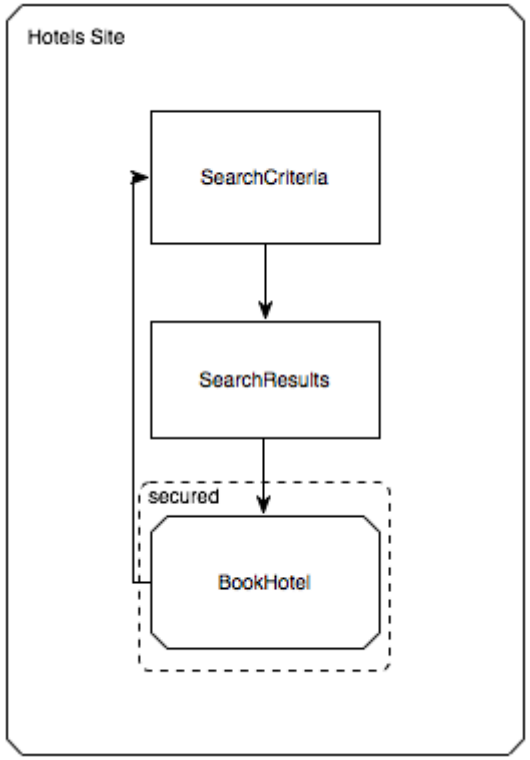
3. Defining Flows

3.1. Introduction

This chapter begins the Users Section. It shows how to implement flows using the flow definition language. By the end of this chapter you should have a good understanding of language constructs, and be capable of authoring a flow definition.

3.2. What is a flow?

A flow encapsulates a reusable sequence of steps that can execute in different contexts. Below is a Garrett Information Architecture diagram illustrating a reference to a flow that encapsulates the steps of a hotel booking process:

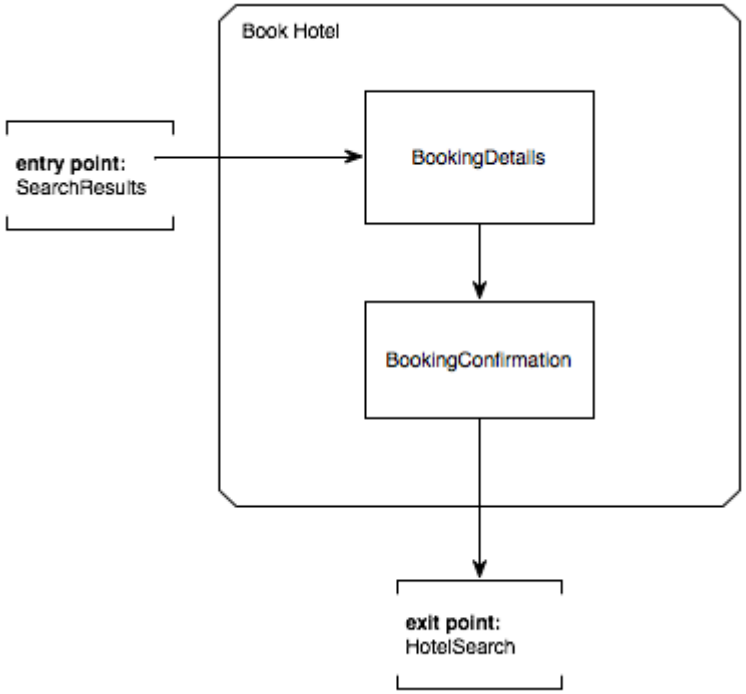


Site Map illustrating a reference to a flow

3.3. What is the makeup of a typical flow?

In Spring Web Flow, a flow consists of a series of steps called "states". Entering a state typically results in a view being displayed to the user. On that view, user events occur that are handled by the state. These events can trigger transitions to other states which result in view navigations.

The example below shows the structure of the book hotel flow referenced in the previous diagram:



Flow diagram

3.4. How are flows authored?

Flows are authored by web application developers using a simple XML-based flow definition language. The next steps of this guide will walk you through the elements of this language.

3.5. Essential language elements

3.5.1. flow

Every flow begins with the following root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow.xsd">

</flow>
```

All states of the flow are defined within this element. The first state defined becomes the flow's starting point.

3.5.2. view-state

Use the `view-state` element to define a step of the flow that renders a view:

```
<view-state id="enterBookingDetails" />
```

By convention, a view-state maps its id to a view template in the directory where the flow is located. For example, the state above might render `/WEB-INF/hotels/booking/enterBookingDetails.xhtml` if the flow itself was located in the `/WEB-INF/hotels/booking` directory.

3.5.3. transition

Use the `transition` element to handle events that occur within a state:

```
<view-state id="enterBookingDetails">
    <transition on="submit" to="reviewBooking" />
```

```
</view-state>
```

These transitions drive view navigations.

3.5.4. end-state

Use the `end-state` element to define a flow outcome:

```
<end-state id="bookingCancelled" />
```

When a flow transitions to a end-state it terminates and the outcome is returned.

3.5.5. Checkpoint: Essential language elements

With the three elements `view-state`, `transition`, and `end-state`, you can quickly express your view navigation logic. Teams often do this before adding flow behaviors so they can focus on developing the user interface of the application with end users first. Below is a sample flow that implements its view navigation logic using these elements:

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow.xsd">

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <view-state id="reviewBooking">
```

```
        <transition on="confirm" to="bookingConfirmed" />
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <end-state id="bookingConfirmed" />

    <end-state id="bookingCancelled" />

</flow>
```

3.6. Actions

Most flows need to express more than just view navigation logic. Typically they also need to invoke business services of the application or other actions.

Within a flow, there are several points where you can execute actions. These points are:

- On flow start
- On state entry
- On view render
- On transition execution
- On state exit
- On flow end

Actions are defined using a concise expression language. Spring Web Flow uses the Unified EL by default. The next few sections will cover the essential language elements for defining actions.

3.6.1. evaluate

The action element you will use most often is the `evaluate` element. Use the `evaluate` element to evaluate an expression at a point within your flow. With this single tag you can invoke methods on Spring beans or any other flow variable. For example:

```
<evaluate expression="entityManager.persist(booking)" />
```

Assigning an evaluate result

If the expression returns a value, that value can be saved in the flow's data model called `flowScope`:

```
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels" />
```

Converting an evaluate result

If the expression returns a value that may need to be converted, specify the desired type using the `result-type` attribute:

```
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels"
          result-type="dataModel"/>
```

3.6.2. Checkpoint: flow actions

Now review the sample booking flow with actions added:

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
```

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

    <input name="hotelId" />

    <on-start>
        <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
                  result="flowScope.booking" />
    </on-start>

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <view-state id="reviewBooking">
        <transition on="confirm" to="bookingConfirmed" />
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <end-state id="bookingConfirmed" />

    <end-state id="bookingCancelled" />

</flow>
```

This flow now creates a Booking object in flow scope when it starts. The id of the hotel to book is obtained from a flow input attribute.

3.7. Input/Output Mapping

Each flow has a well-defined input/output contract. Flows can be passed input attributes when they start, and can return output attributes when they end. In this respect, calling a flow is conceptually similar to calling a method with the following signature:

```
FlowOutcome flowId(Map<String, Object> inputAttributes);
```

... where a `FlowOutcome` has the following signature:

```
public interface FlowOutcome {
   public String getName();
   public Map<String, Object> getOutputAttributes();
}
```

3.7.1. input

Use the `input` element to declare a flow input attribute:

```
<input name="hotelId" />
```

Input values are saved in flow scope under the name of the attribute. For example, the input above would be saved under the name `hotelId`.

Declaring an input type

Use the `type` attribute to declare the input attribute's type:

```
<input name="hotelId" type="long" />
```

If an input value does not match the declared type, a type conversion will be attempted.

Assigning an input value

Use the `value` attribute to specify an expression to assign the input value to:

```
<input name="hotelId" value="flowScope.myParameterObject.hotelId" />
```

If the expression's value type can be determined, that metadata will be used for type coersion if no `type` attribute is specified.

Marking an input as required

Use the `required` attribute to enforce the input is not null or empty:

```
<input name="hotelId" type="long" value="flowScope.hotelId" required="true" />
```

3.7.2. output

Use the `output` element to declare a flow output attribute. Output attributes are declared within end-states that represent specific flow outcomes.

```
<end-state id="bookingConfirmed">
    <output name="bookingId" />
</end-state>
```

Output values are obtained from flow scope under the name of the attribute. For example, the output above would be assigned the value of the `bookingId` variable.

Specifying the source of an output value

Use the `value` attribute to denote a specific output value expression:

```
<output name="confirmationNumber" value="booking.confirmationNumber" />
```

3.7.3. Checkpoint: input/output mapping

Now review the sample booking flow with input/output mapping:

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow.xsd">

    <input name="hotelId" />

    <on-start>
        <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
                  result="flowScope.booking" />
    </on-start>

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <view-state id="reviewBooking">
        <transition on="confirm" to="bookingConfirmed" />
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <end-state id="bookingConfirmed" >
```

```
        <output name="bookingId" value="booking.id"/>
    </end-state>

    <end-state id="bookingCancelled" />

</flow>
```

The flow now accepts a `hotelId` input attribute and returns a `bookingId` output attribute when a new booking is confirmed.

3.8. Variables

A flow may declare one or more instance variables. These variables are allocated when the flow starts. Any `@Autowired` transient references the variable holds are also rewired when the flow resumes.

3.8.1. var

Use the `var` element to declare a flow variable:

```
<var name="searchCriteria" class="com.mycompany.myapp.hotels.search.SearchCriteria"/>
```

Make sure your variable's class implements `java.io.Serializable`, as the instance state is saved between flow requests.

3.9. Variable Scopes

Web Flow can store variables in one of several scopes:

3.9.1. Flow Scope

Flow scope gets allocated when a flow starts and destroyed when the flow ends. With the default implementation, any objects stored in flow scope need to be Serializable.

3.9.2. View Scope

View scope gets allocated when a `view-state` enters and destroyed when the state exits. View scope is *only* referenceable from within a `view-state`. With the default implementation, any objects stored in view scope need to be Serializable.

3.9.3. Request Scope

Request scope gets allocated when a flow is called and destroyed when the flow returns.

3.9.4. Flash Scope

Flash scope gets allocated when a flow starts, cleared after every view render, and destroyed when the flow ends. With the default implementation, any objects stored in flash scope need to be Serializable.

3.9.5. Conversation Scope

Conversation scope gets allocated when a top-level flow starts and destroyed when the top-level flow ends. Conversation scope is shared by a top-level flow and all of its subflows. With the default implementation, conversation scoped objects are stored in the HTTP session and should generally be Serializable to account for typical session replication.

The scope to use is often determined contextually, for example depending on where a variable is defined -- at the start of the flow definition (flow scope), inside a a view state (view scope), etc. In other cases, for example in EL expressions and Java code, it needs to be specified explicitly. Subsequent sections explain how this is done.

3.10. Calling subflows

A flow may call another flow as a subflow. The flow will wait until the subflow returns, then respond to the subflow outcome.

3.10.1. subflow-state

Use the `subflow-state` element to call another flow as a subflow:

```
<subflow-state id="addGuest" subflow="createGuest">
    <transition on="guestCreated" to="reviewBooking">
        <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
    </transition>
    <transition on="creationCancelled" to="reviewBooking" />
</subflow-state>
```

The above example calls the `createGuest` flow, then waits for it to return. When the flow returns with a `guestCreated` outcome, the new guest is added to the booking's guest list.

Passing a subflow input

Use the `input` element to pass input to the subflow:

```
<subflow-state id="addGuest" subflow="createGuest">
    <input name="booking" />
    <transition to="reviewBooking" />
</subflow-state>
```

Mapping subflow output

When a subflow completes, its end-state id is returned to the calling flow as the event to use to continue navigation.

The subflow can also create output attributes to which the calling flow can refer within an outcome transition as follows:

```
<transition on="guestCreated" to="reviewBooking">
    <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
</transition>
```

In the above example, `guest` is the name of an output attribute returned by the `guestCreated` outcome.

3.10.2. Checkpoint: calling subflows

Now review the sample booking flow calling a subflow:

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow.xsd">

    <input name="hotelId" />

    <on-start>
        <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
                  result="flowScope.booking" />
    </on-start>

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <view-state id="reviewBooking">
```

```
        <transition on="addGuest" to="addGuest" />
        <transition on="confirm" to="bookingConfirmed" />
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <subflow-state id="addGuest" subflow="createGuest">
        <transition on="guestCreated" to="reviewBooking">
            <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
        </transition>
        <transition on="creationCancelled" to="reviewBooking" />
    </subflow-state>

    <end-state id="bookingConfirmed" >
        <output name="bookingId" value="booking.id" />
    </end-state>

    <end-state id="bookingCancelled" />

</flow>
```

The flow now calls a `createGuest` subflow to add a new guest to the guest list.

Prev Next

2. What's New Home 4. Expression Language (EL)