

How to Redirect on Spark Completion

Introduction

Redirecting to a separate web page / jsp / asp / or even Spark once having completed an interview is a common requirement, and can be achieved in several ways, dependant upon the desired behaviour.

The methods by which we can achieve this are :

- Use default **redirect on finish** functionality. (Basic)
- Use a **custom web page** at this end of your Spark. (Advanced)
- Use **Sub-Sparks** if directing to another Spark. (Moderate)

Redirect on Finish

The 'Finish Location' option is by far the most simple, but also least flexible. This option simply redirects to a static link (defined on the web server), when you complete your Spark. This does not allow for dynamic link generation, or the inclusion of dynamic variables.

To achieve this simply take the following steps :

1. Design your Spark in the desktop application (No special behaviour is required here).
2. Publish your Spark to the web server as normal.
3. Log into your web server, and click the **Management** link.
4. Click the **Sparks** tab to view all published Sparks.
5. Go to **Edit** on the appropriate Spark, this will open a dialog.
6. Click the **Settings** tab.
7. Make sure **Close the Spark when finished** is unchecked, this will enable the **Finish location** box.
8. In **Finish location** please enter the desired finish link, including protocol, e.g. <http://www.google.com>
9. Click **Save**. Your settings have now been applied.

After performing these steps you can run your Spark, and after clicking the **Finish** button will be redirected to the link specified in your Spark options.

Custom Web Page

A 'Custom Web Page' allows greater flexibility when dealing with externally developed pages. This is a simple item in the spark that redirects process flow to your external page.

To add a custom web page to your Spark simply :

1. Click the 'Custom Web Page' icon and drop it into your Spark as normal.
2. Give your item a reference name (Spark item name).
3. Input the basic **Webpage URL** into the appropriate box (This should be the base URL without any dynamic variables appended).
4. Input a **Verification URL**. This is essentially a location that will be 'pinged' to test a connection exists before the real connection attempt is made. If this is blank then no verification check will be executed.
5. Connect arrows into and out of your 'Custom Web Page' to define the process flow around it (Including fault connectors).

The above should successfully connect your Spark to your custom web page.

To handle passing variables into and out of your 'Custom Web Page', as well as defining your own next, back, finish buttons etc (If required) then you will need to use the **Interview Manager** interface provided by our software.

The **Interview Manager** provides a well defined set of methods that include the functionality to :

- Get Spark item content.
- Get/Set variables.
- Generate standard form JavaScript code / form parameters /dynamic buttons etc.
- Create new interviews for a Spark.

Below is a quick sample of java code explaining how the **InterviewManager** can be set up and utilized :

```
//Import the interview manager class
<%@page import="com.informavores.firefly.integration.InterviewManager"%>

//Get default parameters passed from SparkStudio
String interviewID = request.getParameter(URLParameterName.INTERVIEW_ID);
String customRendererID = request.getParameter(URLParameterName.CUSTOM_WEBPAGE_ID_PARAMETER);
String actionPath = request.getParameter(URLParameterName.ACTION_PATH_PARAMETER);
```

```
String windowingType = request.getParameter(URLParameterName.WINDOWING_TYPE);
String templateID = request.getParameter(URLParameterName.TEMPLATE_ID);
String returnUrl = request.getParameter(URLParameterName.RETURN_URL);
String allowNext = request.getParameter(URLParameterName.ALLOW_NEXT_PARAMETER);
String allowBack = request.getParameter(URLParameterName.ALLOW_BACK_PARAMETER);
String allowFinish = request.getParameter(URLParameterName.ALLOW_FINISH_PARAMETER);
String allowSummary = request.getParameter(URLParameterName.ALLOW_SUMMARY_PARAMETER);
String allowHelp = request.getParameter(URLParameterName.ALLOW_HELP_PARAMETER);
String navigationType = request.getParameter(URLParameterName.NAVIGATION_TYPE);
```

```
//Create an interview manager instance
```

```
InterviewManager intMgr = new InterviewManager(interviewID, customRendererID);
```

```
//Get a map of all variables in the Spark (Key is variable name)
```

```
Map vars = intMgr.getAllVariablesByName();
```

```
//Generate our common JavaScript functions
```

```
intMgr.generateJavascriptFunctions()
```

```
//Generate our template CSS
```

```
intMgr.generateTemplateCSSLink(templateID)
```

```
//Generate the start of our main form content
```

```
intMgr.generateInputFormStart(actionPath)
```

```
//Generate Spark navigation parameters
```

```
intMgr.generateNavigationParameters(templateID, windowingType, returnUrl)
```

```
//Generate standard buttons if certain conditions are met to allow this
```

```
if(allowBack.equals("I"))
{
    out.write(intMgr.generateBackButton());
}
if(allowNext.equals("I"))
{
    out.write(HTMLGenerator.generateInputButton(...);
}
if(allowFinish.equals("I"))
{
    out.write(intMgr.generateFinishButton());
}
if(allowHelp.equals("I"))
{
    out.write(intMgr.generateHelpButton());
}
if(allowSummary.equals("I"))
{
```

```
        out.write(intMgr.generateSummaryButton());  
    }
```

```
//Generate form closing content  
out.write(intMgr.generateInputFormEnd());
```

Please have a look at the 'Custom Web Page' user guide for more detailed, and concise information on the **InterviewManager** and custom web pages in general.

Sub-Sparks

If your desired behaviour is to redirect to another Spark on completion, then this is included within our software as SubSparks. Rather than using complex URLs to redirect and pass information between Sparks, you can simply use a SubSpark reference to move easily into your second Spark.

To create a Spark reference (SubSpark) simply take the following steps.

- 1) Create the Spark to which you wish to link, and save.
- 2) Create your main Spark, and at the end drop in a SubSpark item.
- 3) On the **General** tab give your item a unique name, and Browse for the appropriate Spark file (Spark to which we want to link).
- 4) **Entry Points** and **Exit Points** define the interface between your main Spark and this reference. Please note that if no **Exit Points** are defined then the process will end at the final item of the SubSpark.
- 5) The **Variables In** tab allows you to easily pass data between your main Spark and final Spark. Simply check the boxes next to the variables of the SubSpark that you wish to pre-populate on entry. Once checked you can select from a dropdown of current Spark items, allowing you to seamlessly pass data from our current Spark into the next Spark.
- 6) The **Variables Out** tab works in exactly the same way, but in the opposite direction. Simply define which assignments you wish to make in the direction SubSpark -> Spark. Please note that if no exit points are defined, then you will never exit the SubSpark and so these variables will not be passed across.