|  |  |
| --- | --- |
| **JSF 1.2** | **JSF 2.0** |
| Need to declare everything in a faces-config.xml file. | a faster development cycle because it has been made easier through facelets. |
| It take little more time than JSF 2.0 | What can save time is the additional support libraries that JSF 2.0 opens up for you. For example you have primefaces and Seam 3 at your disposal. Depending on your requirements they can save you plenty of effort. |

# Difference JSF 1.2 ,2.0, 2.1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| **JSF 2.0** | **JSF 2.1** |
| JSF 2.0 is the second release of JSF specification. | JSF 2.1 is the second maintainance release of JSF 2.0. |
| In JSF 2.0 there is no ease of transition from JSF 1.2. In JSF 2.0 separate declaration of facelet tag declarations are needed. | JSF 2.1 converts JSP custom tag libraries to facelet tag libraries. There is an ease of transition from JSF 1.2 to the latest version of specification. |
| JSF 2.0 does not have the feature of pluggable facelet mechanism. | JSF 2.1 has a new feature of pluggable facelet cache mechanism. |
| There is no default mapping of the Faces Servlet in JSF 2.0. | Default mapping for the Faces Servlet is done in JSF 2.1. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |
| --- | --- |
| **JSF** | **JSP** |
| JSF is a web application that is used to simplify development integration of web based user interfaces. | JSP is a Java based technology used specifically in order to help software developers create dynamic web pages. |
| JSF contains multiple core features, including, but not limited to, Managed Beans, a template based component system, and two XML based tag libraries. | JSP must be compiled in Java bytecode in order to function properly. |
| JSF is a framework. e.g. controller:facesservlet Model:backing bean. view:jsf | JSP is not a framework. |
| JSf supports validator and conversion, ajax etc. | JSP does not. |
| you can integrate JSF with richfaces. richfaces supports many library for different UI components and ajax. | JSP does not. |

# 1.  What is JSF (or JavaServer Faces)?

A server side user interface component framework for Java™ technology-based web applications. JavaServer Faces (JSF) is an industry standard and a framework for building component-based user interfaces for web applications.   
JSF contains an API for representing UI components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features.

# 2. What are the advantages of JSF?

The major benefits of JavaServer Faces technology are:

* User interfaces can be created easily with its built-in UI component library, which handles most of the complexities of user interface management.
* Offers a clean separation between behavior and presentation.
* Provides a rich architecture for managing component state, processing component data, validating user input, and handling events.
* Robust event handling mechanism.
* Events easily tied to server-side code.
* Render kit support for different clients
* Component-level control over statefulness
* Highly 'pluggable' - components, view handler, etc
* JSF also supports internationalization and accessibility
* Offers multiple, standardized vendor implementations

# 3. What are differences between struts and JSF?

In a nutshell, Faces has the following advantages over Struts:

* Eliminated the need for a Form Bean, a DTO Class
* Allows the use of the same POJO on all Tiers because of the Backing Bean

**The primary advantages of Struts as compared to JavaServer Faces technology are as follows:**

* Because Struts is a web application framework, it has a more sophisticated controller architecture than does JavaServer Faces technology. It is more sophisticated partly because the application developer can access the controller by creating an Action object that can integrate with the controller, whereas JavaServer Faces technology does not allow access to the controller. In addition, the Struts controller can do things like access control on each Action based on user roles. This functionality is not provided by JavaServer Faces technology.

|  |
| --- |
|  |

* Struts includes a powerful layout management framework, called Tiles, which allows you to create templates that you can reuse across multiple pages, thus enabling you to establish an overall look-and-feel for an application.
* The Struts validation framework includes a larger set of standard validators, which automatically generate both server-side and client-side validation code based on a set of rules in a configuration file. You can also create custom validators and easily include them in your application by adding definitions of them in your configuration file.

**The greatest advantage that JavaServer Faces technology has over Struts is its flexible, extensible UI component model, which includes:**

* A standard component API for specifying the state and behavior of a wide range of components, including simple components, such as input fields, and more complex components, such as scrollable data tables. Developers can also create their own components based on these APIs, and many third parties have already done so and have made their component libraries publicly available.
* A separate rendering model that defines how to render the components in various ways. For example, a component used for selecting an item from a list can be rendered as a menu or a set of radio buttons.
* An event and listener model that defines how to handle events generated by activating a component, such as what to do when a user clicks a button.
* Conversion and validation models for converting and validating component data.

**4.  What are the available implementations of JavaServer Faces?**

The main implementations of JavaServer Faces are:

* Reference Implementation (**RI**) by Sun Microsystems.
* Apache **MyFaces**is an open source JavaServer Faces (JSF) implementation or run-time.
* **ADF Faces** is Oracle’s implementation for the JSF standard.

**6. What typical JSF application consists of?**

A typical JSF application consists of the following parts:

* JavaBeans components for managing application state and behavior.
* Event-driven development (via listeners as in traditional GUI development).
* Pages that represent MVC-style views; pages reference view roots via the JSF component tree.

**7. What Is a JavaServer Faces Application?**

       JavaServer Faces applications are just like any other Java web application. They run in a servlet container, and they typically contain the following:

* JavaBeans components containing application-specific functionality and data.
* Event listeners.
* Pages, such as JSP pages.
* Server-side helper classes, such as database access beans.

**In addition to these items, a JavaServer Faces application also has:**

* A custom tag library for rendering UI components on a page.
* A custom tag library for representing event handlers, validators, and other actions.
* UI components represented as stateful objects on the server.
* Backing beans, which define properties and functions for UI components.
* Validators, converters, event listeners, and event handlers.
* An application configuration resource file for configuring application resources.

**8. What is Managed Bean?**

|  |
| --- |
|  |

**JavaBean objects managed by a JSF implementation are called managed beans. A managed bean describes how a bean is created and managed.** It has nothing to do with the bean's functionalities.

**9. What is Backing Bean?**

**Backing beans are JavaBeans components associated with UI components used in a page. Backing-bean management separates the definition of UI component objects from objects that perform application-specific processing and hold data.**

     The backing bean defines properties and handling-logics associated with the UI components used on the page. Each backing-bean property is bound to either a component instance or its value. A backing bean also defines a set of methods that perform functions for the component, such as validating the component's data, handling events that the component fires and performing processing associated with navigation when the component activates.

**10. What are the differences between a Backing Bean and Managed Bean?**

Backing Beans are merely a convention, a subtype of JSF Managed Beans which have a very particular purpose. There is nothing special in a Backing Bean that makes it different from any other managed bean apart from its usage.

What makes a Backing Bean is the relationship it has with a JSF page; it acts as a place to put component references and Event code.

|  |  |
| --- | --- |
| **Backing Beans** | **Managed Beans** |
| A backing bean is any bean that is referenced by a form. | A managed bean is a backing bean that has been registered with JSF (in faces-config.xml) and it automatically created (and optionally initialized) by JSF when it is needed. |
|  | The advantage of managed beans is that the JSF framework will automatically create these beans, optionally initialize them with parameters you specify in faces-config.xml, |
| Backing Beans should be defined only in the request scope | The managed beans that are created by JSF can be stored within the request, session, or application scopes |

         Backing Beans should be defined in the request scope, exist in a one-to-one relationship with a particular page and hold all of the page specific event handling code.In a real-world scenario, several pages may need to share the same backing bean behind the scenes.A backing bean not only contains view data, but also behavior related to that data.

**11. What is view object?**

A view object is a model object used specifically in the presentation tier. It contains the data that must display in the view layer and the logic to validate user input, handle events, and interact with the business-logic tier. The backing bean is the view object in a JSF-based application. Backing bean and view object are interchangeable terms.

**12. What is domain object model?**

Domain object model is about the business object and should belong in the business-logic tier. It contains the business data and business logic associated with the specific business object.

**13. What is the difference between the domain object model and a view object?**

In a simple Web application, a domain object model can be used across all tiers, however, in a more complex Web application, a separate view object model needs to be used. Domain object model is about the business object and should belong in the business-logic tier. It contains the business data and business logic associated with the specific business object. A view object contains presentation-specific data and behavior. It contains data and logic specific to the presentation tier.

**14. What do you mean by Bean Scope?**

Bean Scope typically holds beans and other objects that need to be available in the different components of a web application.

**15.  What are the different kinds of Bean Scopes in JSF?**

JSF supports three Bean Scopes. *viz.,*

* **Request Scope:**The request scope is short-lived. It starts when an HTTP request is submitted and ends when the response is sent back to the client.
* **Session Scope:** The session scope persists from the time that a session is established until session termination.
* **Application Scope:**The application scope persists for the entire duration of the web application. This scope is shared among all the requests and sessions.

**16. What is the difference between JSP-EL and JSF-EL?**

|  |  |
| --- | --- |
| **JSP-EL** | **JSF-EL** |
| In JSP-EL the value expressions are delimited by ${…}. | In JSf-EL the value expressions are delimited by #{…}. |
| The ${…} delimiter denotes the immediate evaluation of the expressions, at the time that the application server processes the page. | The #{…} delimiter denotes deferred evaluation. With deferred evaluation ,the application server retains the expression and evaluates it whenever a value is needed. |

     *note:As of JSF 1.2 and JSP 2.1 ,the syntax of both expression languages has been unified.*

[More about Unified Expression Language](http://www.developersbook.com/jsf/unified-expression-language.php)

**17. What are The main tags in JSF?**

       JSF application typically uses JSP pages to represent views. JSF provides useful special tags to enhance these views. Each tag gives rise to an associated component. JSF (Sun Implementation) provides 43 tags in two standard JSF tag libraries:

* JSF Core Tags Library.
* JSF Html Tags Library.

**18. How do you declare the managed beans in the faces-config.xml file?**

The bean instance is configured in the faces-config.xml file:

<managed-bean>

<managed-bean-name>login</managed-bean-name>

<managed-bean-class>com.developersBookJsf.loginBean</managed-bean-class>

<managed-bean-scope>request</managed-bean-scope>

</managed-bean>

This means: Construct an object of the class com.developersBookJsf.loginBean, give it the namelogin, and keep it alive for the duration of the request.

**19. How to declare the Message Bundle in JSF?**

We can declare the message bundle in two ways:   
(Let’s assume com.developersBookJsf.messages is the properties file)

**1.**  The simplest way is to include the following elements in *faces-config.xml file:*

<application>

<resource-bundle>

<base-name>com.developersBookJsf.messages</base-name>

<var>message</var>

</resource-bundle>

</application>

**2.**  Alternatively, you can add the *f:loadBundle* element to each JSF page that needs access to the bundle:

<f:loadBundlebaseName = “com.developersBookJsf.messages” var=”message”/>

|  |
| --- |
|  |

**20. How to declare the page navigation (navigation rules) in faces-config.xml file ?**

Navigation rules tells JSF implementation which page to send back to the browser after a form has been submitted. We can declare the page navigation as follows:

<naviagation-rule>

<from-view-id>/index.jsp</from-view-id>

<navigation-case>

<from-outcome>login</from-outcome>

<to-view-id>/welcome.jsp</to-view-id>

</navigation-case>

</naviagation-rule>

This declaration states that the login action navigates to /welcome.jsp, if it occurred inside/index.jsp.

**21. What if no navigation rule matches a given action?**

If no navigation rule matches a given action, then the current page is redisplayed. **22.  What are the JSF life-cycle phases?**

The six phases of the JSF application lifecycle are as follows (note the event processing at each phase):

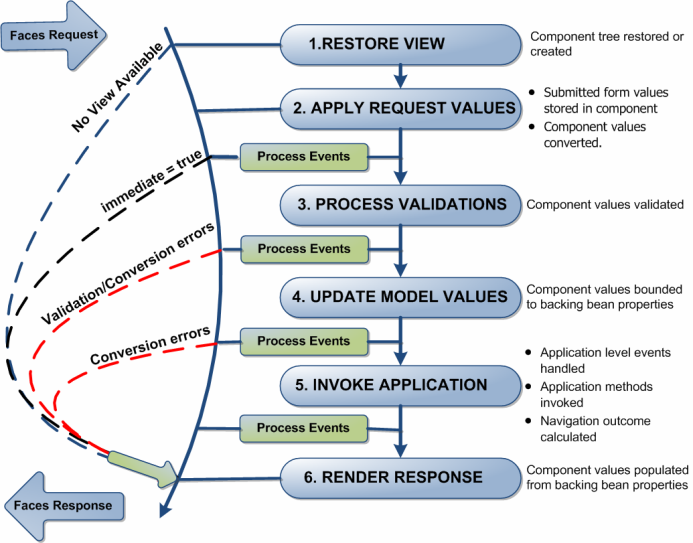
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |  | |  | |

**1.**  Restore view   
**2.**  Apply request values; process events   
**3.**  Process validations; process events   
**4.**  Update model values; process events  
**5.**  Invoke application; process events  
**6.**  Render response

**23. Explain briefly the life-cycle phases of JSF?**

**1. Restore View :**  A request comes through the FacesServlet controller. The controller examines the request and extracts the view ID, which is determined by the name of the JSP page.   
**2. Apply request values:**  The purpose of the apply request values phase is for each component to retrieve its current state. The components must first be retrieved or created from the FacesContext object, followed by their values.   
**3. Process validations:**  In this phase, each component will have its values validated against the application's validation rules.   
**4. Update model values:**  In this phase JSF updates the actual values of the server-side model ,by updating the properties of your backing beans.  
**5. Invoke application:**  In this phase the JSF controller invokes the application to handle Form submissions.  
**6. Render response:**  In this phase JSF displays the view with all of its components in their current state.

[**More about JSF Lifecycle**](http://www.developersbook.com/jsf/jsf-tutorial/jsf-tutorial.php#2)



|  |
| --- |
|  |

**24. What does it mean by render kit in JSF?**

A render kit defines how component classes map to component tags that are appropriate for a particular client. The JavaServer Faces implementation includes a standard HTML render kit for rendering to an HTML client.

**25.  Is it possible to have more than one Faces Configuration file?**

We can have any number of config files. Just need to register in*web.xml*.   
Assume that we want to use faces-config(1,2,and 3),to register more than one faces configuration file in JSF,just declare in the *web.xml file*

<context-param>

<param-name>javax.faces.CONFIG\_FILES</param-name>

<param-value>

/WEB-INF/faces-config1.xml,

/WEB-INF/faces-config2.xml,

/WEB-INF/faces-config3.xml

</param-value>

</context-param>

Over the last few years, a variety of frameworks for building Java–based web applications have been created. For years, Struts have aided developers build web applications using a variation of the classic Model–View–Controller design paradigm. However, JavaServer Faces (JSF) has become a standard part of the Java EE 5 platform, providing both a user interface component framework and the basis for a web application framework.

### JSF Introduction

|  |
| --- |
| http://developersbook.com/jsf/images/caveman.gif |

JavaServer Faces (JSF) is a Java based Web application framework that simplifies the development of user interfaces for enterprise Java applications. JSF applications are implemented in Java on the server, and render as web pages back to clients based on their web requests. JSF provides Web application lifecycle management through a controller servlet; and like Swing, JSF provides a rich component model complete with event handling and component rendering. It is based on other Java standards such as Java Servlets and JavaServer Pages, but it provides a higher–level component layer for UI (user interface) development.

## The major benefits of JavaServer Faces technology are:

* JavaServer Faces architecture makes it easy for the developers to use. In JavaServer Faces technology, user interfaces can be created easily with its built–in UI component library, which handles most of the complexities of user interface management.
* JavaServer Faces technology offers a clean separation between behavior and presentation.
* JavaServer Faces technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.
* Robust event handling mechanism.
* Render kit support for different clients
* Highly 'pluggable' – components, view handler, etc

[[http://www.developersbook.com/images/backToTop.gif](http://www.developersbook.com/jsf/jsf-tutorial/jsf-tutorial.php#top)Back to top](http://www.developersbook.com/jsf/jsf-tutorial/jsf-tutorial.php#top)

### JSF Lifecycle

|  |
| --- |
| JSF Lifecycle Cartoon.png |

In order for you to understand how the framework masks the underlying request processing nature of the Servlet API and to analyze how Faces processes each request, we’ll go through the JSF request processing lifecycle. This will allow you to build better applications.

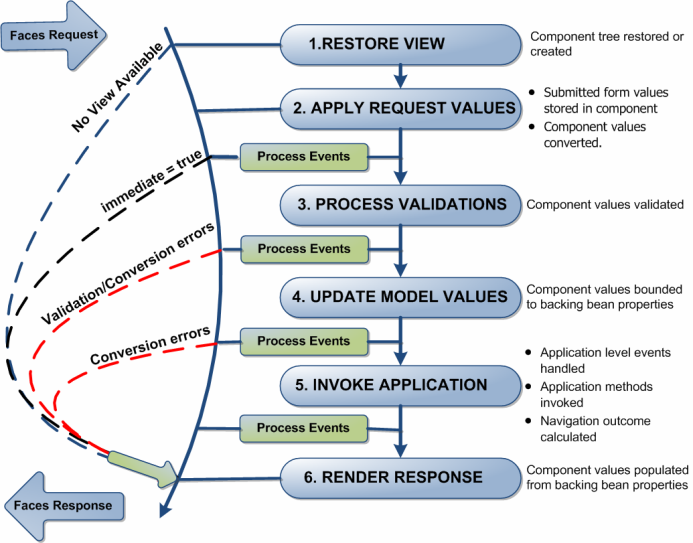
A JavaServer Faces page is represented by a tree of UI components, called a view. During the lifecycle, the JavaServer Faces implementation must build the view while considering state saved from a previous submission of the page. When the client submits a page, the JavaServer Faces implementation performs several tasks, such as validating the data input of components in the view and converting input data to types specified on the server side. The JavaServer Faces implementation performs all these tasks as a series of steps in the JavaServer Faces request–response life cycle.

The phases of the JSF application lifecycle are as follows:

|  |  |
| --- | --- |
|  | 1. Restore view 2. Apply request values; process events 3. Process validations; process events 4. Update model values; process events 5. Invoke application; process events 6. Render response |

|  |
| --- |
|  |

The normal flow of control is shown with solid lines, whereas dashed lines show alternate flows depending on whether a component requests a page redisplay or validation or conversion errors occur.



***Note****: Life – cycle handles two kinds of requests:*

* ***Initial request****: A user requests the page for the first time.*
* ***Postback****: A user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request.*

## Phase 1 : Restore view

In the **RestoreView** phase, JSF classes build the tree of UI components for the incoming request.

* When a request for a JavaServer Faces page is made, such as when a link or a button is clicked, the JavaServer Faces implementation begins the restore view phase.

|  |
| --- |
|  |

* The JSF framework controller uses the view ID (typically JSP name) to look up the components for the current view. If the view isn’t available, the JSF controller creates a new one. If the view already exists, the JSF controller uses it. The view contains all the GUI components and there is a great deal of state management by JSF to track the status of the view – typically using HTML hidden fields.
* If the request for the page is an initial request, the JavaServer Faces implementation creates an empty view during this phase. Lifecycle only executes the restore view and render response phases because there is no user input or actions to process.
* If the request for the page is a postback, a view corresponding to this page already exists. During this phase, the JavaServer Faces implementation restores the view by using the state information saved on the client or the server. Lifecycle continues to execute the remaining phases.
* Fortunately this is the phase that requires the least intervention by application code.

## Phase 2 :ApplyRequest values

During **ApplyRequest values**, the request parameters are read and their values are used to set the values of the corresponding UI components. This process is called decoding.

* If the conversion of the value fails, an error message associated with the component is generated and queued onFacesContext. This message will be displayed during the render response phase, along with any validation errors resulting from the process validations phase.
* If some components on the page have their immediate event handling property is set to true, then the validation, conversion, and events associated with these components takes place in this phase instead of the Process Validations phase. For example, you could have a Cancel button that ignores all values on a form.

|  |
| --- |
|  |

## Phase 3 : Process validations

The **Apply Validations** phase triggers calls to all registered validators.

* The components validate the new values coming from the request against the application's validation rules.
* Any input can be scanned by any number of validators.
* These Validators can be pre-defined or defined by the developer.
* Any validation errors will abort the request–handling process and skip to rendering the response with validation and conversion error messages.

## Phase 4 : Update Model Values

It brings a transfer of state from the UI component tree to any and all backing beans, according to the value expressions defined for the components themselves.

* Converters are invoked to parse string representations of various values to their proper primitive or object types. If the data cannot be converted to the types specified by the bean properties, the life cycle advances directly to the render response phase so that the page is re-rendered with errors displayed.
* Note: The difference between this phase and Apply Request Values - that phase moves values from client–side HTML form controls to server–side UI components; while in this phase the information moves from the UI components to the backing beans.

## Phase 5 : Invoke Application

It handles any application-level events. Typically this takes the form of a call to process the action event generated by the submit button that the user clicked.

* Application level events handled
* Application methods invoked
* Navigation outcome calculated

## Phase 6 : Render Response

Finally, **Render Response** brings several inverse behaviors together in one process:

* Values are transferred back to the UI components from the bean. Including any modifications that may have been made by the bean itself or by the controller.
* The UI components save their state – not just their values, but other attributes having to do with the presentation itself. This can happen server–side, but by default state is written into the HTML as hidden input fields and thus returns to the JSF implementation with the next request.
* If the request is a postback and errors were encountered during the apply request values phase, process validations phase, or update model values phase, the original page is rendered during this phase. If the pages contain message ormessages tags, any queued error messages are displayed on the page.

## Process Events

In this phase, any events that occurred during the previous phase are handled.

* Each Process Events phase gives the application a chance to handle any events (for example, validation failures) that occurred during the previous phase.

***Note:****Sometimes, an application might need to redirect to a different web application resource, such as a web service, or generate a response that does not contain JavaServer Faces components. In these situations, the developer must skip the rendering phase (Render Response Phase) by calling FacesContext.responseComplete. This situation is also shown in the diagram, with ProcessEvents pointing to the response arrow.*

### JSF- Getting started

|  |
| --- |
| JSF Cartoon |

Now that we have a general overview of JavaServer Faces and a basic understanding of the JSF lifecycle, let's get started with some code.   
  
There are more than one JSF implementations available in market. Some of them are:

* Sun (RI) (default)
* Apache MyFaces
* IBM
* Simplica (based on Apache MyFaces)
* Additionally, there are several 3rd party UI components that should run with any implementation.

For our simple application we use Sun (RI) default implementation.

Before you can dive into a full-fledged example, you must lay some groundwork. i.e., configuring your environment to work with JSF. First, you need to get the JSF library files.

* jsf-api.jar
* jsf-impl.jar

|  |
| --- |
|  |

You should place these two JSF JAR files (jsf-api.jar and jsf-impl.jar) in the application's classpath, either in the Web app's lib directory or in the server's classpath. The next thing we'll need to do is download the dependencies our simple project will have. Here are the jar files (apart from above two jars) you will need in your WEB-INF/lib:

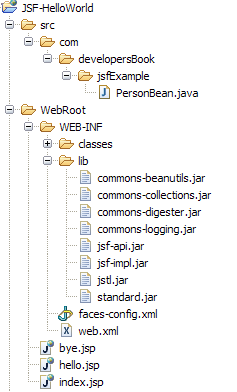
* jstl.jar
* standard.jar
* commons-beanutils.jar
* commons-collections.jar
* commons-digester.jar
* commons-logging.jar

Alternatively, use Ant or Maven to only include these jars when you are testing.

***Note:****Even though, JSF applications typically use JSP tags implemented by the JSF implementation, there are no separate tag library descriptor (TLD) files because that information is contained in the jar files.*

**Figure 2: JSF Helloworld directory structure**

|  |
| --- |
|  |



More advanced JSF applications have the same structure, but they can contain additional Java classes, such as event handlers, validators, and custom components. And optional additional configuration files.

### JSF Basic Example

Now that you know the basic configuration steps, you just need to look at a working example. Let’s start with Java bean class called PersonBean.java (Code Listing 1) . This class has only one property, name

**Code Listing 1: PersonBean.java**

**package**com.developersBook.jsf.bean;

**publicclass**PersonBean {  
      **private** String name;  
  
      **publicvoid**setName(String name) {  
            **this**.name = name;  
      }

**public** String getName() {  
            **returnthis**.name;  
      }  
}

In JSF applications, you use beans for all data that needs to be accessible from a page.  
Next you need is JSF page (index.jsp). This JSP is essentially an HTML file with few additional tags (JSF tags).

**Code Listing 2: index.jsp**

<html>

  <%@tagliburi="http://java.sun.com/jsf/core"prefix="f"%>

  <%@tagliburi="http://java.sun.com/jsf/html"prefix="h"%>

   <f:view>

     <head><title>JSF Hello World</title></head>

     <body>

      <h:formid="helloForm">

        <h:outputLabelfor="promptName"value="Please enter your name" />

        <h:panelGridcolumns="2">

           <h:outputLabelfor="name"value="Name:"/>

           <h:inputTextid="name"value="#{person.name}"/>

           <h:commandButtonvalue="Say Hello"action="sayHello"/>

           <h:commandButtonvalue="Say Bye"action="sayBye"/>

        </h:panelGrid>

       </h:form>

     </body>

   </f:view>

</html>

To use any of the JavaServer Faces tags, you need to include these taglib directives at the top of each page containing the tags defined by these tag libraries:

|  |
| --- |
|  |

  <%@ tagliburi="http://java.sun.com/jsf/core"prefix="f"%>  
  <%@ tagliburi="http://java.sun.com/jsf/html"prefix="h"%>

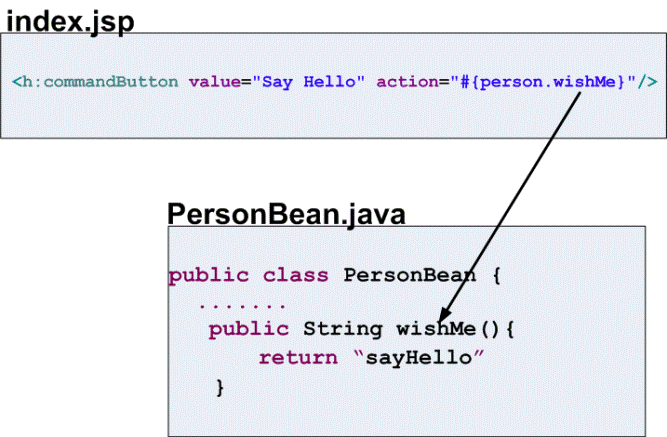
The uri attribute value uniquely identifies the TLD. The prefix attribute value is used to distinguish tags belonging to the tag library. You can use other prefixes rather than the h or fprefixes. However, you must use the prefix you have chosen when including the tag in the page. For example, the form tag must be referenced in the page using the h prefix because the preceding tag library directive uses the h prefix to distinguish the tags defined in html\_basic.tld:  
  
**<h:form....>**

Much of the page is similar to regular an HTML form, except few differences:

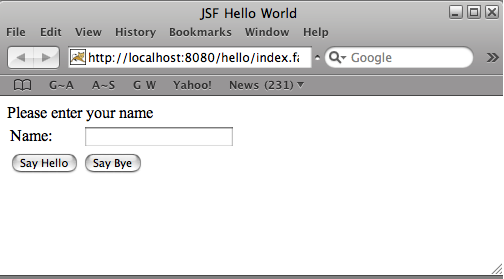
* All JSF tags are contained in an f:view tag.
* All JSF components are enclosed in h:form tag, instead of using HTML form tag.
* Use h:inputText, h:commandButton, instead of regular HTML input tags.

You can find the all the standard JSF tags and their attributes with examples [here](http://www.developersbook.com/jsf/jsf-tags/jsf-standard-components.php).

The value of action attribute (sayHello or sayBye) of h:commandButton shapes the outcome of request. As in our example this action attribute can be literal string or EL method expression, i.e, specific method on a specific bean can be called as part of the request/response cycle.



In this case , the return value from the method will be the outcome .

**Figure 3: index.jsp**  


As the above view (index.jsp) will prompt for your name, and populate the person bean with the value you enter. The others will say hello or goodbye, depending on which action the user chooses.

**Code Listing 3: Hello.jsp**

<%@ tagliburi="http://java.sun.com/jsf/html"prefix="h"%>  
<%@ tagliburi="http://java.sun.com/jsf/core"prefix="f"%>

<html>

  <f:view>  
      <head><title>Hello World!</title></head>  
      <body>  
        <h:form>  
            <h3>Hello,<h:outputTextvalue="#{person.name}"/></h3>

            <h:commandLinkid="Cancel"action="index"immediate="true">  
                  <h:outputTextvalue="Cancel"/>  
             </h:commandLink>  
        </h:form>  
    </body>  
  </f:view>

</html>

**Code Listing 3: Bye.jsp**

<%@ tagliburi="http://java.sun.com/jsf/html"prefix="h"%>  
<%@ tagliburi="http://java.sun.com/jsf/core"prefix="f"%>

<html>

  <f:view>  
      <head><title>GoodBye World!</title></head>  
      <body>  
        <h:form>  
            <h3>GoodBye,<h:outputTextvalue="#{person.name}"/></h3>

            <h:commandLinkid="Cancel"action="index"immediate="true">  
                  <h:outputTextvalue="Cancel"/>  
             </h:commandLink>  
        </h:form>  
    </body>  
  </f:view>

</html>

When working with JSF, you will have a minimum of two XML configuration files, and you will often have even more (ex: tiles.xml). It is important that you become familiar with these config files, as they are the key to the flexibility and loose coupling provided by this architecture.

* Faces config (faces-config.xml) — JavaServer Faces configuration file. Place this file in the WEB-INF directory. This file lists bean resources and navigation rules.
* Web config (web.xml)—this is your standard Web configuration file.

The next step is to create our /WEB-INF/web.xml descriptor file for our hello web application.

**Code Listing 4: web.xml**

<?xmlversion="1.0"?>  
<web-app xmlns="http://java.sun.com/xml/ns/javaee"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
      http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  
      version="2.5">  
      <servlet>  
            <servlet-name>Faces Servlet</servlet-name>  
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>  
            <load-on-startup>1</load-on-startup>  
      </servlet>

      <servlet-mapping>  
            <servlet-name>Faces Servlet</servlet-name>  
            <url-pattern>\*.faces</url-pattern>  
      </servlet-mapping>

      <welcome-file-list>  
            <welcome-file>index.jsp</welcome-file>  
      </welcome-file-list>  
</web-app>

This is your regular web-config file, except the servlet mapping. All JSF pages are processes by a special servlet (FacesServlet) that is a part of the JSF implementation code. The JSF URLs have a special format to ensure that the correct servlet is activated when a JSF page is requested. In our example, they have an extension *.faces*.  
Ex: <http://localhost:8080/hello/index.faces>.

***Note:****You can also define a prefix mapping instead of the*.faces*extension mapping.*

........

url-pattern>/faces/\*</url-pattern>

You are probably wondering how all of this is going to tie together. How does PersonBean get into JSP's and how do the input buttons on index.jsp get me to hello.jsp or bye.jsp? That's where we need a faces-config (faces-config.xml) file.

To cause JSF to proceed from one page to another, you must inform it of the page flows of your application. We do this with navigation rules, which predicate the choice of response view on the outcome of a request. Define navigation rules in your faces-config file.  
Create a file called /WEB-INF/faces-config.xml

**Code Listing 5: faces-config.xml**

<?xmlversion="1.0"?>

<faces-configxmlns="http://java.sun.com/xml/ns/javaee"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
       http://java.sun.com/xml/ns/javaee/web-facesconfig\_1\_2.xsd"  
      version="1.2">

      <navigation-rule>  
            <from-view-id>/index.jsp</from-view-id>  
            <navigation-case>  
                  <from-outcome>sayHello</from-outcome>  
                  <to-view-id>/hello.jsp</to-view-id>  
            </navigation-case>  
            <navigation-case>  
                  <from-outcome>sayBye</from-outcome>  
                  <to-view-id>/bye.jsp</to-view-id>  
            </navigation-case>  
      </navigation-rule>

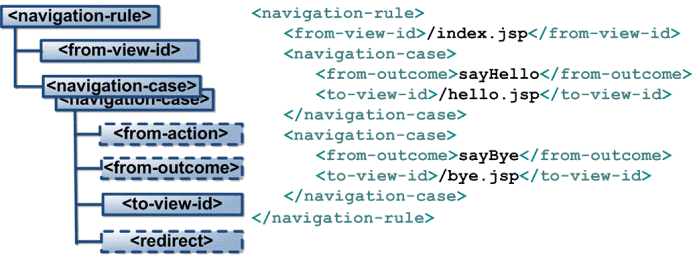
      <managed-bean>  
            <managed-bean-name>person</managed-bean-name>  
            <managed-bean-class>  
                  com.developersBook.jsf.bean.PersonBean  
            </managed-bean-class>  
            <managed-bean-scope>request</managed-bean-scope>  
      </managed-bean>

      <navigation-rule>  
            <from-view-id>\*</from-view-id>  
            <navigation-case>  
                  <from-outcome>index</from-outcome>  
                  <to-view-id>/index.jsp</to-view-id>  
            </navigation-case>  
      </navigation-rule>  
</faces-config>

|  |
| --- |
|  |

**Navigation Rules**

* Navigation rules define almost all the page-flow logic of a JSF application.
* A navigation rule tells the JSF runtime:
  + For what origin we are setting a rule – this is <from–view–id>
  + Where to direct the control in what cases.
* So JSF takes the decision of what page follows what other page away from the page definition, and puts it in the configuration. It decides where to go next based on the identity of the requesting view, plus the outcome.

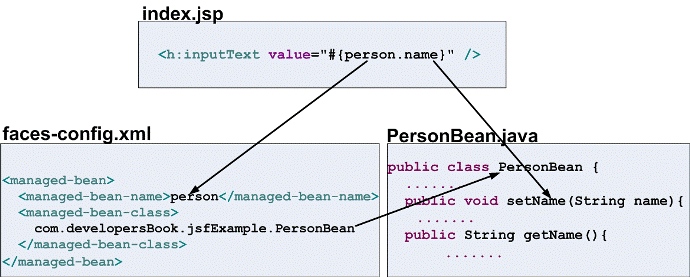
**Figure 4: JSF Navigation Rules**  


|  |
| --- |
|  |

In particular, if we're coming from /index.jsp and the "outcome" is "sayHello", go to the hello.jsp view; if the "outcome" is "sayBye", go to the bye.jsp view. The "outcome" is determined by the action property of the commandButton elements your "from-view", index.jsp.

By contrast, simple HTML forms, and Struts, and Spring, all let the view define the request URLs that originate from various HTML forms, buttons, and links.

Additionally, you can see we've created a managed-bean calledperson that is an instance of our**com.developersBook.jsf.PersonBean** class. personBean has request scope, and is therefore available throughout processing of the entire request. Its properties can be accessed and set using the #{beanname.propertyname}( Ex:**#{person.name}**) convention, as shown in all three JSPs.

**Figure 5: Binding UI to Managed Bean**  


With all this in place, you should now be able to compile, build, deploy and run your web application. You can download the code of this tutorial [here](http://www.developersbook.com/jsf/jsf-tutorials/JSF-Example.zip).

**Conclusion**

JavaServer Faces offers an extremely powerful and flexible framework. Use its power to fit your application’s needs. And don’t forget that it’s all about the lifecycle.

[The Processing Instruction Target Matching “[xX][mM][lL]” is Not Allowed](http://www.anujgakhar.com/2009/02/17/the-processing-instruction-target-matching-xxmmll-is-not-allowed/)

## So, the solution is, there should be no characters (not even white space) before the starting XML. Or do a Trim() before passing it on to the XmlParse() but sometimes that option is not available because XmlParse could be sitting inside a service layer somewhere.

FacesContext.getCurrentInstance().responseComplete();

[**javax.​faces.​context.​FacesContext**](about:blank*0) has public abstract void **responseComplete**()

Signal the JavaServer Faces implementation that the HTTP response for this request has already been generated (such as an HTTP redirect), and that the request processing lifecycle should be terminated as soon as the current phase is completed.

Throws:

[IllegalStateException](http://download.oracle.com/javase/6/docs/api/java/lang/IllegalStateException.html?is-external=true) - if this method is called after this instance has been released

[**javax.​faces.​context.​FacesContext**](about:blank*0) has public abstract void **renderResponse**()

Signal the JavaServer faces implementation that, as soon as the current phase of the request processing lifecycle has been completed, control should be passed to the Render Response phase, bypassing any phases that have not been executed yet.

Throws:

[IllegalStateException](http://download.oracle.com/javase/6/docs/api/java/lang/IllegalStateException.html?is-external=true) - if this method is called after this instance has been released