

# Maven 2.1 Scratchpad

Notes that need to be categorized

- Dependencies of extensions should not be influenced by the dependency requirements of the project.  
MNG-2934
- Transactionality and Shutdown Safety
  - Having a mechanism that writes all files out atomically where we have shutdown hooks to make sure everything is done safely. If all i/o operations pass through a simple Tx manager and the Tx manager is associated with a shutdown hook then we can make sure we don't end up with corrupted files.
- 
- [Artifact Resolution](#)
- Extensions
  - Different categories of extensions: providers vs packaging vs PMD/Checkstyle resources stuff in a JAR
  - Transparent Extension Loading
    - Any Wagon or SCM providers should get picked up automatically from SCM and distributionManagement URLs
    - Any extensions containing packaging/lifecycle related bits need to be picked up automatically
- [Backward Compatibility](#)
  - Provide layer of adapters for plugin backward compatibility, to avoid immediate necessity to recode entire suite of plugins and reports for 2.1 compat.
- Integrity checking
  - Don't allow builds where versions come from non-project sources like local settings and CLI parameters
  - JC: Don't allow builds where versions come from profiles that have to be activated manually
- Queryable Lifecycle
  - The most important change in the embedding environment. You can actually query Maven for the complete execution before it happens
  - I would like to hook in the artifact resolution ghosting to this
    - JC: This tends more toward simulating an entire build, and should include simulating additions to source roots and resources, IMO...but not in the core, in a simulator mojo
- Toolchains
  - Some preliminary work has been done on a branch by Milos and myself, we're basically going to steal all the profile work from Netbeans
  - JC: Do we even have specific requirements documented for this? Where are they?
  - mkleint: the work we've done is not more than a gross sketch. Not sure what is there to be stolen/copied. The usecase for toolchains is basically this:
    - Have a way for plugins to discover what JDK (or other tools) are to be used, without configuring the plugins. The current Maven way of achieving this is to run maven itself with the required JDK. After toolchains, the JDK that maven is running within, shall be irrelevant to the project in question.
- Plugin Testing
  - Results from ApacheCon discussion

- JC: Posted where?
  - We Kenney's and Johns tools which are complementary embedder inside the invoker
- Java5 Annotations
  - We have two implementations for mojos which will be merged
  - We have two implementations for plexus components
- Integration Testing
  - Still not entirely happy with the current setup but it's working and relatively easy to update
  - Too many tools
    - IT plugins: john's with kenney goodies and fold john's into the IT plugins as well
    - Invokers
    - Verifiers
  - JC: IMO, this would benefit greatly from having a decoupled plugin invoker, since it would be really nice to orchestrate several plugins for an IT setup, using a single IT plugin
    - thinking about the component-it-plugin, invoker-plugin, and maybe verifier plugin or similar...
- Custom Components
  - Allowing easy plugging in of new resolvers and such
  - Kenney has finished this and it's working in 2.1
  - JC: IMO we need to address this as a first-class citizen of the container, not simply rewriting the component descriptor for the existing component.
- Specification Dependencies
- POM Changes
  - tags/categories
  - source: Maven, hand bombed and put in the metadata
    - JC: What does this mean? Where the POM came from. A Maven build versus someone doing it manually
  - Coding standard for project
  - dependency excludes && symmetry (JC: consistency?) in the plugins
  - terse attribute based format for the POM
  - properties on dependencies and it was for a C build
    - JC: What was for a C build? What do we need these properties for? If we're thinking that would be a good place to specify prefix for a C dependency, it's not a very scalable approach...
  - tags for dependencies being used by different plugins
    - JC: Can we deprecate scope in favor of this, then? Or, at least trim it back? It seems to me that test scope falls into the tagged-for-surefire category.
    - JC: We should also consider tagging for plugin-execution since a single plugin could be used in multiple ways (think surefire +integration-testing)
  - the profile for an execution environment: development versus production and getting the right specification dependencies, packaging problems
- Plugins
  - Refactor Plugin Manager

- Removal of the Plugin Registry (done)
- Load Plugin dependencies into a separate ClassRealm (done)
- JC: Clean up the API for public consumption (so we can have a good way to orchestrate plugin execution from within a mojo, for example)
- Plugin Execution Environment: Ability to run any version of a plugin where an environment is created which contains all the requirements for a particular version of the Plugin API
  - Expressions: supporting old annotations and allowing for new ones. The expression evaluator would become part of the execution environment
- Plugin API
  - expression
  - DOM related classes into the API
  - JC: function handlers loadable as build extensions
  - JC: XPath expression syntax as alternative to what we have now (maybe look into jxpath)
- Schematron/RelaxNG descriptor for each plugin
  - JC: What's wrong with XSD for this? Far, far more tools support it.
- Remove the use of separate plugin repositories. We only need to pull resources from one repository
  - JC: This forces users with proprietary/custom plugins to run a repository proxy. Why is this critical??
- Symmetric output expressions
- Reporting
  - Report Execution Environment: Ability to run any version of a report where an environment is created which contains all the requirements for a particular version of the Report API
  - Decouple reporting from core
  - JC: Decouple reporting API from Doxia
- Decouple script-based Plugins from the core
  - We should just completely push this out of the core
- Refactor Project Builder
  - Pluggable model readers
    - A new terse format that uses attributes
    - Allow mixin capabilities using an import directive
    - Automatic parent versioning
  - JC: Pipelining of the various steps occurring in the project builder now, according to a strict and well-documented workflow.
- Execution Configuration
  - Remove Settings from the core and make it a user facing configuration
  - Have one configuration model for request
  - Have one configuration model for session: session takes the request in the constructor and delegates
- Domain logging
- Location/Attribute driven behavior
  - JC: This is just going back over the Convention over Configuration stuff, and pushing as much as we can into automated "special" locations where Maven just knows how to handle that sort of content, right?
    - filtering

- velocity pre-processing
- etc?
- Clearly separate project-own dependencies and plugin dependencies
  - the commandline option -U ('update snapshots') needs to either update plugin snapshots+deps, or project deps, not both (MNG-724 - marked won't fix - why?)
- More/Better diagnostic and validation tools
  - better diagnosis for errors
  - better validation of project configurations (dependencies and scopes, plugin config, portability, reproducibility, etc.)