Conditional / context use cases

1. Building the assembly in the first place: Create a new module M, add a package P, add a class C. Oh – now we’re got a condition, and based on that we either will have a property X or not on that class. So we say something like “getContext().beginIfSpecified(“debug”);” to indicate that the following things that we do to the assembly are based on that condition being true. The challenge is that two invocations to the module (or pieces thereof) may have a change to the context that occurred between those two invocations.
2. Using that same assembly to compile against: Load that existing module. Have an assembly context for the new module that is being assembled by the compiler. Hand it to the module to compile against and say “here’s my context; use this!” Now when the module being assembled says “I need module M’s X property on class P.C”, it will succeed if the assembly context has specified “debug” and fail if it has not. The module is not “resolved”, i.e. two questions to the module, one after another, could have different contexts (the context may have changed).
3. Using those two assemblies to link against: Load the first module. Load the second module. There’s some context that is available that says what’s defined. But also, each module needs to have a LinkerContext that is tailored to it, i.e. that can answer all of the LinkerContext questions from the POV of each individual module. The modules are “resolved”, i.e. everything that doesn’t match the context is thrown away.
4. Tool chain: versioning and combining modules.

Versioning

* “dev” – dev builds
* “qa” – test builds
* “alpha” / “beta” – pre-release: alpha / beta / etc. (alpha1, beta2, ..)
* “rc” - release candidate (rc1, rc2, ..)
* release

Do we want to support these concepts as *part of* the version?

Could have v3.1, 3.1dev, 3.1rc2, 3.1

Idea behind a repository is that it handles the abstraction for loading and storing modules.