Dependency Scopes

Added by Jan Dockx, last edited by Eirik Sand on Dec 21, 2006

*This page is started because I (the first author) **don't** understand the dependency scope mechanism. In the first version of this page, I am trying to make sense of different sources about the topic. This means the information here is not correct per definition! The intention of this page is to get it correct and clear, so **please**, if you do know something about dependency scopes, change the nonsense below.*

Introduction to the Dependency Mechanism

Dependencies are defined in the POM, and can be resolved transitively. However, you don't need all dependencies in all situations. That is why dependecies can have a *scope* defined. Furthermore, there is a tag `<optional>true</optional>` you can use for a dependency.

In the following table, we explain the behavior of each scope for different goals. We talk about the **current project**, the **dependency** and a **user project**. The **current project** is the project whose POM we are editing. The **dependency** is a project the **user project** directly depends on. A **user project** is a project for which the **current project** is a direct depency. Consequentially, a **user project** has an indirect depency on the **dependency**. We also presume a maven2-based mechanism (goal) to *run* final projects.

The following *lifecycle phases* are important:

- **compile**: Compile the main source.
- **test**: Compile the test source and run the tests. This requires the main source to be compiled (the **test** goal depends on the **compile** goal).
- ***run***: (non-existing) goal that runs a final artifact. Obviously, this requires the main sources to be compiled (the ***run*** goal depends on the **compile** goal).
- **assembly**: Create an assembly of all kinds off stuff around the artifact. Amongst others, this might contain a `lib`-directory that contains external libraries the **current project** depends on.

Furthermore, it is important to see that some dependencies that are required for compilation (and testing), are optional for using the artifact on a **user project** or for running it if it is a final project itself. Examples are the dependency of *commons-logging* on *log4j* or of *hibernate* on *ehcache* and *c3po* and others. These projects or coded in such a way that they detect whether or not a library is available in the classpath, and use it if it is, but it is not necessary.

Also, indirect dependencies are not necessarily needed for compiling a **user project**: the **current project** might depend on the **dependency** internally, but have no mentioning of the **dependency** in its API that is used by the **user project**. Thus, **dependency** is not needed for compiling the **user project**, but it might be for running the final project.

The following dependency *scopes* are supported:

- **compile**: This **dependency** is needed for compilation of the main source
- **test**: This **dependency** is needed for compiling and running tests. It is not needed for compiling the main source or running the final artifact.
- **runtime**: This **dependency** is needed for running the final artifact. It is not needed for compiling the main source or compiling or running the tests.
- **provided**: This **dependency** is needed for compiling and/or running the artifact but is not necessary to include in the package, because it is *provided* by the runtime environment - for example, jsp-api.jar is provided by your web application container, so you don't include it in your WEB-INF/lib (for the example of a webapp); or a plugin or optional package that is a prerequisite for your application, but is not bundled with your application.
- **system**: This **dependency** is required in some phase of your project's lifecycle, but is system-specific. Use of this scope is discouraged: This is considered an "advanced" kind of feature and should only be used when you truly understand all the ramifications of its use, which can be extremely hard if not actually impossible to quantify. This scope by definition renders your build non-portable. It may be necessarry in certain edge cases. The system scope includes the <systemPath> element which points to the physical location of this dependency on the local machine. It is thus used to refer to some artifact expected to be present on the given local machine an not in a repository; and

whose path may vary machine-to-machine. The systemPath element can refer to environment variables in its path: ${JAVA_HOME} for instance.
- *tag* `<optional />`

Legend:

- U: Download and use **dependency** in the classpath. / Dowload and include **dependency** in the assembly.
- U!O: Download and use **dependency** in the classpath, unless the dependency is `<optional />`. / Dowload and include **dependency** in the assembly, unless the dependency is `<optional />`.
- !: **dependency** is not used

For the **current project**:

scope/phase -->	compile	test	run	assembly
compile	U	U	U	U
test	!	U	!	!
runtime	!	U	U	U
provided	U	!	!	!

For a **user project** that has the **current project** as a dependency with *scope* **compile**:

scope/phase -->	compile	test	run	assembly
compile	U!O	U!O	U!O	U!O
test	!	U	!	!
runtime	!	U!O	U!O	U!O
provided	U!O	!	!	!

For a **user project** that has the **current project** as a dependency with *scope* **test**:

??

For a **user project** that has the **current project** as a dependency with *scope* **runtime**:

??

For a **user project** that has the **current project** as a dependency with *scope* **provided**:

??

Labels None