Programmatic

Programmatic manipulation of the components

This page is for those developers who would like to create JSF-based UIs programmatically. It covers dynamic creation of components and components trees in the runtime.

The problem

Most people who use JSF build static UIs. That is, they write JSPs or Facelets pages explicitly defining the UIs. Such a page typically consists of a number of tags which actually create the component tree.

In case of dynamic component creation you don't know the structure of your page in advance and therefore you can't describe it in any static way. You simply can't design a JSP or Facelets XHTML or anything similar because you simply do not know, which components will make up your page, in which order, and what the structure will be.

The example of such scenario is any model-driven UI. Imagine you'd like to implement a generic bean editor. You know how to edit string, number, date or whatever properties, but the structure of the generic bean is not known in advance. Therefor you will need to build your component subtree in the runtime.

Building component structures in the runtime has quite a lot of peculiarities. We'll try to gather the information on this topic in this page. It's community driven, so if you have an idea, a recipe or and advise, please feel free to append.

Creating and binding components

To make your dynamic component structures be visible to the end-user, you basically need to do the following:

1. Implement a backing bean that creates the component structure and exposes it via a property.
2. Bind a component to this property in the JSP or Facelets page.

```
<h:panelGrid binding="#{someBean.dynamicPanelGrid}"/>
```

The someBean object then needs to provide a method to create and populate the dynamic component. WARNING: the code below is untested.

```
private HtmlPanelGrid grid;

public HtmlPanelGrid getDynamicPanelGrid() {
  if (grid == null) {
    Application app = FacesContext.getCurrentInstance().getApplication();
    grid = (HtmlPanelGrid) app.createComponent(HtmlPanelGrid.COMPONENT_TYPE);

    HtmlOutputText text = (HtmlOutputText) app.createComponent(HtmlOutputText.COMPONENT_TYPE);
    text.setValue("some text");
    grid.getChildren().add(text);
  }

  return grid;
}
```

When the presentation page is initially processed (whether JSP, Facelets or other), the specified binding will first be read; if not null then the returned component is used instead of one being created. Providing a getter method therefore ensures that your code creates the component rather than it being created via the presentation page. Note that all properties (other than the binding) specified in the presentation page are ignored if you do this.

If you want the dynamic part of your page to be generated only as a result of the user selecting an action (eg performing a database search causes dynamic creation of components to display the results) then you can instead define a binding and provide only a setter method, then in the action method modify the component:

```
<h:panelGrid binding="#{someBean.dynamicPanelGrid}"/>
```

```
  private HtmlPanelGrid grid;

  public void setDynamicPanelGrid(UIComponent component) {
    this.grid = (HtmlPanelGrid) component;
  }

  public String doSomeAction() {
      grid.getChildren().clear();

      Application app = FacesContext.getCurrentInstance().getApplication();
      HtmlOutputText text = (HtmlOutputText) app.createComponent(HtmlOutputText.COMPONENT_TYPE);
      text.setValue("some text");
      grid.getChildren().add(text);

      return "someActionSuccess";
    }
  }
```

The Apache Shale ViewController can also be used; its page-based callbacks are convenient places in which to configure dynamic components.