Custom Profile Activators

Custom profile activators allow you to write your own implementation of <tt>org.apache.maven.profiles.activation.ProfileActivator</tt>, import it into a project as a build extension, and use it to activate profiles in your build.

Example

We can declare the profile that uses our new custom activator in the parent POM, along with the extension containing its implementation:

```
<?xml version="1.0"?>
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.myco</groupId>
  <artifactId>parent</artifactId>
  <version>2.1-SNAPSHOT</version>
  <packaging>pom</packaging>

  <build>
    <extensions>
      <extension>
        <groupId>com.myco.maven</groupId>
        <artifactId>custom-profile-activators</artifactId>
        <version>1.0</version>
      </extension>
    </extensions>
  </build>

  <properties>

<distArchiveBaseName>${project.artifactId}-${project.version}</distArchiveBaseName>
  </properties>

  <profiles>
    <profile>
      <id>versionless-archive</id>

      <activation>
        <custom>
          <type>modelProperty</type>
          <configuration>
            <name>uses-versionlessArchive</name>
            <value>true</value>
          </configuration>
        </custom>
      </activation>

      <properties>
          <distArchiveBaseName>${project.artifactId}</distArchiveBaseName>
      </properties>


    </profile>
  </profiles>

</project>
```

Then, to activate the <tt>versionless-archive</tt> profile in our child POM, we add the specified POM property, like this:

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.myco</groupId>
    <artifactId>parent</artifactId>
    <version>2.1-SNAPSHOT</version>
  </parent>

  <artifactId>myutil</artifactId>

  <properties>
    <uses-versionlessArchive>true</uses-versionlessArchive>
  </properties>
</project>
```

This makes it a simple task to customize your projects by using profiles as mix-ins. These profiles can be maintained at a single point, in the parent POM.

Finally, the custom profile activator might look like this:

ModelPropertyProfileActivator.java

```
package org.codehaus.mojo.profile;

import java.util.Iterator;
import java.util.Properties;

import org.apache.maven.context.BuildContextManager;
import org.apache.maven.model.Model;
import org.apache.maven.model.Profile;
import org.apache.maven.profiles.activation.ProfileActivator;
import org.apache.maven.project.build.ProjectBuildContext;
import org.apache.maven.project.build.model.ModelLineage;
import org.codehaus.plexus.logging.LogEnabled;
import org.codehaus.plexus.logging.Logger;
import org.codehaus.plexus.logging.console.ConsoleLogger;

/**
 * @plexus.component role="org.apache.maven.profiles.activation.ProfileActivator"
 *                   role-hint="modelProperty"
 *                   instantiation-strategy="per-lookup"
 *
 * @author jdcasey
 */
public class ModelPropertyProfileActivator
    implements ProfileActivator, LogEnabled
{

    /**
     * Configured as part of the CustomActivator's setup of this ProfileActivator,
before the
     * CustomActivator delegates the profile-activation process to it. This IS a
required element,
```

```
     * and it can be reversed (negated) using a '!' prefix. Reversing the name means
one of two things:
     * <br/>
     * <ul>
     *   <li>If the value configuration is null, make sure the property doesn't exist
in the lineage.</li>
     *   <li>If the value configuration does exist, make sure the retrieved value
doesn't match it.</li>
     * </ul>
     *
     * @plexus.configuration
     */
    private String name;

    /**
     * Configured as part of the CustomActivator's setup of this ProfileActivator,
before the
     * CustomActivator delegates the profile-activation process to it. This is NOT a
required element,
     * and it can be reversed (negated) using a '!' prefix.
     *
     * @plexus.configuration
     */
    private String value;

    /**
     * @plexus.requirement
     */
    private BuildContextManager buildContextManager;

    // initialized by the container, or lazily.
    private Logger logger;

    public ModelPropertyProfileActivator()
    {
        // provided for Plexus activation
    }

    protected ModelPropertyProfileActivator( String name, BuildContextManager
buildContextManager )
    {
        this.name = name;
        this.buildContextManager = buildContextManager;
    }

    protected ModelPropertyProfileActivator( String name, String value,
BuildContextManager buildContextManager )
    {
        this.name = name;
        this.value = value;
        this.buildContextManager = buildContextManager;
    }

    public boolean canDetermineActivation( Profile profile )
    {
        ProjectBuildContext projectContext =
ProjectBuildContext.getProjectBuildContext( buildContextManager, false );

        if ( checkConfigurationSanity() && projectContext != null )
```

```
        {
            return projectContext.getModelLineage() != null;
        }

        return false;
    }

    private boolean checkConfigurationSanity()
    {
        return name != null;
    }

    public boolean isActive( Profile profile )
    {
        // currently, just make sure the name configuration is set.
        if ( !checkConfigurationSanity() )
        {
            getLogger().debug( "Skipping profile: " + profile.getId() + ". Reason:
modelProperty activation is missing 'name' configuration." );
            return false;
        }

        ProjectBuildContext projectContext =
ProjectBuildContext.getProjectBuildContext( buildContextManager, false );

        if ( projectContext == null )
        {
            return false;
        }

        ModelLineage lineage = projectContext.getModelLineage();

        if ( lineage == null )
        {
            return false;
        }

        String propertyName = name;
        boolean reverse = false;

        if ( propertyName.startsWith( "!" ) )
        {
            reverse = true;
            propertyName = propertyName.substring( 1 );
        }

        String checkValue = value;
        if ( checkValue != null && checkValue.startsWith( "!" ) )
        {
            reverse = true;
            checkValue = checkValue.substring( 1 );
        }

        boolean matches = false;

        // iterate through the Model instances that will eventually be calculated as
one
        // inheritance-assembled Model, and see if we can activate the profile based
on properties
```

```
        // found within one of them. NOTE: iteration starts with the child POM, and
goes back through
        // the ancestry.
        for ( Iterator it = lineage.modelIterator(); it.hasNext(); )
        {
            Model model = (Model) it.next();

            getLogger().debug( "Searching model: " + model.getId() + " for property: "
+ propertyName + " having value: " + checkValue + " (if null, only checking for
property presence)." );

            Properties properties = model.getProperties();

            if ( properties == null )
            {
                getLogger().debug( "no properties here. continuing down the lineage."
);
                continue;
            }

            String retrievedValue = properties.getProperty( propertyName );

            if ( value != null )
            {
                // local-most values win, so if the retrievedValue != null in the
current POM, NEVER
                // look in the parent POM for a match.
                // If the retrievedValue == null, though, we need to stop looking for
a match here.
                if ( retrievedValue == null )
                {
                    getLogger().debug( "property not found here. continuing down the
lineage." );
                    continue;
                }

                matches = checkValue.equals( retrievedValue );

                // if we get here, retrievedValue != null, and we're looking at the
local-most POM, so:
                //
                // if startsWith '!' (reverse == true) and values don't match (match
== false), return true
                // if NOT startsWith '!' (reverse == false) and values DO match (match
== true), return true
                // else return false
                if ( reverse != matches )
                {
                    getLogger().debug( "Searching for property-value match: matches: "
+ matches + "; reversed: " + reverse + "; profile: " + profile.getId() + " should be
activated." );
                    break;
                }
            }
            // if the value is not specified, then we have to search the entire
ancestry before we
            // can say for certain that a property is missing.
            else
            {
```

```
                // if startsWith '!' (reverse == true) and retrievedValue == null
(true), return true
                // if NOT startsWith '!' (reverse == false) and NOT retrievedValue ==
null (false), return true
                matches = retrievedValue != null;

                getLogger().debug( "Searching for property presence: matches: " +
matches + "; reversed: " + reverse + "; stopping lineage search." );

                if ( matches )
                {
                    break;
                }
            }

            // if we can't definitely say we're activating the profile, go to the next
model in the
            // lineage, and try again.
        }

        // if we get to the end of the lineage without activating the profile, return
false.
        return reverse != matches;
    }

    protected Logger getLogger()
    {
        if ( logger == null )
        {
            logger = new ConsoleLogger( Logger.LEVEL_DEBUG,
"ModelPropertyProfileActivator:internal" );
        }

        return logger;
    }

    public void enableLogging( Logger logger )
    {
        this.logger = logger;
    }
```

```
}
```

Notes

1. The Java source code above relies on the new Build Context feature of Maven 2.1. This feature allows components in the Maven runtime - including mojos - to import and export shared, structured information. It's important to note that this feature is not yet completely stabilized; therefore, the code above will be changing in the future, in response to changes in the Build Context.