

Maven Parent POMs

What is this ?

We have several parent poms. They pre-configure a whole array of things, from plugin versions to deployment on our infrastructure. They should be used by all public and internal projects at Magnolia.

There is no compelling reason to use these for external (i.e not developed or hosted at Magnolia) projects.

GroupId

Your project should define its own `groupId`. In the past, our parent poms were using `info.magnolia`, which led to a lot of pollution of this namespace. The poms now have their own `groupId`.

Arguments

There are some case where we have(or had) to pass arguments to the release plugin and its forked Maven processes. As from version 30, doing `-Darguments=fooBar` won't work anymore, due to how we configured the staging profile. Use `-DadditionalReleaseArguments` instead.

Properties

There are a number of properties that can be overridden by projects; here's a current list with default values and explanations. There are actually more properties than those listed below; do `mvn help:effective-pom` if you're curious. Those listed below are the properties that are actually *meant* for projects to override.

```
<!-- Production code; used for -source and -target -->
<javaVersion>1.4</javaVersion>

<!-- Defaults to ${javaVersion}, used to tests only -->
<javaVersionForTests>${javaVersion}</javaVersionForTests>

<!-- By default, SCM tags will be prefixed with the project's artifactId.
Typically, in a
    multi-module project, you'll want to override this property to have
more appropriate scm tags
    (e.g "foo-1.2.3" instead of "foo-parent-1.2.3") -->
<scmTagPrefix>@{project.artifactId}</scmTagPrefix>

<!-- These get overridden by our specialized parent poms to deploy releases
to the appropriate repositories.
    The only case where a project would need to change this is if the
project _was_ public and enters maintenance mode.
    In that case, set:
    <distriRepoPrefix>magnolia.public.maintenance</distriRepoPrefix>
    (See http://www.magnolia-cms.com/services/support/maintenance-policy.
html) -->
<distriRepoPrefix>magnolia.public</distriRepoPrefix>
<distriSiteId>${distriRepoPrefix}.sites</distriSiteId>
<distriSiteRoot>dav:https://nexus.magnolia-cms.com/content/sites
/${distriRepoPrefix}.sites</distriSiteRoot>

<!-- To enable or disable specific profiles during releases, use this
property. It is to enable-clover by default. Keep in mind this only works
when set on a whole project, not on sub-modules. -->
<additionalReleaseProfiles>enable-clover</additionalReleaseProfiles>
<!-- Use -DadditionalReleaseArguments instead of -Darguments to pass
arguments to the release Maven forks -->
<additionalReleaseArguments />
```

- [What is this ?](#)
- [GroupId](#)
- [Arguments](#)
- [Properties](#)
 - [Other available properties](#)
- [Site](#)
- [Assemblies, bundles](#)
- [Profiles](#)
 - [Clover](#)
- [Pre-configured plugins](#)
- [Reference](#)

More:

- [BANNED dependencies](#)
- [POMs v30 Release Notes](#)
- [POMs v31 Release Notes](#)
- [POMs v32 Release Notes](#)
- [POMs v33 Release Notes](#)
- [POMs v34 Release Notes](#)
- [POMs v37 Release Notes](#)
- [POMs v38 Release Notes](#)
- [POMs v39 Release Notes](#)

Also see:

- [Maven tips](#)
- [Release process](#)

```

<!-- Staging is enabled by default for the community and enterprise poms.
To enable or disable it, set the following 3 properties need to be
set accordingly (thanks Maven for the absence of boolean logic in property
substitution) -->
<nexusStagingProfileId />
<disableNexusStaging>true</disableNexusStaging>
<enableNexusStaging>false</enableNexusStaging>

<!-- The coverage the build needs to reach to pass. -->
<cloverCoverageThreshold>0</cloverCoverageThreshold>
<!-- Ignore getter and setters by default. -->
<cloverContextFilters>property</cloverContextFilters>
<!-- To skip Clover in all builds, set the following property to false.
Works at sub-module level, but does not prevent Clover from forking a
lifecycle. -->
<skipClover>false</skipClover>

<!-- Set this property to false to build and attach test and test-sources
jars.
This sounds like a double-negative, but much like the
disableNexusStaging/enableNexusStaging properties,
this is due to Maven's inability to "process" properties; we can't do
${!buildTestJar}, for example.
Both plugins involved (jar-plugin and source-plugin) use a "skip"
property, so.. that's it.
Mojo's build-helper plugin doesn't seem to be able to help either,
presumably because it's too late
to reconfigure said plugins.
-->
<skipTestJar>true</skipTestJar>

<!-- Set this property to ${asciidoclet}, ${standardDoclet}, or any other
Javadoc Doclet class.
Don't forget the ${} around the property name you're referring to ! --
>
<javadocDoclet>${standardDoclet}</javadocDoclet>
<!-- The real name of the doclet and the project is "Asciidoclet", but
here's an opinionated choice: I find it less confusing and more consistent
to call this property "Asciidoc Doclet": -->
<asciidocDoclet>org.asciidoctor.Asciidoclet</asciidocDoclet>
<standardDoclet />
<!-- Value for javadoc's -Xdoclint option. Set to 'none' by default. Since
this is only available on JDK8, this is only set via the jdk8 profile. -->
<javadocDoclint>none</javadocDoclint>

<!-- -D properties are normally not able to override plugin configuration
from pom.xml. With these, we work around this limitation. -->
<installAtEnd>true</installAtEnd>
<javadocQuiet>true</javadocQuiet>

<!-- Used in various other properties and plugin configurations; one of
dual, mit, cpal, mna,
generic (avoid the latter). Our specialized parent poms already
override this property. -->
<magnoliaLicenseStyle>dual</magnoliaLicenseStyle>
<!-- Additionally, you might want to change the below if you provide your
own README.txt templates,
checkstyle configuration etc -->
<magnoliaBuildResourcesArtifactId>magnolia-build-
resources-${magnoliaLicenseStyle}-licensed<
/magnoliaBuildResourcesArtifactId>
<magnoliaBuildResourcesVersion>1.6.1</magnoliaBuildResourcesVersion>
<checkstyleHeader>magnolia-build-resources/license-
header-${magnoliaLicenseStyle}.regex</checkstyleHeader>
<!-- If you want to use the default Checkstyle rules but ignore the
license header (don't): -->
<!-- <checkstyleHeader>magnolia-build-resources/license-header-ignored.
regex</checkstyleHeader> -->

<!-- Set this to the ID of one of Maven's default assemblies or one
provided by

```

```

    info.magnolia.maven.assemblies:magnolia-maven-bundle-assemblies. If
you need another assembly,
    you're probably better off skipping the default assembly and re-
configuring the plugin. -->
<defaultAssemblyDescriptor>module-assembly-descriptor<
/defaultAssemblyDescriptor>

<!-- This property can be used by certain bundles to drive the contents of
the README.txt from
    magnolia-build-resources; currently only "tomcat-bundle" is
recognized. -->
<magnoliaProjectBundleType></magnoliaProjectBundleType>

<!-- Projects can use or try a newer version of our site skin. -->
<mavenSiteSkinVersion>1.3</mavenSiteSkinVersion>

<!-- Used for site generation; this is our generic "nexus" GA property. -->
<googleAnalyticsAccountId>UA-6575210-21</googleAnalyticsAccountId>

```

Other available properties

There are also a couple of properties that projects can use to further configure plugins or resource filtering:

- `${parsedVersion}` and `${javaVersion.XXX}`: the parsed version for current project's version, and the current project's value for `<javaVersion>`, respectively. This means one can use `${parsedVersion.majorVersion}` to get 5 if `<version>` is 5.4.2-m2, for example. For details, see the [ParseVersion mojo of the build helper plugin](#). (since v31)
- `${buildScmRevisionNumber}` and `${buildScmBranch}` contain the current Git hash and branch name, respectively. The plugin is pre-configured in our parent poms, but not executed by default, so you need to enable it with the following snippet in your `<build><plugins>` section:

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>buildnumber-maven-plugin</artifactId>
</plugin>

```

Site

The parent poms declare a default location for generated sites which works for single module projects. If you have a multi-module project, you **have to** define the following in your pom. It seems silly, but this solves a number of issues with how Maven generated/resolves URLs for sites.

```

<distributionManagement>
  <site>
    <id>${distribSiteId}</id>
    <url>${distribSiteRoot}/<name-of-your-choice>/${project.version}<
/url>
  </site>
</distributionManagement>

```

For Community modules, please use this snippet instead (note the `/modules/` path element)

```
<distributionManagement>
  <site>
    <id>${distribSiteId}</id>
    <url>${distribSiteRoot}/modules/<name-of-your-choice>/${project.
version}</url>
  </site>
</distributionManagement>
```

... where `<name-of-choice>` will typically be the same as the `scmTagPrefix` property. (current examples: `standard-templating-kit`, `magnolia-ldap`, ...) We can unfortunately not avoid this re-configuration by using this property in the parent's url, because its usage is precisely done to avoid the same issues as those we have with the site deployment: its default value is `project.artifactId($-prefixed properties are substituted when the pom is computed, as opposed to @-prefixed properties, which are substituted at runtime but are specific to the release-plugin)`

At the time of writing, we are +/- consistently using `${project.version}` for all sites, while the `${project.artifactId}` is used similarly to the `scmTagPrefix` property: for multi-module, it sometimes makes sense to "hardcode" the path here instead of using the `artifactId`.

Due to how our parent poms are configured (because we want to deploy sites in a directory named after the project's version), combined with Maven's URL generation/resolving, sites get deployed in what seems like an unnecessary directory: the website for `foobar-1.0` will get deployed under <https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/foobar/1.0/foobar/>

While that's fine for single-module projects, it's not so good for multi-module projects, because their modules are then seemingly spread around: <https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/module-a/1.0/foobar/module-a/>, <https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/module-b/1.0/foobar/module-b/>, and so on ...

More details:

 [BUILD-114](#) - Simplify distributionManagement sections, and provide way to fix site URLs CLOSED

Assemblies, bundles

If your project needs to generate a "bundle" (.zip / .tgz), we have a default configuration for the assembly plugin. Pasting the following in your pom should be enough:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

This will generate a tgz and zip containing your project and it's compiled-scoped dependencies.

In the case of multi-module projects, we typically do this in a dedicated submodule. In this case, you also need to set the assembly-plugin's `appendAssemblyId` property to `false` (or you'll likely end up with a file name `foobar-bundle-1.2.3-bundle.zip`)

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <appendAssemblyId>false</appendAssemblyId>
      </configuration>
    </plugin>
  </plugins>
</build>

```

If you need to use a different assembly descriptor, you can set the `<defaultAssemblyDescriptor>` property, but we'd rather you configure the assembly plugin explicitly:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <executions>
        <!-- First, disable the default-assembly execution : -->
        <execution>
          <id>default-assembly</id>
          <configuration>
            <skipAssembly>true</skipAssembly>
          </configuration>
        </execution>
        <!-- Then configure your own : -->
        <execution>
          <id>bundle-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <skipAssembly>false</skipAssembly>
            <descriptors>
              <descriptor>path/to/your/assembly.xml</descriptor>
            </descriptors>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Profiles

Some profiles are pre-configured in our parent poms. Some are used "behind the scenes", and are not likely to be useful when invoking Maven yourself:

- `release-perform-profile`. Used by/for the release plugin. Stay away.
- `continuous-integration`. Activated on Jenkins/Hudson. Could be useful to debunk some hairy situations, but use with caution; could have unwanted side-effects on your local install.

Some could be, though:

- `show-compiler-warnings - naming` should be explicit
- `enable-clover` - the profile is enabled automatically on Jenkins but can also be used manually. Is also enabled during releases. Generating sites also uses Clover but not via this particular profile.
- `jrebel` - generates `jrebel.xml` files for your project. Recent version of the IDE plugins take care of this AFAIK, but some might still need it.

- staging - see [Staging releases](#).
- purge-magnolia-deps - do `mvn -P purge-magnolia-deps clean` to remove all dependencies (transitive ones too) of the `info.magnolia*` groupIds. To be able to resolve them correctly, Maven needs to download all of them first, so if you're depending on snapshots and/or haven't built the project completely yet, this might create a lot of unnecessary traffic. Sometimes, `rm -rf ~/.m2/repository/info/magnolia` works just as well 🙄.

Clover

By default, Clover is only enabled during the perform phase of the release plugin. It is also enabled on Jenkins by default, via the `BUILD_NUMBER` system properties that Jenkins sets up. To disable it, set the `skipClover` property to `true` in the appropriate pom's `<properties>` section.

This should also work for certain submodules of a multi-module project (such as module bundles - no code, nothing to check coverage for), or entire projects (i.e ce-bundle or ee-bundle).

To disable it on Jenkins, negate the `enable-clover` profile in the job settings with `-P !enable-clover`.

Pre-configured plugins

Many plugins are pre-configured in our parent POMs. Here's a rundown of a few noticeable items

- enforcer: is configured to enforce correct Maven versions, have all plugins with a specified version, ...
- deploy: the deploy plugin is now configured to "deploy at end", which means all your project's artifacts will be uploaded together at the end of a successful build.
- install: likewise, the install plugin is now configured to "install at end". A reactor build failing halfway will result in no artifacts being copied to your local Maven repository.
- animal-sniffer: pre-configured to use signatures derived from your `javaVersion` property.

Reference

Below are the `<parent>` snippets you can use:

```
<parent>
  <groupId>info.magnolia.maven.poms</groupId>
  <artifactId>magnolia-parent-pom-community</artifactId>
  <version>38</version>
</parent>
<parent>
  <groupId>info.magnolia.maven.poms-enterprise</groupId>
  <artifactId>magnolia-parent-pom-enterprise</artifactId>
  <version>38</version>
</parent>
<parent>
  <groupId>info.magnolia.maven.poms-addons</groupId>
  <artifactId>magnolia-parent-pom-addons</artifactId>
  <version>38</version>
</parent>
<parent>
  <groupId>info.magnolia.maven.poms-internal</groupId>
  <artifactId>magnolia-parent-pom-internal</artifactId>
  <version>38</version>
</parent>
<parent>
  <groupId>info.magnolia.maven.poms-project</groupId>
  <artifactId>magnolia-parent-pom-project</artifactId>
  <version>38</version>
</parent>
```