, use the `fields` parameter. For example, to show only the name field for each service:

```
http://localhost/mws/rest/services?fields=name
```

This returns:

```
{
  "totalCount": 9,
  "resultCount": 3,
  "results":   [
    {"name": "aliceService.1"},
    {"name": "machine0.1"},
    {"name": "OSStoremachine0.1"}
  ]
}
```

To show the name, type, and user:

```
http://localhost/mws/rest/services?fields=name,type,user
```

This returns:

```
{
  "totalCount": 9,
  "resultCount": 3,
  "results":   [
    {
      "name": "aliceService.1",
      "type": "container",
      "user": "alice"
    },
        {
      "name": "machine0.1",
      "type": "vm",
      "user": "alice"
    },
        {
      "name": "OSStoremachine0.1",
      "type": "storage",
      "user": "alice"
    }
  ]
}
```

4.19.1.2 Get Single Service

URLs and Parameters

```
GET http://localhost/mws/rest/services/<id>[?[show-recursive-vc|show-vc]=true]
GET http://localhost/mws/rest/services/<name>[?[show-recursive-vc|show-vc]=true]
```

Parameter	Required	Valid Values	Description	Example
id	Yes	String	The unique identifier of the service.	
name	Yes	String	The name of the service.	
show-recursive-vc	No	*true*	Show extended details about the service's virtual container including nested virtual containers and nested jobs.	show-recursive-vc=true
show-vc	No	*true*	Show details about the service's virtual container.	show-vc=true

Parameter	Required	Type	Valid Values	Description

See Global URL Parameters for available URL parameters.

Only one of **id** or **name** are required.

Samples

GET http://localhost/mws/rest/services/bobService.1?

```
{
  "dateCreated": "2011-12-07 16:03:40 MST",
  "lastUpdated": "2011-12-07 16:03:40 MST",
  "name": "bobService.1",
  "version": 1,
  "type": "container",
  "label": null,
  "user": "bob",
  "account": "bamboo",
  "status": "A custom status message",
  "statusCode": 0,
  "includedServices":   [
    "machine0.1",
    "OSStoremachine0.1"
  ],
  "parent": null,
  "serviceTemplate":   {
    "id": "4fbd42cfc4aa4c444cc54112",
    "name": "CentosVmPlusStorage"
  },
  "attributes": {"moab":   {
    "vc": {"id": "vc56"},
    "dependencies": [    {
      "service": "machine0.1",
      "dependency": ["OSStoremachine0.1"]
    }]
  }},
  "id": "4edff0cc6852f709fa777826"
}
```

4.19.2 Creating Services

The HTTP POST method is used to create a **Service**.

Quick Reference

```
POST http://localhost/mws/rest/services
```

4.19.2.1 Create Service From Service Template

URLs and Parameters

```
POST http://localhost/mws/rest/services
```

Simple Case

To create a service from the template named "Rhel54VmPlusStorage":

POST http://localhost/mws/rest/services

```
{
  "user": "steve",
  "account": "cloud",
  "data":   [
    {
      "name": "MyRhel54VmPlusStorage",
      "serviceTemplate": "Rhel54VmPlusStorage",
    }
  ]
}
```

Alternatively you can submit:

POST http://localhost/mws/rest/services

```
{
  "user": "steve",
  "account": "cloud",
  "data":   [
    {
      "name": "MyRhel54VmPlusStorage",
      "serviceTemplate": {
        "name":"Rhel54VmPlusStorage"
      }
    }
  ]
}
```

To create a service based on the service template with id "4fbd2d90c4aa4996400bsa5m"

POST http://localhost/mws/rest/services

```
{
  "user": "steve",
  "account": "cloud",
  "data":   [
    {
      "name": "MyRhel54VmPlusStorage",
      "serviceTemplate": {
        "id":"4fbd2d90c4aa4996400bsa5m"
      }
    }
  ]
}
```

Extending a Service Template

If you want to create a service from a service template, but wish to extend the service template with some additional variables or generic resources, you can use the **extends** field. Extending a service template is also helpful when you wish to override certain values, such as the amount of memory or processors the service requires.

To extend a service template, you will need to determine the extends path for the service you wish to override. The extends path is the name of the top level service, followed by one or more localNames as described in the includedServices field. All but the last <localName> are nested containers inside the top level container. For example:

```
<top level service name>::<localName>[:<localName>]+
```

For example, suppose you want to create a new service from the "Rhel54VmPlusStorage" service template, and you want to name this new service "MyRhel54VmPlusStorage". In this example, "Rhel54VmPlusStorage" contains a service template named "SubContainer1". The localName for "SubContainer1" in the "Rhel54VmPlusStorage" **includedServices** field is "sc1".

Rhel54VmPlusStorage Service Template

```
{
   "name":"Rhel54VmPlusStorage",
   "type":"container",
   …
   "includedServices":[
      {
         "localName":"sc1",
         "serviceTemplate":"SubContainer1"
      }
   ]
}
```

The extends path for the instance of "SubContainer1" in your "MyRhel54VmPlusStorage" is:

```
MyRhel54VmPlusStorage::sc1
```

Let's say inside "SubContainer1" is another service template called "SubContainer2". The localName for "SubContainer2" as defined in the includedServices field for "SubContainer1" is "sc2".

SubContainer1 Service Template

```
{
   "name":"SubContainer1",
   "type":"container",
   …
   "includedServices":[
      {
         "localName":"sc2",
         "serviceTemplate":"SubContainer2"
      }
   ]
}
```

The extends path for the instance of "SubContainer2" in "MyRhel54VmPlusStorage" is:

```
MyRhel54VmPlusStorage::sc1:sc2
```

Now let's say that "SubContainer2" contains two service templates, "Rhel54Vm" and "OpsysStorage" with localNames "rvm" and "oss" respectively.

SubContainer1 Service Template

```
{
   "name":"SubContainer2",
   "type":"container",
   …
   "includedServices":[
      {
         "localName":"rvm",
         "serviceTemplate":"Rhel54Vm"
      },
      {
         "localName":"oss",
         "serviceTemplate":"OpSysStorage"
      }
   ]
}
```

The extends paths for the instances of "Rhel54VM" and "OpSysStorage" in "MyRhel54VmPlusStorage" are:

```
MyRhel54VmPlusStorage::sc1:sc2:rvm
MyRhel54VmPlusStorage::sc1:sc2:oss
```

Now that we have the extends paths for all the services that will be created from the "Rhel54VmPlusStorage" template, we can add variables to these services that were not in the service templates.

POST http://localhost/mws/rest/services

```
{
  "user": "steve",
  "account": "cloud",
  "data":   [
        {
      "name": "MyRhel54VmPlusStorage",
      "serviceTemplate": "Rhel54VmPlusStorage",
      "attributes": {
         "sharedData":{ "extraAttribute":"some attribute not in the Rhel54VmPlusStorage
template" }
      }
    },
    {
      "name": "MyRhel54Vm",
      "extends": "MyRhel54VmPlusStorage::sc1:sc2:rvm",
      "attributes": {
          "moab": {"job": {"variables": {"extraVar": "An additional variable not in the
Rhel54Vm template"}}},
          "sharedData":{ "extraAttribute":"some attribute not in the Rhel54Vm template"
}
      }

},
    {
      "name": "MyOsStorage",
      "extends": "MyRhel54VmPlusStorage::sc1:sc2:oss",
      "attributes": {
          "moab": {"job": {"variables": {"extraVar2": "An additional variable not in
the OpSysStorage template"}}},
          "sharedData":{ "extraAttribute":"some attribute not in the OpSysStorage
template" }
      }
    }
  ]
}
```

When the "MyRhel54Vm" service is created, it will have a variable named "extraVar" even though this variable was not defined in the "Rhel54Vm" service template. Likewise, when the "MyOsStorage" service is created, it will have a variable named "extraVar2", even though no such variable was defined in the "OsStorage" service template. All three services will have an attribute named "extraAttribute" in their attributes.sharedData sections though "extraAttribute" does not appear in any service template.

Extending Services and Dependencies in a Container Service

To add a services to a container service that were not in the container's service template you first define the new services in the service request. Then you extend the includedServices field of the container with the newly defined services. This will add the new services to any that are already in the container as defined in the service template. It is only possible to add services to a container. It is not possible to remove services from a container that were defined in the container's service template.

For example, say the CentosVmPlusStorage service template contains an OpSysStorage service template and a CentosVm service template.

CentosVmPlusStorage Service Template

```
{
   "name":"CentosVmPlusStorage",
   "type":"container",
   …
   "includedServices":[
      {
         "localName":"oss",
         "serviceTemplate":"OpSysStorage"
      },
      {
         "localName":"cvm",
         "serviceTemplate":"CentosVm"
      }
   ]
}
```

To add two storage services to the service created from the CentosVmPlusStorage service template submit the following service request:

POST http://localhost/mws/rest/services

```
{
   "user":"bob",
   "account":"cloud",
   "data":[
      {
         "name":"BobsCentosVmPlusStorage",
         "serviceTemplate":"CentosVmPlusStorage",
         "includedServices":[
            "NewStorageToAdd1",
            "NewStorageToAdd2"
         ]
      },
      {
         "name":"NewStorageToAdd1",
         "serviceTemplate":"ExtraStorage"
      },
      {
         "name":"NewStorageToAdd2",
         "serviceTemplate":"ExtraStorage"
      }
   ]
}
```

The resulting service BobsCentosVmPlusStorage will contain NewStorageToAdd1, NewStorageToAdd2, a service created from the OpSysStorage template, and a service created from the CentosVm template. To add a dependency such that the CentosVm service will not be able to start until both NewStorageToAdd1 and NewStorageToAdd2 have been set up:

POST http://localhost/mws/rest/services

```
{
   "user":"bob",
   "account":"cloud",
   "data":[
      {
         "name":"BobsCentosVmPlusStorage",
         "serviceTemplate":"CentosVmPlusStorage",
         "includedServices":[
            "NewStorageToAdd1",
            "NewStorageToAdd2"
         ],
         "attributes":{
            "moab":{
               "dependencies":[
                  {
                     "service":"BobsCentosVm",
                     "dependency":[
                        "NewStorageToAdd1",
                        "NewStorageToAdd2"
                     ]
                  }
               ]
            }
         }
      },
      {
         "name":"BobsCentosVm",
         "extends":"CentosVmPlusStorage:cvm"
      },
      {
         "name":"NewStorageToAdd1",
         "serviceTemplate":"ExtraStorage"
      },
      {
         "name":"NewStorageToAdd2",
         "serviceTemplate":"ExtraStorage"
      }
   ]
}
```

Extendable Fields

You can only extend certain fields. Below is a table of fields that can be extended:

Extendable Fields	Notes
attributes.moab.dependencies	Dependencies can be added but not removed. Only applicable to containers.
attributes.moab.job.features	Features can be added but not removed.
attributes.moab.job.requestedHosts	Hosts can be added but not removed.
attributes.moab.job.resources	Including procs, mem, disk, and any generic resource.
attributes.moab.job.variables	Can either change the value of variables in the template or add new variables.
attributes.sharedData	A place for arbitrary, site-specific data.
image	
includedServices	Services can be added but not removed. Only applicable to containers.
label	

Sample Response

If the request was successful, the response includes the unique ID of the new Service. On failure, the response is an error message.

JSON Response

```
{"name":"MyRhel54VmPlusStorage.1"}
```

4.19.2.2 Create Custom Service

URLs and Parameters

```
POST http://localhost/mws/rest/services
```

Request Body

POST http://localhost/mws/rest/services

```
{
   "user":"adaptive",
   "account":"cloud",
   "data":[
       {
         "name":"myNewService",
         "type":"container",
         "label":"My New Service",
         "includedServices":[
            "myVmContainer",
            "myNetworkStorageWorkflow",
            "myPmContainer"
         ],
         "attributes":{
            "moab":{
               "dependencies":[
                  {
                     "dependency":[
                        "myNetworkStorageWorkflow"
                     ],
                     "service":"myVmWorkflow"
                  }
               ]
            },
            "sharedData":{
               "extraAttribute":"Some arbitrary value",
               "extraAttribute2":"Another arbitrary value"
            }
         }
      },
      {
         "name":"myVmContainer",
         "type":"container",
         "includedServices":[
            "myVmWorkflow",
            "myOsStorageWorkflow"
         ],
         "attributes":{
            "moab":{
               "dependencies":[
                  {
                     "dependency":[
                        "myOsStorageWorkflow"
                     ],
                     "service":"myVmWorkflow"
                  }
               ]
            }
         }
      },
      {
         "name":"myVmWorkflow",
         "type":"vm",
         "includedServices":[

],
         "attributes":{
            "moab":{
               "job":{
                  "resources":{
                     "procs":2,
                     "mem":2048,
                     "disk":80
                  },
                  "variables":{
                       "QOS":"Premium"
                  },
                  "image":"centos5.5-stateless",
                  "template":"genericVM",
                  "requestedHosts":["i16"],
                  "features":["vlan3"]
               }
            }
         }
      },
      {
         "name":"myOsStorageWorkflow",
         "type":"storage",
         "includedServices":[
],
         "attributes":{
            "moab":{
               "job":{
                  "template":"OSStorage",
                  "resources":{
                     "OS":2500
                  }
               }
            }
         }
      },
      {
         "name":"myNetworkStorageWorkflow",
         "type":"storage",
         "includedServices":[

],
         "attributes":{
            "moab":{
               "job":{
                  "template":"extraStorage",
                  "resources":{
                     "gold":500
                  },
                  "variables":{
                     "mount":"/path/to/mount"
                  }
               }
            }
         }
      },
      {
         "name":"myPmContainer",
         "type":"container",
         "includedServices":[
            "myPmWorkflow"
         ]
      },
      {
         "name":"myPmWorkflow",
         "type":"pm",
         "includedServices":[

],
         "attributes":{
            "moab":{
               "job":{
                  "resources":{
                     "procs":2,
                     "mem":2048,
                     "disk":100
                  },
                  "variables":{
                     "QOS":"Premium"
                  },
                  "image":"centos5.5-stateless",
                  "template":"genericPM"
               }
            }
         }
      }
   ]
}
```

Sample Response

If the request was successful, the response includes the unique ID of the new Service. On failure, the response is an error message.

JSON Response

```
{"name":"myNewService.1"}
```

4.19.3 Modifying Services

The HTTP PUT method is used to modify **Services**.

Quick Reference

```
PUT http://localhost/mws/rest/services/<id>
PUT http://localhost/mws/rest/services/<name>
```

4.19.3.1 Modify Service

URLs and Parameters

```
PUT http://localhost/mws/rest/services/<id>
PUT http://localhost/mws/rest/services/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the Service.
name	Yes	String	-	The name of the Service .

See Global URL Parameters for available URL parameters.

Only one of **id** or **name** are required.

Example Request

Only the `attributes`, `status`, and `statusCode` fields may be modified in services. Note that the `status` field must be a valid string, and the `statusCode` field must be a valid number (long). Any arbitrary string and number may be used to represent the current state of the service through `status` and `statusCode` respectively.

PUT http://localhost:8080/mws/rest/services/myStorageService

```
{
    "status": "Done provisioning!",
    "statusCode": 200,
    "attributes": {
        "mount": "/mnt/myMount",
        "size": "2500",
        "sharedData":{
            "extraAttribute":"Some arbitrary value",
            "extraAttribute2":"Another arbitrary value"
        }
    }
}
```

The `moab` element of attributes cannot be modified. An error will be returned if this is attempted.

Sample Response

JSON Response

```
{
    "name": "myStorageService",
    "dateCreated": "2012-02-01 14:54:52 MST",
    "lastUpdated": "2012-02-01 14:54:5    2 MST",
    "type": "storage",
    "label": null,
    "user": "john",
    "account": "corp",
    "status": "Done provisioning!",
    "statusCode": 200,
    "includedServices": [],
    "parent": "myVmWithStorage",
    "attributes": {
        "moab": {
            "vc    ": {
                "id": "vc3"
            },
            "job": {
                "id": "Moab.1",
                "template": "extraStorage",
                "resources": {
                    "gold": 2500
                }
            }
        },
        "sharedData":{
            "extraAttribute":"Some arbitrary value",
            "extraAttribute2":"Another arbitrary value"
        },
        "mount": "/mnt/myMount",
        "size": "2500"
    },
    "id": "4f29b4abe4b03c2f8e3a1a40"
}
```

4.19.4 Deleting Services

The HTTP DELETE method is used to delete **Services**.

Quick Reference

```
DELETE http://localhost/mws/rest/services/<id>
DELETE http://localhost/mws/rest/services/<name>
```

4.19.4.1 Delete Service

URLs and Parameters

```
DELETE http://localhost/mws/rest/services/<id>
DELETE http://localhost/mws/rest/services/<name>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the Service.
name	Yes	String	-	The name of the Service.
force-delete	No	Boolean	-	If true MWS will not check service dependencies before deleting it.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** are required.

Sample Response

JSON Response

```
{}
```

4.20 Service Templates

This section describes the behavior of the **Service Template** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Service Template API contains the type and description of all fields in the **ServiceTemplate** object. It also contains details regarding which fields are valid during PUT and POST actions.

⚠ See Create Service From Service Template to create Services from Service Templates.

⚠ The Service Template name has the following constraints:

- It must only contain letters, digits, spaces, and these special characters: hyphen, period, question mark, at sign, tilde, pound sign, square brackets, angle brackets, vertical bar, equals sign, ampersand, parentheses, asterisk, curly braces, grave accent, and dollar sign.
- It cannot have the same form as a MongoDB ID (24 characters of 0-9 and a-f)
- It must be unique in the database.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/service-templates	Get all Service Templates		Create ServiceTemplate	
/rest/service-templates/**id or name**	Get specified Service Template	Modify ServiceTemplate		Cancel Service Template

4.20.1 Getting Service Templates

The HTTP GET method is used to retrieve **Service Template** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/service-templates[?query={"field":"value"}&sort={"field"
:<1|-1>}]
GET http://localhost/mws/rest/service-templates/<id>
GET http://localhost/mws/rest/service-templates/<name>
```

4.20.1.1 Get All Service Templates

URLs and Parameters

```
GET http://localhost/mws/rest/service-templates[?query={"field":"value"}&sort={"field"
:<1|-1>}]
```

Parameter	Required	Valid Values	Description	Example
query	No	JSON	Query for specific results.	query={"type":"vm","createdBy":"name"}
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"name":1}

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "totalCount": 5,
  "resultCount": 5,
  "results": [
    {
      "id": "4f04a93f84ae17912ae2763e",
      "label": "Linux ESA",
      "type": "vm",
      "name": "LinEsaTemplate",
      "modified": "2011-07-04 00:00:00 MDT",
      "createdBy": "TempName",
      "includedServices": [],
      "tags": [
        "tag0",
        "tag1"
      ],
      "attributes": {
        "dependencies": {
          "service": "tid.1",
          "dependency": [
            "tid.2",
            "tid.3"
          ]
        },
        "job": {
          "image": "rhel54-stateless",
          "resources": {
            "procs": 1,
            "mem": 1024,
            "ipaddress": 1
          },
          "template": "new-vm",
          "variables": {
            "foo": "bar"
          }
        },
        "viewpoint": {
          "name": "",
          "service-description": "",
          "form": {
            "f0": "zero",
            "f1": "one"
          },
          "access": {}
        }
      }
    },
    {
      "id": "4f05dd1484ae18e002b22d92",
      "label": "Linux ESA",
      "type": "vm",
      "name": "LinEsa004",
      "modified": "2011-07-04 00:00:00 MDT",
      "createdBy": "TempName",
      "includedServices": [
        {
          "localName": "SQLServ004",
          "serviceTemplate": "LinEsaTemplate"
        }
      ],
      "tags": [
        "tag0",
        "tag1"
      ],
      "attributes": {
        "dependencies": {
          "service": "tid.1",
          "dependency": [
            "tid.2",
            "tid.3"
          ]
        },
        "job": {
          "image": "rhel54-stateless",
          "resources": {
            "procs": 1,
            "mem": 1024,
            "ipaddress": 1
          },
          "template": "new-vm",
          "variables": {
            "foo": "bar"
          }
        },
        "viewpoint": {
          "name": "",
          "service-description": "",
          "form": {
            "f0": "zero",
            "f1": "one"
          },
          "access": {}
        }
      }
    },
    {
      "id": "4f05dd7484ae18e002b22d93",
      "label": "Linux ESA",
      "type": "vm",
      "name": "R",
      "modified": "2011-07-04 00:00:00 MDT",
      "createdBy": "TempName",
      "includedServices": [
        {
          "localName": "SQLServ004",
          "serviceTemplate": "LinEsaTemplate"
        }
      ],
      "tags": [
        "tag0",
        "tag1"
      ],
      "attributes": {
```

```
        "dependencies": {
          "service": "tid.1",
          "dependency": [
            "tid.2",
            "tid.3"
          ]
        },
        "job": {
          "image": "rhel54-stateless",
          "resources": {
            "procs": 1,
            "mem": 1024,
            "ipaddress": 1
          },
          "template": "new-vm",
          "variables": {
            "foo": "bar"
          }
        },
        "viewpoint": {
          "name": "",
          "service-description": "",
          "form": {
            "f0": "zero",
            "f1": "one"
          },
          "access": {}
        }
      }
    },
    {
      "id": "4f05e41f84ae18e002b22d94",
      "label": "Linux ESA",
      "type": "vm",
      "name": "5",
      "modified": "2011-07-04 00:00:00 MDT",
      "createdBy": "TempName",
      "includedServices": [
        {
          "localName": "SQLServ004",
          "serviceTemplate": "LinEsaTemplate"
        }
      ],
      "tags": [
        "tag0",
        "tag1"
      ],
      "attributes": {
        "dependencies": {
          "service": "tid.1",
          "dependency": [
            "tid.2",
            "tid.3"
          ]
        },
        "job": {
          "image": "rhel54-stateless",
          "resources": {
            "procs": 1,
            "mem": 1024,
            "ipaddress": 1
          },
          "template": "new-vm",
          "variables": {
            "foo": "bar"
          }
        },
        "viewpoint": {
          "name": "",
          "service-description": "",
          "form": {
            "f0": "zero",
            "f1": "one"
          },
          "access": {}
        }
      }
    },
    {
      "id": "4f05e4a284ae18e002b22d95",
      "label": "Linux ESA",
      "type": "vm",
      "name": "LinEsaServ001",
      "modified": "2011-07-04 00:00:00 MDT",
      "createdBy": "TempName",
      "includedServices": [
        {
          "localName": "SQLServ004",
          "serviceTemplate": "LinEsaTemplate"
        }
      ],
        "tags": [
          "tag0",
          "tag1"
        ],
        "attributes": {
          "dependencies": {
            "service": "tid.1",
            "dependency": [
              "tid.2",
              "tid.3"
            ]
          },
          "job": {
            "image": "rhel54-stateless",
            "resources": {
              "procs": 1,
              "mem": 1024,
              "ipaddress": 1
            },
            "template": "new-vm",
            "variables": {
              "foo": "bar"
            }
          },
          "viewpoint": {
            "name": "",
            "service-description": "",
            "form": {
              "f0": "zero",
              "f1": "one"
            },
            "access": {}
          }
        }
        }
    ]
}
```

Querying Service Templates

It is possible to query service templates by one or more fields based on the MongoDB query syntax.

Simple Queries

To see only service templates that are associated with the user "bob", use a query like the following:

```
http://localhost/mws/rest/service-templates?query={"user":"bob"}
```

To see only service templates that are of type "vm":

```
http://localhost/mws/rest/service-templates?query={"type":"vm"}
```

To see only bob's vm service templates:

```
http://localhost/mws/rest/service-templates?query={"user":"bob","type":"vm"}
```

To see only service templates that are NOT associated with bob:

```
http://localhost/mws/rest/service-templates?query={"user":{"$ne":"bob"}}
```

More Complex Queries

When the field values of the desired service templates are a finite set, use the `$in` operator. For example, to see service templates that belong to either bob, alice, or charlie, do the following:

```
http://localhost/mws/rest/service-templates?query={"user":{"$in":["alice","bob",
"charlie"]}}
```

You can also query on embedded JSON objects within the service template JSON. For example, to see service templates requesting 3 processors, do the following:

```
http://localhost/mws/rest/service-templates?query={
"attributes.moab.job.resources.procs":3}
```

Conditional Operators

You can perform <, <=, >, >= comparisons using the `$lt`, `$lte`, `$gt`, `$gte` operators.

Operator	Comparison
$lt	<
$lte	<=
$gt	>
$gte	>=

To see service templates requesting < 2 processors:

```
http://localhost/mws/rest/service-templates?query={
"attributes.moab.job.resources.procs":{"$lt":2}}
```

To see service templates requesting >= 1024 memory:

```
http://localhost/mws/rest/service-templates?query={"attributes.moab.job.resources.mem"
:{"$gte":1024}}
```

Querying Service Templates by Date

To see all service templates modified after July 4, 2011 at 10:30:00 PM Mountain Standard Time (MST):

```
http://localhost/mws/rest/service-templates?query={"modified":{"$gt":"2011-07-04
22:30:00 MST"}}
```

To see service templates modified before July 6, 2011 at 12:00 AM Pacific Standard Time (PST):

```
http://localhost/mws/rest/service-templates?query={"modified":{"$lt":"2011-07-06
00:00:00 PST"}}
```

To see service templates modified between 12:00 AM and 11:59 PM (inclusive) Eastern Standard Time (EST) on July 5, 2011

```
http://localhost/mws/rest/service-templates?query={"modified":{"$gte":"2011-07-05
00:00:00 EST","$lte":"2011-07-05 23:59:00 EST"}}
```

Sorting

See the sorting section in Global URL Parameters.

Limiting the Number of Results

To limit the size of the result set, use the `max` parameter. For example, to see only 10 of bob's services:

```
http://localhost/mws/rest/service-templates?query={"user":"bob"}&sort={"name":1}&max=10
```

To see bob's service templates 91-100 when sorted by name in ascending order, combine `max` with `offset` as follows:

```
http://localhost/mws/rest/service-templates?query={"user":"bob"}&sort={"name"
:1}&max=10&offset=90
```

Retrieving a Subset of Fields

To retrieve only certain fields, use the `fields` parameter. For example, to show only the `name` field for each service:

```
http://localhost/mws/rest/service-templates?fields=name
```

This returns:

```
{
  "totalCount": 9,
  "resultCount": 3,
  "results":   [
    {"name": "aliceService.1"},
    {"name": "machine0.1"},
    {"name": "OSStoremachine0.1"}
  ]
}
```

To show the name, type, and user:

```
http://localhost/mws/rest/service-templates?fields=name,type,user
```

This returns:

```
{
  "totalCount": 9,
  "resultCount": 3,
  "results":   [
    {
      "name": "aliceService.1",
      "type": "container",
      "user": "alice"
    },
        {
      "name": "machine0.1",
      "type": "vm",
      "user": "alice"
    },
        {
      "name": "OSStoremachine0.1",
      "type": "storage",
      "user": "alice"
    }
  ]
}
```

4.20.1.2 Get Single Service Template

URLs and Parameters

```
GET http://localhost/mws/rest/service-templates/<id>
GET http://localhost/mws/rest/service-templates/<name>
```

Parameter	Required	Valid Values	Description
id	Yes	String (24 character alphanumeric)	The unique identifier of the service template.
name	Yes	String	The name of the service template.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** is required.

Response

JSON Response

```
{
  "totalCount": 1,
  "resultCount": 1,
  "results": [  {
    "id": "...",
    …
  }]
}
```

4.20.2 Creating Service Templates

The HTTP POST method is used to create **Service Templates**.

Quick Reference

```
POST http://localhost/mws/rest/service-templates
```

4.20.2.1 Create Service Template

URLs and Parameters

```
POST http://localhost/mws/rest/service-templates
```

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows some of the fields that are available when creating a Service Template, along with some sample values.

JSON Request Body

```
{
    "attributes": {
        "moab": {
            "dependencies": [
                {
                    "dependency": [
                        "oss",
                        "ns"
                    ],
                    "localName": "rvm"
                }
            ],
            "job": {
                "features": [
                    "vlan3"
                ],
                "image": "centos5.5-stateless",
                "requestedHosts": [
                    "i16"
                ],
                "resources": {
                    "disk": 80,
                    "mem": 2048,
                    "procs": 1
                },
                "template": "genericVM",
                "variables": {
                    "QOS": "Premium"
                }
            }
        }
    },
    "createdBy": "bob",
    "includedServices": [
        {
            "localName": "rvm",
            "serviceTemplate": "Rhel54Vm"
        },
        {
            "localName": "oss",
            "serviceTemplate": "OpSysStorage"
        },
        {
            "localName": "ns",
            "serviceTemplate": "NetworkStorage"
        }
    ],
    "label": "Redhat Enterprise Linux 5.4 VM Plus OS and Network Storage",
    "modified": "2011-07-04 00:00:00 MDT",
    "name": "Rhel54VmPlusStorage",
    "tags": [],
    "type": "container"
}
```

⚠ includedServices is a key-value pair of the internal service name and the serviceTemplate. The service name is unique for each service container.

Sample Response

JSON Response for successful POST

```
{"id":"4f06111184ae2bbfa31fa4c7"}
```

If the Service Template name is not unique:

JSON Response

```
{
    "messages": [
        "Service template Rhel54Vm could not be created",
        "Request has a non-unique service template name 'Rhel54Vm'",
        "Please correct the request and try again"
    ]
}
```

If the Service Template included service local name is not unique to this service template:

JSON Response

```
{
    "messages": [
        "Service template CentOS5 could not be created",
        "Service template request has a non-unique included service template local name
([SQLServ05])",
        "Please correct the request and try again"
    ]
}
}
```

If the Service Template depends on a non-existent included service:

JSON Response

```
{
    "messages": [
        "Service template NSStor34 could not be created",
        "Service template requires service template(s) [NewRhel54Vm] which do not
exist",
        "Please correct the request and try again"
    ]
}
```

If the Service Template depends on more than one non-existent included service:

JSON Response

```
{
    "messages": [
        "Service template NSStor34 could not be created",
        "Service template requires service template(s) [NewRhel54Vm, Storage003] which
do not exist",
        "Please correct the request and try again"
    ]
}
```

If the Service Template name contains a colon:

JSON Response

```
{
{
    "messages": [
        "Service template Rhel54Vm:C could not be created",
        "Request contains a colon (:) in the service template name 'Rhel54Vm:C'",
        "Please correct the request and try again"
    ]
}
```

If the Service Template name has the same format as a MongoDB ID (Service Template ID):

JSON Response

```
{
    "messages": [
        "Service template 4f2049a684ae6e1d4f09bd71 could not be created",
        "Request has a MongoDB Object ID format for the service template name
'4f2049a684ae6e1d4f09bd71'",
        "Please correct the request and try again"
    ]
}
```

4.20.3 Modifying Service Templates

The HTTP PUT method is used to modify **Service Templates**.

⚠ The `modified` field is not automatically updated. It will need to be changed by the user.

Quick Reference

```
PUT http://localhost/mws/rest/service-templates/<id>
PUT http://localhost/mws/rest/service-templates/<name>
```

4.20.3.1 Modify Service Template

URLs and Parameters

```
PUT http://localhost/mws/rest/service-templates/<id>
PUT http://localhost/mws/rest/service-templates/<name>
```

Parameter	Required	Valid Values	Description
id	Yes	String (24 character alphanumeric)	The unique identifier of the service template.
name	Yes	String	The name of the service template.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** is required.

Request Body

This is similar to create, except you change the request body to what you need modified.

The request body below shows some of the fields that are available when modifying a Service Template, along with some sample values.

```
{
    "attributes": {
        "dependencies": {
            "dependency": [
                "tid.2",
                "tid.3"
            ],
            "service": "tid.1"
        },
        "job": {
            "image": "rhel54-stateless",
            "resources": {
                "ipaddress": 1,
                "mem": 1024,
                "procs": 1
            },
            "template": "new-vm",
            "variables": {
                "foo": "bar"
            }
        },
        "viewpoint": {
            "access": {},
            "form": {
                "f0": "zero",
                "f1": "one"
            },
            "name": "",
            "service-description": ""
        }
    },
    "createdBy": "Newname",
    "includedServices": [],
    "modified": "2011-07-04 00:00:00 MDT",
    "name": "A",
    "tags": [
        "database",
        "ele45",
        "tag56"
    ],
    "type": "RhOs"
}
```

Sample Response

JSON Response for successful PUT

```
{
    "resultCount": 1,
    "results": [
        {
            "attributes": {
                "dependencies": {
                    "dependency": [
                        "tid.2",
                        "tid.3"
                    ],
                    "service": "tid.1"
                },
                "job": {
                    "image": "rhel54-stateless",
                    "resources": {
                        "ipaddress": 1,
                        "mem": 1024,
                        "procs": 1
                    },
                    "template": "new-vm",
                    "variables": {
                        "foo": "bar"
                    }
                },
                "viewpoint": {
                    "access": {},
                    "form": {
                        "f0": "zero",
                        "f1": "one"
                    },
                    "name": "",
                    "service-description": ""
                }
            },
            "createdBy": "Newname",
            "id": "4f0746f684ae23bbd6726852",
            "includedServices": [],
            "label": "Linux ESA",
            "modified": "2011-07-04 00:00:00 MDT",
            "name": "RhOs004",
            "tags": [
                "database",
                "ele45",
                "tag56"
            ],
            "type": "RhOs"
        }
    ],
    "totalCount": 1
}
```

If the Service Template depends on a non-existent included service:

JSON Response

```
{
    "messages": [
        "Service template NewR could not be updated",
        "Service template requires service template(s) [RhOs045] which do not exist",
        "Please correct the request and try again"
    ]
}
```

If the Service Template depends on more than one non-existent included service:

JSON Response

```
{
    "messages": [
        "Service template NewR could not be updated",
        "Service template requires service template(s) [Stor45, Stor12] which do not
exist",
        "Please correct the request and try again"
    ]
}
```

An attempt to modify the Service Template name to an existing template name:

JSON Response

```
{
    "messages": [
        "Service template NewR could not be updated",
        "Request has a non-unique service template name 'Stor44'"
    ]
}
```

4.20.4 Deleting (Canceling) Service Templates

The HTTP DELETE method is used to delete **Service Templates**.

Quick Reference

```
DELETE http://localhost/mws/rest/service-templates/<id>
DELETE http://localhost/mws/rest/service-templates/<name>
```

4.20.4.1 Cancel Service Template

URLs and Parameters

```
DELETE http://localhost/mws/rest/service-templates/<id|name>
```

Parameter	Required	Valid Values	Description
id	Yes	String (24 character alphanumeric)	The unique identifier of the service template.
name	Yes	String	The name of the service template.

See Global URL Parameters for available URL parameters.

⚠ Only one of **id** or **name** is required.

Response

A successful deletion

JSON Response

```
{}
```

If the Service Template ID does not exist

JSON Response

```
{
    "messages": [
        "Service template not found with ID '4f2049a684ae6e1d4f09bd71'"
    ]
}
```

If the Service Template name does not exist

JSON Response

```
{
    "messages": [
        "Service template not found with ID 'Stor44'"
    ]
}
```

If other Service Templates depend on the one being deleted

JSON Response

```
{
    "messages": [
        "Service template Cent5 could not be deleted",
        "Service template 'Cent5' cannot be deleted because Service template '[Cent5]' 
depends on it "
    ]
}
```

4.21 Standing Reservations

This section describes behavior of the **Standing Reservation** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

> ⚠ The Standing Reservation API contains the type and description of all fields in the **Standing Reservation** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/standing-reservations	Get all standing reservations			
/rest/standing-reservations/**id**	Get specified standing reservation			

4.21.1 Getting Standing Reservations

The HTTP GET method is used to retrieve **Standing Reservation** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/standing-reservations/<id>
```

4.21.1.1 Get All Standing Reservations

URLs and Parameters

```
GET http://localhost/mws/rest/standing-reservations
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/standing-reservations?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "sr1"},
    {"id": "sr2"},
    {"id": "sr3"}
  ]
}
```

4.21.1.2 Get Single Standing Reservation

URLs and Parameters

```
GET http://localhost/mws/rest/standing-reservations/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "access": "DEDICATED",
  "accounts": ["account1"],
  "aclRules": [  {
    "affinity": "POSITIVE",
    "comparator": "EQUAL",
    "type": "USER",
    "value": "adaptive",
  }],
  "chargeAccount": "account2",
  "chargeUser": "user2",
  "classes": ["class1"],
  "clusters": ["cluster1"],
  "comment": "comment",
  "days": ["Monday"],
  "depth": 2,
  "disabled": false,
  "endOffset": 86415,
  "flags": ["ALLOWJOBOVERLAP"],
  "groups": ["group1"],
  "hosts": ["host1"],
  "id": "fast",
  "jobAttributes": ["TEMPLATESAPPLIED"],
  "maxJob": 2,
  "maxTime": 0,
  "messages": ["message1"],
  "nodeFeatures": ["feature1"],
  "os": "Ubuntu 10.04.3",
  "owner":   {
    "name": "root",
    "type": "USER"
  },
  "partition": "ALL",
  "period": "DAY",
  "procLimit":   {
    "qualifier": "<=",
    "value": 5
  },
  "psLimit":   {
    "qualifier": "<=",
    "value": 60
  },
  "qoses": ["qos1"],
  "reservationAccessList": [],
  "reservationGroup": "group2",
  "resources":   {
    "PROCS": -1,
    "tapes": 1
  },
  "rollbackOffset": 43200,
  "startOffset": 347040,
  "taskCount": 0,
  "tasksPerNode": 0,
  "timeLimit": -1,
  "triggers": [],
  "type": "type1",
  "users": ["user1"]
}
```

4.22 Usage Records

This section describes behavior of the **Usage Record** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

> The Usage Record API contains the type and description of all fields in the **UsageRecord** object.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/usage-records	Get all usage-records			
/rest/usage-records/**id**	Get specified usage-record			

4.22.1 Getting Usage Records

The HTTP GET method is used to retrieve **Usage Record** information.

Quick Reference

```
GET http://localhost/mws/rest/usage-records?proxy-user=<USER>
[&custom-fields=QualityOfService][&query={"id":"2"}][&sort={"Stage":-1}]
GET http://localhost/mws/rest/usage-records/<id>?proxy-user=<USER>
[&custom-fields=QualityOfService]
```

4.22.1.1 Get All Usage Records

URLs and Parameters

```
GET http://localhost/mws/rest/usage-records?proxy-user=<USER>
[&custom-fields=QualityOfService][&query={"id":"2"}][&sort={"stage":-1}]
```

Parameter	Required	Valid Values	Description	Example
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
custom-fields	No	Comma-Separated String	Includes custom MAM usage-record attributes.	custom-fields=QualityOfService
query	No	JSON	Query for specific results.	query={"priority":"2","allocation.active":"fals
sort	No	JSON	Sort the results. Use `1` for ascending and `-1` for descending.	sort={"stage":-1}

⚠ The `query` parameter does not support the full Mongo query syntax. Only querying for a simple, non-nested JSON object is allowed.

See Global URL Parameters for available URL parameters.

Sample Response

```
GET
http://localhost/mws/rest/usage-records?proxy-user=amy&custom-fields=qualityOfService&fields=id,q
```

```
{
  "totalCount": 8,
  "resultCount": 2,
  "results":   [
        {
      "id": 1,
      "qualityOfService": "premium",
      "type": "Job",
      "instance": "job.123"
    },
        {
      "qualityOfService": "premium",
      "id": 2,
      "type": "Job",
      "instance": "job.1234"
    }
  ]
}
```

4.22.1.2 Get Single Usage Record

URLs and Parameters

```
GET http://localhost/mws/rest/usage-records/<id>?proxy-user=<USER>
[&custom-fields=QualityOfService]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	Yes	String	Perform action as defined MAM user.	proxy-user=amy
custom-fields	No	Comma-Separated String	Includes custom MAM usage-record attributes.	custom-fields=QualityOfService

See Global URL Parameters for available URL parameters.

Sample Responses

```
GET
http://localhost/mws/rest/usage-records/1?proxy-user=amy&custom-fields=qualityOfService
```

```
{
  "id": 1,
  "qualityOfService": "premium",
  "type": "Job",
  "instance": "job.123",
  "charge": 0,
  "stage": "Create",
  "quote": "",
  "user": "doug",
}
```

4.23 Virtual Containers

This section describes behavior of the **Virtual Container** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

The Virtual Container API contains the type and description of all fields in the **Virtual Container** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/vcs	Get all Virtual Containers		Create Virtual Container	
/rest/vcs/**id**	Get specified Virtual Container	Modify Virtual Container		Destroy Virtual Container

4.23.1 Getting Virtual Containers

The HTTP GET method is used to retrieve **Virtual Container** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/vcs/<id>
```

4.23.1.1 Get All Virtual Containers

URLs and Parameters

```
GET http://localhost/mws/rest/vcs
```

See Global URL Parameters for available URL parameters.

Sample Response

```
GET http://localhost/mws/rest/vcs?fields=id
```

```
{
  "totalCount": 5,
  "resultCount": 5,
  "results":    [
    {"id": "vc3"},
    {"id": "vc1"},
    {"id": "vc4"},
    {"id": "vc5"},
    {"id": "vc2"}
  ]
}
```

4.23.1.2 Get Single Virtual Container

URLs and Parameters

```
GET http://localhost/mws/rest/vcs/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "aclRules": [  {
    "affinity": "POSITIVE",
    "comparator": "LEXIGRAPHIC_EQUAL",
    "type": "USER",
    "value": "root"
  }],
  "createDate": "2011-11-15 14:01:40 MST",
  "creator": "root",
  "description": "vc2",
  "flags": ["DESTROYWHENEMPTY"],
  "id": "vc2",
  "jobs": [
    {"id":"Moab.1"}
  ],
  "nodes": [
    {"id":"node1"}
  ],
  "owner":   {
    "name": "root",
    "type": "USER"
  },
  "reservations": [
    {"id":"system.1"}
  ],
  "variables":   {
    "a": "b",
    "c": "d"
  },
  "virtualContainers": [
    {"id":"vc3"}
  ],
  "virtualMachines": [
    {"id":"vm1"}
  ]
}
```

4.23.2 Creating Virtual Containers

The HTTP POST method is used to create **Virtual Containers**.

Quick Reference

```
POST http://localhost/mws/rest/vcs
```

4.23.2.1 Create Virtual Container

URLs and Parameters

```
POST http://localhost/mws/rest/vcs
```

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when creating a Virtual Container, along with some sample values.

JSON Request Body

```
{
  "description": "ted's vc",
  "owner":   {
    "name": "ted",
    "type": "USER"
  }
}
```

Sample Response

JSON Response for successful POST

```
{"id": "vc8"}
```

Restrictions

- When creating a Virtual Container, the owner field is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]           USERS=root,ted ENABLEPROXY=TRUE
```

- If `ENABLEPROXY` is set and you pass in the owner field as above, then the new Virtual Container will have creator and owner set to that user.

4.23.3 Modifying Virtual Containers

The HTTP PUT method is used to modify **Virtual Containers**.

Quick Reference

```
PUT http://localhost/mws/rest/vcs/<id>?change-mode=<add|remove|set>
```

4.23.3.1 Modify Virtual Container

URLs and Parameters

```
PUT http://localhost/mws/rest/vcs/<id>?change-mode=<add|remove|set>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
change-mode	Yes	String	add	Add the given objects (jobs, VMs, etc) to the objects that already exist.
			remove	Delete the given objects from the objects that already exist.
			set	Modify the attributes of the virtual container itself and **not** the associated objects.

See Global URL Parameters for available URL parameters.

Request Body

Here are three examples of Virtual Container updates: add objects, remove objects, and update attributes. In each case, the examples below show all the fields that are available, along with some sample values.

Add objects with /rest/vcs/vc1?change-mode=add

```
{
  "jobs":     [
    {"id": "Moab.37"},
    {"id": "Moab.38"}
  ],
  "nodes":     [
    {"id": "node1"},
    {"id": "node2"}
  ],
  "reservations":     [
    {"id": "system.48"},
    {"id": "system.49"}
  ],
  "virtualContainers":     [
    {"id": "vc93"},
    {"id": "vc94"}
  ],
  "virtualMachines":     [
    {"id": "vm2"},
    {"id": "vm4"}
  ]
}
```

Remove objects with /rest/vcs/vc1?change-mode=remove

```
{
  "jobs":     [
    {"id": "Moab.37"},
    {"id": "Moab.38"}
  ],
  "nodes":     [
    {"id": "node1"},
    {"id": "node2"}
  ],
  "reservations":     [
    {"id": "system.48"},
    {"id": "system.49"}
  ],
  "virtualContainers":     [
    {"id": "vc93"},
    {"id": "vc94"}
  ],
  "virtualMachines":     [
    {"id": "vm2"},
    {"id": "vm4"}
  ]
}
```

Modify VC attributes with /rest/vcs/vc1?change-mode=set

```
{
  "description": "This is a new description.",
  "flags": ["HOLDJOBS"],
  "owner":     {
    "name": "ted",
    "type": "USER"
  },
  "variables":     {
    "a": "b",
    "c": "d"
  }
}
```

Sample Responses

These messages may not match the messages returned from Moab exactly, but they are given as examples of the structure of the responses.

JSON response for adding objects

```
{
  "messages":[
    "job '147' added to VC 'vc3'",
    "job 'Moab.1' added to VC 'vc3'"
  ]
}
```

JSON response for removing objects

```
{
  "messages":[
    "job '147' removed from VC 'vc3'",
    "job 'Moab.1' removed from VC 'vc3'"
  ]
}
```

JSON response for updating attributes

```
{"messages":["VC 'vc3' successfully modified"]}
```

Restrictions

- You can change the ACL Rules on a Virtual Container, but not using this resource. See Create or Update ACLs.

4.23.4 Destroying Virtual Containers

The HTTP DELETE method is used to destroy **Virtual Containers**.

Quick Reference

```
DELETE http://localhost/mws/rest/vcs/<id>
```

4.23.4.1 Destroy Virtual Container

URLs and Parameters

```
DELETE http://localhost/mws/rest/vcs/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response for successful DELETE

```
{}
```

4.24 Virtual Machines

This section describes behavior of the **Virtual Machine** object in Moab Web Services. It contains the URLs, request bodies, and responses delivered to and from Moab Web Services.

⚠ The Virtual Machine API contains the type and description of all fields in the **Virtual Machine** object. It also contains details regarding which fields are valid during PUT and POST actions.

Supported Methods

Resource	GET	PUT	POST	DELETE
/rest/vms	Get all VMs		Create VM	
/rest/vms/**id**	Get specified VM	Modify VM		Destroy VM
/rest/nodes/**nodeId**/vms	Get all VMs on a Node			

4.24.1 Getting Virtual Machines

The HTTP GET method is used to retrieve **Virtual Machine** information. Queries for all objects and a single object are available.

Quick Reference

```
GET http://localhost/mws/rest/vms/<id>
GET http://localhost/mws/rest/nodes/<nodeId>/vms
```

4.24.1.1 Get All Virtual Machines

URLs and Parameters

```
GET http://localhost/mws/rest/vms
```

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/vms?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "vm1"},
    {"id": "vm2"},
    {"id": "vm3"}
  ]
}
```

4.24.1.2 Get All Virtual Machines On Node

URLs and Parameters

```
GET http://localhost/mws/rest/nodes/<nodeId>/vms
```

Parameter	Required	Type	Valid Values	Description
nodeId	Yes	String	-	The ID of the node of interest.

See Global URL Parameters for available URL parameters.

Sample Response

GET http://localhost/mws/rest/nodes/hv1/vms?fields=id

```
{
  "totalCount": 3,
  "resultCount": 3,
  "results":   [
    {"id": "vm1"},
    {"id": "vm2"},
    {"id": "vm3"}
  ]
}
```

4.24.1.3 Get Single Virtual Machine

URLs and Parameters

```
GET http://localhost/mws/rest/vms/<id>
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response

```
{
  "aliases": [],
  "availableDisk": 1024,
  "availableMemory": 512,
  "availableProcessors": 0,
  "cpuLoad": 0.823,
  "description": "",
  "effectiveTimeToLive": 0,
  "flags":   [
    "CREATION_COMPLETED",
    "CAN_MIGRATE"
  ],
  "genericEvents": [],
  "genericMetrics": {"watts": 250},
  "id": "vm3",
  "job": {"id": "Moab.1"},
  "lastMigrationDate": null,
  "lastSubstate": "",
  "lastSubstateModificationDate": null,
  "lastUpdateDate": null,
  "migrationCount": 0,
  "networkAddress": "10.0.0.5",
  "node": {"id": "hv2"},
  "osList": [],
  "os": "stateless1",
  "powerSelectState": "NONE",
  "powerState": "ON",
  "rack": 0,
  "requestedTimeToLive": 0,
  "slot": 0,
  "startDate": null,
  "state": "BUSY",
  "substate": "",
  "totalDisk": 1024,
  "totalMemory": 512,
  "totalProcessors": 1,
  "trackingJob": {"id": "Moab.5"},
  "triggers": [],
  "variables": {}
}
```

4.24.2 Creating Virtual Machines

The HTTP POST method is used to create **Virtual Machines**.

Quick Reference

```
POST http://localhost/mws/rest/vms[?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]           USERS=root,ted ENABLEPROXY=TRUE
```

4.24.2.1 Create Virtual Machine

URLs and Parameters

```
POST http://localhost/mws/rest/vms[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when creating a Virtual Machine, along with some sample values. Note that you can pass in an ID for the Virtual Machine. If you do not, Moab will choose an ID for you.

JSON Request Body

```
{
  "totalDisk": 1024,
  "totalMemory": 512,
  "totalProcessors": 1,
  "id": "vm3",
  "node": {"id": "hv2"},
  "os": "stateless1",
  "sovereign":true,
  "storage":"os:5'c'%os:10'd'",
  "template":"CustomTemplate",
  "requestedTimeToLive":10000,
  "triggers": [],
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample Response

JSON Response for successful POST

```
{"jobId": "vmcreate-25"}
```

⚠ The jobId in the response identifies the job that will create the virtual machine.

4.24.3 Modifying Virtual Machines

The HTTP PUT method is used to modify **Virtual Machines**.

Quick Reference

```
PUT http://localhost/mws/rest/vms/<id>[?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]         USERS=root,ted ENABLEPROXY=TRUE
```

4.24.3.1 Modify Virtual Machine

URLs and Parameters

```
PUT http://localhost/mws/rest/vms/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows all the fields that are available when modifying a Virtual Machine, along with some sample values.

JSON Request Body for VM Modify

```
{
  "genericEvents": [],
  "genericMetrics": {"watts": 250},
  "os": "stateless1",
  "powerState": "ON",
  "state": "BUSY",
  "triggers": [],
  "variables":   {
    "var1": "val1",
    "var2": "val2"
  }
}
```

Sample Response

This message may not match the message returned from Moab exactly, but is given as an example of the structure of the response.

JSON Response

```
{"messages":["successfully updated VM variables"]}
```

4.24.3.2 Migrate Virtual Machine

URLs and Parameters

```
PUT http://localhost/mws/rest/vms/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Request Body

The request body below shows how to migrate a Virtual Machine to a node with ID "hv2".

JSON Request Body for VM Migrate to a specific node

```
{"node": {"id": "hv2"}}
```

The request body below shows how to migrate a Virtual Machine to any available node by using the destination ID of ANY, which for this operation is a reserved word.

JSON Request Body for VM Migrate to any available node

```
{"node": {"id": "ANY"}}
```

Sample Response

⚠ The HTTP response code for this operation is 202 Accepted. See the responses section for more information.

JSON Response

```
{"jobId": "vm-migrate1"}
```

Restrictions

- If a migration is requested by setting the node as shown in the above examples, any other properties in the same request body will be ignored.

4.24.4 Destroying Virtual Machines

The HTTP DELETE method is used to destroy **Virtual Machines**.

Quick Reference

```
DELETE http://localhost/mws/rest/vms/<id>[?proxy-user=<username>]
```

Restrictions

- The proxy-user parameter is ignored unless you set `ENABLEPROXY=TRUE` in the `moab.cfg` file. Example:

```
ADMINCFG[1]          USERS=root,ted ENABLEPROXY=TRUE
```

4.24.4.1 Destroy Virtual Machine

URLs and Parameters

```
DELETE http://localhost/mws/rest/vms/<id>[?proxy-user=<username>]
```

Parameter	Required	Type	Valid Values	Description
id	Yes	String	-	The unique identifier of the object.
proxy-user	No	String	-	Perform the action as this user.

See Global URL Parameters for available URL parameters.

Sample Response

JSON Response for successful DELETE

```
{"jobId": "vmdestroy-26"}
```

⚠ The jobId in the response identifies the job that will destroy the virtual machine.

5 Reporting Framework

The reporting framework is a set of tools to make time-based reports from numerical data. The following sections will (1) provide an overview of the framework and the concepts related to it, and (2) work through an example report (CPU Utilization) with details regarding which web services to use and with what data.

The REST API reference is located in the Report Resource section.

5.1 Overview

5.1.1 Concepts

The reporting framework uses 3 core concepts: reports, datapoints, and samples.

- **Report** - A report is a time-based view of numerical data.
- **Datapoint** - A datapoint is a consolidated set of data for a certain time period.
- **Sample** - A sample is a snapshot of a certain set of data at a particular point in time.

To illustrate, consider the memory utilization of a virtual machine: at any given point in time, you can get the memory utilization by using your operating system's performance utilities (top for Linux, Task Manager for Windows):

```
2400/12040MB
```

By recording the memory utilization and time constantly for 1 minute, you could gather the following data:

Time	Memory Utilization
3:53:55 PM	2400/12040 MB
3:54:13 PM	2410/12040 MB
3:54:27 PM	2406/12040 MB
3:54:39 PM	2402/12040 MB
3:54:50 PM	2409/12040 MB

Each of the rows in the table above represent a **sample** of data. By averaging the rows we can consolidate them into one or more **datapoints**:

Start time	End Time	Memory Utilization
3:53:30 PM	3:54:00 PM	2400/12040 MB
3:54:00 PM	3:54:30 PM	2408/12040 MB
3:54:30 PM	3:55:00 PM	2406/12040 MB

Note that each datapoint covers exactly the same amount of time, and averages all samples within that period of time.

A **report**, then, is simply a list of datapoints with some additional configuration information:

Field	Value
Name	**Memory Utilization Report**
Datapoint Duration	30 seconds
Report Size	3 datapoints

Datapoints:

Start time	End Time	Memory Utilization
3:53:30 PM	3:54:00 PM	2400/12040 MB
3:54:00 PM	3:54:30 PM	2408/12040 MB
3:54:30 PM	3:55:00 PM	2406/12040 MB

5.1.2 Capabilities

While storing simple information like memory utilization is nice, the reporting framework is built to automatically handle much more complex information.

Consolidating Samples

Samples are JSON documents which are pushed into the report using the samples API. Samples are then stored until the consolidation operation creates a datapoint out of them. The table below shows how different data types are handled in this operation:

Type	Consolidation Function Handling
Numbers	Numerical data is averaged
Strings	Strings are aggregated into an array
Objects	The consolidation function recursively consolidates sub-objects
Lists	Lists are combined into a single flat list containing all elements
Mixed	If samples have different types of data for the same field, the values are aggregated into an array.
Null	These values will be ignored unless all values for a sample field are set to null, resulting in a null result.

⚠ If the mixed data types contains at least one number, it will be treated as numerical data. The non-numerical data will be ignored and the result will be averaged.

Below is an example of how the consolidation function works:

Samples:

Time	NumberEx	StringEx	ListEx	MixedEx	MixedNumberEx
3:53:55 PM	2400	"str1"	["elem1"]	"str1"	"str1"
3:54:13 PM	2410	"str2"	["elem2", "elem3"]	["elem1"]	["elem1"]
3:54:27 PM	2405	"str3"	["elem4"]	null	5

Resulting Datapoint after consolidation:

Time	NumberEx	StringEx	ListEx	MixedEx	MixedNumberEx
3:55:00 PM	2405	["str1", "str2", "str3"]	["elem1", "elem2", "elem3", "elem4"]	["str1", "elem1"]	5

Minimum Number of Samples

If your dataset is highly variable (i.e. values contained in samples are not very close together), converting a single sample into a datapoint may provide misleading information. It may be better to have a datapoint with an "Unknown" value. This can be accomplished by setting the minimum number of samples for a datapoint in the report.

The `minimumSampleSize` field in the Report API explains that if the specified size of samples is not met when the consolidation function is performed, the datapoint is considered "null" and no data is available for it. When this occurs, the sample data is discarded and the `data` field of the datapoint is set to "null".

For information on how to set this option, see the REST API Report Resource section in the documentation.

Report Size

Reports have a predetermined number of datapoints, or size, which sets a limit on the amount of data that can be stored. After the report size has been reached, as newly created datapoints are pushed into the report, the oldest datapoints will automatically be deleted. This is to aid in managing the storage capacity of the server hosting MWS.

On report creation, a Mongo collection will be initialized that is the maximum size of a single entry (currently 16 MB) multiplied by the report size. Be careful in setting a large report size as this will quickly allocate the entire disk if many reports with large report sizes are created.

5.2 Example Report (CPU Utilization)

To understand how the behavior and usage of the reporting framework, a sample report covering CPU Utilization will be shown in this section. It will not cover how to gather or display data for reports, but will cover some basic operations that are available with Moab Web Services to facilitate reporting.

5.2.1 Creating A Report

Before any data is sent to Moab Web Services, a report must first be created. A JSON request body with a HTTP method of POST must be used to do this.

POST /rest/reports

```
{
  "name":"cpu-util",
  "description":"An example report for cpu utilization",
  "consolidationFunction":"average",
  "datapointDuration":600,
  "reportSize":288
}
```

This will result in a report being created which can then be retrieved by sending a GET request to `/rest/reports/cpu-util`. The `datapointDuration` of `600` signifies that the datapoint consolidation should occur once every 10 minutes, while the `reportSize` (i.e. number of the datapoints) shows that the report will retain up to 2 days worth of the latest datapoints.

GET /rest/reports/cpu-util

```
{
    "consolidationFunction": "average",
    "datapointDuration": 600,
    "datapoints": [],
    "description": "An example report for cpu utilization",
    "id": "aef6f6a3a0bz7bf6449537c9d",
    "keepSamples": false,
    "minimumSampleSize": 1,
    "name": "cpu-util",
    "reportSize": 288,
    "version": 0
}
```

Note that an ID has been generated automatically and that no datapoints are associated with the report.

5.2.2 Adding Samples

Until samples are added and associated with the report, datapoint consolidation will generate datapoints with a `data` field equal to `null`. Once samples are added, however, they will be averaged and inserted into the next datapoint.

Create samples for the `cpu-util` by sending a POST request as follows:

POST /rest/reports/cpu-util/samples

```
[
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:00:00 MST",
    "data": {
      "minutes1": 0.5,
      "minutes5": 0,
      "minutes15": 0
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:01:00 MST",
    "data": {
      "minutes1": 1,
      "minutes5": 0.5,
      "minutes15": 0.05
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:02:00 MST",
    "data": {
      "minutes1": 1,
      "minutes5": 0.5,
      "minutes15": 0.1
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:03:00 MST",
    "data": {
      "minutes1": 0.75,
      "minutes5": 1,
      "minutes15": 0.25
    }
  },
  {
    "agent": "cpu-monitor",
    "timestamp":"2012-01-01 12:04:00 MST",
    "data": {
      "minutes1": 0,
      "minutes5": 1,
      "minutes15": 0.85
    }
  }
]
```

This sample data contains average load for the last 1, 5, and 15 minute intervals. The samples were recorded at one-minute intervals starting at noon on January 1st, 2012.

5.2.3 Consolidating Data

A consolidation function must run to generate datapoints from the given samples. This scheduled consolidation will occur at intervals of `datapointDuration` seconds. For each field in the `data` object in samples, all values will be averaged. If non-numeric values are included, the following strategies will be followed:

1. All fields which contain a single numeric value in any included sample will be averaged and the non-numeric or null values will be ignored.
2. All fields which contain a list will be consolidated into a single, flat list.
3. All fields which contain only non-numeric or null values will be consolidated into a single, flat list.

If no historical datapoints are provided in the creation of a report as in this example, the next consolidation will be scheduled for the current time plus the `datapointDuration`. In this example, the scheduled consolidation is at 10 minutes from the creation date. If historical datapoints are included in the report creation, the latest datapoint's `endDate` plus the `datapointDuration` will be used as the scheduled time. If this date was in the past, the next scheduled consolidation will occur at the appropriate interval from the last `endDate`.

5.2.4 Retrieving Report Data

To retrieve the consolidated datapoints, simply perform a GET request on the report once again. Alternatively, the GET for a report's datapoints may be used.

GET /rest/reports/cpu-util

```
{
    "consolidationFunction": "average",
    "datapointDuration": 600,
    "datapoints": [
        {
            "firstSampleDate": null,
            "lastSampleDate": null,
            "data": null,
            "startDate": "2012-01-01 11:49:00 MST",
            "endDate": "2012-01-01 11:59:00 MST"
        },
        {
            "firstSampleDate": "2012-01-01 12:00:00 MST",
            "lastSampleDate": "2012-01-01 12:04:00 MST",
            "data": {
                "minutes1": 0.65,
                "minutes15": 0.25,
                "minutes5": 0.6
            },
            "startDate": "2012-01-01 11:59:00 MST",
            "endDate": "2012-01-01 12:09:00 MST"
        }
    ],
    "description": "An example report for cpu utilization",
    "id": "aef6f6a3a0bz7bf6449537c9d",
    "keepSamples": false,
    "minimumSampleSize": 1,
    "name": "cpu-util",
    "reportSize": 288,
    "version": 0
}
```

Note that of the two datapoints above, only the second actually contains data, while the other is set to `null`. Only samples lying within the datapoint's duration, or from the `startDate` to the `endDate`, are included in the consolidation. Therefore the first datapoint, which covered the 10 minute period just before the samples' recorded timestamps, contained no data. The second, which covers the 10 minute period matching that of the samples, contains the averaged sample data. This data could be used to display consolidated report data in a custom interface.

5.2.5 Possible Configurations

Configuration options may be changed to affect the process of report generation. These are documented in the API for the Report object and the Sample object.

6 MWS Plugins (Beta)

This section describes MWS Plugins, their use, and their creation in Moab Web Services.

> ⚠ MWS Plugins are currently in beta. Interfaces may change significantly in future releases.

6.1 Plugin Overview

This section provides an overview of the plugin layer in web services. The following areas will be covered:

- An introduction to the concept of MWS plugins
- How to configure Moab Workload Manager to interact with MWS plugins
- A description of the plugin lifecycle
- How plugins are driven by events
- How to expose web services from a plugin
- How plugin utility services may be used
- How data report collisions between plugins are consolidated
- How calls from MWM are routed to MWS plugins

6.1.1 Introduction

Moab Web Services plugins provide a highly extensible interface to interact with Moab, MWS, and external resources. Plugins can perform some of the same functions as Moab Resource Managers (RMs), while also providing many other features not available to RMs. This section will discuss the main features of plugins, some basic terminology, and how MWS plugins can interact with Moab.

Features

Plugins can

- be created, modified, and deleted without restarting Moab Workload Manager or MWS.
- be defined in Groovy and uploaded to MWS without restarting.
- have individual data storage space and configuration.
- access MWS configuration and RESTful web services.
- log to a standard location configured in MWS.
- be polled at a regular interval (configured on a per-plugin basis).
- report current state data to Moab Workload Manager through the Resource Manager interface.
- be informed of important system events.
- be individually stopped, started, paused, and resumed.
- expose secured and unsecured custom web services for external use.
- be manipulated via a full RESTful API (see Resources for more information).
- be manipulated via a full user interface in a web browser.

Terminology

There are two distinct terms in the plugin layer: plugin types and plugins (also called plugin instances).

Plugin Types

Plugin Types can be considered plugin templates with built-in logic. In object-oriented programming languages, this relates to the concept of a class. They possess certain abilities, or methods, that can be called by Moab Web Services to query or update information about certain resources. They also can define methods which will be exposed to external clients as web services. They do not contain any configuration or current data, but they are often tied to a *type* of component, such as components that communicate with Moab's WIKI Protocol or those that are built on a certain product.

They can define several types of methods:

1. **Instance methods** that return information about the current plugin, such as `getState`. While these are defined in the plugin type, the plugin type itself does not have a state.
2. The **poll event method** that is called at a configured interval.
3. **Lifecycle event methods** of plugins created from the plugin type, such as `beforeStart` and `afterStart`.
4. **RM event methods** that are called by Moab when certain events occur.
5. **Web service methods** that expose custom functionality as public web services.

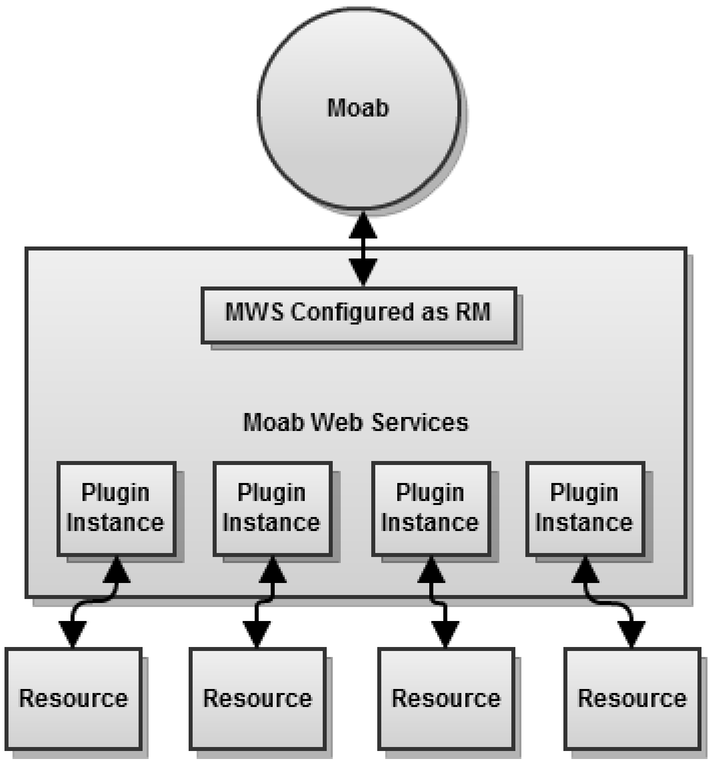
Some examples of plugin types include the Native and MSM plugin types.

Plugins (Instances)

Plugins (also called plugin instances) are created from plugin types. They contain current data or configuration and use the plugin type methods to interact with resources.

Interactions with MWM as a Resource Manager

The plugin layer in MWS is integrated with Moab Workload Manager via the Native Resource Manager (RM) interface. When utilizing plugins, MWS is configured as a RM in Moab as explained in the next section. Events from Moab are pushed through the RM interface to MWS which is then pushed to each plugin in turn. The relationship between Moab Web Services, Moab, and plugins is shown in the following image:



See Consolidating Data and Reporting State Data for more information.

6.1.2 Configuring Moab

To use the full functionality of MWS plugins (including RM events), Moab must be configured to use MWS as a resource manager. The following lines must be in the `/opt/moab/etc/moab.cfg` file or one of its included files:

```
RMCFG[mws]                TYPE=NATIVE
RMCFG[mws]                FLAGS=UserSpaceIsSeparate
RMCFG[mws]                CLUSTERQUERYURL=exec://$TOOLSDIR/mws/cluster.query.mws.pl
RMCFG[mws]                WORKLOADQUERYURL=exec://$TOOLSDIR/mws/workload.query.mws.pl
RMCFG[mws]                JOBCANCELURL=exec://$TOOLSDIR/mws/job.cancel.mws.pl
RMCFG[mws]                JOBMIGRATEURL=exec://$TOOLSDIR/mws/vm.migrate.mws.pl
RMCFG[mws]                JOBMODIFYURL=exec://$TOOLSDIR/mws/job.modify.mws.pl
RMCFG[mws]                JOBREQUEUEURL=exec://$TOOLSDIR/mws/job.requeue.mws.pl
RMCFG[mws]                JOBRESUMEURL=exec://$TOOLSDIR/mws/job.resume.mws.pl
RMCFG[mws]                JOBSTARTURL=exec://$TOOLSDIR/mws/job.start.mws.pl
RMCFG[mws]                JOBSUBMITURL=exec://$TOOLSDIR/mws/job.submit.mws.pl
RMCFG[mws]                JOBSUSPENDURL=exec://$TOOLSDIR/mws/job.suspend.mws.pl
RMCFG[mws]                NODEMODIFYURL=exec://$TOOLSDIR/mws/node.modify.mws.pl
RMCFG[mws]                NODEPOWERURL=exec://$TOOLSDIR/mws/node.power.mws.pl
RMCFG[mws]                RESOURCECREATEURL=exec://$TOOLSDIR/mws/resource.create.mws.pl
RMCFG[mws]                SYSTEMMODIFYURL=exec://$TOOLSDIR/mws/system.modify.mws.pl
RMCFG[mws]                SYSTEMQUERYURL=exec://$TOOLSDIR/mws/system.query.mws.pl
```

The next step is to edit the MWS values in `/opt/moab/etc/cloud.cfg`. Here are the default values:

```
CONFIG[default]                MWS_URL=http://localhost:8080/mws
CONFIG[default]                MWS_USERNAME=admin
CONFIG[default]                MWS_PASSWORD=adminpw
```

⚠ MWS_USERNAME and MWS_PASSWORD must match the values of auth.defaultUser.username and auth.defaultUser.password, respectively, found in /opt/mws/etc/mws-config.groovy.

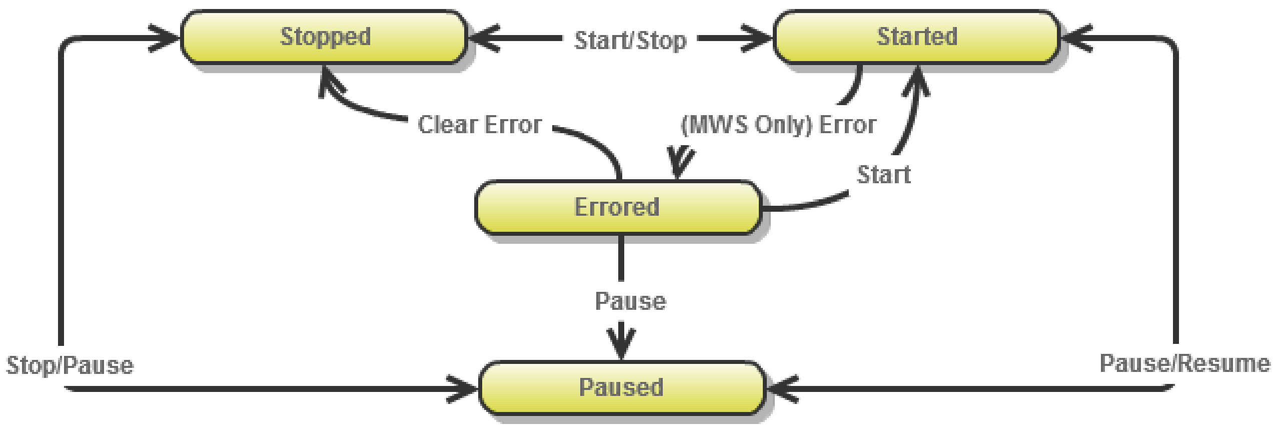
The *.mws.pl scripts should be located in the tools/mws of the Moab home directory. The Moab/WebServices.pm module must also be available to the scripts. All of these files may be found in the tools/mws and lib/perl5 directories of the Moab tar file. They are automatically installed if Moab is configured with the --with-mws flag or they can be copied directly from there to the tools folder in your Moab home directory.

To enable such actions as submitting jobs as different users, the ENABLEPROXY=TRUE option must be present in the ADMINCFG configuration line and the OSCREDLOOKUP option must be set to NEVER as follows:

```
ADMINCFG[1]          USERS=root      ENABLEPROXY=TRUE
OSCREDLOOKUP         NEVER
```

6.1.3 Lifecycle States

During the course of a plugin's use, the state of the plugin may change many times. Plugins have four possible states: Stopped, Started, Paused, and Errored. The flow of a plugin through the states is shown in the following image:



To see descriptions of each state, see the PluginState API.

Events that occur during lifecycle state changes may be found in the Events section.

6.1.4 Events

Plugins use an event based model in that methods are called on the plugin when certain criteria are met or situations arise. Events currently exist for polling, lifecycle state changes, and RM events from Moab. See Handling Events for more information.

6.1.5 Custom Web Services

Although the events interface typically serves most cases, there are some instances where an event is not supported that is desired. This is especially true when an external resource is the source of the event. To address these issues, plugins can expose custom web services to external resources. These web services may be named freely and do anything they wish within the plugin framework.

For example, suppose a resource needs to notify a plugin that provisioning of a virtual machine has been completed. Instead of having the plugin poll the resource to verify that the provisioning was finished, the plugin could expose a custom web service to handle notification from the resource itself.

Sample custom web service

```
def vmProvisionFinished(Map params) {
    // Handle event
    return [messages:["Event successfully processed"]]
}
```

Additionally, plugin types may define web services which are unsecured, meaning that a user or application account is not required to access it. A full explanation of the syntax and creation of custom secured and unsecured web services may be seen on the Exposing Web Services page.

For information how resources can access plugin web services, see Accessing Plugin Web Services.

6.1.6 Utility Services

Several features of plugins are only available by utilizing bundled services. These include:

- Accessing the individual datastore.
- Reporting state data to Moab through the Resource Manager interface.
- Manipulating other plugins and controlling their lifecycle.
- Accessing REST resources from Moab Web Services.

All bundled services are covered on their respective pages in the Quick Reference guide under "Plugin Services".

It may also be necessary or desired to create additional utility services when creating new plugin types. The easiest way to do this is to create a utility service which is called by convention a Translator, due to the fact that it typically can "translate" from a specific resource or API to data which can be used by the plugin type.

Finally, custom components may be used to fulfill use cases not covered by bundled services or custom translators.

6.1.7 Data Consolidation

At times, plugins can report differing or even contradictory data for nodes, virtual machines, and jobs. This is called a data "collision". The act of resolving these collisions is called "Consolidation". Currently, when data from one plugin "collides" with another, the last plugin to report the data will be considered the authoritative source for information. In addition, node, virtual machine, and job unique identifiers (IDs) are all converted to lower-case before consolidation, so a node "NODE1" is equivalent to "node1".

For example, suppose two plugins exist, `pluginA` and `pluginB`. These plugins both report data for a node with an ID of `node1`. However, each reports a different node power state. Plugin A reports the power as `ON`, while plugin B reports the power as `OFF`. The data collision that occurs due to these two contradictory reports is resolved by the timing of when they save their node reports. Assume that both plugins report node data when polling only. If plugin A is polled first and plugin B second, the node will be reported as `OFF` until plugin A is polled again and vice versa.

The simple workaround for this issue is to ensure that no two plugins report the same resource or that they report different properties of the same resource. For example, if plugin A only modified the power state and plugin B only modified the available disk resource, these two plugins would work in harmony to provide a consistent view of the node resource.

6.1.8 Routing

Due to the fact that Moab Web Services is configured as a Resource Manager (RM) in Moab Workload Manager, events are sometimes triggered by Moab through the RM interface. These actions could be migrating a virtual machine, starting a job, submitting a job, modifying a node, and so forth. The decisions of which plugins are affected and notified is termed *routing* .

Currently all plugins receive all commands from Moab. This means that each plugin will receive the command to start a job if sent from Moab, even if that plugin does not handle the job. This means that plugins must ensure they handle only actions or commands for resources which they report or handle.

6.2 Plugin Developer's Guide

Plugin types comprise the methods by which Moab may communicate with resource managers or other external components. They define all operations that can be performed for a "type" or "class" of plugins, hence the name "plugin type".

Several plugin types are provided with Moab Web Services (see Plugin Types in the Quick Reference), but it is easy to create additional plugin types and add their functionality to web services. This involves using Groovy, which is based on the Java programming language. This section describes the general guidelines and specifics of implementing new plugin types.

API Classes and Interfaces

There are several packages and classes available to assist in creating plugin types. These can all be found in the API documentation under the com.ace.mws.plugins package.

6.2.1 Requirements

This section discusses the requirements to create a basic functional plugin. The `com.ace.mws.plugins` package contains the abstract class AbstractPlugin that should form the basis of any new plugin type. However, this class need not be extended to create a functional plugin type. Only two requirements must be fulfilled for this:

1. The class name must end in `Plugin`.
2. There must exist `id` field getter and setter methods:

```
* public String getId();
* public void setId(String id);
```

The `id` field may be stored in whichever way desired as long as the getter and setter are available as shown above, but will most likely be implemented as follows:

```
class BasicPlugin {
    String id
}
```

In this case, `String id` will be expanded by the Groovy compiler to the full getter and setter method definitions given above. In other words, no explicit method definitions are actually needed. Note that the `BasicPlugin` shown above is able to be uploaded as a plugin type to MWS, but does not actually do anything.

6.2.2 Dynamic Methods

Several methods are dynamically inserted onto each plugin. These methods do not need to be included in the plugin class, and will be overwritten if included. Additionally, a logger is inserted into each plugin as discussed in the next section. The inserted methods are shown below:

```
// Full definitions can be found in
[AbstractPlugin|api:com.ace.mws.plugins.AbstractPlugin]
public void start() throws PluginStartException;    // Equivalent to the start method
in the [Plugin Control Service|Plugin Services]
public void stop() throws PluginStopException;  // Equivalent to the stop method in the
[Plugin Control Service|Plugin Services]
public Log getLog();    // Discussed in [Logging|guide:pluginLogging]
public ConfigObject getAppConfig();    // Discussed in
[Configuration|guide:pluginConfiguration]

// Full definition for these methods can be found in
[AbstractPluginInfo|api:com.ace.mws.plugins.AbstractPluginInfo]
public String getPluginType();
public PluginState getState();
public Integer getPollInterval();
public Boolean getAutoStart();
public Map<String, Object> getConfig();    // See
[Configuration|guide:pluginConfiguration]
```

Many of these methods are provided for convenience and are discussed in the linked pages or the following sections.

6.2.3 Logging

Logging in plugin types is uses the Apache Commons Logging and log4j libraries. Each plugin is injected with a method called `getLog` which can be used to access the configured logger. It returns an instance of org.apache.commons.logging.Log. Examples of using the logger are shown below.

The logger may used to register messages to the MWS log at several levels (in order of severity):

1. trace
2. debug
3. info
4. warn
5. error
6. fatal

Each of these levels is available as a method on the logger, for example:

```
public void poll() {
    getLog().debug("getLog() is equivalent to just using 'log' in Groovy")
    log.debug("This is a debug message and is used for debugging purposes only")
    log.info("This is a informational message")
    log.warn("This is a warning")
    log.error("This is an error message")
}
```

Logger Name

Each logger in the MWS logging configuration has a name. In the case of plugins, it is comprised of the full class name, including the package, prepended by "plugins.". For example, a plugin class of "example.LoggingPlugin" will have access to a logger configured as "plugins.example.LoggingPlugin".

Logging Configuration

The logging configuration is done through the MWS configuration file. See the MWS Configuration page for more information on configuring loggers. A good configuration for developing plugin types may be to add "plugins" at the debug level. Be sure to set the log level threshold down for the desired appender.

```
log4j = {
  …
  // Appender configuration
  ...
debug "plugins"
}
```

6.2.4 Configuration

Plugin types can access two different kinds of configuration: an individual plugin's configuration, and the global MWS application configuration.

Individual Plugin Configuration

The individual plugin configuration is separate for each instance of a plugin. This may be used to store current configuration information such as access information for linked resources. It should **not** be used to store cached information or non-configuration related data. The Individual Datastore should be used instead for these cases.

It is accessed by using the `getConfig` method discussed in Dynamic Methods.

```
public void poll() {
  def configFromMethod = getConfig()
  // OR an even simpler method…
  def configFromMethod = config
}
```

A common case is to retrieve the configuration in the `configure` method, verify that it matches predetermined criteria, and utilize it perform initial setup of the plugin (e.g. initialize libraries needed to communicate with external resources). For example, to verify that the configuration contains the keys "username" and "password", the following code may be used.

```
public void configure() throws InvalidPluginConfigurationException {
  def myConfig = config
  // This checks to make sure the key exists in the configuration Map and that the
value is not empty or null
  if (!myConfig.containsKey("username") || !myConfig.username)
      throw new InvalidPluginConfigurationException("The username configuration
parameter must be provided")
  if (!myConfig.containsKey("password") || !myConfig.password)
    throw new InvalidPluginConfigurationException("The password configuration parameter
must be provided")
}
```

Access MWS Configuration

The MWS application configuration can also be accessed in plugin types. This configuration is global for the entire application and can be modified by the administrator as shown in the MWS Configuration section.

It is accessed by using the `getAppConfig` method discussed in Dynamic Methods. This is demonstrated below.

```
public void poll() {
  // Retrieve the current MWS_HOME location
  def mwsHome = appConfig.mws.home.location
  // OR an even simpler method…
  def mwsHome = getAppConfig().mws.home.location
}
```

Any of the properties shown in the Configuration reference may be accessed. Custom properties may also be registered and accessed:

mws-config.groovy

```
plugins.custom.property = "This is my custom property"
```

CustomAppPropertyPlugin

```
public void poll() {
  assert appConfig.plugins.custom.property=="This is my custom property"
}
```

6.2.5 Individual Datastore

Each plugin has access to an individual, persistent datastore which may be used for a variety of reasons. The datastore is not designed to store Moab data such as nodes, jobs, or virtual machines, but custom, arbitrary data pertinent only to the individual plugin. This may include storing objects in a persistent cache, state information for currently running processes, or any other arbitrary data. The individual datastore has the following properties:

- Data is persisted to the Mongo database and will be available even if the plugin or MWS is restarted.
- The data must be stored in groups of data called **collections**. These correspond directly to MongoDB collections.
- Each plugin may have an arbitrary number of collections.
- Collections are guaranteed not to collide if there are identically named collections between two plugin types or even two plugin instances.
- Each collection contains multiple objects or **entries**. These correspond directly to MongoDB documents.
- The values of entries may be any object which can be serialized to MongoDB: simple types (int or Integer), Maps, and Lists.
- A collection is automatically created whenever an entry is added to it, it does not need to be specifically initialized.

To utilize the datastore, the Plugin Datastore Service must be used. Operations are provided to add, query, and remove data from each collection.

Example

The example below demonstrates two web services. The first adds multiple entries containing various types of data to an arbitrarily named collection. The second retrieves the data and returns it to the user.

```
package example

import com.ace.mws.plugins.*

class DatastorePlugin extends AbstractPlugin {
    IPluginDatastoreService pluginDatastoreService

def storeData(Map params) {
        def collectionName = params.collectionName
        def data = [[boolVal:true], [stringVal:"String"], [intVal:1], [nullVal:null]]
        if (pluginDatastoreService.addData(collectionName, data))
            log.info("Data successfully added")
        else
            log.info("There was an error adding the data")
    }

def retrieveData(Map params) {
        def collectionName = params.collectionName
        return pluginDatastoreService.getCollection(collectionName)
    }
}
```

6.2.6 Exposing Web Services

Any number of methods may be exposed as public, custom web services by satisfying several criteria:

1. The method must declare that it returns `Object` or `def`.
2. The method must define a single argument of type Map.
3. The method must actually return a List, Map, or a complex object; no simple types such as `int` or `String` can be returned.
4. The method must not be declared as private or protected; only public or unscoped methods will be recognized as web services.

Parameters and Request Body

The Map argument will contain all parameters passed into the web service by the client. See Accessing Plugin Web Services for additional details.

Parameters may be passed into the web service call as normal URL parameters such as `?param=value&param2=value2`, as key-value pairs in the POST body of a request, or as JSON in the body.

For the first two cases, the parameters will be available on the Map argument passed into the web service call as key value pairs matching those of the request. Note that in these cases all keys and values will be interpreted as strings. However, the parameters object has several helper methods to convert from Strings to simple types, such as booleans, integers, doubles, floats, and lists. If the value is not a valid simple type, null is returned.

```
GET <webServiceUrl>?key=value&key2=true&key3=5&list=1&list=2

def serviceMethod(Map params) {
    assert params.key=="value"
    assert params.key2=="true"
    assert params.bool('key2')==true
    assert params.key3=="5"
    assert params.int('key3')==5
    assert params.list('list')==[1, 2]

// Null is returned if the conversion is invalid
    assert params.int('key')==null
}
```

When the body possesses JSON, the parsed JSON object or array will be available within a parameter called `body` in the Map argument. In this scenario, the types of the values are preserved by the JSON format.

```
POST <webServiceUrl> with JSON body of
{"key":"value","key2":true,"key3":5}

def serviceMethod(Map params) {
    assert params.body.key=="value"
    assert params.body.key2==true
    assert params.body.key3==5
}
```

Unsecured Web Services

There are times when it is desirable to create a plugin with a publicly available web service that does not require a valid Application Account in order to access it. In these cases, the Unsecured annotation may be used on the plugin web service method. No authentication will be performed on Unsecured web services. An example of using the annotation is given below.

Sample unsecured custom web service

```
@Unsecured
def retrievePublicData(Map params) {
    return [data:["data item 1", "data item 2"]]
}
```

Be cautious in using this annotation as it may potentially present a security risk if sensitive data is returned from the web service.

Returning Errors

In order to signify an error occurred or invalid data was provided, the WebServiceException class may be thrown from any custom web service. This exception contains constructors and fields for a list of messages and a HTTP response code. For example, suppose that the user provided inadequate information. The web service could use the following code to notify the user and prompt them to take action with custom messages.

```
def service(Map params) {
    // Handle invalid input
    if (!params.int('a'))
        throw new WebServiceException("Invalid parameter 'a' specified, please specify
an integer!", 400)
    // Use params.a correctly …
}
```

For the example above, a 400 response code (bad request) would be returned with a response body as follows:

```
{
  "messages":[
    "Invalid parameter 'a' specified, please specify an integer!"
  ]
}
```

If any other exception is thrown from a web service (ie Exception, IllegalArgumentException, etc), a 500 response code will be returned with the following response body:

```
{
  "messages":[
    "A problem occurred while processing the request",
    "Message provided in the exception constructor"
  ]
}
```

See the Responses and Return Codes section for more information on error formats in MWS.

Accessing the HTTP Request Method

The HTTP method used for the request is available from the Map parameters argument. The key used to access it is stored as a static field in PluginConstants called `WEB_SERVICES_METHOD`. The value is a string which can be `GET`, `POST`, `PUT`, or `DELETE`. The following example demonstrates how this could be used with the `WebServiceException` to create a REST API with a plugin.

```
def serviceMethod(Map params) {
    // Check to make sure that this request used the HTTP GET method
    // Throw a 405 error (method not supported) if not
    if (params[PluginConstants.WEB_SERVICES_METHOD]!="GET")
        throw new WebServiceException("Method is not supported", 405)
}
```

6.2.7 Reporting State Data

As long as Moab Workload Manager is configured with MWS as a Resource Manager (RM), plugins may report state information on nodes, virtual machines, and jobs to MWM. This is done through **Reports** that are generated by the plugin and passed to the bundled RM services (Node RM Service, Virtual Machine RM Service, and Job RM Service). Each report is for a specific type of object: node, virtual machine, or job. Each contains current state information on the specific attributes of the type it is for.

Generating Reports

To generate a report, simply create a new instance of a report depending on the type of object to be reported:

Object Type	Report Type
Node	NodeReport
Virtual Machine	VirtualMachineReport
Job	JobReport

Each report has a single required parameter for creating a new instance - the ID of the object which is being reported. Once the report instance has been created, any property may be modified as shown in the API documentation links in the table above. The following example shows the creation of a simple node report and modification of a few properties:

```
public void poll() {
    NodeReport node = new NodeReport("node1")
    node.timestamp = new Date()
    node.image = "centos-5.4-stateless"
    … // Set other properties and persist the report
}
```

Special Cases in Field Values

All complex types, such as Lists, Maps, and objects (not including Enumerated values such as NodeReportState and JobReportState) have default values set for them and are not required to be instantiated before use. For example, the `metrics` property of a node report may be modified as follows:

```
public void poll() {
    NodeReport node = new NodeReport("node1")
    // The following assignments are equivalent in their functionality
    node.features.add("FEAT1")
    node.features << "FEAT2"
    // The following assignments are equivalent in their functionality
    node.metrics.METRIC1 = 4d
    node.metrics["METRIC2"] = 125.5
    … // Set other properties and persist the report
}
```

For the `resources` and `requirements` (jobs only) properties, assignments may be made easily without checking for previously existing values or null objects. For example, resources may be added to the `resources` property simply by accessing it as a Map:

```
public void poll() {
    NodeReport node = new NodeReport("node1")
    node.resources.RES1.total = 10
    node.resources.RES1.available = 3
    node.resources["RES2"].total = 10
    node.resources["RES2"].available = 10
    … // Set other properties and persist the report
}
```

The job report's `requirements` property has some additional handling to allow it to be accessed as a single JobReportRequirement object, such as in the following example:

```
public void poll() {
    JobReport job = new JobReport("job.1")
    job.nodeCountMinimum = 4
    job.processorCountMinimum = 2
    job.requiredNodeFeatures << "FEAT1"
    job.preferredNodeFeatures << "FEAT2"
    … // Set other properties and persist the report
}
```

⚠ Although multiple requirements may be added to the `requirements` list to provide consistent with the MWS Job resource, only the first requirement object's properties will be reported to MWM through the RM interface.

Managing Images for Nodes

In order to have Moab Workload Manager recognize a node as a virtual machine hypervisor, it must have a valid associated Image. In particular, the `image` property on a node report must set to a valid image name. The image's `extensions.xcat.hvType` and `virtualizedImages` properties are then used to report the correct hypervisor type and supported virtual machine images to MWM. If the `image` is invalid, it will be ignored and the node will not be recognized as a hypervisor.

Persisting a Report

After a report has been generated and all desired fields have been updated, the report must be sent to one of the three bundled RM services for persisting. If this is not done, the report will be discarded and will not be considered when reporting state information to MWM. The RM services are shown below according to the object type that they handle:

Object Type	RM Service
Node	Node RM Service
Virtual Machine	Virtual Machine RM Service
Job	Job RM Service

Each service has a `save` method that must be called with a list of reports to persist as shown in the following example.

```
INodeRMService nodeRMService

public void poll() {
    NodeReport node = new NodeReport("node1")
    // Change the state
    node.state = NodeReportState.BUSY
    // Persist
    nodeRMService.save([node])
}
```

Once this is done, the reports will be persisted to MongoDB and will be included in consolidation.

Report Consolidation

During each iteration of Moab Workload Manager's cycle, it will query MWS through the RM interface to access current node, virtual machine, and job information. At this point, all reports are loaded from the database and consolidated into a single report of each object as explained in the Data Consolidation section.

Several points must be noted when considering data consolidation:

- All unset, or null, values for properties on reports are ignored.
- Once consolidation is performed, node and virtual machine reports that were included are removed from the database (see `delete-old` below).
- Job reports are kept in the database until the Job state has been reported as a completed active (see JobReportState) and a configurable amount of time has passed. See the `plugins.job.purge.duration` Configuration parameter.

In some cases it may be desired to query MWS directly for the current consolidated node, virtual machine, and job reports. This may be done using the following URLs which return data in the Wiki interface format (see the Native plugin type for more information).

query	Description
/admin/plugins/moab/clusterQuery	Retrieves consolidated node and virtual machine reports. All VMs will have a `CONTAINERNODE` attribute present.
/admin/plugins/moab/workloadQuery	Retrieves consolidated job reports.

Additionally, URL parameters may be used to modify the output of the query as shown in the table below.

Parameter	Type	Default Value	Description
delete-old	Boolean	false	If true, reports included in the consolidation will be removed from the database as described above. This is mainly used by MWM and should not be used directly.
previous	Boolean	false	If true, the results of the last query with the `delete-old` parameter set to `true` will be returned. Typically this is the result of the last MWM query.

In effect, each iteration of MWM's cycle clears out all old reports using the `delete-old` parameter. Therefore it is recommended to use the default values (`delete-old` and `previous` both set to `false`) in between MWM scheduling iterations (after new reports are generated but before MWM queries for the consolidated report), and to use the `previous` parameter set to `true` after a MWM scheduling iteration has run but no new reports have been generated.

6.2.8 Controlling Lifecycle

At times a plugin developer may wish to modify the current state of a plugin or even create plugins programatically. This may be done with the Plugin Control Service. Operations exist on the service to:

- create plugin instances dynamically with specific configuration.
- retrieve plugin instances by ID or based on configuration properties.
- start or stop plugin instances.
- verify plugin instance configuration.

Creating Plugins

Several methods are provided to allow on-the-fly creation of new plugins. Generally, they allow a plugin with a specific ID and plugin type (as a string or as a Groovy Class) to be created with optional configuration properties. These properties should match the fields in the Plugin API.

If any configuration properties are omitted, the defaults will be used as described in the Setting Default Plugin Configuration section. A boolean value is also returned indicating whether the creation succeeded or not.

Note that the `createPlugin` methods will initialize the plugin for retrieval or usage and attempt to start the plugin if the `autoStart` property is true.

Retrieving Plugins

Plugins may be retrieved by using an ID, querying by plugin type, or even querying based on configuration parameters. Several methods are provided to perform these functions as shown on the Plugin Control Service page.

Starting and Stopping Plugins

Plugins may also be started or stopped on demand. These two methods are exposed directly as `start` and `stop` on the plugin control service. Although each method does not return any data, exceptions are thrown if errors are encountered.

Verifying Plugin Configuration

Finally, the plugin control service may be used to verify plugin configuration at any point instead of just when the plugin is started or modified. This may be useful to attempt to modify plugin configuration directly through the `setConfig` dynamic method and then verify that the new configuration is valid for the plugin. Exceptions are thrown if the plugin or the configuration is invalid.

Examples

If an error state is detected it may be necessary to stop the current plugin instance until corrective action can be taken. This may be done using the following code:

```
package example

import com.ace.mws.plugins.*

class ErrorPlugin {
    IPluginControlService pluginControlService

public void poll() {
        // Error is detected, stop plugin instance!
        try {
            log.warn("An error was detected, trying to stop the plugin ${id}")
            pluginControlService.stop(id)
            log.warn("The plugin was successfully stopped")
        } catch(PluginStopException e) {
            log.error("Plugin instance ${id} could not be stopped", e)
        }
    }
}
```

6.2.9 Accessing MWS REST Resources

Often a plugin type may need to access existing MWS REST Resources in order to extend or complement default MWS functionality. This may be done with the Moab Rest Service, which allows a plugin type developer to utilize the existing Resources documentation to perform these tasks.

All accesses to resources require a HTTP method to use (such as GET, POST, PUT, or DELETE) and a relative URL (such as /rest/jobs). Although it mimics the REST resource interface, no actual requests are made and no data is transmitted through the network.

Authentication

All resources are available to the Moab REST Service, and no authentication or Application Accounts are needed.

⚠ Caution must be used when developing plugin types, as there are no restrictions to what may be done with the Moab REST Service. This is especially true when not utilizing hooks as discussed below.

Hooks

If Pre and Post-Processing Hooks are utilized in MWS, the plugin type developer may choose whether or not they are executed when performing a "request" through the Moab REST service. This is done through the `hooks` option as documented on the Moab Rest Service page.

Examples

This code retrieves a list of all nodes, and is equivalent to the Get All Nodes task.

```
package example

import com.ace.mws.plugins.*
import net.sf.json.*

class RestPlugin {
    IMoabRestService moabRestService

public void poll() {
        def result = moabRestService.get("/rest/nodes")
        // OR with the hook enabled…
        def result = moabRestService.get("/rest/nodes", hooks:true)

assert result instanceof MoabRestResponse
        assert nodes instanceof List

log.debug("Nodes list:")
        nodes.each { JSON node ->
            log.debug(node.id)
        }
    }
}
```

This code adds a flag to a job, and is equivalent to the Modify Job Attributes task. This request also enables the hook (if one is configured) for the "request" and uses a URL parameter. This is the equivalent of making a call to `/rest/jobs/job.1?proxy-user=adaptive`.

```
package example

import com.ace.mws.plugins.*
import net.sf.json.*

class RestPlugin {
    IMoabRestService moabRestService

public void poll() {
        def jobId = "job.1"
        def result = moabRestService.put("/rest/jobs/"+jobId, hooks:true,
params:['proxy-user':'adaptive']) {
            [flags:["RESTARTABLE"]]
        }
        assert result.isSuccess()
    }
}
```

6.2.10 Handling Events

Plugin types may handle specific events by containing methods defined by the conventions below. All events are optional.

The Polling Event

To maintain current information, each plugin is polled at a specified time interval. The following method definition is required to utilize the polling event.

```
void poll() { … }
```

Typically this polling method is used to report node and virtual machine information. By default, the polling interval is set to 30 seconds, but can be modified for all or individual plugins as explained in Plugin Management.

When a polling event occurs, the `poll` method on the target plugin is called. This method may perform any function desired and should typically make calls to the Node RM Service, the Virtual Machine RM Service, and the Job RM Service services to report the current state of nodes and virtual machines. For example, the `poll` method in the Native plugin type is implemented as follows:

This is an extremely simplified version of what is actually implemented in the Native plugin type.

```
INodeRMService nodeRMService;
IVirtualMachineRMService virtualMachineRMService;

public void poll() {
    nodeRMService.save(getNodes());
    virtualMachineRMService.save(getVirtualMachines());
}
```

This simple poll method calls two other helper methods called `getNodes` and `getVirtualMachines` to retrieve node and virtual machine reports. These reports are then sent to the appropriate RM service. See Reporting State Data for more information on the RM services, but the objective of this example is to demonstrate one possible use of the poll event handler. Other plugin types, on the other hand, may use the poll event to update internal data from pertinent resources or make calls to external APIs.

Lifecycle Events

Events are also triggered for certain lifecycle state changes. The following method definitions are required to receive lifecycle events.

```
public void configure() throws InvalidPluginConfigurationException { … }
public void beforeStart() { … }
public void afterStart() { … }
public void beforeStop() { … }
public void afterStop() { … }
```

Each event is described in the table below with the associated state change when the event is triggered.

State Change	Event	Description
configure	Configure	Triggered before `beforeStart` and after the plugin has been configured. May be used to verify configuration and perform any setup needed any time configuration is loaded or modified.
beforeStart	Start	Triggered just before starting a plugin.
afterStart	Start	Triggered just after a plugin has been started.
beforeStop	Stop	Triggered just before stopping a plugin.
afterStop	Stop	Triggered just after stopping a plugin.

Currently, no events are triggered for pausing, resuming, erroring, or clearing errors for plugins.

RM Events

When MWS is configured as a Moab Resource Manager (see Configuring Moab), RM events are sent from Moab to each plugin according to the Routing page. The following method definitions are required to receive these events.

```
public boolean jobCancel(List<String> jobs) { … }
public boolean jobModify(List<String> jobs, Map<String, String> properties) { … }
public boolean jobRequeue(List<String> jobs) { … }
public boolean jobResume(List<String> jobs) { … }
public boolean jobStart(String jobId, String taskList, String username) { … }
public boolean jobStart(String jobId, String taskList, String username, Map<String,
String> properties) { … }
public boolean jobSubmit(Map<String, String> properties) { … }
public boolean jobSuspend(List<String> jobs) { … }
public boolean nodeModify(List<String> nodes, Map<String, String> properties) { … }
public boolean nodePower(List<String> nodes, NodeReportPower state) { … }
public boolean resourceCreate(String type, String id, Map<String, String> attributes) {
… }
public boolean systemModify(Map<String, String> properties) { … }
public List<String> systemQuery(List<String> attributes) { … }
public boolean virtualMachineMigrate(String vmId, String hypervisorId, String
operationId) { … }
```

These calls are equivalent to the Moab RM URLs as described in the Native "Native Plugin Interface Comparison" section. All method definitions are documented in the AbstractPlugin API documentation.

6.2.11 Handling Exceptions

The `com.ace.mws.plugins` package contains several exceptions that may be used and in some cases, should be caught. All exceptions end with "Exception", as in PluginStartException.

There are several specific cases where Exceptions should or can be used:

- The `reload` method on the Plugin Control Service can throw the InvalidPluginConfigurationException to signify that the configuration contains errors.
- Various methods on the Plugin Control Service throw plugin exceptions which must be caught to diagnose errors when creating plugin types.
- Any exception (including the Exception class) can be thrown from a custom web service to display a 500 Internal Server Error to the client requesting the service with the given error message.

6.2.12 Managing SSL Connections

At times it is desirable to load and use self-signed certificates, certificates generated from a single trusted certificate authority (CA), or even simple server certificates. It may also be necessary to use client certificates to communicate with external resources. To ease this process, the SSL Service may be utilized. This service provides methods to load client and server certificates from the filesystem. Methods are also present to aid in creating connections which automatically trust all server certificates and connections.

Several points should be noted when using the SSL Service:

- Certificate files may be in the PEM file format and do not need to be in the DER format (as is typical of Java security).
- Each method returns an instance of SSLSocketFactory, which may then be used to create simple sockets or, in combination with another client library of choice, create a connection.
- If the client certificate password is non-null, it will be used to decrypt the protected client certificate.
- This service is **not** needed when performing SSL communications with trusted certificates, such as those for HTTPS enabled websites that do not have a self-signed certificate.
- If the file name of the certificate file (client or server) is relative (no leading '/' character), it will be loaded from the `mws.certificates.location` Configuration parameter.
    - The default value of `mws.certificates.location` is `MWS_HOME/etc/ssl.crt`.
- Both the client certificate alias and password may be `null`. In this case, the client certificate must not be encrypted and the client certificate's default alias (the first subject CN) will be used.
- The lenient socket factory and hostname verifier automatically trust all server certificates. Because of this, they present a large security hole. Only use these methods in development or in fully trusted environments.

Example

To create a socket to a server that requires a client certificate, the following code may be used.

```
package example

import com.ace.mws.plugins.*

class SSLConnectionPlugin extends AbstractPlugin {
    ISslService sslService

public void poll() {
        // This certificate is not encrypted and will be the only certificate presented
to the
        // connecting end of the socket.
        // This file will be loaded from MWS_HOME + mws.certificates.location +
my-cert.pem.
        String clientCert = "my-cert.pem"

def socketFactory = sslService.getSocketFactory(clientCert, null, null)
        def socket = socketFactory.createSocket("hostname.com", 443)
        // Write and read from the socket as desired…
    }
}
```

To create a HTTPS URL connection to a server that has a self-signed certificate, the following code may be used. Note that this is very typical of client libraries - they have a method to set the SSL socket factory used when creating connections.

```
package example
import com.ace.mws.plugins.*
class SSLConnectionPlugin extends AbstractPlugin {
    ISslService sslService
public void poll() {
        // This certificate represents either the server public certificate or the CA's
certificate.
        // Since the path is absolute it will not be loaded from the MWS_HOME
directory.
        String serverCert = "/etc/ssl/certs/server-cert.pem"
def socketFactory = sslService.getSocketFactory(serverCert)
// Open connection to URL
        HttpsURLConnection conn = "https://hostname.com:443/test"
.toURL().openConnection()
        conn.setSSLSocketFactory(socketFactory)
// Retrieve page content and do with as desired…
        def pageContent = conn.getInputStream().text
    }
}
```

6.2.13 Utilizing Services or Custom "Helper" Classes

There are three general types of services available for use in plugins:

1. Bundled services such as the Moab Rest Service.
2. Custom built translators loaded by convention of their name.
3. Other custom built helper classes registered with Annotations.

These will each be described in this section.

6.2.13.1 Bundled Services

Bundled services are utility classes that are included and injected by default onto all plugin types. It is not required to use any of these services, but they enable several core features of plugin types as discussed in the Utility Services section.

More information may be found on each bundled service in the Quick Reference section under "Plugin Services". See especially the "Usage" page under "Plugin Services" to understand generally how they are to be used.

6.2.13.2 Using Translators

Often a plugin type class file becomes so complex that it is desirable to split some of its logic into separate utility service classes. The most typical use case for this is to split out the logic for "translating" from a specific resource API to a format of data that the plugin type can natively understand and utilize. For this reason, there is a convention defined to easily add these helper classes called "Translators".

Simply end any class name with "Translator", and it will be automatically injected just as bundled services onto plugin types, other translators, or even custom registered components. The injection occurs only if a field exists on the class matching the name of the translator with the first letter lower-cased. For example, a translator class called "MyTranslator" would be injected on plugin types, other translators, and custom components that define a field called "myTranslator" as `def myTranslator` or `MyTranslator myTranslator`.

Do not use two upper-case letters to start the class name of a Translator. Doing this may cause injection to work improperly. i.e. use RmTranslator instead of RMTranslator as the class name.

Be careful not to declare translator and custom component injection such that a cyclic dependency is created.

Example

Suppose that a translator needs to be created to handle a connection to access an external REST resource. The translator could be defined as follows:

```
package example

class ExampleTranslator {
    public int getExternalNumber() {
        def number = … // Make call to external resource
        return number
    }
}
```

A plugin type can then use the translator by defining a field called "exampleTranslator". Note that an instance does not need to be explicitly created.

```
package example

class ExamplePlugin {
    def exampleTranslator
    // OR …
    //ExampleTranslator exampleTranslator

public void poll() {
        // Use the translator
        log.info("The current number is "+exampleTranslator.getExternalNumber())
    }
}
```

To extend the example, the translator may also be injected into another translator:

```
package example

class AnotherTranslator {
    def exampleTranslator

public int modifyNumber(int number) {
        return number + exampleTranslator.getExternalNumber()
    }
}
```

This translator may be used in the plugin type just as the other translator.

6.2.13.3 Registering Custom Components

There are cases where the concept of a "Translator" does not fit the desired use of a utility class. In these cases, it is possible to register any arbitrary class as a component to be injected just as a translator would be. This is done using the Spring Framework's annotation `org.springframework.stereotype.Component`. When this annotation is used, the class is automatically registered to be injected just as translators onto plugin types and translators.

> ⚠ All annotations are available in the dependencies declared by the plugins-commons artifact.

> ⚠ Do not use two upper-case letters to start the class name of a custom component. Doing this may cause injection to work improperly. i.e. use RmUtility instead of RMUtility as the class name.

Changing Scope

By default, when a custom component is injected, only a single instance is created for all classes which inject it. This is referred to as the 'singleton' scope. Another scope that is available is 'prototype', which creates a new instance every time it is injected. This is useful when the class contains state data or fields that are modified by multiple methods. To change the scope, use the `org.springframework.context.annotation.Scope` on the class with a single String parameter specifying 'singleton' or 'prototype'.

Injecting Translators or Components

The need may arise to inject translators or other custom components onto custom components. This is done using the `org.springframework.beans.factory.annotation.Autowired` or `javax.annotation.Resource` annotations. The `Autowired` annotation is used to inject class instances by the type (i.e. `MyTranslator myTranslator`) while the `Resource` annotation is used to inject class instances by the name (i.e. `def myTranslator`). Add the desired annotation to the field that needs to be injected.

> ⚠ There is a known issue with dynamically updating plugin types with typed field injection, such as that required when using the `Autowired` annotation. See the Add or Update Plugin Types section for more information.

> ⚠ Note that using the `Autowired` annotation does injection by type which differs from translator and plugin type injection. These are done by name just as the `Resource` annotation allows. Due to this fact, a type of "def" cannot be used when doing injection onto custom components using the `Autowired` annotation. See the example below.
>
> Injection of custom components *onto* translators and plugin types are still done by name, only fields injected using the `Autowired` annotation are affected.

Be careful not to declare translator and custom component injection such that a cyclic dependency is created.

Example

Suppose that a custom utility class is needed to perform complex logic. A custom component could be defined as follows (notice the optional use of the `Scope` annotation):

```
package example

import org.springframework.stereotype.Component
import org.springframework.context.annotation.Scope

@Component
@Scope("prototype")
class ComplexLogicHandler {
    def handleLogic() {
        … // Perform complex logic and return
    }
}
```

A plugin type or translator could then be defined to inject this component:

```
package example

class CustomPlugin {
    def complexLogicHandler

public void poll() {
        complexLogicHandler.handleLogic()
    }
}
```

Now suppose another custom component needs to use the ComplexLogicHandler in its code. It can inject it using the `Autowired` annotation:

```
package example

import org.springframework.stereotype.Component
import org.springframework.beans.factory.annotation.Autowired

@Component
class AnotherHandler {
    // Note that this is injected by type, so 'def' may not be used
    @Autowired
    ComplexLogicHandler complexLogicHandler

def wrapLogic() {
        complexLogicHandler.handleLogic()
    }
}
```

To perform the same injection but by name (as translators and plugin types are injected), use the `Resource` annotation:

```
package example

import org.springframework.stereotype.Component
import javax.annotation.Resource

@Component
class AnotherHandler {
    // Note that this is injected by name based solely on the name defined in
    //    the annotation.  The name of the field itself does not affect the injection.
    @Resource(name="complexLogicHandler")
    def complexLogicHandler

def wrapLogic() {
        complexLogicHandler.handleLogic()
    }
}
```

6.2.14 Packaging Plugins

Plugin types may be packaged in two different ways to upload to MWS:

1. A simple Groovy file containing a single plugin type definition.
2. A JAR file containing one or more plugin types, translators, and custom components.

While each may be uploaded to MWS using the REST API or the User Interface as described in Add or Update Plugin Types, using a JAR file is recommended. Using a simple Groovy file is useful for testing and generating proof of concept work, but does not allow the use of several features of plugins.

The principles of packaging a plugin type or set of plugin types in a JAR file are very simple. Simply compile the classes and package in a typical JAR file. All classes ending in "Plugin" are automatically attempted to be loaded as a plugin type, all classes ending in "Translator" are attempted to be loaded as a translator, and all classes annotated as a custom component will be attempted to be loaded. It is recommended that a build framework is used to help with compiling and packaging the JAR file, such as Gradle. This makes it easy to declare a dependency on the necessary JAR files used in plugin development and to debug, compile, and test plugin code.

In addition to using utility services such as translators, packaging plugin types in JAR files allows the creation of a single project for multiple related plugin types and bundling of external dependencies. These two features are discussed in the following sections.

6.2.14.1 Plugin Projects and Metadata

Each plugin type has information attached to it, called metadata, which describes the origin and purpose of the plugin type. Additionally, a JAR file may also contain a project file which defines default metadata attributes for all plugin types in the JAR. Initial plugins, or plugins that will be created on loading of the JAR file if they do not exist, are also able to be defined on a project file. In all cases, metadata declared on a plugin type will override the metadata defined on the project file.

To define a project file, simply add a class to JAR file that ends in "Project". This file will attempted to be loaded as the project file. Every field on a project file, and even the file itself, is optional. All available fields are shown in the example below.

```
class SampleProject {
    // Plugin information
    String title = "Sample"
    String description = "Sample plugin types"
    String author = "Adaptive Computing Enterprises, Inc."
    String email = "mws.plugins@adaptivecomputing.com"

// Versioning properties
    String version = "0.1"
    String mwsVersion = "7.1 > *"
    String license = "APACHE"

// Documentation properties
    String issueManagementLink = "http://example.com/ticket-system/sample-plugins"
    String documentationLink = "http://example.com/docs/sample-plugins"
    String scmLink = "http://example.com/git/sample-plugins"

// Plugins that are to be created with these properties only when they do NOT exist
    // This does not override any existing plugin instance configuration
    def initialPlugins = {
        /*
        // Multiple instances of plugins may be defined here.
        // In this case, 'sample' is the id of the plugin
        sample {
                pluginType = "Sample"
                // All properties except for "pluginType" are optional
                pollInterval = 30
                autoStart = true
                config {
                    configParam = "value"
                }
            }
        }
        // Another plugin with an ID of 'sample2'
        sample2 {
            …
        */
    }
}
```

As can be seen, metadata information about the plugin type(s), versions, and documentation are available. These are displayed when viewing plugin information in the User Interface or through the REST API.

Any of these properties except for `initialPlugins` may be overwritten by the plugin type class itself by using static properties. A simple example is shown below.

```
package example

class SamplePlugin {
    // Properties may be typed, untyped, final, or otherwise,
    // but they MUST be static
    static version = "0.2"
    static title = "Sample plugin"
    static description = "This sample plugin is used to demonstrate metadata
information"
    static author = "Separate Division"

… // Rest of the plugin type definition
}
```

Initial Plugins

The initial plugins closure provides the flexibility to insert plugin instances when the JAR is loaded. This occurs at two points: when the plugin JAR is first uploaded to MWS, and when MWS is restarted. As shown in the example above, the ID, pluginType, and other properties may be configured for multiple plugins.

The nature of Groovy closures means that programmatic definition of initial plugins is possible. This may even be based on the MWS application configuration. Two properties are automatically available in the initialPlugins closure:

- `appConfig` - Contains the MWS application configuration. Any configuration parameter is available for access as documented on the Configuration page.
- `suite` - Contains the currently configured suite that MWS is running in. This is equivalent to the `mws.suite` configuration parameter, and is an instance of Suite.

Native Plugin Case Study

The Native JAR file utilizes many of the features discussed above. In the root of the JAR file, a compiled class called NativeProject exists which defines all of the metadata fields, including `initialPlugins`. Trying to create an initial plugin presents two distinct problems:

- The plugin should only be initialized if the suite is CLOUD.
- The plugin type configuration must contain an entry referencing the configured `mws.home.location` parameter, or the configured MWS_HOME location.

The `initialPlugins` closure is defined as follows:

```
import com.ace.mws.plugins.Suite

class NativeProject {
    … // Metadata fields

def initialPlugins = {
        // Only initialize the cloud-native plugin if the suite is CLOUD
        if (suite==Suite.CLOUD) {
            'cloud-native' {
                pluginType = "Native"
                pollInterval = 30
                config {
                    // Use the appConfig property to retrieve the current MWS_HOME
location
                    getCluster = "file://${appConfig.mws.home.location}/etc/nodes.txt"
                }
            }
        }
    }
}
```

6.2.14.2 Managing External Dependencies

External dependencies (e.g. JAR files) may be included and referenced in JAR files. Certain rules must also be followed in order to have the dependencies loaded from the JAR file correctly:

1. The plugin type must bundle all external dependency JARs in the root of the plugin type JAR file.
2. An entry must be included in the `MANIFEST.MF` file that references each of these bundled JAR files as a space separated list:

```
Class-Path: dependency1.jar dependency2.jar dependency3.jar
```

Assuming that these rules are followed and that the plugin type is uploaded using the REST API or the User Interface, the dependent JARs will first be loaded and then the new plugin type and associated files will be loaded.

6.2.15 Example Plugin Types

Several plugin types are provided by Adaptive Computing for use in Moab Web Services. Examples of these include the Native and MSM plugin types.

Please see the Plugin Types item in the Quick Reference menu for all bundled plugin types.

A sample plugin type in Groovy would resemble the following:

```
package sample

import com.ace.mws.plugins.*

class SamplePlugin extends AbstractPlugin {
    static author = "Adaptive Computing"
    static description = "A simple plugin in groovy"
    static version = "0.1"

INodeRMService nodeRMService

public void configure() throws InvalidPluginConfigurationException {
        def myConfig = config    // "config" is equivalent to getConfig() in groovy
        def errors = []
        if (!myConfig.arbitraryKey)
            errors << "Missing arbitraryKey!"
        if (errors)
            throw new InvalidPluginConfigurationException(errors)
    }

public void poll() {
        NodeReport node = new NodeReport("node1")
        node.resources.RES1.total = 5
        node.resources.RES1.available = 5
        node.state = NodeReportState.IDLE
        nodeRMService.save([node])
    }

// Access at /rest/plugins/<id>/services/example-service
    public def exampleService(Map params) {
        return [success:true]
    }
}
```

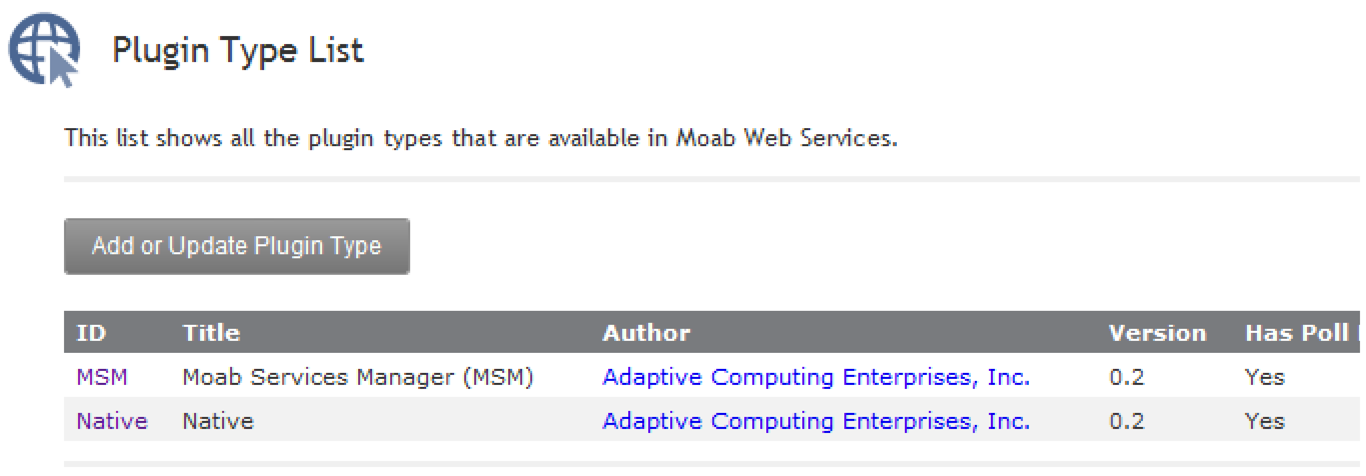
6.3 Plugin Type Management

Plugin types may be managed and accessed with Moab Web Services dynamically, even while running. Operations are provided to upload (add or update) plugin types and to list or show current plugin types. The available fields that are displayed with plugin types are given in the PluginType API. For more information on how these fields are set, see the Plugin Projects and Metadata section.

Plugin Type JAR or groovy files should never be manually copied into the `MWS_HOME/plugins` directory. They must be managed using the methods shown in this section or through the REST API.

6.3.1 Listing Plugin Types

To list all plugin types, browse to the MWS home page (`https://servername/mws` for example). Log in as the admin user, then click **Plugins** and then **Plugin Types**.

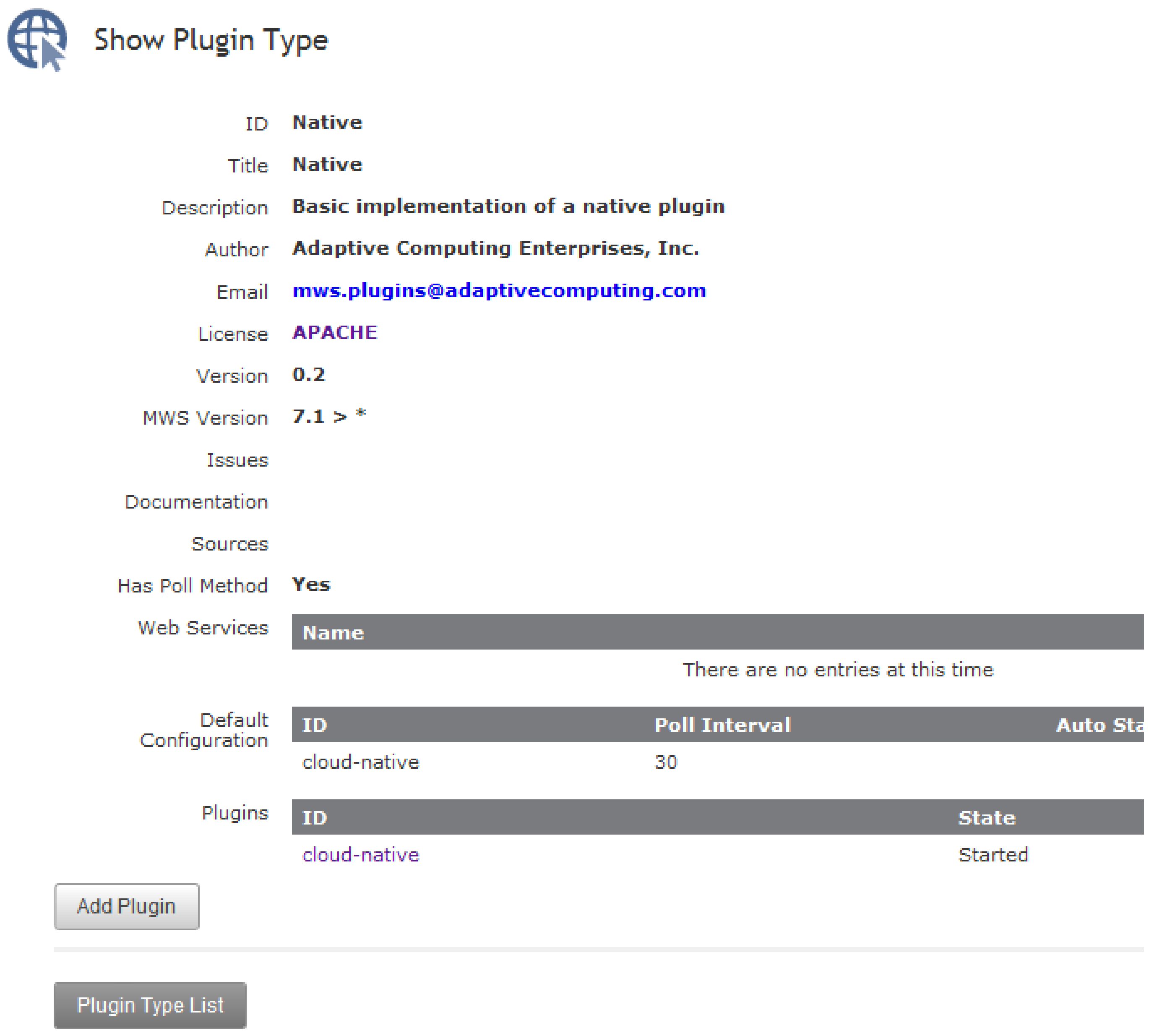

Plugin Type List

This list shows all the plugin types that are available in Moab Web Services.

ID	Title	Author	Version	Has Poll
MSM	Moab Services Manager (MSM)	Adaptive Computing Enterprises, Inc.	0.2	Yes
Native	Native	Adaptive Computing Enterprises, Inc.	0.2	Yes

6.3.2 Displaying Plugin Types

To show information about a plugin type, go to the **Plugin Type List** page and click the desired plugin type.


Show Plugin Type

ID	Native
Title	Native
Description	Basic implementation of a native plugin
Author	Adaptive Computing Enterprises, Inc.
Email	mws.plugins@adaptivecomputing.com
License	APACHE
Version	0.2
MWS Version	7.1 > *
Issues	
Documentation	
Sources	
Has Poll Method	Yes

Web Services

Name
There are no entries at this time

Default Configuration

ID	Poll Interval	Auto Sta
cloud-native	30	

Plugins

ID	State
cloud-native	Started

6.3.3 Add or Update Plugin Types

Plugin types can be uploaded into Moab Web Services using a Groovy file, a Java Archive (JAR) file, or pasted Groovy code. To access the plugin type upload page, navigate to the **Plugin Type List** page and click **Add or Update Plugin Type**. The default interface of this page enables the uploading of a single Groovy class file or a JAR file.

When a plugin type is updated, by default all corresponding plugins created from the plugin type will be recreated. If this behavior is not desired, clear the "Do you want to reload all plugins to use this new version?" checkbox before uploading the plugin type.

There is a known issue when dynamically updating plugin types that use typed fields for injected classes, such as:

```
MyTranslator myTranslator
```

instead of

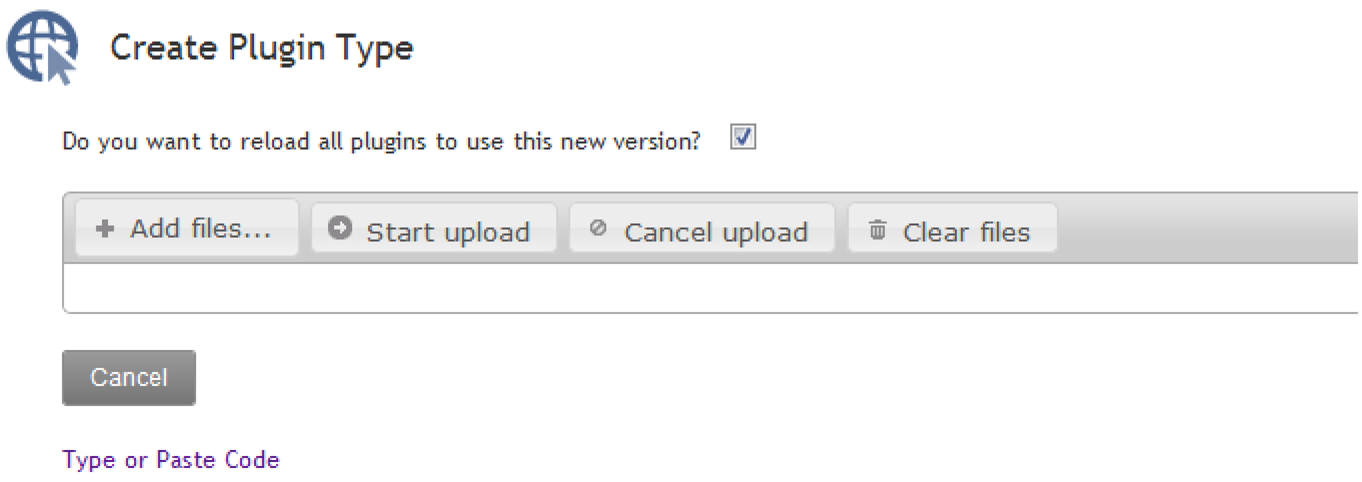
```
def myTranslator
```

Change all injections to 'def' in order to work around the issue. Note that this issue makes injection of instances on custom components using the `Autowired` annotation work improperly since they must be typed. Use the `Resource` annotation as documented in the Registering Custom Components section instead.

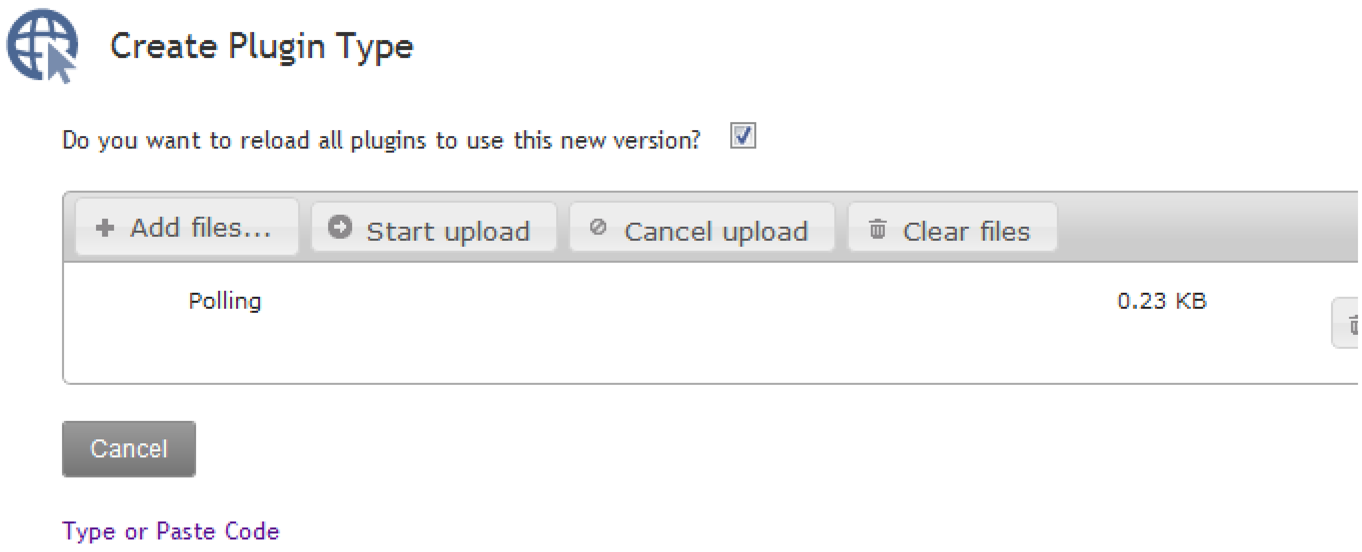
In all cases, and as a last resort, restarting MWS after updating a plugin type resolves the issue.

Single Class File

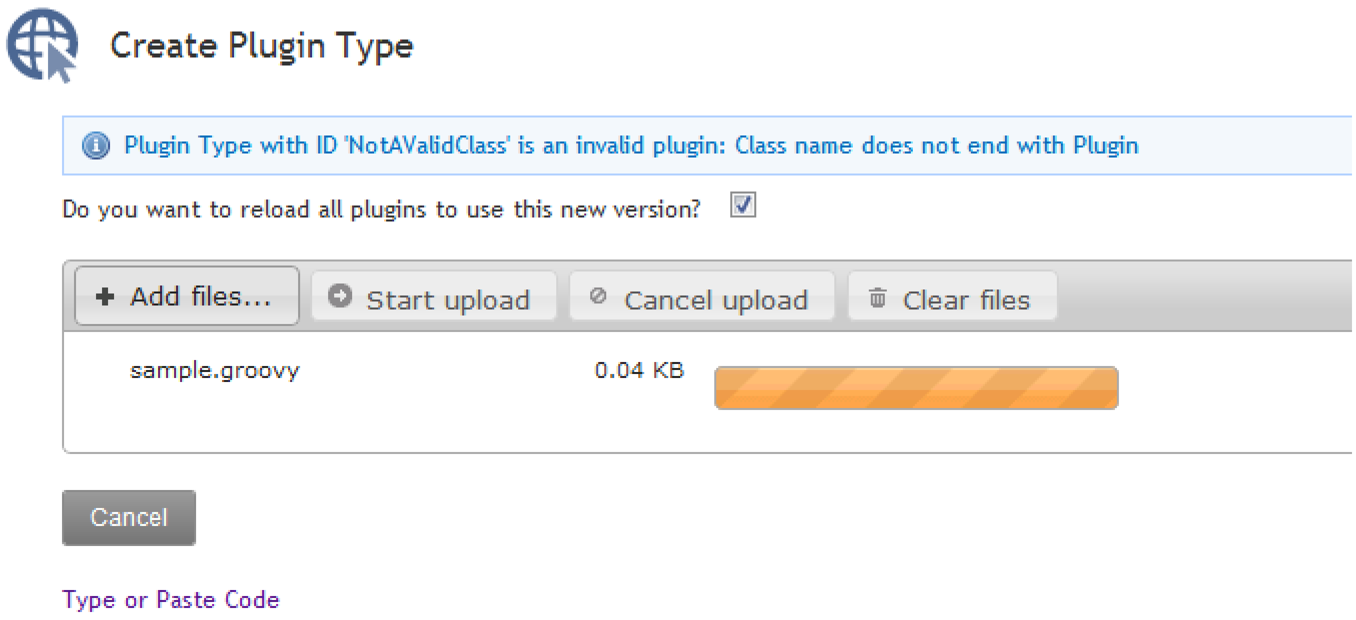
Groovy files containing a single plugin type may be uploaded at the /mws/admin/plugin-types/create URL.



Simply click **Add files...**, select the `.groovy` class file, and click the **Start upload** button. If the plugin type was successfully uploaded and initialized, the size of the file uploaded will be displayed along with the name of the plugin loaded.



If the upload failed or an error occurred during initialization of the plugin, an error message will be displayed.

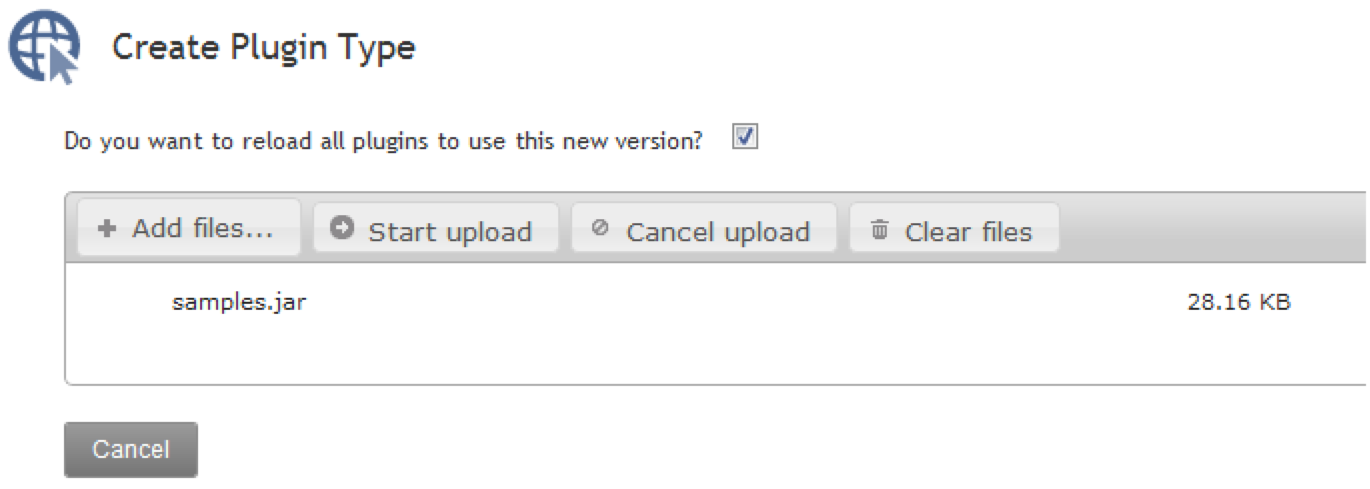

Create Plugin Type
Plugin Type with ID 'NotAValidClass' is an invalid plugin: Class name does not end with Plugin
Do you want to reload all plugins to use this new version? ✓
Add files...
Start upload
Cancel upload
Clear files
sample.groovy
0.04 KB
Cancel
Type or Paste Code

JAR File

A JAR file as described in the Packaging Plugins section containing one or more plugins may also be uploaded using the same process as the Groovy file.

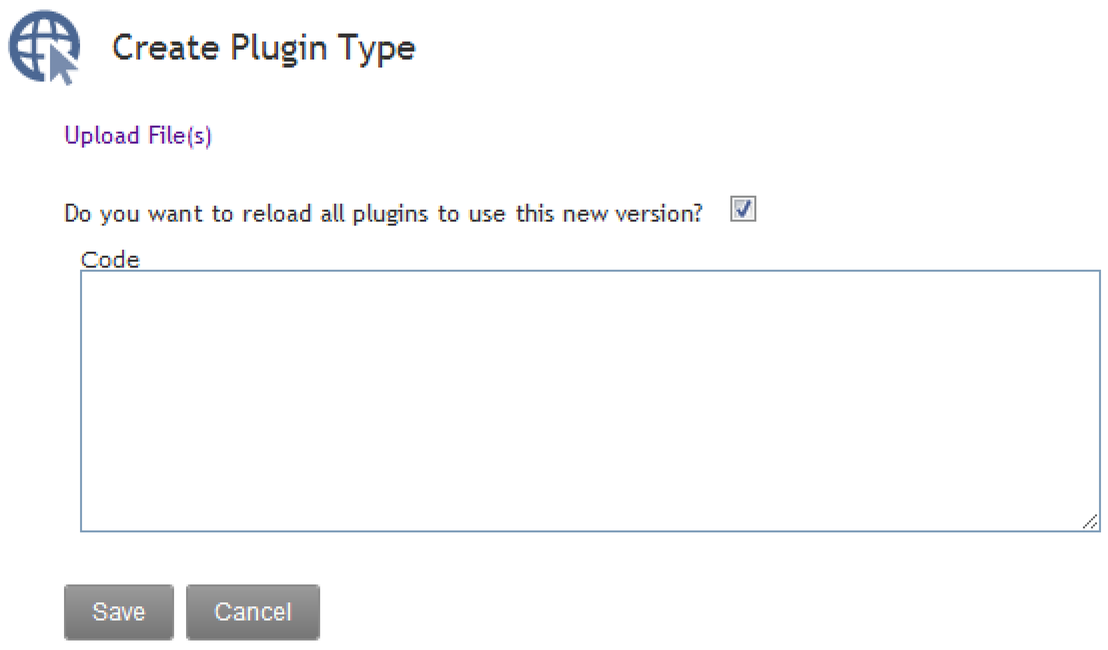
Click **Add files...**, select the `.jar` file, and click the **Start upload** button. If the upload failed or an error occurred during initialization of the plugin(s), an error message will be displayed.

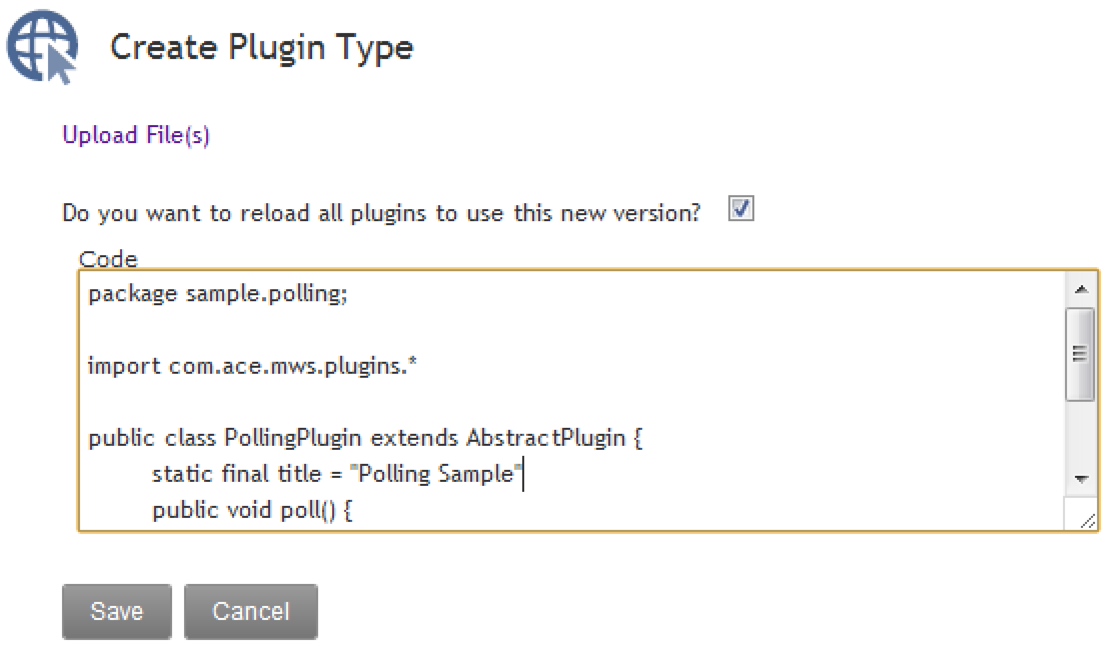
The JAR upload process differs from the single file in that if successful, the name of the JAR file itself is displayed instead of the plugin name(s).



Code

To paste or type code directly into MWS and have it be loaded as a single class file, click **Type or Paste Code** and type or paste the code into the presented text box.


Create Plugin Type
Upload File(s)
Do you want to reload all plugins to use this new version? ✓
Code
Save
Cancel


Create Plugin Type

Upload File(s)

Do you want to reload all plugins to use this new version? ☑

Code

```
package sample.polling;

import com.ace.mws.plugins.*

public class PollingPlugin extends AbstractPlugin {
	static final title = "Polling Sample"
	public void poll() {
```

When the code is in the box, click **Create**. If the upload succeeded and the code was able to be compiled as Groovy, the browser will be redirected to the **Show Plugin Type** page. If the upload failed or an error occurred during compilation or initialization of the plugin, an error message will be displayed.

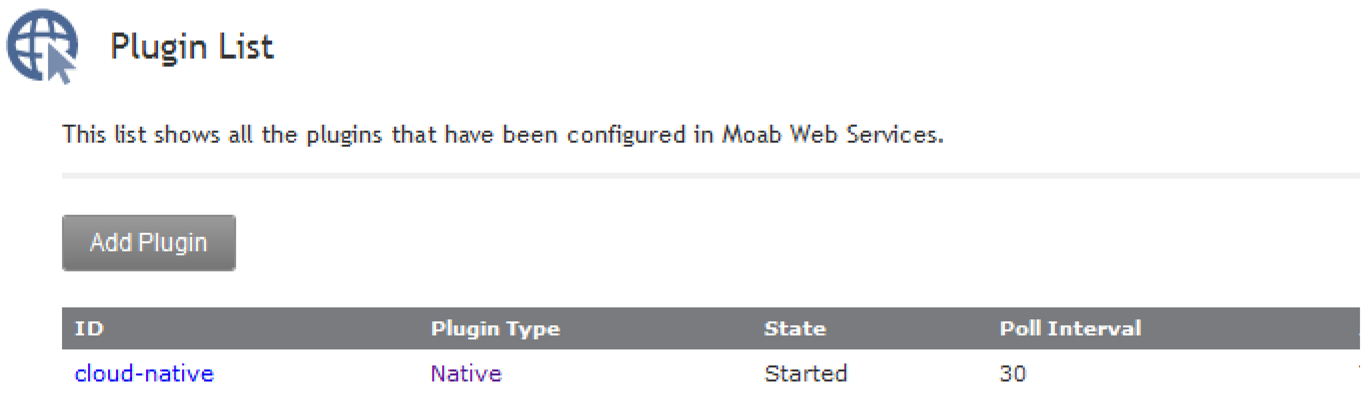
> ⚠ The MWS log file file may need to be referred to for additional details and error messages in the case of a failure.

6.4 Plugin Management

Plugins may be managed and accessed with Moab Web Services dynamically, even while running. This includes plugin instance and lifecycle management. Additionally, default configuration values may be set for new plugins. In order to access custom web services, the REST API must be utilized as described in the Accessing Plugin Web Services section. The available fields that are displayed with plugins are given in the PluginInstance API.

6.4.1 Listing Plugins

To list all plugins, browse to the MWS home page (`https://servername/mws` for example). Log in as the admin user, then click **Plugins** and then **Plugins**.

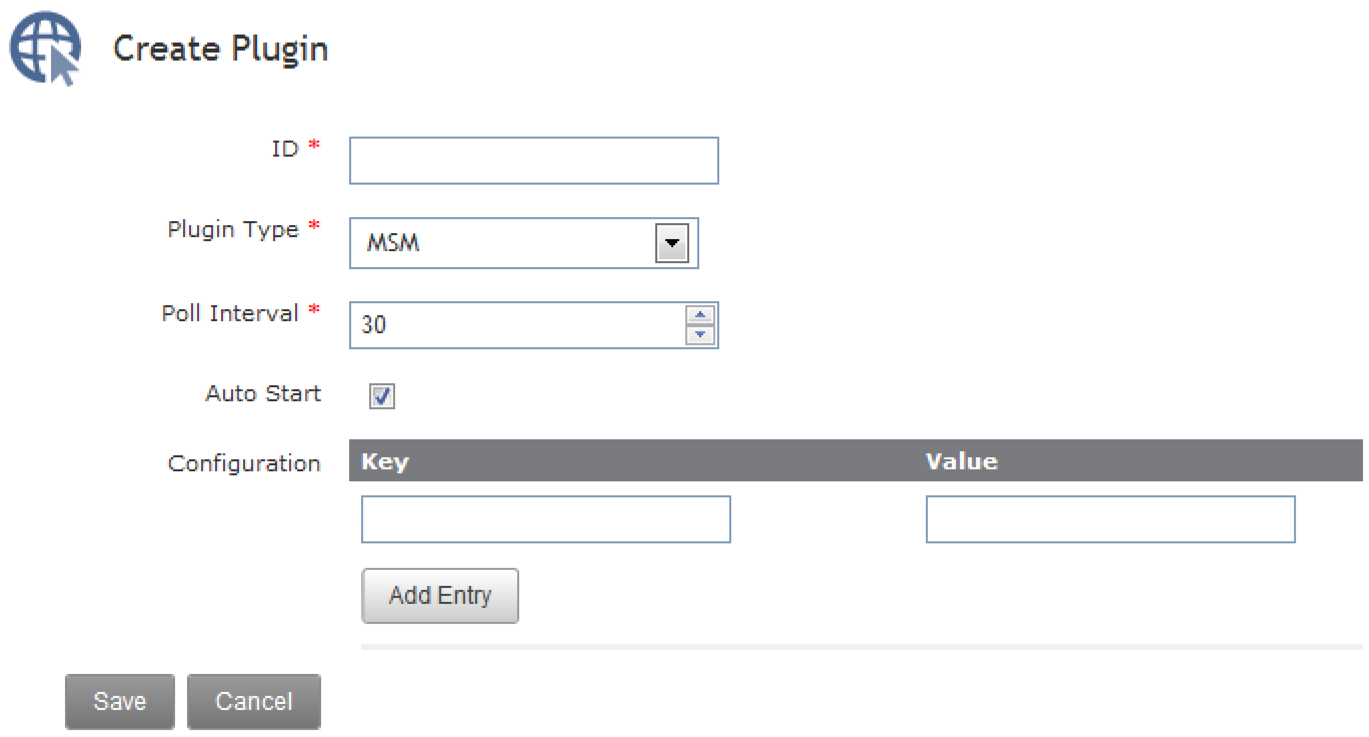

Plugin List

This list shows all the plugins that have been configured in Moab Web Services.

ID	Plugin Type	State	Poll Interval
cloud-native	Native	Started	30

6.4.2 Creating a Plugin

To create a plugin, go to the **Plugin List** page and click **Add Plugin**. The **ID** and **Plugin Type** are required fields. See the PluginInstance API for more information on the fields.

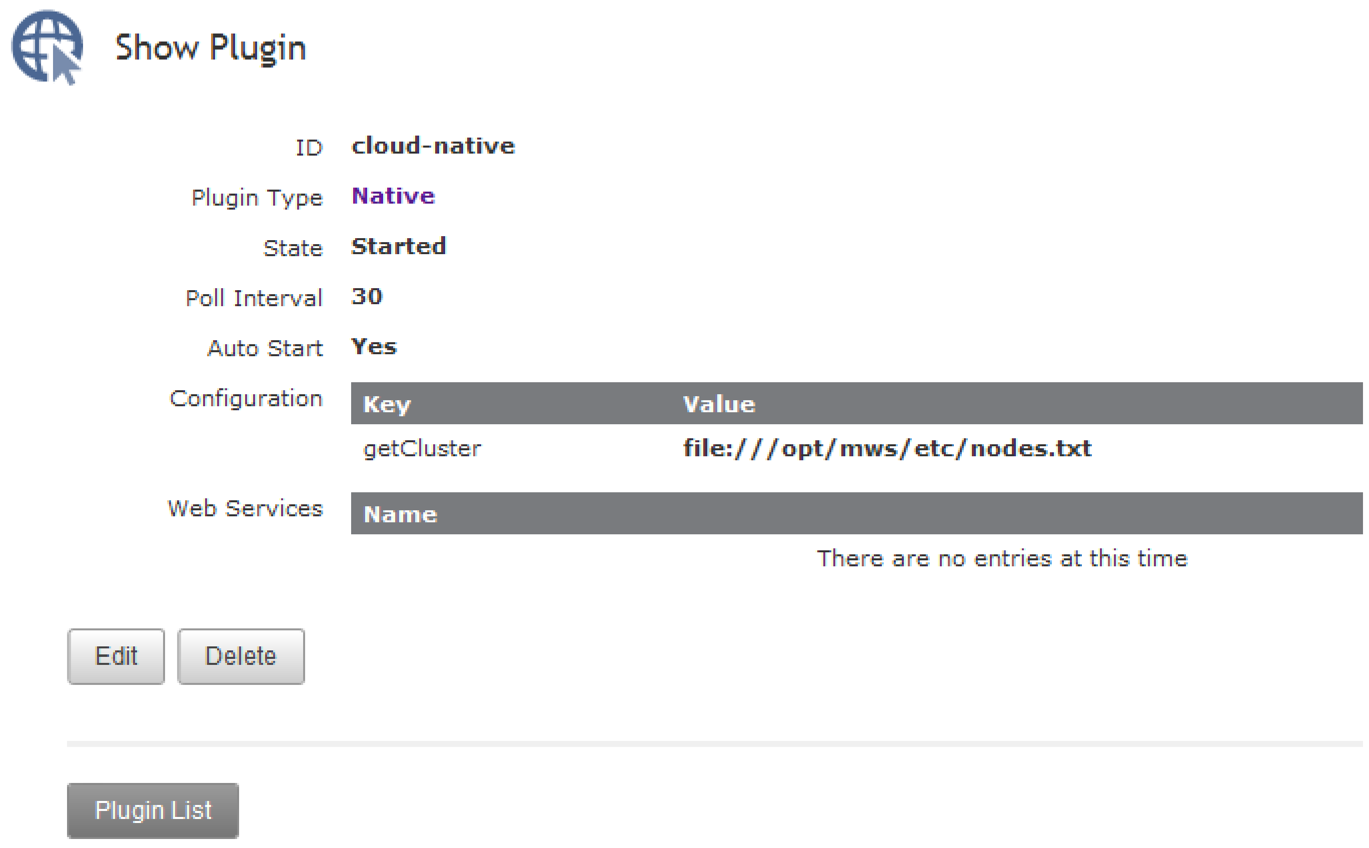


Create Plugin

ID *	
Plugin Type *	MSM
Poll Interval *	30
Auto Start	✓
Configuration	Key / Value

6.4.3 Displaying a Plugin

To show information about a plugin, go to the **Plugin List** page and click the desired plugin ID.



Show Plugin

ID	cloud-native
Plugin Type	Native
State	Started
Poll Interval	30
Auto Start	Yes

Configuration

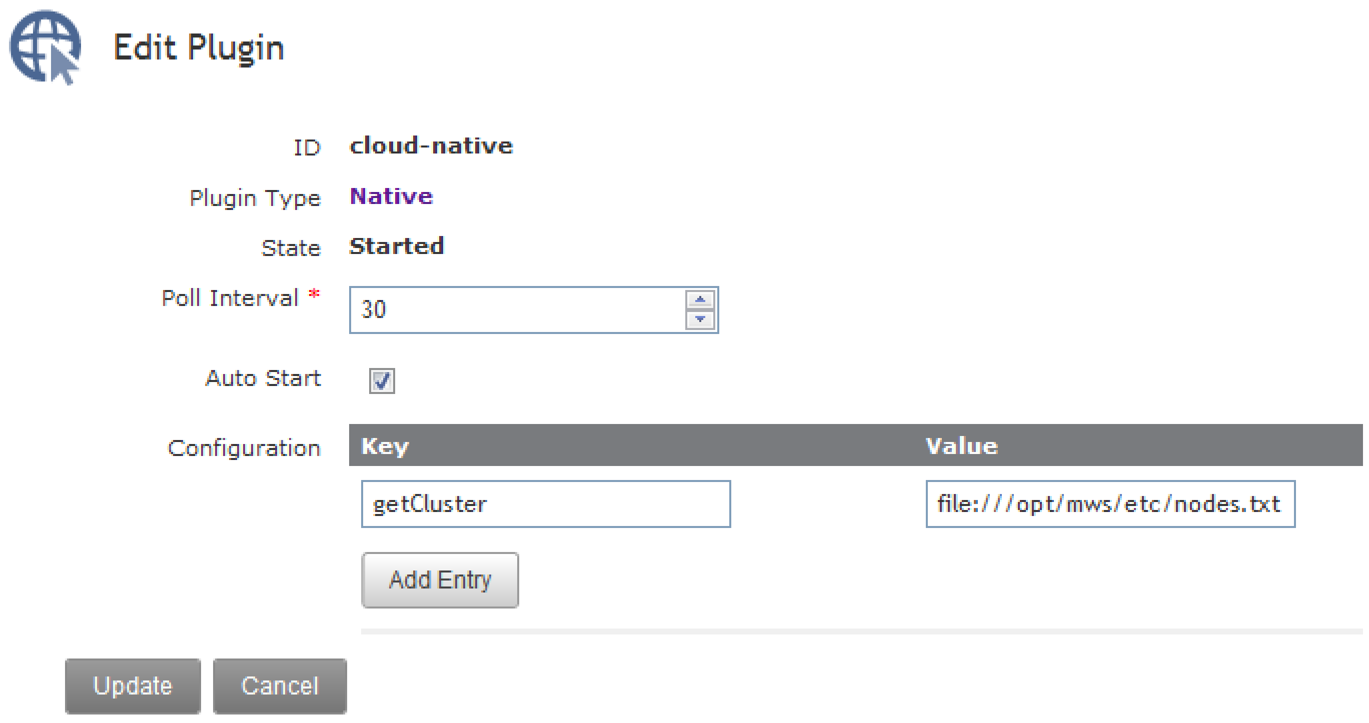
Key	Value
getCluster	file:///opt/mws/etc/nodes.txt

Web Services

Name
There are no entries at this time

6.4.4 Modifying a Plugin

To modify a plugin, go to the **Plugin List** page, click the desired plugin ID, and then click **Edit**. See the PluginInstance API for more information on available fields.



Edit Plugin

ID	cloud-native
Plugin Type	Native
State	Started
Poll Interval *	30
Auto Start	✓

Configuration

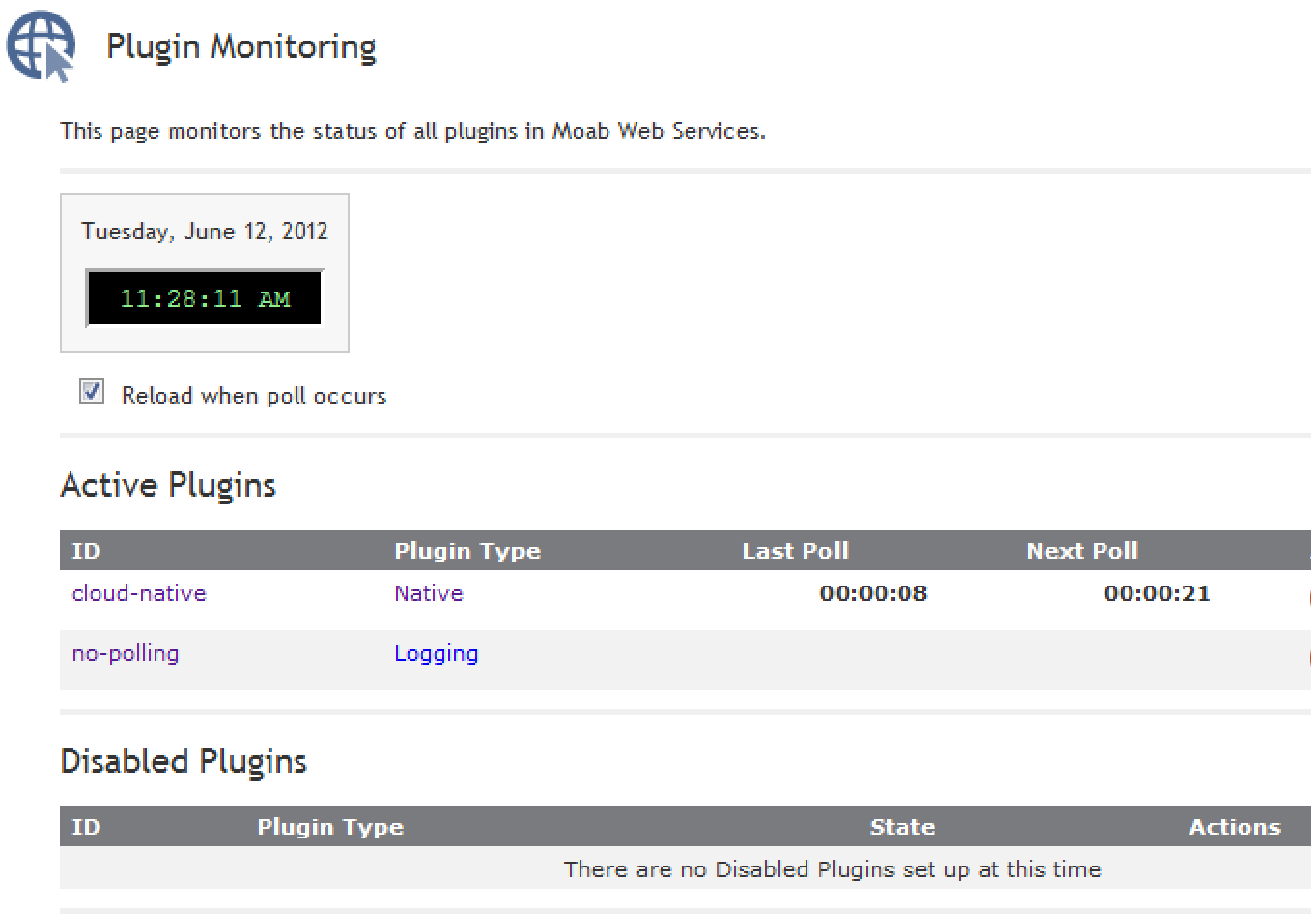
Key	Value
getCluster	file:///opt/mws/etc/nodes.txt

6.4.5 Deleting a Plugin

To delete a plugin, go to the **Plugin List** page, click the desired plugin ID, and then click **Delete**. A confirmation message is shown. If the **OK** button is clicked, the plugin is deleted from the system and cannot be recovered, including all configuration.

6.4.6 Monitoring and Lifecycle Controls

To monitor and control the lifecycle of plugins, browse to the MWS home page (`https://servername/mws` for example). Log in as the admin user, then click **Plugins** and then **Plugin Monitoring**. This page displays the current state of all plugins as well as their polling status.


Plugin Monitoring

This page monitors the status of all plugins in Moab Web Services.

Tuesday, June 12, 2012

11:28:11 AM

☑ Reload when poll occurs

Active Plugins

ID	Plugin Type	Last Poll	Next Poll
cloud-native	Native	00:00:08	00:00:21
no-polling	Logging		

Disabled Plugins

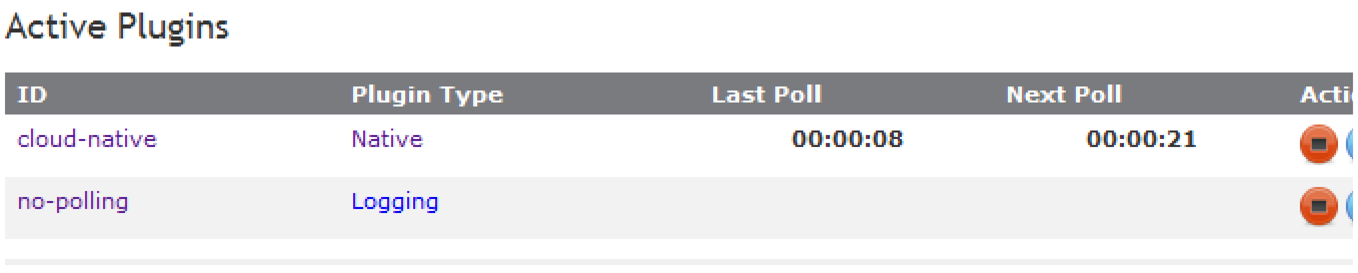
ID	Plugin Type	State	Actions
There are no Disabled Plugins set up at this time			

> ⚠ If plugins are created from plugin types which do not have a `poll` method, their lifecycle controls will be limited. Any information below which mentions polling does not apply to the 'no-polling' plugin shown in the screenshots.

Active Plugins

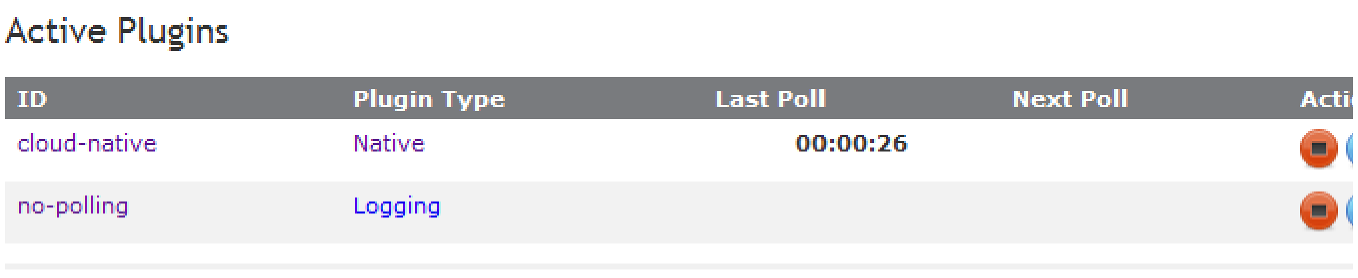
Active plugins are those which are in the Started or Paused states. These are available to receive events such as polling. If paused, a plugin will not receive events but is not actually stopped, therefore no stop events are triggered.

The following images demonstrate the status of plugins in the active states.


Active Plugins

ID	Plugin Type	Last Poll	Next Poll	Acti
cloud-native	Native	00:00:08	00:00:21	
no-polling	Logging			

Started plugins which can include the relative time of the last poll as well as the time of the next poll in a countdown format. Action buttons are available to stop or pause the plugin as well as trigger an immediate poll event.


Active Plugins

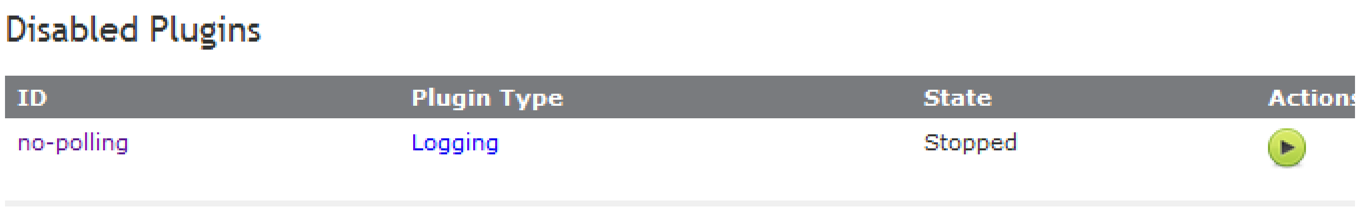
ID	Plugin Type	Last Poll	Next Poll	Acti
cloud-native	Native		00:00:26	
no-polling	Logging			

Paused plugins which can include only the last polling time. Action buttons are available to stop or resume the plugin, as well as trigger an immediate poll event.

Disabled Plugins

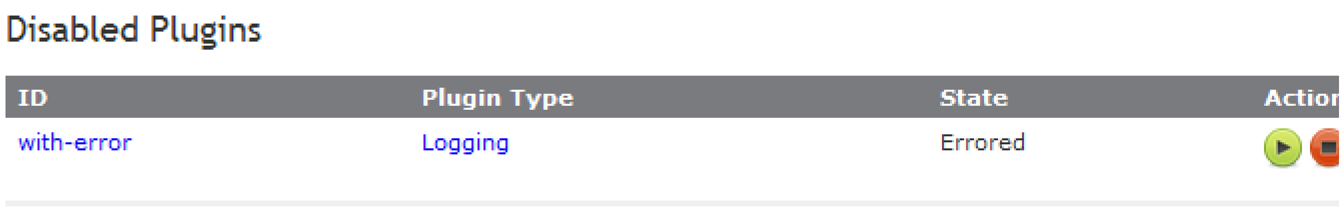
Disabled plugins are those which are in the Stopped or Errored states. These plugins do not receive events such as polling. If errored, a plugin may either be stopped, which represents a "clearing" of the error, or started normally. However, if no action is taken on an errored plugin, it likely will not start due to the fact that most plugins are put into the errored state during startup of the plugin.

The following images demonstrate the representation of plugins in the disabled states.


Disabled Plugins

ID	Plugin Type	State	Actions
no-polling	Logging	Stopped	▶

Stopped plugins. A single action button is available to attempt to start the plugin.


Disabled Plugins

ID	Plugin Type	State	Actio
with-error	Logging	Errored	▶ ■

An errored plugin. As mentioned previously, action buttons are available to stop the plugin or clear the error as well as attempt to start the plugin. If the start fails, an error message will be displayed.

6.4.7 Setting Default Plugin Configuration

Configuration of default values for plugin configuration parameters involves setting fields in the MWS configuration file. These values are used if no values are provided when creating a new plugin. Additionally, the default values will be displayed to the user on the **Create Plugin** page.

The parameters to configure are documented on the MWS Configuration page and comprise most values starting with `plugins`.