JavascriptOpenNewWindow

How to open a new browser window from my JSF page?

The article describes different ways to open a new browser window from a JSF page and the involved advantages and drawbacks.

Using the "target" attribute of the commandLink component

```
<h:commandLink target="popupWindow" action="navigateToPopup" actionListener="#{bean.linkClicked}" value="open
popup" />
```

This approach has many drawbacks:

- In FireFox a new tab is opened instead of a new browser window.
- The browser window is opened although there are validation errors.
- Validation errors are displayed in the newly opened window which is very confusing.
- The appearance of the new window cannot be controlled (e.g. hiding of the location bar)

If no model update is needed then setting the attribute "immediate" to "true" can be used to work around the validation problems.

Note: This is the only way to open a new browser window without using Javascript.

Using an OutputLink and Javascript

```
<h:outputLink onclick="window.open('popup.faces', 'popupWindowName', 'dependent=yes, menubar=no, toolbar=no');
return false;" value="#">
        <h:outputText value="open popup" />
</h:outputLink>
```

With this solution we got control over the appearance of the new browser window. And since there is no postback, there is no validation at all. This is the easiest way to open a new browser window when no model update is needed and no action has to be executed.

Advances techniques

In order to implement a proper action handling we need to move the decision whether to open a new window to the action listener.

```
<h:commandLink actionListener="#{bean.openPopupClicked}" value="open popup" />
```

```
public void openPopupClicked(ActionEvent event) {
    // code to open a new browser window goes here
}
```

How to initiate a Javascript call from server side?

Well, unfortunately there's no standardized way to do so yet. However there are two ways to initiate a Javascript call from server side:

- Create a JSF page containing the script and navigate to that page. (Will not be described in this tutorial.)
- Use the Tomahawk's ExtestionFilter to inject the Javascript in the rendered page. (Needs an initial setup but is easier to maintain afterwards.)

Information about how to setup the Tomahawk's ExtensionFilter can be found here: http://myfaces.apache.org/tomahawk/extensionsFilter.html

NOTE: The following code example will only work, if the DefaultAdd'***Resource or similar implementation is configured to be used, which does not attempt to write to the ResponseWriter immediately (like e.g. StreamingAdd***'Resource class) but caches the resources until the render response phase is completed! (see Javadoc of class DefaultAdd_'_'Resource for details)

Now we can inject the Javascript using Tomahawk's ExtensionFilter AddResource class:

```
public void openPopupClicked(ActionEvent event) {
    // View's id in the same form as used in the navigation rules in faces-config.xml
    // This value could be passed as parameter (using <f:param>)
    final String viewId = "/popup.faces";

    FacesContext facesContext = FacesContext.getCurrentInstance();

    // This is the proper way to get the view's url
    ViewHandler viewHandler = facesContext.getApplication().getViewHandler();
    String actionUrl = viewHandler.getActionURL(facesContext, viewId);

    String javaScriptText = "window.open('"+actionUrl+"', 'popupWindow', 'dependent=yes, menubar=no,
toolbar=no');";

    // Add the Javascript to the rendered page's header for immediate execution
    AddResource addResource = AddResourceFactory.getInstance(facesContext);
    addResource.addInlineScriptAtPosition(facesContext, AddResource.HEADER_BEGIN, javaScriptText);
}
```

Closing the popup window

If the close button has a "cancel" semantic, we can use a simple OutputLink:

```
<h:outputLink onclick="window.close(); return false;" value="#">
        <h:outputText value="cancel" />
</h:outputLink>
```

The window will simply close. The server will not notice anything.

Submit form and close the popup window

For a proper action handling which goes through the whole request lifecycle we need to use an action listener:

```
<h:commandLink actionListener="#{popupBean.closeWindowClicked}" value="submit and close" />
```

```
public void closeWindowClicked(ActionEvent event) {
    FacesContext facesContext = FacesContext.getCurrentInstance();

    String javaScriptText = "window.close();";

    // Add the Javascript to the rendered page's header for immediate execution
    AddResource addResource = AddResourceFactory.getInstance(facesContext);
    addResource.addInlineScriptAtPosition(facesContext, AddResource.HEADER_BEGIN, javaScriptText);
}
```

Note: If there is a validation error the window will not close unless the "immediate" attribute of the commandLink is set to "true".