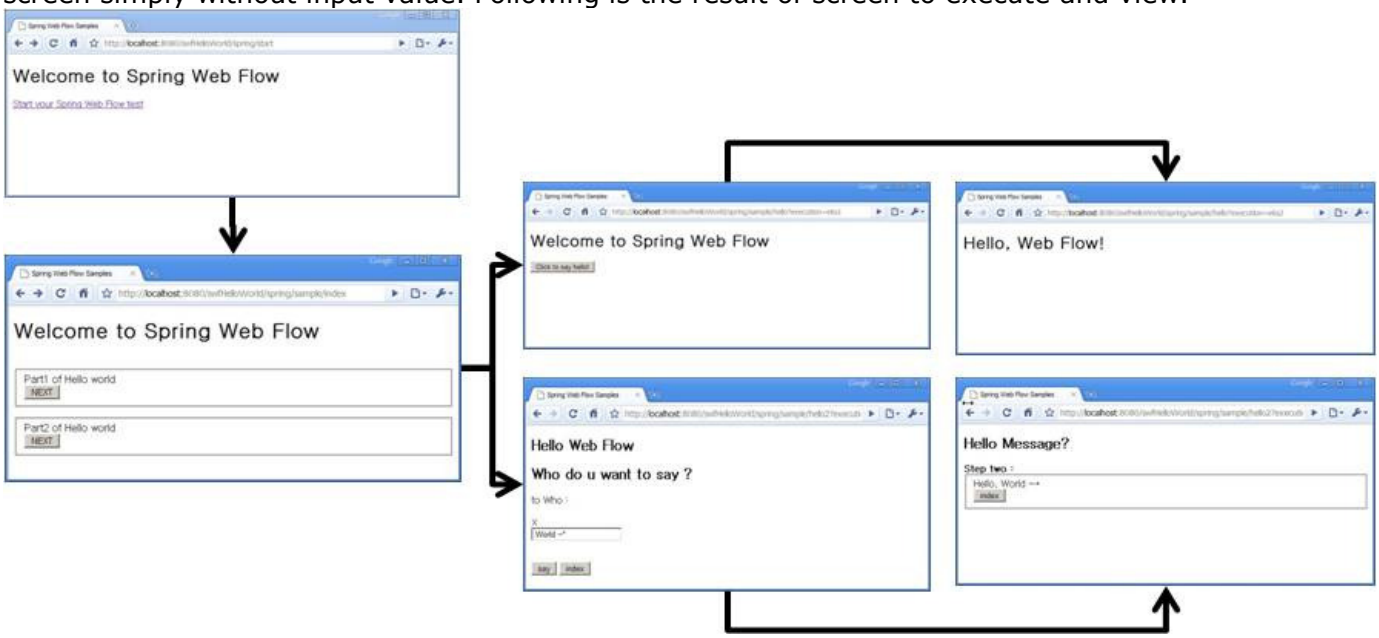
Hello, World

Summary

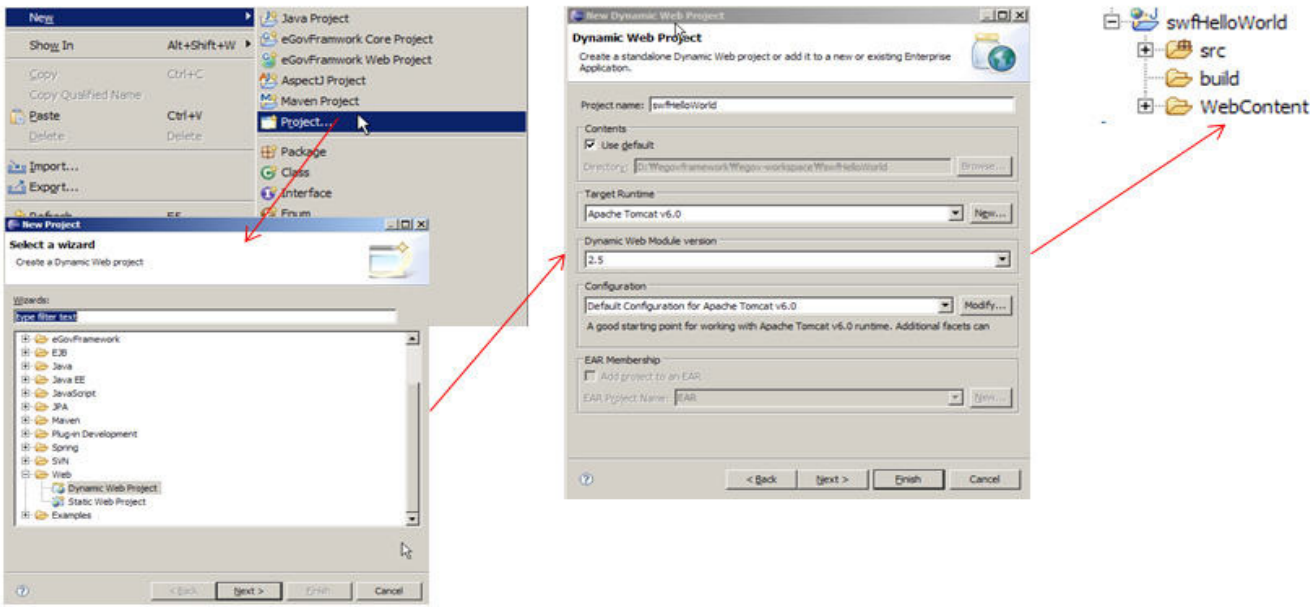
Let's find out how to run Hello World for the first time.
Hello World will be explained in the version of showing the results in screen after executing the service method such as branch processing with input value, as well as the case of calling the Hello, Web Flow screen simply without input value. Following is the result of screen to execute and view.



Description

Spring Web Flow (SWF) is a component of the Spring Framework's web stack focused on the definition and execution of UI flow within a web application

swfHelloWorld Project Environment Generation



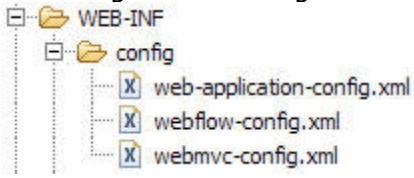
web.xml

Create web.xml under webContent/WEB-INF as shown below.
Set /WEB-INF/config/web-application-config.xml as the value of contextConfigLocation.
Register org.springframework.web.servlet.DispatcherServlet as servlet, and map /spring/* URL information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
	version="2.4">

	<!—Register main setting file for Spring Web application. -->
	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>
			/WEB-INF/config/web-application-config.xml
		</param-value>
	</context-param>

	<!—Load the Spring Web application context. -->
	<listener>
		<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
	</listener>

	<!—Register Controller(DispatcherServlet) in front of Spring Web application. -->
	<servlet>
		<servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
		<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<init-param>
			<param-name>contextConfigLocation</param-name>
			<param-value></param-value>
		</init-param>
		<load-on-startup>0</load-on-startup>
	</servlet>

	<!—Enable to process the request for all spring request by mapping it with DispatcherServlet.
-->
	<servlet-mapping>
		<servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
		<url-pattern>/spring/*</url-pattern>
	</servlet-mapping>

	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
	</welcome-file-list>

</web-app>
```

web-application-config.xml

Following is the setting file for Spring MVC and Spring Web Flow.



First of all, we'll examine web-application-config.xml .

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

```
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-2.5.xsd">

        <!—Scan and load the application source. -->
        <context:component-scan base-package="org.egovframe.swf.sample.service" />


        <!—For convenience, separate and get the setting for Spring MVC setting and Spring Web
Flow.-->
        <import resource="webmvc-config.xml" />
        <import resource="webflow-config.xml" />

</beans>
```

webmvc-config.xml

Setting file for Spring MVC .

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        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-2.5.xsd">

        <!--
            Perform the role of mapping flow registered in flowRegistry with requested path.
                In the example, use requested .../swfHelloWorld/spring/sample/hello   URL
information and find sample/hello ID in flow.
        -->
        <bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
                <property name="order" value="0" />
                <property name="flowRegistry" ref="flowRegistry" />
        </bean>

        <bean

        class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
                <property name="order" value="1" />
                <property name="defaultHandler">
                    <!-- UrlFilenameViewController extracts the View name and returns the View using
path information accessing to spring/start. Here, view of tiles is returned. -->
                        <bean
class="org.springframework.web.servlet.mvc.UrlFilenameViewController" />
                </property>
        </bean>

        <!--
            Send the view name returned by Controller and show the screen defined in tiles in advance.
        -->
        <bean id="tilesViewResolver"
class="org.springframework.js.ajax.AjaxUrlBasedViewResolver">
                <property name="viewClass"
                        value="org.springframework.webflow.mvc.view.FlowAjaxTilesView" />
        </bean>

        <!—Define tiles setting information. -->
        <bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
                <property name="definitions">
                        <list>
```

```
                                <value>/WEB-INF/layouts/layouts.xml</value>
                                <value>/WEB-INF/views.xml</value>
                                <value>/WEB-INF/sample/views.xml</value>
                                <value>/WEB-INF/sample/hello/views.xml</value>
                        </list>
                </property>
        </bean>


        <!-- Dispatches requests mapped to POJO @Controllers implementations-->
        <bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

        <!--
                Dispatches requests mapped to
                org.springframework.web.servlet.mvc.Controller implementations
        -->
        <bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />

        <!--
                Connect registered FlowHandler implementation part suitable for request.
        -->
        <bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
                <property name="flowExecutor" ref="flowExecutor" />
        </bean>

        <!-- Custom FlowHandler for the hello flow-->
        <bean name="sample/hello" class="org.egovframe.web.HelloFlowHandler" />


</beans>
```

webflow-config.xml

Setting file related to Web Flow.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:webflow="http://www.springframework.org/schema/webflow-config"
        xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/webflow-config
          http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd">

        <webflow:flow-executor id="flowExecutor" />

        <!-- Get the file defining flow and configure flow registry. -->
        <webflow:flow-registry id="flowRegistry"
                flow-builder-services="flowBuilderServices" base-path="/WEB-INF">
                <webflow:flow-location-pattern value="/**/*-flow.xml" />
        </webflow:flow-registry>

        <!—Expand and use to enable customizing in Web Flow views. -->
        <webflow:flow-builder-services id="flowBuilderServices"
                view-factory-creator="mvcViewFactoryCreator" conversion-
service="conversionService"
                development="true" />


        <!—Set to use tiles in Web Flow. -->
```

```
	<bean id="mvcViewFactoryCreator"
		class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
		<property name="viewResolvers" ref="tilesViewResolver" />
	</bean>

</beans>
```

Details: Expand and Use to Enable Customizing in Web Flow views.

tiles

URL: index.html file opens when approaching http://localhost:8080/swfHelloWorld for the first time.
index.html

```
<html>
  <head>
    <meta http-equiv="Refresh" content="0; URL=spring/start">
  </head>
</html>
```

As seen above, call "spring/start" URL.
spring/start the other terminal session, start the Producer application.

```
<tiles-definitions>
	<definition name="start" extends="standardLayout">
		<put-attribute name="body" value="/WEB-INF/main.jsp" />
	</definition>
</tiles-definitions>
```

See http://tiles.apache.org/ for details related to tiles.
Registered tiles setting file is from the above setting as shown below..

```
...
	<!-- Define tiles setting information. -->
	<bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
		<property name="definitions">
			<list>
				<value>/WEB-INF/layouts/layouts.xml</value>
				<value>/WEB-INF/views.xml</value>
				<value>/WEB-INF/sample/views.xml</value>
				<value>/WEB-INF/sample/hello/views.xml</value>
			</list>
		</property>
	</bean>
...
```

Hello, Web Flow

The application starts, creates the cache and region and then asks you to start the producer:

hello-flow.xml

```
<?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-2.0.xsd">

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

```
        <transition on="say" to="helloworld" />
    </view-state>

    <view-state id="helloworld">
        <transition on="return" to="return" />
    </view-state>

    <end-state id="return"      view="externalRedirect:servletRelative:/start" />

</flow>
```

Details are discussed in flow definition.
Simply,
it is classified into view-state, end-state. View-state that exists first can be regarded as a start point. In addition, end-state is literally the last point. If a screen "hello" appears first, and helloworld screen appears, then, perform last execution of return.
Transition inside view-state is the execution of button that is clicked to move. Here, press the Say to execute the view-state of helloworld. Likewise, press Return to move to externalRedirect:servletRelative:/start.

- Reference: externalRedirect , servletRelative

If not defining separate view in view-state, get the View with ID. Here, the id of hello becomes view name.
default is to find the screen source(JSP, xhtml, etc) in the directory same as flow.xml. Here, refer to the area defined as titles.
…Examine the contents of /hello/views.xml file,

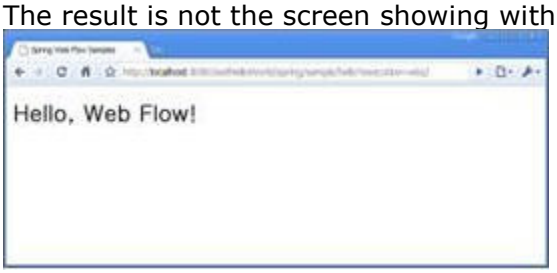
```
...
    <definition name="hello" extends="standardLayout">
        <put-attribute name="body" value="/WEB-INF/sample/hello/hello.jsp" />
    </definition>
...
```

Can be checked to get corresponding hello.jsp to the screen.

Then, does transition perform event and mapping occurred in the screen? Let's examine hello.jsp source briefly.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Welcome to Spring Web Flow</title>
</head>
<body>
<h1>Welcome to Spring Web Flow</h1>
<form:form id="start">
    <input type="submit" name="_eventId_say" value="Click to say hello!" />
</form:form>
</body>
</html>
```

The answer lies in the area surrounded by form as shown above. The answer can be found with **_eventId_say** examining the name in <input type="submit" name="_eventId_say" …. />. _eventId is answer. It can be checked that Say is the same as on of transition. Let's analyze the transition with character string in the specific position defined in eventId.
Examine the contents of transition in details in flow definition. If eventId has "say" and form is delivered, find the transition according to flow definition and go to the stage accordingly.
The result is not the screen showing with separate value, but shows the following screen.



The application starts, creates the cache and region, puts data entry key/values pairs into its cache, and then exits

Hello, Web Flow with input value

When they start, both programs create their cache, specifying the XML file to use. The cache creation automatically creates the distributed system using the gemfire.properties file settings. The distributed system is what connects the two programs to each other. **hello2-flow.xml**

```
<?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-2.0.xsd">


        <on-start>
                <evaluate expression="helloService.sayMessage()" result="flowScope.message" />
        </on-start>

        <view-state id="hello2" model="message">
                <binder>
                        <binding property="str" required="true" />
                </binder>
                <transition on="proceed" to="actionHello" />
                <transition on="return" to="return" />
        </view-state>

        <action-state id="actionHello">
                <evaluate expression="helloService.addHello(message)" />
                <transition on="yes" to="moreDecision" />
                <transition on="no" to="hello" />
        </action-state>

        <decision-state id="moreDecision">
                <if test="helloService.getDecision(message)" then="helloworld2" else="return" />
        </decision-state>

        <view-state id="helloworld2">
                <transition on="return" to="return" />
```

```
        </view-state>


        <end-state id="return"      view="externalRedirect:servletRelative:/start" />
</flow>
</xml>
```

To show, bind the input data to the object in hello2 screen(view-state),
execute addHello method through helloService service object, pass through branch statement (decision-state) according to result and go to the helloworld2 screen.
Briefly speaking,
on-start is executed first when executing flow for the first time. Here, execute sayMessage of helloService and save as the message object in flowScope.

HelloService .java

```
...
@Service("helloService")
public class HelloService implements Iservice {

        public Message sayMessage() {
                return new Message();
        }
        ...

}
```

When flow is executed, view-stage met first is regarded as a starting point. Accordingly, view-state "hello2" corresponds to starting point.
hello2.jsp shows the screen as shown in the above example. We'll show using the tiles of Spring MVC.

views.xml

```
...
        <definition name="hello2" extends="standardLayout">
                <put-attribute name="body" value="inhello2.body" />
        </definition>

        <definition name="inhello2.body" template="/WEB-INF/sample/hello2/main.jsp">
                <put-attribute name="helloSection" value="/WEB-INF/sample/hello2/hello.jsp" />
        </definition>
...
```

hello.jsp

```
...
<form:form method="post" >
        <p>
       to Who: <input type="text" id="str" name="str" value=" World ~*"/>
                                <script type="text/javascript">
                                        Spring.addDecoration(new Spring.ElementDecoration({
                                                elementId: "str",
                                                widgetType: "dijit.form.ValidationTextBox",
                                                widgetAttrs: { promptMessage: "for who ? ",
required: true }}));
                                </script>
                                <br>
        </p>

  <input id="proceed" type="submit" class="button"
      name="_eventId_proceed" value="say">
      <script type="text/javascript">
      Spring.addDecoration(new Spring.ValidateAllDecoration({elementId:'proceed', event:'onclick'}));
      </script>
```

```
  <input type="submit" class="button"
      name="_eventId_return" value="index">

</form:form>
...
```

The above screen is mapped to view-state in the following hello2-flow.xml.
What should be noticed here is the area of binding the input data of str name in the screen to the object called message.

```
...
        <view-state id="hello2" model="message">
                <binder>
                        <binding property="str" required="true" />
                </binder>

                <transition on="proceed" to="actionHello" />
                <transition on="return" to="return" />
        </view-state>
...
```

Click the Proceed button that corresponds to event, and it moves to the next state.

```
...
<transition on="proceed" to="actionHello" />
...
```

actionHello is as follow. The function is the addHello method calling of helloService object.

```
...
        <action-state id="actionHello">
                <evaluate expression="helloService.addHello(message)" />
                <transition on="yes" to="moreDecision" />
                <transition on="no" to="hello" />
        </action-state>
...
```

addHello method is as follow. It is required to pay attention that the returned value is boolean.
Returned boolean value is mapped to yes, no of transition.

```
...
        public boolean addHello(Message msg){
                try{
                msg.setStr("Hello,"+msg.getStr());
                }catch (Exception e) {
                        return false;
                }
                return true;
        }
...
```

Next decision-state performs the branch inquiry function as shown below.

```
...
        <decision-state id="moreDecision">
                <if test="helloService.getDecision(message)" then="helloworld2" else="return" />
        </decision-state>
...
```

Move to helloworld2 screen and the jsp source as shown below can be checked as follows. It shows as ${message.str} using the str value of message object with EL.

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form" %>

<h2>Hello Message?</h2>
<form:form>
  <b>Step two:</b>
        <fieldset>
                <div class="field">
                        <div class="label">
                                <label>${message.str}</label>
                        </div>

                </div>
                <div class="buttonGroup">
                        <input type="submit" class="button"   name="_eventId_return"
value="index">
                </div>

    </fieldset>
</form:form>
```

The screen is:

Press say button,

Hello , the next sentence follows.