

Repository Metadata

Repository Metadata

Currently we have a number of small metadata files:

- `RELEASE.version.txt` (determines the latest release of the artifact)
- `SNAPSHOT.version.txt` (determines the timestamp/build# of the latest build of the artifact)
- `LATEST.version.txt` (determines the latest build of the artifact, whether it be a timestamp or a release)
- `plugins.xml` (maps plugin prefixes to artifact Id within the group)

With the advent of version ranges, this will require use to get a list of versions for an artifact also. This has gotten to "critical mass" of metadata that we should move to handling general directory metadata.

We can safely remove reading of `plugins.xml` now since there hasn't been a release cut that uses it, and there are not significant amounts to convert in the repo. We should continue to read the others, but deprecate it. They only need to be looked for if the repository metadata file is missing.

Location of metadata

Metadata will be stored at the directory level, allowing different metadata for each granularity. The file will be called `maven-metadata.xml`. Note that the format is the same for each directory, though it is expected that only a subset will be used for each. This is in preference to having 3 different metadata formats.

Version Information for Artifact Directories

```
<metadata>
  <versioning>
    <release>1.0.2</release>
    <latest>1.0.3-SNAPSHOT</latest>
    <versions>
      <version>1.0.0-SNAPSHOT</version>
      <version>1.0.0</version>
      <version>1.0.1-SNAPSHOT</version>
      <version>1.0.1</version>
      <version>1.0.2-SNAPSHOT</version>
      <version>1.0.2</version>
      <version>1.0.3-SNAPSHOT</version>
    </versions>
  </versioning>
</metadata>
```

Version Ordering

The latest and even release metadata may not be needed if we are honoring a versioning scheme as determined from the POM - however, if that scheme changes over time the version order might not be consistent so it is worth keeping here.

It is also possible that the list be ordered and the last is used for latest (and the last non-snapshot for release).

Version Information for Snapshot Version Directories

```
<metadata>
  <versioning>

<snapshot>1.0.3-20051011.123014-5</snapshot>
  </versioning>
</metadata>
```

Plugin Information for Group Directories

```
<metadata>
  <groupId>org.codehaus.mojo</groupId>
  <plugins>
    <plugin>
      <prefix>myPlugin</prefix>

<artifactId>myPlugin-maven-plugin</artifactI
d>
    </plugin>
  </plugins>
</metadata>
```

Local metadata storage

As changes made locally may need to be merged with the remote version, they will be stored in `maven-metadata-local.xml` alongside the other file which is downloaded from the remote repository (when newer).

`maven-metadata-local.xml` will be merged with `maven-metadata.xml` with the local changes being considered newer regardless of time. If this later becomes an issue we can timestamp each entry.

This means that multiple local changes can be made, but when it comes to deploy one, all that is needed is:

- check `maven-metadata.xml` is up to date
- update `maven-metadata.xml`
- deploy `maven-metadata.xml`

- remove entry from `maven-metadata-local.xml`

For removals, they should be removed from `maven-metadata.xml` directly in the local repository. This will not be overwritten until the remote file changes so is persistent long enough and avoids having to have "deleted" semantics in `maven-metadata-local.xml`.

Handling Multiple Remote Repositories

Due to the fact that each remote repository will have a subset of the data, when the metadata is stored locally, the filename will include the *repository key*. Currently, the `id` used to access the repository will serve as the repository key, but in the future a repository will be created by an admin program and metadata stored at its root which gives it a repository key, canonical URL and other such metadata.

For example, a directory in the local repository may look like:

```
maven-metadata-local.xml
maven-metadata-central.xml
maven-metadata-myrepo.xml
```

The `id` will be that of the source repository, so is not changed for mirrors (as mirrors are considered to be identical). This does mean that files may be doubled up if your download `id` and upload `id` differ (a problem that will also be resolved by keys). This is not harmful.

When looking for metadata from a given repository, only the file for that repository (plugin `-local` merged in) will be consulted.

Keeping Up To Date

We will need to get a process running over the repository that monitors its health. We should be able to reuse `repoclean` in an "m2 reporting mode" to do this. It can check for `md5s`, `sha1s` and create and/or warn if they are missing, check if the metadata matches the directory, and so on.

This will be a later feature to add. It could be part of the repo management application so that errors can be shown and rectified via the web interface.