

# Spring Integration

This page outlines all possible integration points from Activiti with the Spring Framework. On the kick-off meeting the team agreed that there MUST NOT be a needed runtime dependency to Spring, but a decent and optional integration will help Spring users to add Activiti into their existing and new Spring application.

## Configuration support

It should be easy to configure Activiti with the Spring Framework inside an application context. Activiti should provide a namespace configuration support to simplify the configuration of the Activiti engine inside Spring. A possible namespace based configuration could look like this

```
<beans
xmlns="http://www.springframework.org/schema
/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xmlns:activiti="http://www.activiti.org/sch
ema/spring/config"
  xsi:schemaLocation="http://www.springframew
ork.org/schema/beans

http://www.springframework.org/schema/beans/
spring-beans.xsd

http://www.springframework.org/schema/spring
/config

http://www.springframework.org/schema/spring
/config/spring-config.xsd">

  <!-- Activiti setup with references to
spring based extension points -->
  <activiti:process-engine
```

```
data-source="dataSource"
transaction-manager="transactionManager"/>

    <bean id="dataSource" class="....">
        <!-- data source specific properties
like username, url, driver etc. -->
    </bean>

    <bean id="transactionManager"
class="....">
        <!-- transaction manager
implementation class like JPA, Hibernate,
```

```
plain JDBC or JTA -->
    </bean>
</beans>
```

The namespace could also be implemented with default references for the most common extension points like the bean "transactionManager" for the transaction manager bean definition. The bean name "transactionManager" is a common name used in Spring for the transaction manager in particular if used with the built-in jta namespace support. With the defaults, the configuration could look like this in its simplest form (assuming that we are using a JTA transaction manager)

```
<beans...>

    <activiti:process-engine />

    <bean id="dataSource" class="...">
        <!-- data source specific properties
like username, url, driver etc. -->
    </bean>

    <!-- Chooses automatically the best
transaction manager strategy -->
    <tx:jta-transaction-manager/>
</beans>
```

Starting with a minimal and simple default configuration there could be lots of extension points that can be configured with Spring to plugin the Spring-based implementation like transaction managers or their own SPI-based implementation.

The next chapters will outline possible configurable extension points for Activiti along with the affected classes/modules in Activiti

## DataSource Configuration

Because of the fact that Spring users will integrate Activiti into their Spring application and vice-versa they will build a new Spring application based on Activiti it is crucial to offer an extension point for injected datasources through Spring. Based on the unified datasource inside the application context it is possible to use Activiti with its own persistence implementation (ibatis) and the application persistence (probably JPA) inside one transaction without JTA.

The extension point would be a `javax.sql.DataSource` class that can be configured in Spring and referenced through the `Activiti` namespace.

A sample for a standalone data source that utilizes the Apache Commons DataBase Connection Pool could look like this

```
<beans...>

    <activiti:process-engine
data-source="dataSource"/>

    <bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSou
ce">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver"/>
        <property name="url"
value="jdbc:mysql://localhost:3306/schema"/>
        <property name="username"
value="appUser"/>
        <property name="password"
value="verySecret"/>
    </bean>
</beans>
```

likewise the user could use a JNDI based data source for its application and `Activiti`

```
<beans...>

    <activiti:process-engine
data-source="dataSource" />

    <jee:jndi-lookup id="dataSource"
jndi-name="jdbc/myDS" />
</beans>
```

## Implementation Notes

The support to add an external data source is already supported in the code based through the method `org.activiti.test.cfg.spring.ProcessEngineFactoryBean#setDataSource` although there is no namespace support right now.

## Transaction Manager Configuration

Spring provides a rich set of transaction manager implementations to be used in a standalone environment (like plain JDBC, Hibernate and JPA) and also different implementations to be used inside a JEE application server (like plain JTA, Websphere and Weblogic specific and optimized ones). With both strategies supported, it is possible to have one transaction for the application and the engine even in a standalone environment!

A namespace based configuration could look like this for the standalone environment with plain JDBC

```
<beans...>

    <activiti:process-engine
data-source="dataSource" />

    <bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSou
ce">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver" />
        <property name="url"
value="jdbc:mysql://localhost:3306/schema" />
        <property name="username"
value="appUser" />
        <property name="password"
value="verySecret" />
    </bean>

    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.D
ataSourceTransactionManager">
        <property name="dataSource"
ref="dataSource" />
    </bean>
</beans>
```

whereas the setup in combination with a JEE application server could look like this

```
<beans...>

    <activiti:process-engine
data-source="dataSource" />

    <jee:jndi-lookup id="dataSource"
jndi-name="jdbc/myDS" />

    <tx:jta-transaction-manager/>
</beans>
```

Where the tag `<tx:jta-transaction-manager/>` automatically detects the best available transaction manager (Websphere/Weblogic specific ones before the `JTATransactionManager` as a fallback)

## Implementation Notes

There is already support to use the Spring Framework transaction abstraction inside Activiti. The method `org.activiti.test.cfg.spring.ProcessEngineFactoryBean#setTransactionManager` already support Spring based `org.springframework.transaction.PlatformTransactionManager` implementations.

## MyBatis Configuration

Although Spring doesn't have any support for (My)iBatis yet it should be possible to configure the `org.apache.ibatis.session.SqlSessionFactory` with Spring to customize the iBatis configuration used in Activiti in the following areas

\*Interceptors

\*Caching

**NOTE** This decision can be deferred until Spring 3.1 ships with the iBatis 3.x configuration support. The iBatis 3.x support will be tackled in this [issue](#)

## Service Activating Activity Configuration

Activiti has to be able to activate Java-based services as part of a process definition (not yet implemented as of alpha4). Spring users will want to be able to resolve services from Spring configuration.

## Implementation Notes

The `ExpressionManager` and `ScriptingEngines` are the key abstractions in Activiti. The `ProcessEngineFactory` allows customization of the `ExpressionManager` by injecting an `ELResolver` and this might already be enough.

## **IdentityService**

Users need to strategise the Activiti IdentityService. It can be injected into the ProcessEngineFactory directly.

## **Spring Infrastructure**

Spring provides a fully fledged infrastructure to run JAVA based application in a standalone and JEE server environment with special support for the most common application servers like Websphere, Weblogic. The most important infrastructure implementation and their benefits for Activiti will be outlines below.

## **Task Abstraction**

Some JEE Application servers disallow the creation of native Threads while running a application inside the server. It is mandatory to use the vendor specific APIs to create "managed" threads. Spring has an built-in abstraction Task abstraction that allows to run tasks (which are essentially Runnable) with different implementations like a `java.util.concurrent.Executor` for standalone environments and a CommonJ based implementation for Websphere and Weblogic.

## **Implementation Note**

Activiti uses native Threads like `org.activiti.impl.jobexecutor.JobAcquisitionThread` which must be converted to a Runnable instance. There is also a new abstraction needed to run Runnable instances with different implementations (for the beginning a `java.util.concurrent.Executor` one and a Spring based one).