

# Activiti Classloading

[As suggested by Tom](#), at this page we collect ideas concerning a classloading scheme for activiti which works in as many environments as possible.

## 1) Why is classloading an issue?

Activiti offers the possibility to call "client code" (i.e. custom Java Classes) from BPMN Processes using different ways:

- Directly: Java Delegates, Listeners etc.
- Indirectly: using EL

In order to achieve this, the activiti-engine must be able to load the class definitions for delegates, which means that they must either be on the same classpath as activiti or activiti needs a custom classloader to load these classes.

This is trivial under the following circumstances:

- a standalone Java SE application (unit test, Swing client etc...)
- a single "deployment" in a managed environment (i.e. a single war, ear, OSGi Bundle, ...) such that the activiti engine is deployed alongside with the client code.

Unfortunately, this is often not possible, as we want to be able to redeploy processes and client code at runtime in a managed environment without stopping / starting the process engine. In such a case, the engine needs to be made aware of the classloader of a deployment.

## 2) What is already in place?

On the ProcessEngineConfiguration, a custom ClassLoader can be set. This Classloader is used by org.activiti.engine.impl.util.ReflectUtil to load JavaDelegates. To my knowledge, the custom classloader is currently used

- by activiti-osgi (see org.activiti.osgi.blueprint.ProcessEngineFactory)
- [at camunda we are building a deployment manager for activiti and JBoss AS 5/6](#)

## 3) Discussion

I raised the issue [here](#) whether to also set the classloader used to load delegates as a context-ClassLoader on the current thread when calling delegates.

The problem with the context classloader is that its role in the Java Platform is somewhat vaguely specified. On many platforms (like Jboss As), the context classloader is used to load classes.

What we should collect here is a set of situations, where, when invoking delegates, Thread.currentThread().setContextClassLoader( CLASSLOADER of the delegate ) leads to unexpected behavior.