Settings Reference

Introduction

Quick Overview

The `settings` element in the `settings.xml` file contains elements used to define values which configure Maven execution in various ways, like the `pom.xml`, but should not be bundled to any specific project, or distributed to an audience. These include values such as the local repository location, alternate remote repository servers, and authentication information.

There are two locations where a `settings.xml` file may live:

- The Maven install: `$M2_HOME/conf/settings.xml`
- A user's install: `${user.home}/.m2/settings.xml`

The former `settings.xml` are also called global settings, the latter `settings.xml` are referred to as user settings. If both files exists, their contents gets merged, with the user-specific `settings.xml` being dominant.

Tip: If you need to create user-specific settings from scratch, it's easiest to copy the global settings from your Maven installation to your `${user.home}/.m2` directory. Maven's default `settings.xml` is a template with comments and examples so you can quickly tweak it to match your needs.

Here is an overview of the top elements under `settings`:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
```

```
    <pluginGroups/>
    <servers/>
    <mirrors/>
    <proxies/>
    <profiles/>
    <activeProfiles/>
  </settings>
```

The contents of the `settings.xml` can be interpolated using the following expressions:

1. `${user.home}` and all other system properties *(since Maven 3.0)*
2. `${env.HOME}` etc. for environment variables

Note that properties defined in profiles within the `settings.xml` cannot be used for interpolation.

Settings Details

Simple Values

Half of the top-level `settings` elements are simple values, representing a range of values which describe elements of the build system that are active full-time.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>${user.home}/.m2/repository</localRepository>
  <interactiveMode>true</interactiveMode>
  <usePluginRegistry>false</usePluginRegistry>
  <offline>false</offline>
  ...
</settings>
```

- **localRepository**: This value is the path of this build system's local repository. The default value is `${user.home}/.m2/repository`. This element is especially useful for a main build server allowing all logged-in users to build from a common local repository.
- **interactiveMode**: `true` if Maven should attempt to interact with the user for input, `false` if not. Defaults to `true`.
- **usePluginRegistry**: `true` if Maven should use the `${user.home}/.m2/plugin-registry.xml` file to manage plugin versions, defaults to `false`. *Note that for the current version of Maven 2.0, the plugin-registry.xml file should not be depended upon. Consider it dormant for now.*
- **offline**: `true` if this build system should operate in offline mode, defaults to `false`. This element is useful for build servers which cannot connect to a remote repository, either because of network setup or security reasons.

Plugin Groups

This element contains a list of `pluginGroup` elements, each contains a groupId. The list is searched when a plugin is used and the groupId is not provided in the command line. This list automatically contains `org.apache.maven.plugins` and `org.codehaus.mojo`.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <pluginGroups>
    <pluginGroup>org.mortbay.jetty</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

For example, given the above settings the Maven command line may execute `org.mortbay.jetty:jetty-maven-plugin:run` with the truncated command:

```
mvn jetty:run
```

Servers

The repositories for download and deployment are defined by the `repositories` and `distributionManagement` elements of the POM. However, certain settings such as `username` and `password` should not be distributed along with the `pom.xml`. This type of information should exist on the build server in the `settings.xml`.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <servers>
    <server>
      <id>server001</id>
      <username>my_login</username>
      <password>my_password</password>
      <privateKey>${user.home}/.ssh/id_dsa</privateKey>
      <passphrase>some_passphrase</passphrase>
      <filePermissions>664</filePermissions>
      <directoryPermissions>775</directoryPermissions>
      <configuration></configuration>
    </server>
  </servers>
  ...
</settings>
```

- **id**: This is the ID of the server *(not of the user to login as)* that matches the `id` element of the repository/mirror that Maven tries to connect to.
- **username**, **password**: These elements appear as a pair denoting the login and password required to authenticate to this server.
- **privateKey**, **passphrase**: Like the previous two elements, this pair specifies a path to a private key (default is `${user.home}/.ssh/id_dsa`) and a `passphrase`, if required. The

`passphrase` and `password` elements may be externalized in the future, but for now they must be set plain-text in the `settings.xml` file.
- **filePermissions**, **directoryPermissions**: When a repository file or directory is created on deployment, these are the permissions to use. The legal values of each is a three digit number corrosponding to *nix file permissions, ie. 664, or 775.

Note: If you use a private key to login to the server, make sure you omit the `<password>` element. Otherwise, the key will be ignored.

Password Encryption

A new feature - server password and passphrase encryption has been added to 2.1.0+. See details on this page

Mirrors

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <mirrors>
    <mirror>
      <id>planetmirror.com</id>
      <name>PlanetMirror Australia</name>
      <url>http://downloads.planetmirror.com/pub/maven2</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
  ...
</settings>
```

- **id**, **name**: The unique identifier and user-friendly name of this mirror. The `id` is used to differentiate between `mirror` elements and to pick the corresponding credentials from the `<servers>` section when connecting to the mirror.
- **url**: The base URL of this mirror. The build system will use this URL to connect to a repository rather than the original repository URL.
- **mirrorOf**: The `id` of the repository that this is a mirror of. For example, to point to a mirror of the Maven `central` repository (`http://repo.maven.apache.org/maven2/`), set this element to `central`. More advanced mappings like `repo1,repo2` or `*,!inhouse` are also possible. This must not match the mirror `id`.

For a more in-depth introduction of mirrors, please read the Guide to Mirror Settings.

Proxies

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <proxies>
```

```
    <proxy>
      <id>myproxy</id>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.somewhere.com</host>
      <port>8080</port>
      <username>proxyuser</username>
      <password>somepassword</password>
      <nonProxyHosts>*.google.com|ibiblio.org</nonProxyHosts>
    </proxy>
  </proxies>
  ...
</settings>
```

- **id**: The unique identifier for this proxy. This is used to differentiate between `proxy` elements.
- **active**: `true` if this proxy is active. This is useful for declaring a set of proxies, but only one may be active at a time.
- **protocol**, **host**, **port**: The `protocol://host:port` of the proxy, seperated into discrete elements.
- **username**, **password**: These elements appear as a pair denoting the login and password required to authenticate to this proxy server.
- **nonProxyHosts**: This is a list of hosts which should not be proxied. The delimiter of the list is the expected type of the proxy server; the example above is pipe delimited - comma delimited is also common.

Profiles

The `profile` element in the `settings.xml` is a truncated version of the `pom.xml` `profile` element. It consists of the `activation`, `repositories`, `pluginRepositories` and `properties` elements. The `profile` elements only include these four elements because they concerns themselves with the build system as a whole (which is the role of the `settings.xml` file), not about individual project object model settings.

If a profile is active from `settings`, its values will override any equivalently ID'd profiles in a POM or `profiles.xml` file.

Activation

Activations are the key of a profile. Like the POM's profiles, the power of a profile comes from its ability to modify some values only under certain circumstances; those circumstances are specified via an `activation` element.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <profiles>
    <profile>
      <id>test</id>
      <activation>
```

```
        <activeByDefault>false</activeByDefault>
        <jdk>1.5</jdk>
        <os>
          <name>Windows XP</name>
          <family>Windows</family>
          <arch>x86</arch>
          <version>5.1.2600</version>
        </os>
        <property>
          <name>mavenVersion</name>
          <value>2.0.3</value>
        </property>
        <file>
          <exists>${basedir}/file2.properties</exists>
          <missing>${basedir}/file1.properties</missing>
        </file>
      </activation>
      ...
    </profile>
  </profiles>
  ...
</settings>
```

Activation occurs when all specified criteria have been met, though not all are required at once.

- **jdk**: `activation` has a built in, Java-centric check in the `jdk` element. This will activate if the test is run under a jdk version number that matches the prefix given. In the above example, `1.5.0_06` will match. Ranges are also supported as of Maven 2.1. See the maven-enforcer-plugin for more details about supported ranges.
- **os**: The `os` element can define some operating system specific properties shown above. See the maven-enforcer-plugin for more details about OS values.
- **property**: The `profile` will activate if Maven detects a property (a value which can be dereferenced within the POM by `${name}`) of the corresponding `name=value` pair.
- **file**: Finally, a given filename may activate the `profile` by the `existence` of a file, or if it is `missing`.

The `activation` element is not the only way that a `profile` may be activated. The `settings.xml` file's `activeProfile` element may contain the profile's `id`. They may also be activated explicitly through the command line via a comma separated list after the `-P` flag (e.g. `-P test`).

To see which profile will activate in a certain build, use the `maven-help-plugin`.

```
mvn help:active-profiles
```

Properties

Maven properties are value placeholder, like properties in Ant. Their values are accessible anywhere within a POM by using the notation `${X}`, where `X` is the property. They come in five different styles, all accessible from the `settings.xml` file:

1. `env.X`: Prefixing a variable with "env." will return the shell's environment variable. For

example, `${env.PATH}` contains the \$path environment variable (`%PATH%` in Windows).
2. `project.x`: A dot (.) notated path in the POM will contain the corresponding element's value. For example: `<project><version>1.0</version></project>` is accessible via `${project.version}`.
3. `settings.x`: A dot (.) notated path in the `settings.xml` will contain the corresponding element's value. For example: `<settings><offline>false</offline></settings>` is accessible via `${settings.offline}`.
4. Java System Properties: All properties accessible via `java.lang.System.getProperties()` are available as POM properties, such as `${java.home}`.
5. `x`: Set within a <properties /> element or an external files, the value may be used as `${someVar}`.

```xml
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <profiles>
    <profile>
      ...
      <properties>
        <user.install>${user.home}/our-project</user.install>
      </properties>
      ...
    </profile>
  </profiles>
  ...
</settings>
```

The property `${user.install}` is accessible from a POM if this profile is active.

Repositories

Repositories are remote collections of projects from which Maven uses to populate the local repository of the build system. It is from this local repository that Maven calls it plugins and dependencies. Different remote repositories may contain different projects, and under the active profile they may be searched for a matching release or snapshot artifact.

```xml
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <profiles>
    <profile>
      ...
      <repositories>
        <repository>
          <id>codehausSnapshots</id>
          <name>Codehaus Snapshots</name>
          <releases>
```

```
            <enabled>false</enabled>
            <updatePolicy>always</updatePolicy>
            <checksumPolicy>warn</checksumPolicy>
          </releases>
          <snapshots>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
            <checksumPolicy>fail</checksumPolicy>
          </snapshots>
          <url>http://snapshots.maven.codehaus.org/maven2</url>
          <layout>default</layout>
        </repository>
      </repositories>
      <pluginRepositories>
        ...
      </pluginRepositories>
      ...
    </profile>
  </profiles>
  ...
</settings>
```

- **releases**, **snapshots**: These are the policies for each type of artifact, Release or snapshot. With these two sets, a POM has the power to alter the policies for each type independent of the other within a single repository. For example, one may decide to enable only snapshot downloads, possibly for development purposes.
- **enabled**: `true` or `false` for whether this repository is enabled for the respective type (`releases` or `snapshots`).
- **updatePolicy**: This element specifies how often updates should attempt to occur. Maven will compare the local POM's timestamp (stored in a repository's maven-metadata file) to the remote. The choices are: `always`, `daily` (default), `interval:X` (where X is an integer in minutes) or `never`.
- **checksumPolicy**: When Maven deploys files to the repository, it also deploys corresponding checksum files. Your options are to `ignore`, `fail`, or `warn` on missing or incorrect checksums.
- **layout**: In the above description of repositories, it was mentioned that they all follow a common layout. This is mostly correct. Maven 2 has a default layout for its repositories; however, Maven 1.x had a different layout. Use this element to specify which if it is `default` or `legacy`.

Plugin Repositories

Repositories are home to two major types of artifacts. The first are artifacts that are used as dependencies of other artifacts. These are the majority of plugins that reside within central. The other type of artifact is plugins. Maven plugins are themselves a special type of artifact. Because of this, plugin repositories may be separated from other repositories (although, I have yet to hear a convincing argument for doing so). In any case, the structure of the `pluginRepositories` element block is similar to the `repositories` element. The `pluginRepository` elements each specify a remote location of where Maven can find new plugins.

Active Profiles

```xml
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  ...
  <activeProfiles>
    <activeProfile>env-test</activeProfile>
  </activeProfiles>
</settings>
```

The final piece of the `settings.xml` puzzle is the `activeProfiles` element. This contains a set of `activeProfile` elements, which each have a value of a `profile` `id`. Any `profile` `id` defined as an `activeProfile` will be active, reguardless of any environment settings. If no matching profile is found nothing will happen. For example, if `env-test` is an `activeProfile`, a profile in a `pom.xml` (or `profile.xml` with a corrosponding `id` will be active. If no such profile is found then execution will continue as normal.