

POM Interpolation Refactor

[PROPOSAL] Refactored POM Interpolation in Maven 2.0.x

Context

Related JIRA Issues

- [MNG-3535](#)
- [MNG-3536](#)
- [MNG-3475](#)
- [MNG-2562](#)

Related Subversion Feature Branch

- <http://svn.apache.org/repos/asf/maven/components/branches/john-2.0.x-plxInterpolation>

Background

Since before the 2.0 release of Maven, POM interpolation has been essentially a slightly enhanced version of static property resolution, drawing from multiple sources but always substituting static information in place of expressions. The interpolator implementation contained a hard-coded order of operations, specifying in code which sources were checked in which order, without any higher-level processing of any values.

One example where higher-order processing would help had to do with expressions referencing build paths. Specific elements within the POM build section were post-processed to transform relative paths into absolute ones, and remove any sensitivity to the varying base directories that came with multi- versus single-module builds. However, when these build paths were referenced from expressions in arbitrary locations of the POM, tracking these injection points for later path translation was very difficult. This meant that build paths had to be handled in a very specific way, to avoid injecting relative paths into arbitrary locations within the POM then losing track of those injection points when it came time to translate build paths from relative to absolute.

As this interpolation logic evolved through subsequent releases of Maven 2.0.x, it took on aspects of layered resolution coming from command-line properties, system properties, POM properties, and POM instance values (not necessarily in that order). Environment variables from the operating system were added to the mix at one point. Through all of this, elements coming from the POM itself were required to be available in three forms: `${pom.*}`, `${project.*}`, and unprefixed `${*}` expressions. Added to this, environment variables were distinguished from system and other properties using the `${env.*}` expression prefix.

All of these prefix variants and layered value sources strained the original interpolation logic more and more with each successive release, and use cases always dropped through the cracks...new variants with each change, each Maven release.

Problem

At this point, it seems obvious that the original interpolation logic used in Maven 2.0.x is naive and simplistic when compared to the very rich set of sources and operations that need to be applied to resolve POM expressions. To continue patching the original logic - which was never designed for the level of complexity we deal with now when interpolating POMs - is to continue to ask for newly broken use cases, since the solutions for many sets of use cases are directly at odds with another within the current implementation.

Additionally, interpolation has become a widespread feature either present or requested in many Maven plugins, not

to mention applications that use Plexus (the underlying container for Maven). In maintaining separate libraries for supporting interpolation inside Maven's core and from plugins, we only decrease consistency across the whole system, and create more work in terms of maintaining feature parity between the two implementations.

In order to address these problems, a solution should support modularized layering of interpolation strategies. Additionally, to reduce the overhead and inconsistency associated with the current collection of three or more interpolation libraries, a solution here should support both the needs of Maven's core POM interpolation /and/ that of Maven's plugins. These requirements play well off of one another, since plugins often require the traditional value sources required by POM interpolation, plus some additional strategies. One perfect example of this sort of interpolation extension can be found in the Assembly plugin, which allows interpolation from dependency-artifact or module-artifact metadata, in addition to POM metadata.

Proposed Solution

Since the 2.2 refactor of the Assembly plugin, a new interpolation framework has been under development in the plexus-utils library. This framework meets all of the requirements above, interpolating expressions from plain-vanilla String input, and allowing value-resolution strategies to be encapsulated in ValueSource implementations which are then layered in as successive strategies for resolving a particular expression's value. Recent additions go much further than the other interpolation implementations in Maven, even supporting multi-expression cycle detection (a trivial example of this might be: `project.build.testSourceDirectory` references `project.build.sourceDirectory`, which references `project.build.directory`, which references `project.build.testSourceDirectory`). Such multi-expression cycles can be particularly insidious where descriptor inheritance is allowed, for example between configurations in a POM and its parent.

Since Maven's history concerning the plexus-utils library is a bit tortured to say the least, and owing to the maturity of this interpolation framework, it has recently been migrated out of plexus-utils to its own library, called plexus-interpolation. The original classes in plexus-utils have been deprecated, and now depend on a shaded copy of plexus-interpolation to function. Once this library was released as version 1.0 - since it's a stable api that has been through several releases alongside the Assembly plugin already - it became a suitable piece of infrastructure for Maven's core to depend on.

By converting the logic implemented in Maven's current `DefaultModelInterpolator` into a series of ValueSource implementations using plexus-interpolation, the logic involved in interpolating a POM can be encapsulated into a series of targeted strategies that is easy to understand and maintain. By wrapping the `RegexBasedInterpolator` implementation from plexus-interpolation with an implementation of `ModelInterpolator` from maven-project, we can further encapsulate the layering of ValueSources required for standard POM interpolation behind a single, easy-to-use class. At the same time, this `ModelInterpolator` implementation will simply serialize the POM and delegate to plexus-interpolation - reusing the same essential String-oriented interpolation logic available to plugins from this library. In fact, implementing this refactor leaves very little interpolation-related code in maven-project itself. It will only provide a few small ValueSource implementations essential for doing build-path translation on the fly and supplying a build timestamp based on the start date in the `MavenExecutionRequest` used to drive the build. Beyond these, the rest of the Maven-core-specific interpolation code is primarily concerned with configuring the plexus-interpolation library, along with marshaling and unmarshaling the project model.

Notes

Status of plexus-interpolation

Plexus-interpolation is a fully-documented, time-tested interpolation library that has already been shown to work for POM interpolation in Maven's trunk. Its sources can be found at: <http://svn.codehaus.org/plexus/plexus-components/trunk/plexus-interpolation>.

Future releases of the Assembly plugin will be modified to depend on this new library, and the deprecation message

for the old interpolation classes in plexus-utils urges other users to switch to direct use of plexus-interpolation as well.