Maven Ecosystem Cleanup

- Drop Plexus Container for JSR-330 + Sisu Guice extension
- Maven
  - Maven Artifact Transfer
  - Maven Plugins
  - Maven SCM
    - Phase 1
    - Phase 2 (post release)
- Maven Wagon
  - Phase 1
  - Phase 2 (post release)
- Questions
  - Maven Resolver (aka Maven Artifact Resolver)
    - Phase 1
    - Phase 2
    - Phase 3 (later)
    - Resolver and Maven
  - Maven Doxia
    - Phase 1
    - Phase 2 (post release)
  - Maven Doxia Sitetools
    - Phase 1
    - Phase 2 (post release)

This page discusses emerging ideas of "clean up" of Maven project and its related sub-projects.

Drop Plexus Container for JSR-330 + Sisu Guice extension

Whenever you "touch" a project, would be good to perform following steps to migrate from old Maven-specific Plexus Container IoC to JSR-330 + Sisu Guice extension:

- Make sure **Plexus AbstractLogEnabled** is not used (extended) any more: Maven since 3.x uses SLF4J API instead.
- Make sure ancient org.codehaus.plexus:plexus-container-default is NOT used as dependency IN ANY SCOPE: use Sisu Plexus Shim instead (but only in test scope) = org.eclipse.sisu:org.eclipse.sisu.plexus (currently 0.3.5, soon 1.0.0), because in general, all shared components should be migrated to JSR330, while use of Plexus Container MAY BE still needed in UTs (due to Plexus Components present in dependencies).
- Make sure **components in project use JSR330 annotations** and not org.codehaus.plexus:plexus-component-annotations or EVEN WORSE, the Plexus QDox Javadoc.
- Use JSR330 (javax.inject) consistently.
- Sisu tutorials:
  - https://eclipse.github.io/sisu.inject/
  - https://eclipse.github.io/sisu.plexus/

Maven

- ... should really drop Maven 2.x support (was it 10 years?), so maven-compat must go.

Maven Artifact Transfer

- in addition to maven-install-plugin and maven-deploy-plugin usage, this is widely used by non-core, non-plugin, maybe "shared component".
- we need to keep its "bridging" capability (see m-artifact-resolver 2.x plans, package renames from Maven version to version).
- unsure about what we want with this: given planned rename, resolver API should not be used directly in upcoming code changes?
- Given baseline increase (decided to go with 3.2.5, and even 3.5+ is considered, see here), the bridging aspect of this library is not useful any more (before Maven 4), so this library should perhaps be proactively dropped whenever a plugin is updated.
- Maven4 has API worked on, that WILL replace currently (3.1-3.9 and even 4.0.x) "safe" direct use (as in 3.1+ there is no sonatype/eclipse package problem, is gone) of resolver.
- Maven 3.0 should be loudly unsupported (mostly is)
- Maven 3.1-3.999 using direct resolver APIs are the only way to go with them, these versions have no "Maven API"
- Maven 4 will come with Maven API, but will provide transition period and support both, Maven 3.1+ way and Maven API
- Cut off is currently undecided (seal off maven internals, stop supporting Maven 3 way), may happen in 4.1, 4.5 or maybe even Maven 5.0

Maven Plugins

- ... should really make up the 10 year lag, and stop depending on maven-compat stuff. Baseline for plugins should be Maven 3.2.5+ line.
- We must forget the "ill fated" Maven 3.0.x line, not only due org.sonatype/org.eclipse Resolver problem (m-artifact-transfer kinda covers it), but also due it's "sealed" Plexus/Sisu implementation and ancient version. We must comprehend, that we have no resources to support 10+ years spanning compatibility. Move on.
- (to be reviewed) mark Maven core dependencies as scope=provided
Note about plugins carrying custom lifecycle mappings: while Plexus XML will work across 3.2.5-3.99 versions, converting lifecycle mapping to JSR330 is not so simple: there was a type erasure change that renders mappings in plugin between 3.2.5 and 3.3.9 broken (well, not trivial to migrate). In this case, simplest is to just "avoid" the breakage, and raise the bar on that plugin to 3.3.9+

See here:
- [https://github.com/apache/maven-rar-plugin/pull/3/commits/8a23374ffdf5e9d592559e0b213a0b6ee6c06](https://github.com/apache/maven-rar-plugin/pull/3/commits/8a23374ffdf5e9d592559e0b213a0b6ee6c06)
- [MNG-5958](https://issues.apache.org/jira/secure/IssueDetail.jspa?ix=5958) - Getting issue details... STATUS

**Maven SCM**

Maven SCM is too big given our resources and contains many providers we cannot test, given they bridging for are non OSS SCMs.

Proposal: keep only those SCM providers that are OSS and not obsolete (like cvs...) in Maven project, and provide a clear path for SCM integrators to maintain their providers in their community. Proposed plan:

**Phase 1**

- mark as @Deprecated all except following providers: git (gitexe, jgit), svn (svnexe), mercurial
  - maven-scm-provider-accurev
  - maven-scm-provider-bazaar
  - maven-scm-provider-clearcase
  - maven-scm-provider-integrity
  - maven-scm-provider-jazz
  - maven-scm-provider-perforce
  - maven-scm-providers-cvs
  - maven-scm-providers-standard (skim POM)
  - maven-scm-provider-starteam
  - maven-scm-provider-synergy
  - maven-scm-provider-tfs
  - maven-scm-provider-vss
- release as-is as next minor version = 1.12.0 [https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12317828&version=12350600](https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12317828&version=12350600)

**Phase 2 (post release)**

- drop all deprecated modules/providers (including svnkit)
  - write down how to move these to a new home?
- up major version: 2.0.0
- make it Java 8?
- provide "clear path" (documentation, repository, mini-site?) for SCM integrators (prev codebase will be still available under release tag done in phase 1)
  - SCM-1002 - Getting issue details... STATUS
- prepare orphaned Git branches for each removed SCM provider with the code as it was before
- Get rid of Plexus DI

**Maven Wagon**

Similarly, Wagon contains many providers we have no feedback whatsoever, or, we do have feedback but are dead-end (JSch/SSH).

**Phase 1**

- mark as @Deprecated all except following providers: file, http, ssh (sshexe), scm
  - wagon-ftp
  - wagon-http-lightweight
  - wagon-ssh
  - wagon-webdav-jackrabbit
- Drop support for JSoup-based file listing
- release as-is as next minor version

**Phase 2 (post release)**

- drop all deprecated modules/providers
- up major version?
- make it Java 8?
- Wagon specific changes, like collapse DAV into HTTP (and drop webdav-jackrabbit), probably collapse http-shared into HTTP (as lightweight is gone as well)
- Get rid of Plexus DI

**Questions**
• if maven-compat goes away, how does Wagon fit/interface with Maven?

Maven Resolver (aka Maven Artifact Resolver)

Phase 1

• release 1.7.0 (soon): DONE (Java8 + locking) https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12320628&version=12349416
• release 1.9.x (to be used in Maven 3.9.0+) - DONE (current 3.9.6 + 1.9.18)

Phase 2

• up major version (2.x)
• Drop ServiceLocator, as it currently forces us to use bad practices: create mutable components, and is just cruft (member population is "circumvented" by init method, no way for ctor injection). Anyway, anyone integrating resolver w/o guice should just manually craft the "component graph", basically all they need is a "factory pattern" w/o the half-ass DI-like cruft in SL (that forces how components are created), see MRESOLVER-157
• make all things proper ctor injection and final (lot of code breakage)

Phase 3 (later)

• Java package change from `org.eclipse.aether` to `org.apache.maven.resolver`? Package change is postponed post Maven 4 and serves the purpose of smoother transitioning from Maven 3 plugins to Maven 4 plugins: as Maven 4.0 will allow still "old style" plugins (using maven-core and other, so 3.x plugins will work with Maven 4.0 unchanged) but at the same time it will introduce new Maven API as well. This "transitioning state" will be kept as long as needed, but at some point (4.1 or later, undecided, maybe even only in 5.0, yet to be seen, to keep kangaroos who overslept last two years calm) Maven will seal off internal access and require plugins to use new Maven API only. That will be the point when 4.x plugin line will require to switch. Renaming packages before this introduces abrupt breakage for plugins. This also means that there will be a "window" from 4.0 to 4.x something where both, 3.x style and 4.x style plugins will both be able to work unchanged, giving time and allowing plugin developers to migrate gradually to new Maven API.

Have to note, that as long as Maven internals are exposed to plugins, we are doomed at "slow pace" as we cannot make huge leaps in progress, as changing private internals (that plugins currently depends on due lack of API) will introduce breakage. IMHO, our and also plugin developers benefit is to perform this switch to new API as fast as possible (so not in 10-ish years).

### Resolver and Maven

<table>
<thead>
<tr>
<th>Maven version line</th>
<th>Java level</th>
<th>Plexus XML</th>
<th>Sisu index</th>
<th>Resolver</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0.x</td>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>org.sonatype.aether:org.sonatype.aether-api:1.11</td>
<td>3.0.5, 2013-02-19</td>
</tr>
<tr>
<td>3.1.x</td>
<td></td>
<td>Yes</td>
<td></td>
<td>org.eclipse.aether:org.eclipse.aether-api:0.9.0.M2</td>
<td>3.1.1, 2013-09-17</td>
</tr>
<tr>
<td>3.2.x</td>
<td>6</td>
<td></td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:1.1.1</td>
<td>3.5.4, 2018-06-21</td>
</tr>
<tr>
<td>3.3.x</td>
<td>7</td>
<td></td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:1.4.1</td>
<td>3.6.3, 2019-11-19</td>
</tr>
<tr>
<td>3.5.x</td>
<td></td>
<td></td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:1.6.3</td>
<td>3.8.8, ?</td>
</tr>
<tr>
<td>3.9.x ?</td>
<td>8</td>
<td></td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:1.9.x (1.9.18)</td>
<td>3.9.6, ?</td>
</tr>
<tr>
<td>4.0.x (old + new Maven API)</td>
<td>8?</td>
<td></td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:2.x (compa</td>
<td>2.x (direct resolver access removed)</td>
</tr>
<tr>
<td>4.1.x (only Maven API)</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>5.x ?</td>
<td>?</td>
<td>No ?</td>
<td></td>
<td>org.apache.maven.resolver:maven-resolver-api:3.x (direct resolver access removed)</td>
<td>?</td>
</tr>
</tbody>
</table>

Note: this table contains latest releases only and just for quick reference. For details history of Maven, go to Maven Releases History page.

### Maven Doxia

Phase 1

• mark as @Deprecated all except following providers: xdoc, fml, apt, markdown, asciidoc, xhtml, xhtml5
  ◦ doxia-logging-api
  ◦ doxia-module-confluence
  ◦ doxia-module-docbook-simple
  ◦ doxia-module-fo
  ◦ doxia-module-itext
  ◦ doxia-module-latex
doxia-module-rtf
  doxia-module-twiki
  release as-is as next minor version = 1.11.1 https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12317230&version=12350888

Phase 2 (post release)
  • drop all deprecated modules/providers/classes
  • up major version?
  • make it Java 8?
  • Get rid of Plexus DI
  • Get rid of Doxia Logging

Maven Doxia Sitetools

Phase 1
  • mark as @Deprecated all except following renderers: site
doxia-doc-renderer
  • release as-is as next minor version = 1.11.1 https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12317320&version=12351046

Phase 2 (post release)
  • drop all deprecated modules/providers/classes
  • up major version?
  • make it Java 8?
  • Get rid of Plexus DI
  • Get rid of Doxia Logging