Merging WAR files

Using Maven2 and Cargo to merge WAR files

If you have an application that you are building with maven2 that you wish to merge two WAR files together, then you may wish to use the 'uberwar' mojo present in the cargo maven2 plugin.

An "uberwar" is a war file constructed from 2 or more War files, where the deployment descriptors in files such as web.xml have been combined.

Setting up your project to output an uberwar

Create a maven2 project whose output artifact is going to be the uberwar. Don't forget to add as dependencies the War files you're going to make the uberwar from.

Your packaging type should be uberwar:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycorp</groupId>
  <artifactId>myproj</artifactId>
  <packaging>uberwar</packaging>
  <version>1.0-SNAPSHOT</version>
  <!-- etc, etc -->
  <dependencies>
    <dependency>
      <groupId>com.mycorp</groupId>
      <artifactId>baseWarFile</artifactId>
      <version>1.0</version>
      <type>war</type>
    </dependency>
    <dependency>
      <groupId>com.mycorp</groupId>
      <artifactId>funkyApp</artifactId>
      <version>1.0</version>
      <type>war</type>
    </dependency>
  </dependencies>
</project>
```

Next, you need to add the cargo maven plugin to your `<plugins>` section, and set it up with the path to a merging descriptor:

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <descriptor>src/assemble/merge.xml</descriptor>
  </configuration>
</plugin>
```

Here the `src/assemble/merge.xml` is the descriptor of how you want the merge to work (see next section for detailed explanations).

Maven Archiver
Note that plugin configuration also supports the `<archive>` element. For full list of supported values see maven-archiver reference page.
Example:

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <descriptor>src/assemble/merge.xml</descriptor>
    <archive>
      <addMavenDescriptor>false</addMavenDescriptor>
      <manifest>
        <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
        <addDefaultSpecificationEntries>true</addDefaultSpecificationEntries>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

Creating a merge descriptor

The merge descriptor file specifies the content of the resulting uberwar file.

Here is a sample of the minimal merge descriptor:

```
<?xml version="1.0"?>
<uberwar>
  <wars>
    <war>com.mycorp:baseWarFile</war>
    <war>com.mycorp:funkyApp</war>
  </wars>
</uberwar>
```

Merge descriptor content

The following merge descriptor elements are supported:

<wars>

In order to know which war files to merge, and in which order (this is significant - for example, when you choose overwrite strategy), specify them in the `<wars>` element. Each `<war>` sub-element should be of form `<groupId>:<artifactId>` to be properly resolved.

<merges>

If you have files that need to be merged together by some kind of processor, then specify it here. The classname can be `org.codehaus.cargo.module.webapp.DocumentMerger`, which is a simple class that will merge by copying all xml nodes from the descriptor files. Alternatively, you can provide any class that is available on the classpath, provided it implements `org.codehaus.cargo.module.merge.MergeProcessor`. The `<merges>` element is optional. You can have as many `<merge>` sub-elements as you need.

For example:

```
<?xml version="1.0"?>
<uberwar>
  ...
  <merges>
    <merge>
      <document>WEB-INF/classes/properties.xml</document>
      <classname>org.codehaus.cargo.module.webapp.DocumentMerger</classname>
    </merge>
  </merges>
</uberwar>
```

Merging the `web.xml` file

The uberwar plugin will merge your `web.xml` file using the default merge strategy even if no explicit definition was provided. Keep reading to learn more about merge strategies!

The ability to merge `web.xml` is probably the most important piece of uberwar functionality. This is achieved by providing a special `<merge>` element, as shown in the example below:

```
<?xml version="1.0"?>
<uberwar>
  ...
  <merges>
    ...
    <merge>
      <type>web.xml</type>
      <parameters>
        <default>
          <tag name="display-name">
            <strategy name="Overwrite"/>
          </tag>
          <tag name="welcome-file-list">
            <strategy name="NodeMerge">
              <welcome-file-list>
                <welcome-file>$left:welcome-file</welcome-file>
                <welcome-file>$right:welcome-file</welcome-file>
              </welcome-file-list>
            </strategy>
          </tag>
          <tag name="context-param">
            <strategy name="ChooseByName">
              <default>
                <strategy name="Preserve"/>
              </default>
              <choice name="contextConfigLocation">
                <strategy name="NodeMerge">
                  <context-param>
                    <param-name>$left:param-name</param-name>
                    <param-value>$left:param-value,$right:param-value</param-value>
                  </context-param>
                </strategy>
              </choice>
            </strategy>
          </tag>
        </default>
      </parameters>
    </merge>
  </merges>
</uberwar>
```

Note the `<default>` element that contains merge strategies by tag and is the only child of `<parameters>` element.

For each tag in the `web.xml`, the descriptor is defining how you want the merge to happen (the 'strategy'). There are 4 out-of-the-box strategies, and you can provide your own by passing the name of a class that implements `org.codehaus.cargo.module.merge.strategy.MergeStrategy`. These strategies define what should happen if an entry is just in one file, or what to do if there is a conflict. When merging, each successive file is merged in sequence - so if there are 3 merges to do, the first two files will be merged together, then the result will be merged with the last file. In each merge operation, the first of the two files is the 'left hand side', and the second of the two files is the 'right hand side' (think of source code merge utilities).

The defaults strategies are:

Preserve

Preserve means "if there is a conflict, then preserve the original".

Overwrite

Overwrite means "if there is a conflict, then use the values in the right-hand side". This is the default strategy if nothing else is specified.

NodeMerge

NodeMerge means "I want to do some merge resolution". The configuration of this strategy is the XML that the conflicting node will be replaced with. Anything starting with `$left:` will be translated as an XPath expression, executed against the left-hand descriptor. The effect of `$right:` is the same for the right-hand side.

ChooseByName

ChooseByName means "most of the time, I want to use one strategy, but for a particular named item, I want to do something else". The configuration of this strategy must contain single `<default>` sub-element (specifying default merge strategy) and multiple `<choice>` sub-elements (specifying merge strategy for specific tag instances).

In the example above:

- the `<display-name>` element from the last war file being merged will be used
- the `<welcome-file-list>` definitions will be merged (assuming that we are merging two war files, each one specifies at most single welcome file)
- the list of Spring configuration files will be merged, the rest of `<context-param>` elements will be preserved

Known Problems

- If you build as a part of a multiproject build, maven2 may incorrectly store your output war file in the repository with a `.uberwar` extension. It does not do this if the project is built on its own.
- If you're experiencing issues with the plugin because Maven is telling you that some Xerces classes cannot be found, please also add `xerces:xercesImpl` to the plugin's dependencies. That error should not be happening as of Java 6.