ProfileActions

A profile action is a discrete piece of code, packaged into a Java bean, that peforms a specific function, such as building a SAML assertion, signing a message, encrypting an attribute, validating a password, or displaying a message to a user. Just as the functionality in the IdP is made up of "profiles", a profile is implemented by connecting together a series of profile action beans into a sequence of steps. The steps can be sequential or include conditional branches based on data or user input.

Spring Web Flow is the mechanism used to orchestrate the sequence of actions that implement a complete profile. Authentication is also implemented using a reusable set of web flows that support the login mechanisms a deployer wants to enable, and new mechanisms can be added by implementing small amounts of code and configuring new flows.

The interface used by Spring to invoke an action bean is **org.springframework.webflow.execution.Action** and a single method is used:

SWF Action interface

```
package org.springframework.webflow.execution;

public interface Action {
    public Event execute(RequestContext context) throws Exception;
}
```

All action beans ultimately support this interface. However, the vast majority of beans in the IdP will implement an OpenSAML interface, ProfileAction, which is a Spring-independent interface with a similarly simple signature:

OpenSAML ProfileAction interface

```
package org.opensaml.profile.action;

public interface ProfileAction<InboundMessageType,OutboundMessageType> extends InitializableComponent {

        public void execute(@Nonnull final ProfileRequestContext<InboundMessageType,OutboundMessageType>
profileRequestContext)
            throws ProfileException;
}
```

There is intentional symmetry between these interfaces. The purpose of the OpenSAML interface is to support independence from Spring when possible, and make a larger amount of functional SAML processing code available within OpenSAML for those that are implementing their own applications.

Note that some profile action beans may require other IdP services and components, and so may not be independent of the IdP. They often, though, will still be mostly independent of Spring.

Spring-Independent Actions

More often than not, a profile action bean will not need access to the Spring Web Flow context or the Spring container, in which case it will not need to implement the Web Flow interface itself, just the OpenSAML interface. Usually this is done by deriving a bean from the AbstractProfileAction base class and overriding this method:

Implementing an OpenSAML ProfileAction

```
protected void doExecute(@Nonnull ProfileRequestContext<InboundMessageType,OutboundMessageType>
profileRequestContext)
        throws ProfileException;
```

The action bean interacts with the current request using the ProfileRequestContext, which is the root of an extensible tree of context objects representing the state of a profile execution, the inputs to the action, and the outputs of the action when it completes. If executing in a servlet container, the action can access the servlet objects by requiring their injection as properties on the action bean.

When an action completes, it signals a transition by attaching an EventContext object to the context tree. The "proceed" Event ID is generally used to signal a standard successful transition. Helper methods for attaching events are provided in the ActionSupport helper class. In most cases, actions do nothing on success, but call a helper method to signal a non-proceed event. Exceptions may be, but generally should not be, thrown. Using events provides the best model for control and customization.

Spring-Based Actions

If an action bean requires access to the Spring Web Flow context object or Spring container, it has a couple of options that are functionally equivalent. Both start by deriving from the AbstractProfileAction base class. It can choose to directly implement the Spring Web Flow **Action** interface by overriding this method:

Implementing a Spring Web Flow Action

```
protected Event doExecute(@Nonnull final RequestContext springRequestContext,
        @Nonnull final ProfileRequestContext<InboundMessageType,OutboundMessageType> profileRequestContext)
                throws ProfileException;
```

An alternative approach that is more consistent with the rest of the action beans that don't involve Spring is to override the ProfileAction method shown in the Spring-Independent example above. As in that case, results are signaled by adding an EventContext to the context tree. Access to Spring is via a SpringRequestContext object available as a child of the context tree.

Web Flow and Bean Configuration

This is not an exhaustive discussion as to how web flows can or will be configured, but to illustrate the two cases discussed, here is an example of how a couple of action beans are declared so that they can be referenced by a web flow:

Spring Web Flow Bean Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/util
                           http://www.springframework.org/schema/util/spring-util.xsd"
       default-init-method="initialize">

    <!--
    Enables property replacement, auto-wraps OpenSAML actions in a Spring action wrapper, and
    auto-adds a component identifier to each bean based on the Spring bean ID.
    -->
    <bean class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer"
        p:placeholderPrefix="%{" p:placeholderSuffix="}" />
    <bean class="net.shibboleth.idp.profile.impl.ProfileActionBeanPostProcessor" />
    <bean class="net.shibboleth.ext.spring.config.IdentifiableBeanPostProcessor" />

    <!-- An action that can run without Spring and is adapted. -->
    <bean id="SpringIndependentBean" class="org.example.profile.impl.SpringIndependent" scope="prototype" />

    <!-- An action that is Spring-dependent. -->
    <bean id="SpringAware" class="org.example.profile.impl.SpringAware" scope="prototype" />

</beans>
```

The ProfileActionBeanPostProcessor bean is a Spring post-processor that detects any OpenSAML ProfileAction beans that do not implement the Spring Action interface, and wraps them in a WebFlowProfileActionAdaptor. This is a simple bean that turns any implementation of the OpenSAML ProfileAction interface into a Spring Web Flow action.

The "SpringIndependentBean" bean is an example of such a bean. It will automatically be wrapped. The "SpringAware" bean uses Spring internally and implements the Action interface, so will not be wrapped. Either of the techniques described earlier to implement a bean can be used. Because of the post-processor, declaration of the two types of actions is the same.

The examples shown use the "prototype" scope to allow the beans to be stateful, such that each invocation of the action will involve a unique bean instance. This is the norm. The default scope can be used for stateless action beans, but declaring it as a prototype means it can eventually acquire state without having to worry about how it's been declared. Using non-prototype beans is a source of potentially serious concurrency bugs so it is strongly discouraged and we may implement bean processors to enforce this at some point.