**JSF**

**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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs.php); 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:

* 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.
* 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
* Eliminated the need for 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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs.php), 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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs.php). 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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs.php) 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.

**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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-2.php) 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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-2.php) Implementation) provides 43 tags in two standard JSF [tag libraries](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-2.php):

* 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 name *login*, 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:loadBundle baseName = “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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-2.php) 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.

**What are the JSF life-cycle phases?**

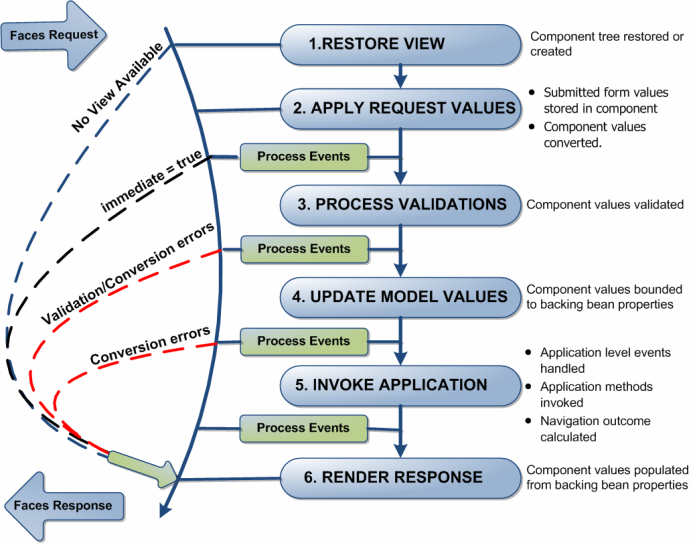
The six phases of the JSF application [lifecycle](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-3.php) are as follows (note the event processing at each phase):

|  |
| --- |
|  |

**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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-3.php) 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.



|  |
| --- |
|  |

**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](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-3.php) 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*](http://www.developersbook.com/jsf/interview-questions/jsf-interview-questions-faqs-3.php).   
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>

1. **Who are the users of JSF technology?**JSF is that it has not only been designed for coding experts but for others also Page authors, Component writers etc. Because of integrating MVC model and flexible in its architecture JSF allows a wide range of users :
   1. **Page authors :**  
      Web designers have experience with graphic design. They can design look and feel of web application in html/jsp using custom [tag libraries](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml) of JSF.
   2. **Application developers :**  
      Application developers can integrate this design with UI components. They program objects, event handling, converters, validators.
   3. **Component writers :**  
      Component developer can build custom UI components because of JSF’s extensible and customizable nature. They can create their own components directly from UI component classes or extending the standard components of JSF.
   4. **Application architects :**  
      Application architects are responsible for designing web applications. Defining page navigation, ensuring Scalability of application, configuring beans object registration are the key points that an application architect handles.
   5. **Tool vendors :**  
      JSF is well suited for tool vendors, for example **Sun Java Studio Creator** application development tool, who provide tools that take advantages of JSF to create UI easier.
2. **What are JSF releases?**  
   JSF started its journey from version 1.0 and now it has come to its latest version JSF1.2. The listing of versions released so far is :
   1. **JSF 1.2** (11 may **2006**) -  **Latest release of JSF specification.**
   2. **JSF 1.1** (27 may **2004**) -  **Bug fix release. No specification changes. No HTML renderkit changes.**
   3. **JSF 1.0** (11 mar **2004**) -  **Initial release** **of JSF specification.**

There are many releases of 1.1 and 1.2 and these are listed below showing released date also:

* 1. **1.2\_04 P01**  (20 Mar **2007**)
  2. **1.2\_04**       (5 Mar **2007**)
  3. **1.2\_02**        (25 Aug **2006**)
  4. **1.2\_01**         (14 July **2006**)
  5. **1.1\_02**         (24 Apr **2006**)
  6. **1.1\_01**         (07 Sep **2004**)

1. **How JSF Fits For Web Applications?**   
   JSF has many  advantages over other existing frameworks that makes it a better choice for Java web [application development](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml). Some of the reasons are below:
   1. **Easy creation of UI:**  
      It makes easier to create complex UI for an applicaton using jsf tags.Its APIs are layered directly on  top of servlet APIs that enables us to use  presentation technology other than JSP,creating your own custom components and rendering output for various client devices.
   2. **Capacity to handle complexities of UI management:**  
      It handles cleanly the complexities of UI management like input validation, component-state management, page navigation, and event handling.
   3. **Clean separation between presentation and logic:**  
      One of the greatest advantage of jsf is to clearly separate behaviour and presentation in an application. JSF is based on the Model View Controller (MVC) architecture
   4. **Shorter development cycle:**  
      This  separation between logic and presentation enables a wide range of users( from [web-page designers](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml) to component developers). It allows members of team to focus on their own work only , resulting in division of labour and shorter development cycle.
   5. **Standard Java framework:**  
      JSF is a [Java standard](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml) which is being developed through Java Community Process ([JCP](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml)). Several prominent tool vendors are members of the group and are committed  to provide easy to use, visual, and productive develop environments for JavaServer Faces.
   6. **An extensible architecture:**JSF architecture has been designed to be extensible.Extensible means additional functionality can be given on the top of JSF core i.e. we can customize the functionality. JSF UI components are customizable and reusable elements. You can extend standard components and create your own complex  components like stylish calendar, menu bar etc.
   7. **Support for multiple client devices:**  
      Component developers can extend the component classes to generate their own component tag libraries to support specific client. JSF flexible and extensible architecture allows developers to do so.
   8. **Flexible rendering model:**  
      Renderer separates the functionality and view of the component. So we can create multiple renderers and give them different functionality to get different appearance of the same component for the same client or different .
   9. **International language support:**  
      Java has excellent support for internationalization . It allows you to localize messages  with user specific locale. A locale is a combination of a country, a language, and a variant code. Java [Server](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions1.shtml) Faces adopts this property and let you specify which locale your application supports. So you can display you messages in different languages.
   10. **Robust tool support:**  
       There are several standard tool vendors like Sun Java Studio Creator who provide robust tools that take advantages of JSF to create server side UI  easily.
2. **What does component mean and what are its types?**Components in JSF are elements like [text box](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml), button, table etc. that are used to create [user interfaces](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml) of JSF Applications. These are objects that manage interaction with a user. Components help developers to create UIs by assembling a number of components , associating them with object properties and event handlers. Would u like to repeat the same code again & again and waste time if u want to create many tables in hundreds of pages in your web [application](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml)? Not at all. Once you create a component, it’s simple to drop that component onto any JSP. Components in JSF are of two types :
   1. **Simple Components** like text box, button and
   2. **Compound Components** like table, data grid etc.

A component containing many components inside it is called a **compound component**.

JSF allows you to create and use components of two types:

1. **Standard UI Components:**JSF contains its basic set of  UI components like text box, check box, list boxe, button, label, radio button, table, panel etc. These are called standard components.
2. **Custom UI Components:**Generally UI designers need some different , stylish components like fancy calendar, tabbed panes . These types of components are not standard JSF components. JSF provides this additional facility to let you create and use  your own set of reusable components. These components are called custom components.
3. **What are third party components and how they are useful?**One of the greatest power of JSF is to support third party components . Third party components are custom components created by another vendor. Several components are available in the market, some of them are commercial and some are [open source](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml) . These pre-built & enhanced components can be used in UI of your web application. For example, we are in need of a stylish calendar then we have an option to take it from third party rather than creating it our own. This will help saving time & cost creating effective & robust UI and to concentrate on business logic part of web application.
4. **What are 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:
   1. **JSF Core Tags**  **Library**
   2. **JSF Html Tags Library**

Even a very simple page uses tags from both libraries. These tags can be used adding the following lines of code at the head of the page.

<%@ taglib uri=”<http://java.sun.com/jsf/core> “ prefix=”f” %> (For Core Tags)  
<%@ taglib uri=”<http://java.sun.com/jsf/html> “ prefix=”h” %> (For Html Tags)

1. **What is the difference between JSP and JSF?**JSP simply provides a Page which may contain markup, embedded [Java code](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml), and tags which encapsulate more complicated logic / html.   
   JSF may use JSP as its template, but provides much more. This includes validation, rich component model and [lifecycle](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions2.shtml)

, more sophisticated EL, separation of data, navigation handling, different view technologies (instead of JSP), ability to provide more advanced features such as AJAX, etc

1. **What is the role of Renderer in JSF? and justify the statement "JSF supports multiple client devices".**After creating JSF components, it is also necessary  for each component to be rendered to the client so that it can be visible to the client’s device. Each of the tag gives rise to an associated component. A renderer is a type of class that is responsible for encoding and decoding components. Encoding displays the component while decoding translates the user’s input into components value i.e. transform it into values the component can understand.  
   Now a days there are many devices that are web enabled. So application developers have challenge to develop components that can work across various platforms. For example, if we have an application that works on standard web [browser](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml) and we want to extend it to make it enable to work on a WAP device. So, to handle this case we need components to be rendered in more than one way. Here JSF can be helpful. This is a simple task for JSF. The solution is to develop separate renderers for the component. JSF components use different renderers depending on the device used.
2. **What is Render Kit in JSF?**Component classes generally transfer the task of generating output to the renderer. All JSF components follow it. Render kit is a set of related renderers. **javax.faces.render.RenderKit** is the class which represents the render kit. The default render kit contains renderers for html but it’s up to you to make it for other markup languages. Render kit can implement a skin (a look & feel). Render kit can target a specific device like phone, [PC](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml) or markup language like HTML, WML, SVG. This is one of the best benefit of  JSF because JSF doesn't limit to any device or markup.
3. **What is conversion and validation? and how are they related?**This is one of the phase of JSF life cycle that happens before binding the component data to the related backing bean in the Update model values phase. Conversion is the process of transforming the component data from String to [Java objects](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml) and vice versa. For example, an user enters a value (String) in an input component and this value is to store in a Date field in the backing bean then this String value is converted to a [java](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml).util.Date value when request is sent to the [server](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml) and vice versa. This process is called Conversion.  
   Validation is the process of ensuring data contains the expected content. For example, checking the Date value is in MM./dd/YYYY format or any integer value is between 1 to 10.  
   The main purpose of conversion and validation is to ensure the values are of correct type and following the required criteria before updating model data. So this step allows you to focus on business logic rather than working on tedious qualifications of input data such as null checks, length qualifiers, range boundaries, etc.
4. **When automatic conversion is supplied by JSF Implementation?**  
   JSF implementation automatically converts component data between [presentation](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions3.shtml) view and model when the bean property associated with the component is of one of the types supported by the component's data.  
   For example, If a UISelectBoolean component is associated with a bean property of type Boolean, then JSF implementation will automatically convert the data from String to Boolean.
5. **Which type of converters can we use in our** [**application**](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions4.shtml)**?**A JSF application can use two types of [converters](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions4.shtml) :
   1. **JSF standard Converters**  
      JSF supplies built-in converters known as standard converters. All standard converters implements **javax.faces.convert.Converter** interface. These converter classes are listed below :
      1. BigDecimalConverter
      2. BigIntegerConverter
      3. BooleanConverter
      4. ByteConverter
      5. CharacterConverter
      6. DateTimeConverter
      7. DoubleConverter
      8. FloatConverter
      9. IntegerConverter
      10. LongConverter
      11. NumberConverter
      12. ShortConverter
   2. **Custom Converter**Custom [data converter](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions4.shtml) is useful in in converting field data into an application-specific value object. For example,
      1. String to User object.
      2. String to Product object    etc.
6. **Which standard converters have their own tags?**  
   DateTimeConverter and NumberConverter are two standard converters that have their own tags **convertDateTime** and **convertNumber** respectively that help configuring the format of the component data by configuring tag attributes.  
     
   **DateTimeConverter :** A component data can be converted to a Date object using convertDateTime tag within the component tag like,

|  |
| --- |
| <h:inputText value="#{Bean.RegisterDate}">         <f:convertDateTime pattern="MMM,dd,YYYY" /> </h:inputText> |

1. **NumberConverter :** Converter for dealing with numbers such as type of number,  maximum integer digits, currency symbol etc.

|  |
| --- |
| <h:inputText id="it" value="#{Bean.value}">    <f:convertNumber maxFractionDigits="3"                     type="currency"                      currencySymbol="$"                     maxIntegerDigits="4" /> </h:inputText> |

2. **What are the ways of using standard converters of JSF?**Standard converters other than the two which have their own tags (DateTimeConverter and NumberConverter) can be used in one of the following three ways. The first one converts the model value of the component and the other two ways convert the component's local value.
   1. **Bind UI Component to Backing Bean Property :**  
      Make sure that the component has its value bound to a backing bean property of the same type as the converter. For example, converting component data to a float number requires binding the component to the property like :

|  |
| --- |
| Float value = 0.0; public Float getValue(){ return value;} public void setValue(Float value) {this.value = value;} |

* 1. **Use “converter” attribute on the UI Component :**  
     Using the converter attribute of the component tag. Just specify the fully qualified class name or ID of the converter in the converter attribute. If the component is not bound to a bean property then this option can be used. For example,

|  |
| --- |
| <h:inputText converter="javax.faces.Integer" /> |

* 2. **Use <f:converter> Tag with ConverterId Attribute :**  
     Using converter tag within the component tag and refer the converter by specifying the ID of the converter in the convertId attribute. For example:

|  |
| --- |
| <h:inputText value="#{Bean.Value}" />         <f:converter converterId="Float" /> </h:inputText> |

1. **When to create and use custom convertors?**The main reasons behind creating our [converter](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions5.shtml) are :
   1. When we want to convert a component's data to a type other than a standard type
   2. When we want to convert the format of the data.  
      Read more at <http://www.roseindia.net/jsf/customconverter.shtml>
2. **What are the steps of creating and using custom converter in our application?**Creating and using a custom converter requires the following steps :

**Steps to follow :**

* 1. Create a class that implements **javax.faces.converter.Converter** interface.
  2. Import necessary packages and classes.
  3. Implement two abstract methods **"getAsObject()"**, **"getAsString()"** provided by **Converter** interface. **getAsObject()** method converts the String (User Input) to Object and **getAsString()** method converts the Object to String to send back to the page.
  4. Register the converter class in [configuration file](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions5.shtml) **(faces-config.xml)** adding **<converter>** element. This element has child elements **<converter-id>** **(name of the converter to be used while programming )**and **<converter-class>** **( name of the converter class which we have created)**.
  5. Create **view** page where **<f:converter>** tag is used with attribute **"converterId"** which specifies the name of the converter which we have specified in **<converter-id>** element of **<converter>** element in **"faces-config.xml"** file.
  6. Use **<h:message>** tag to display the [error message](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions5.shtml).  
     Read more at <http://www.roseindia.net/jsf/customconverter.shtml>

1. **What are the ways to register the custom converters in faces context?**After creating custom converter class implementing Converter interface it needs to register in the faces context. This can be done in one of the two ways :
   1. Register the converter class with the id. This id is used in <f:convertrer> tag in our view page (For example, [JSP](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions5.shtml)).

|  |
| --- |
| <converter>     <converter-id>**ms\_Converter**</converter-id>     <converter-class>**ms\_Converter**</converter-class> </converter> |

* 1. Use this converter in the view page as :

|  |
| --- |
| <h:inputText id="input\_text">     **<f:converter  converterId="ms\_Converter" />** </h:inputText> |

* 2. Register the converter class to handle all "[Email](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions5.shtml)" objects, for example,  automatically.

|  |
| --- |
| <converter>    <converter-for-class>**Email**</converter-for-class>    <converter-class>**EmailConverter**</converter-class> </converter> |

* 1. If  we register the EmailConverter class to handle all Email objects automatically then there is no need to use the <f:converter/> tag in view page. Use this converter as :

|  |
| --- |
| <h:inputText id="phone" value="#{Bean.email}"> </h:inputText> |

1. **What is the use of immediate attribute?**UIInput components and command components can set the immediate attribute to true. This attribute, when set to true, forces the **conversion and validation** phase to occur earlier in the [lifecycle](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml), during the **apply request values** phase.  In other words, if some components on the page have their immediate attributes set to true, then the validation, conversion, and events associated with these components will be processed during **apply request values** phase.

The immediate attribute can be used for the following purposes :

* 1. Immediate attribute, when set to true, allows a commandLink or commandButton to process the back-end logic and ignore validation process related to the fields on the page. This allows navigation to occur even when there are validation errors.
  2. To make one or more input components "high priority" for validation, so validation is performed, if there is any invalid component data, only for high priority input components and not for low priority input components in the page. This helps reducing the number of [error messages](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml) shown on the page.  
       
     **For example :**  
     In the code below, button performs navigation without validating the required field.

|  |
| --- |
| <h:inputText id="it" required="true"/> <t:message for="it"/> <t:commandButton value="submit" immediate="true" action="welcome"/> |

* 1. In the code below, validation is performed only for the first component when button is clicked in spite of being both the input components required.

|  |
| --- |
| <h:inputText id="it1" immediate="true" required="true"/> <h:inputText id="it2" required="true"/> <t:message for="it1"/> <t:message for="it2"/> <t:commandButton value="submit" action="welcome"/> |

1. **Explain the usage of immediate attribute in an application.**Take an example of [shopping cart application](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml) in which a page contain **quantity** fields for each product and two hyperlinks "**Continue Shopping**" and "**Update quantities**". Now we have set the immediate attributes to "**false**" for all the **quantity** fields, "**true**" for "**Continue Shopping**" hyperlink and "**false**" for "**Update quantities**" hyperlink. If we click the "**Continue Shopping**" hyperlink, **none** of the changes entered into the **quantity** [input fields](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml) will be processed. If you click the "**Update Quantities"** hyperlink, the values in the **quantity** fields will be updated in the [shopping cart](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml).
2. **How to get the error messages displayed?**Error messages can be displayed using "**message**" and "**messages**" tag. "message" tag is used to display single error message for a particular component. The ID of the component for which the message is to be displayed is specified is specified in "**for**" attribute of the message tag.  
   Error messages for all the components can be displayed at a place using "**messages**" tag. It [supports](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions6.shtml) two layouts "**table**" and "**list**". List layout shows all the messages in a line and table layout places all the messages in a tabular format.
3. **How to avoid that all the messages are shown on the same line?**"**messages**" tag displays all error messages in a line because the default layout it supports is "**list**". This tag also supplies one more value "**table**" for "**layout**" attribute which displays all error messages in a tabular format i.e. in subsequent lines. So specifying explicitly the value of "**layout**" attribute to "**table**" can solve the problem of displaying all messages in the same line.
4. **How can we replace the JSF Standard Error Message?**Create the message bundle file and set the value of the key for a particular type of error specified in JSF specification.  
   For example, JSF specification supplies value "Value is required" for key "javax.faces.component.UIInput.REQUIRED" so replace the value in our message bundle file similar to the following :

|  |
| --- |
| javax.faces.component.UIInput.REQUIRED= Please enter the required value. |

Register the message bundle within <application> tag  in the [configuration file](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml) (faces-config.xml) and restart the [server](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml). Now when we use message or messages tag in the view page then the value specified in this message bundle file for a particular error is displayed.

1. **How we can change the appearance of error messages in a JSF Page?**The appearance can be changed by any of the two methods :  
   Using "**style**" attribute or "**styleClass**" attribute. "**style**" attribute is used to set the **CSS style definition** for the component while styleClass attribute is used to set the **CSS class** for the component. Using "**styleClass**" attribute is same as html "**class**" attribute.
2. **What is the significance of properties file (Resource Bundle) and how to use this in our JSF page?**  
   Properties file is a collection of **param=value** pairs. This provides a great benefit to the [application](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml) like we can modify these values easily and there is no need to change the JSP file. For example, we can create "**message.properties**" like :

|  |
| --- |
| prompt=Enter Your Name: greeting\_text=Welcome In Roseindia button\_text=Submit |

1. Now edit the configuration file using <message-bundle> element which tells the application where the message resource file is located.

|  |
| --- |
| <application>  <message-bundle>roseindia.messages</message-bundle> </application> |

1. Now, message resource bundle is loaded first using core tag <f:loadBundle> in view page. That loads the bundle and stores it in the request scope.

|  |
| --- |
| <f:loadBundle basename="roseindia.messages" var="message"/> |

1. We can now use this in our JSP like below :

|  |
| --- |
| <h:outputText value="#{message.prompt}"/> |

2. **How can I use several configuration resource files in one single application?**JSF finds configuration file or files looking in context initialization parameter, javax.faces.CONFIG\_FILES  in **web.xml**, that specifies one or more paths to multiple configuration files for your web application. These multiple paths must be comma separeted. The important point to remember is not to register /WEB-INF/faces-config.[xml file](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml) in the web.xml. Otherwise, the JSF implementation will process it twice. For example, make changes in web.xml like below :

|  |
| --- |
| <context-param> <param-name>javax.faces.CONFIG\_FILES</param-name> <param-value> /WEB-INF/test1-config.xml,/WEB-INF/test2-config.xml </param-value> </context-param> |

2. **Can we use a different configuration resource file in place of traditional "faces-config.xml" file in our application?**JavaServer Faces technology provides an [XML document](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml) for configuring resources. This file is used to register application's resources, such as validators, converters, managed beans, and navigation rules. This application configuration resource file is usually called faces-config.xml. You can have more than one application configuration resource file but it must be valid against the DTD located at http://[java](http://www.roseindia.net/interviewquestions/jsf/JSF-interview-questions7.shtml)

.sun.com/dtd/web-facesconfig\_1\_0.dtd. Now register the file within context-param element in web.xml file.

# **JSF 2.0 New Features**

***JSF 2.0*** [proposal](http://weblogs.java.net/blog/edburns/archive/2007/03/prejcpfiled_dra_1.html) has been posted on March 2007 and planned for the release in last quarter of 2008. It will be shipped with ***JEE 6.0*** release. Looking into the features lits of the ***JSF 2.0***, it is added lots of interesting features in the list and also they are aiming to attract more new developers to use ***JSF*** technology.

#### **JSF 2.0 Features**

Some of the interesting features includes *Bookmarking for URLs, expanding the existing lifecycle mechanism, more support for the* ***AJAX*** *requests, use annotations instead of faces-config.xml, introducing JSFTemplating concepts* and much more. It seems the list size is big it will be great to see when the final version is released. Basically it is categorized into four parts: Ease of development, ***New Features***, Performance and Adoption. We will look into the each category with brief explanation. This tips based on the ***JSR 314*** and the features list is taken from the original JSR document.

#### **Ease of development**

* Component aggregation. Allow development of custom component with little or no Java coding. [A] Perhaps consider a standard set of aggregations.
* Allow for "zero configuration" web applications. No faces-config.xml, no web.xml. If necessary, annotations will be used to supplement the configuration data.
* Vastly improve the developer experience with regard to error messages that happen during iterative development, and even during deployment of the application. For example the EG will consider options including, allowing the developer to tell the runtime what stage of development the web application is in, and take action accordingly: Development, Test, Deployment. Line precise error reporting. Pluggable Exception Handling.
* Provide default exception handling mechanisms. For example, all runtime errors get forwarded to an error page.
* Eliminate the need to author a JSP tag handler when writing JSF components.
* Leverage annotations to declare JSF artifacts (components, managed beans, navigation rules, etc) to the runtime.
* Declarative Renderers, otherwise known as Renderers without resorting to out.println().
* Real world, production view description technology, including templating: include something influenced by Facelets, JSFTemplating or Tiles in the specification.
* Make it very easy to create CRUD based applications.

#### **New Features**

* Expand the request processing lifecycle to be aware of Ajax. This may include describing a small developer-contract footprint JavaScript library as part of the JavaServer Faces specification.
* Separate the "build the tree" and "render the tree" processes into two separate lifecycle phases.
* Allow for partial tree traversel during lifecycle execution via Ajax.
* Allow for bookmarkable JSF pages. More broadly, if HTTP GET can be used, it should be used.
* Decent client side, inter-component and form-level validation (leveraging JSR-303 validation if possible).
* First class support for bundling and delivering static resources associated with a component: images, stylesheets, scripts, etc. Should be able to specify default locations for resources.
* Strategic additions to the Standard HTML RenderKit: Date Picker, Tree, Tab View, File Upload components.
* Provide a mechanism to minimize the "Lost Update" and "Duplicate Button Press" problems. These mechanisms must be pluggable or customizable from the user.
* Page actions: The ability to say, "when this page loads, invoke this action (via Ajax if necessary)."
* Provide a mechanism to easily access persistent store.

#### **Performance**

* Saving and restoring page deltas rather than saving and restoring the full view state each time.
* Re-do UIComponent state saving with a view towards making stateless components the default.
* Allow for moving as much of the event processing and inter-page component interaction into the client as possible. This may imply specifying client side controller framework.
* Streamline the rendering process via caching if possible.
* Improve the interceptor mechanism delivered via the PhaseListener feature so that the developer can control exactly what kinds of requests are allowed to be processed by each PhaseListener instance.

#### **Adoption**

* Enable components to have a client based lifecycle in addition to, or instead of the server based request/response lifecycle. Such a client based lifecycle would enable use-cases such as drag-and-drop, master-detail and sub-dialogs on a single page interface web application.
* Improve the UIComponent specification to allow an increase in the interoperability of UIComponent libraries from arbitrary vendors.
* Enable applications to use features that are defined in the JavaServer Faces specification but are intended for use in the Java EE platform as a whole. For example, the managed bean facility is useful even in an application that only uses Servlets from Java EE but provides the rest of the web application features from software outside of the platform.
* Allow JSF application resources to be accessed via REST.
* Enable components that publish events via RSS/Atom.
* Add support for REST (JSR 311)
* Broad and pervasive support for scripting at all levels of the specification. It will be possible to use a scripting language to write all executable artifacts in a JSF application.
* Specify command line interface for authoring JSF applications.
* Support for passing values from page to page. This will be a standardization of the many different Dialog/Conversation/Scope/Flash ideas currently in use.
* "Skinning", or "Themeing" of components.
* An important target user for JavaServer Faces technology is the developer who relies on an Internet Service Provider (ISPv) to host their application, as well as hosting their development environment. It must be easy to iteratively develop a JavaServer Faces application that is hosted entirely on an ISP.

JSF 1.1 VS JSF 1.2

JSF started its journey from version 1.0 and now it has come to the latest version of JSF is 1.2. The listing of versions released so far are :

1. **JSF 1.2** (11 may **2006**) -  **Latest release of JSF specification.**
2. **JSF 1.1** (27 may **2004**) -  **Bug fix release. No specification changes. No HTML renderkit changes.**
3. **JSF 1.0** (11 mar **2004**) -  **Initial release** **of JSF specification.**

There are many releases of 1.1 and 1.2 and these are listed below showing released date also:

1. **1.2\_04 P01**  (20 Mar **2007**)
2. **1.2\_04**       (5 Mar **2007**)
3. **1.2\_02**        (25 Aug **2006**)
4. **1.2\_01**         (14 July **2006**)
5. **1.1\_02**         (24 Apr **2006**)
6. **1.1\_01**         (07 Sep **2004**)

The JSF specification was developed under the **Java Community Process** **(JCP)** as **JSR 127**, which released **JSF 1.0** and **1.1**, and **JSR 252** which released **JSF 1.2.** The first release of JSF is JSF 1.0 in 2004. **JSF 1.0** supports **servlet 2.3 and jsp 1.2**. After JSF 1.0, JSF 1.1 was released. The main purpose of this release was bug-fixing. There were no specification or HTML renderkit changes. This version also works with same version of **servlet 2.3** and **jsp 1.2** as in the case of JSF 1.0.

**JSF 1.1\_01** release features **bug fixes** and **performance improvements**. Improvements include compression of views serialized to the client, failover support when storing views on the [server](http://www.roseindia.net/jsf/jsf-versions.shtml), renderer fixes, and several other bug fixes and enhancements.It supports **JDK 1.3.1 or later.**  Many bugs have been fixed so if [your application](http://www.roseindia.net/jsf/jsf-versions.shtml) depends on one of these bugs, it needs to be changed.

**JSF  1.2** is the **latest release** and it works with **servlet 2.5** and **jsp 2.1.** If you want to run **JSF 1.2 on Tomcat**, you want **6.0 not 5.5**. So JSF needs a web container that supports atleast servlet 2.3 and jsp 1.2 and these are part of J2EE 1.3.

**New in JSF 1.2**

In this section we will go through some new features and changes in JSF 1.2 specification. JSF 1.1 was designed for JSP 1.2 and so could not take advantage of newer JSP features and JSP could not be changed to fill the needs of JSF. This resulted in a number of new features and changes. For ex.

1. **Unified Expression Language(EL) :**  
   Unified EL has been added to JSTL to overcome problems when integrating JSP EL with the JSF EL.EL was used to access data objects in a simple way. It is used mainly as a short-hand for accessing values. Later it was included in JSP 2.0 with more functionality.When JSF was released it also needed EL.JSF and JSP each has its own EL.The JSP EL is much flexibile to the [web application developer](http://www.roseindia.net/jsf/jsf-versions.shtml) but JSF people were unsatisfied by the JSP EL.One of the reason was that it requires JSP engine to evaluate expression at the appropriate time during the lifecycle, but JSP expressions are evaluated immediately. The second reason behind it was that JSP functions could only call static methods in TLD and they cannot be used to invoke public methods dynamically on server side objects during lifecycle to validate data and handle component evevts.  
   So these were the reasons for developing such an EL that would work for both JSP and JSF and make using JSP and JSF together easier. This new EL is called unified EL. So this was inspired by EL used in JSTL and JSP. There is a package "el" in javax package that represents EL.  
     
   In addition to the features already available in the JSP EL, unified EL has the following features :  
   1.  Deferred evaluation of expressions i.e.evaluation of the expression at the appropriate time during the page lifecycle from JSP engine.  
   2.  Support for expressions that can set and get values  
   3.  Support for expressions that can invoke methods.  
   4.  A pluggable API for resolving expressions
2. **Ajax Support :**JSF framework can make things a little easier to write AJAXian JSF components.JSF allows to map different instances of the FacesServlet with different lifecycles.For example, one mapping for standard JSF requests and another for AJAX JSF requests.
3. **New Tree Creation and Content Interweaving Model for Faces applications that use JSP :**We can make JSF application without using any JSP page but it can also be developed using both JSP and JSF.Using JSP makes some developers easy to handle.Sometimes When integrated it did not respond as we expect. These problems have been resolved in JSF 1.2.There have been made some changes in specification of the implementation of the FacesViewHandler for JSP and JSP custom tag base class.
4. **Integration with JSTL :**Using JSTL's <c:forEach> tag to contain Faces input components was one of the problem, because JSP has no notion of a postback, it was not possible to apply the values correctly to the nested input components on postback. Some new concepts in EL make it possible to use this tag fully with any kind of JSF component.
5. **Back Button issues and Multi Frame or Multi Window Faces Apps :**JSF have fixed the problem related to Multi Frame or Multi Window applications.The browser back button also created some problems. These problems were because of deficiency in the State Management API.
6. **Associating a message with a particular component in the page :**Previously we could not include label of a component dynamically in an error message for that component. Now it's possible in new JSF 1.2.It's also possible to override the conversion or validation message that is displayed to the user on a per-instance basis.
7. **Expose an application wide ResourceBundle to the EL :**<resource-bundle> element in faces-config allows listing many resource bundles that should be exposed to the EL using the new ELResolver chain.This optimize performance and prevent the need to create a ResourceBundle for every request.
8. **Use of multiple renderKits.**
9. **Provide XML Schema for the config files, instead of using DTD.**
10. **Security enhancements for client side state saving.**
11. **Solve the "duplicate button press" problem.**
12. **The portlet related bug-fixes.**

**JSP 2.1** is developed under **JSR-245** and **JSF 1.2** is developed under **JSR-252**. These two groups have experts and they are working independently but the main focus of [development](http://www.roseindia.net/jsf/jsf-versions.shtml) was to provide more support and compatibility between these two technology that are powerful web presentation technology so that they can be useful for each other.

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 provides 43 tags in two standard JSF tag libraries:

1. **JSF Core Tags**  **Library** and
2. **JSF Html Tags Library**

Even a very simple page uses tags from both libraries.

<%@ taglib uri=”<http://java.sun.com/jsf/core> “ prefix=”f” %>  
<%@ taglib uri=”<http://java.sun.com/jsf/html> “ prefix=”h” %>  
<f:view>  
<h:form>  
……………  
……………  
</h:form>  
</f:view>  
  
In the above code fragment we have imported two JSF tag libraries with the help of taglib directive. JSF Core Tag Library contains set of JSF core tags while JSF Html Tags Library contains set of html tags. Prefix is used to use tags defined in tag library. Here we are using conventional names f and h for Core & Html tags respectively. We have the choice to choose any name for the prefixes.

* **JSF Html Tags:**These tags represent html components like text fields, buttons, form.  
  Html tags can be divided into following categories:  
    
  **Inputs**                           (inputText, inputTextarea)  
  **Outputs** (outputText, outputLabel)  
  **Commands** (commandButton)  
  **Selections** (selectOneRadio, selectOneListbox, selectOneMenu for radio buttons, list boxes, menu etc)  
  **Layouts** (panelGrid)  
  **Data table** (dataTable)  
  **Errors and messages** (message, messages)  
    
  Some examples have been given below to understand how to use these tags and its attributes:

        <**h:inputText** id=”ID1” value=”value”/>                   
creates a single line text input control  where  id attribute is used to uniquely identify the component rendered by this tag and value attribute sets the current value of the component.

         <**h:outputText** id="ID2" value="Welcome"/>  
creates a single line text output where id attribute uniquely identifies the rendered component and current value is set by value attribute .  
  
         <**h:commandButton**  
                       id="submit"  
                       value="go"  
                       action="nextPage">  
           <**/h:commandButton**>   
  
creates a command button where value attribute sets the value that is displayed on the button when it is rendered and action attribute is used to invoke a method defined in backing bean when a user does an action on the component .According to the return of the invoked method it is determined which view is to be displayed next.  
  
In JSF Html Tag Library there are 25 core tags .

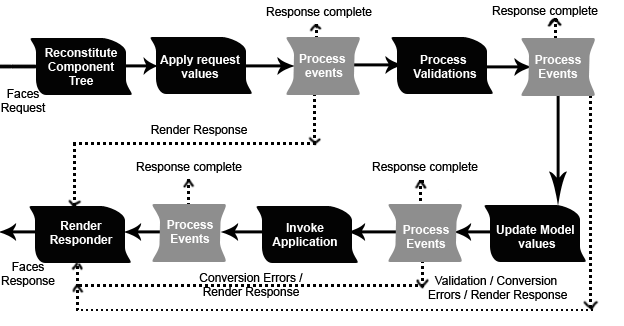
### **All JSF Html Tags :**

* + column                              creates column in a dataTable
  + commandButton                creates button
  + commandLink                   creates link that acts like a pushbutton
  + dataTable                         creates a  table control
  + form                                 creates a form
  + graphicImage                    displays an image
  + inputHidden                      creates hidden field
  + inputSecret                       creates input control for password
  + inputText                          creates  text input control (single line)
  + inputTextarea                    creates  text input control (multiline)
  + message                           displays the most recent message for a component
  + messages                          displays all messages
  + outputFormat                    creates  outputText, but formats compound messages
  + outputLabel                      creates label
  + outputLink                        creates anchor
  + outputText                        creates single line text output
  + panelGrid                         creates html table with specified number of columns
  + panelGroup                      used to group other components where the specification requires one child element
  + selectBooleanCheckbox   creates checkbox
  + selectManyCheckbox       creates set of checkboxes
  + selectManyListbox           creates multiselect listbox
  + selectManyMenu              creates multiselect menu
  + selectOneListbox              creates single select listbox
  + selectOneMenu                creates single select menu
  + selectOneRadio                creates set of radio buttons
* **JSF Core Tags:**  
  These tags allows you to take advantages of features of JSF framework, like validation, conversion , event handling. Core library is stepchild of Html library. i.e. core library supports the html library. Core tag library also contains tags for views and sub-views , loading resource bundle, adding arbitrary text to a page. Some examples of JSF core tags are:  
    
  **f: view**  tag is used to create top level view  
  **f: subview** tag is used to create subview of  a view.   
  **f: validator** tag is used to add a validator to a component.  
  **f: converter** tag is used to add an arbitrary converter to a component.  
  **f: actionListener** tag is used to add an action listener to a component.  
  **f:valueChangeListener** tag is used to add a valuechange listener to a component  
    
  Some examples have been given below to understand how to use these tags:  
    
  <**f:view** locale="en">  
      <h:outputText value="label" />  
  <**/f:view**>   
  **f: view**  tag is used to create top level view and is a container for all JSF component tags on a page. Where locale attribute  provides several options for presenting  localized views of [your application](http://www.roseindia.net/jsf/jsftags.shtml). Here "en" represents English and if we give velue "fr" to locale attribute then french view will be displayed. So this attribute is useful for internationalization purpose.  
    
  <f:view>  
    <h1>head</h1>  
    <p>view</p>  
    <**f:subview** id="sub\_id">  
      <c:import url="second.jsp" />  
     <**/f:subview**>  
  </f:view>   
  Here **f:subview** tag is like container for the JSF components contained in an included JSP page (second.jsp).

<h:inputText id="txt\_id"   
             value="txt\_value">  
  <**f:validator** validatorId="Txt\_Validator" />   
</h:inputText>    
The Validator tag registers a  Validator  on the component associated with the enclosing tag. In validatorId field, we give the value of one of the validator-id element of a validator in your Faces configuration file.

In JSF Core Tag Library there are 18 core tags .  
  
**All JSF Core Tags:**

* f **:**view                                 Creates the top-level view
* f**:**subview                             Creates a subview of a view
* f:attribute           Adds an attribute  to a component
* f**:**param                                Constructs a parameter component
* f:converter            Adds an arbitrary converter to a component
* f**:**converterDateTime      Adds a datetime converter to a component
* f**:**converterNumber       Adds a number converter to a component
* f**:**actionListener          Adds an action listener to a component
* f**:**valueChangeListener     Adds a valuechange listener to a component
* f**:**validator              dds a validator to a component
* f**:**validateDoubleRange    Validates a double range for a component’s value
* f**:**validateLength         Validates the length of a component’s value
* f**:**validateLongRange      Validates a long range for a component’s value
* f**:**facet                 Adds a facet to a component
* f**:**loadBundle            Loads a resource bundle, stores properties as a Map
* f**:**selectitems            Specifies items for a select one or select many component
* f**:**selectitem             Specifies an item for a select one or select many component
* f**:**verbatim              Adds markup to a JSF page



A JSF application typically follows six steps in its life  :

**1. Restore view phase  :**   
This phase starts when a user requests a JSF page by clicking a link, button etc. In this phase  view generation of the page, binding of components to its event handlers and validators are performed and view is saved in the FacesContext object. The FacesContext object contains all the state information JSF needs to manage the GUI component's state for the current request in the current session. The FacesContext stores the view in its viewRoot property.All the JSF components are contained by viewRoot for the current view ID. Component tree of a page is newly built or restored.  
A request comes through the FacesServlet controller. The controller checks the request and takes the view ID i.e. name of the JSP page. View ID is used to look up the components in the current view. JSF controller uses this ID if the view already exists . If the view doesn't already exist, the JSF controller creates it. The created view contains all components.

**2. Apply request values phase  :**The purpose of this phase is for each component to retrieve its current state. After restoring of component tree in previous phase each component in the tree retrieves its new value and store it locally. Component values are typically retrieved from the request parameters.If immediate attribute of a component is set to true, then the validation, conversion, and events associated with the component is processed in this phase.  
  
If a component's immediate event handling property is not set to true, the values are converted. Suppose field is bound to be an Integer property, the value is converted to an Integer. An error message associated with the component is generated if this conversion fails, and queued in the FacesContext. This message will  be displayed during the render response phase, along with any validation errors resulting from next  process validation phase.  
  
At the end of this phase, the components are set to their new values, and messages and events have been queued.

**3. Process validations phase :**  
During this phase local values stored for the component in the tree are compared to the validation  rules registered for the components. If local value is invalid, an error message is added to FacesContext, and the component is treated invalid then JSF proceeds to the render response phase and display the current view showing the validation error messages. If there were conversion errors from the apply request values phase, the messages for these errors are also displayed. If there are no validation errors, JSF proceeds ahead to the update model values phase.

**4. Update model values phase :**After confirming that data is valid in the previous phase local values of components can be set to corresponding server side object properties i.e. backing beans. So bean properties will be updated .If the local data cannot be converted to the types specified by the bean properties, the life cycle proceeds directly to the render response phase and errors are displayed.

**5. Invoke application phase :**Before this phase the component values have been converted, validated, and applied to the bean objects, so you can now use them to execute the application's business logic. Application-level code is executed such as submitting a form or linking to another page.For example user moves to the next page you will have to create a mapping in the faces-config.xml file. Once this navigation occurs, you move to the final phase of the lifecycle.

**6. Render response phase:**  
In this phase JSP container renders the page back to the user,if jsp is used by application i.e. view is displayed with all of its components in their current state.If this is an initial request, the components will be added to the component tree. If this is not an initial request, the components are not added because they are already added to the tree.The state of the response is saved after rendering of the content of the view, so that subsequent requests can access it and it is available to the restore view phase.

JSF MVC

JSF was developed integrating MVC design pattern so that applications can be designed well with greater maintainability. To understand this fact we need to understand what is MVC design pattern, how MVC helps to design an application well and how can we make our web application easy to maintain.The MVC design pattern splits an [application design](http://www.roseindia.net/jsf/jsfandmvc.shtml)

into three separate parts:

**Model :**        handles data and logic.

**View :**           handles output (presentation)

**Controller:**    handles processing of an application.

MVC model’s purpose is to separate model and [presentation](http://www.roseindia.net/jsf/jsfandmvc.shtml) to enable developers to set focus on their core skills and [collaborate](http://www.roseindia.net/jsf/jsfandmvc.shtml) more clearly. If you have to create many pages for presentation then you have to concentrate only on view layer rather than model and controller layer because you can reuse code for controller and model.  
  
In the same way if you want to change the code for model then you typically need not to change view layer.  
  
Controllers are used to process user actions. In this process layer model and views may be changed.

The best advantages of JSF is that it is both a Java Web user-interface standard and a framework that fits well with the Model-View-Controller (MVC) design pattern. It offers a clean separation between presentation and behavior. MVC pattern helps persons of different skill sets to work separately so tasks can be completed in parallel. UI can be created by page author using reusable UI components and business logic part can be implemented using managed beans.

JSF Renders

After creating JSF components, it is also necessary for each component to be rendered to the client so that it can be visible to the client’s device. Each of the tag gives rise to an associated component. A renderer is a type of class that is responsible for encoding and decoding components. Encoding displays the component while decoding translates the user’s input into components value i.e. transform it into values the component can understand.  
  
Now a days there are many devices that are web enabled. So application developers have challenge to develop components that can work across various [platforms](http://www.roseindia.net/jsf/jsfrenderers.shtml). For example, if we have an application that works on [standard web browser](http://www.roseindia.net/jsf/jsfrenderers.shtml) and we want to extend it to make it enable to work on a [WAP](http://www.roseindia.net/jsf/jsfrenderers.shtml) device. So, to handle this case we need components to be rendered in more than one way. Here JSF can help you . It is a simple task for JSF. Solution is to develop separate renderers for the component. JSF components use different renderers depending on the device used.  
  
**Encoding:**  
For example: Suppose we have used **h:inputText** tag . So the renderer of the component associated with this tag produces the following output:

<input type=”text” name=”ID” value=”current\_value”/>

This is called encoding. The encoded page is sent to the [browser](http://www.roseindia.net/jsf/jsfrenderers.shtml) and displayed.  
  
**Decoding:**  
Now if fields in form are filled by user and page is submitted by clicking the button, the browser sends the form data to the web [server](http://www.roseindia.net/jsf/jsfrenderers.shtml) as a “POST” request”. POST” request contains form data and URL of the form. This form data is placed in hash table and can be accessed by all components. Each component gets a chance to look that hash table to interpret the form data. This is called decoding.

Html output Tag handler asks each component to render itself. Tag handler call two rendering methods for each component :

1-encodeBegin() in doStartTag() and

2-encodeEnd() in doEndTag().

Opening tag, like <form> , is written by encodeBegin() method and closing tage </form>, is written by encodeEnd() method. Single tag, like <input>, that dosen’t require separate opening and closing tag is also written by encodeEnd() method.  
  
JSF tag handler may call third rendering method encodeChildren().if rendersChilden property of component is set to true i.e. if component has child components(compound component). For example: Html table component composed of input field components as column values for each row. So for a complex component tag handler calls encodeBegin(), then encodeChildren() and encodeEnd() on the component.If the child component also has its own children component then encodeChildren() method calls encodeBegin() and encodeEnd() on the child component.

Some components renders itself (**Direct rendering**) and some components uses renderer to render itself (**Delegated rendering**). So both are different. If component uses renderer, then JSF calls encoding methods of the renderer, not the encoding method of the component. Which renderer will be used is determined by getRenderer() method of the component. So rendering of UI components is separated out that makes it pluggable with other possible rendering i.e.if we want  a new functionality, like a new look and feel or rendering output to different client types, then its easy to add or plug a new renderer.

**Render kit :**  
Component classes generally transfer the task of generating output to the renderer. All JSF components follow it. Render kit is a set of related renderers. javax.faces.render.RenderKit is the class which represents the render kit.The default render kit contains renderers for html but it’s up to you to make it for other markup languages.Render kit can implement a skin (a look & feel).Render kit can target a specific device like phone, PC or markup language like HTML,WML, SVG. This is one of the best benefit of  JSF because JSF doesn't limit to any device or markup.  
  
Sometimes we may wish to customize the renderers of an existing RenderKit and sometimes create our own RenderKit. We typically create renderers for our  custom components.In this case we have to register renderers with existing RenderKits in JSF configuration file faces-config.xml.This renderer should define renderer-type of the original component. Also, you should provide the renderer-class that points to your custom renderer. The renderer class should extend javax.faces.render.Renderer class.  
  
<render-kit>  
<renderer>  
<renderer-type>........</renderer-type>  
<renderer-class>........</renderer-class>  
</renderer>  
</render-kit>  
  
The information mentioned in configuration file registers the renderer with the default html RenderKit. UI component's geRendererType() method is called that returns a string to identify the type of renderer that would be used by component and to see if it should delegate rendering to a renderer. If no matching render type is found then component renders itself without delegating to any renderer.  
  
<render-kit>  
    <render-kit-id>...........</render-kit-id>  
    <render-kit-class>........</render-kit-class>  
     
    <renderer>  
      <renderer-type>.........</renderer-type>  
      <renderer-class>........</renderer-class>  
    </renderer>  
      
</render-kit>

In this case, configuration file shows how to register the renderer (for ex. ButtonRenderer) specified in <renderer-class> which renders a component (for ex. Button) specified in <renderer-type> to a client (for ex. SVG) specified in <render-kit-id>.  
  
**Standard render kits:**  
JSF defines a standard RenderKit and set of associated Renderers that generate html markup. Renderer will be determined and handled automatically  according to the tag used.

JSF HTML Tag Reference

In this section, you will learn more about html tags provided in JSF. JSF HTML tags are used for design the page by using server side code. This tag is used by specifying the uri and prefix attribute of the taglib directive at the top of your [JSP](http://www.roseindia.net/jsf/JSFHTMLTags.shtml) file. These are illusrated below as follows:

***http://java.sun.com/jsf/html***

This is the uri for the JSF html tags. And for the prefix value commonly "h" is used but you can put anything as value of the prefix attribute of the taglib directive. When you declare any html tag as a JSF tag then you should write the tag as follows:

**<htmlprefixvalue:tagName attribute1="value" attribute2="value"></htmlprefixvalue:tagName attribute1="value" attribute2="value">.**

This is the syntax of specifying the html tag as a jsf tag. You can take an example of creation an input box by using jsf html tag as follows:

If the prefix value is "h" then the syntax will be seen like : **<h:inputText value="This is a text box."></h:inputText>.**

All the html tags have been used in the Rose India JSF Tutorial as you will get one by one ahead. You can skip for next section for getting html tags overview with complete code examples that can be used directly in your web [application](http://www.roseindia.net/jsf/JSFHTMLTags.shtml) for creating several html components as the need of [your application](http://www.roseindia.net/jsf/JSFHTMLTags.shtml).

# **JSF column Tag**

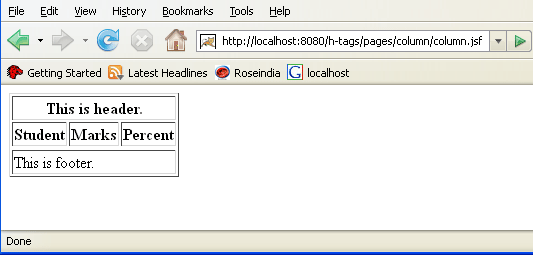
This section tells you about the JSF html column tag which is used for creating columns of a table. This tag creates a data column inside a data table. By this table you can specify number of column and fix these with some specific values. You can specify the value for specific column by using data array. JSF data table creates multiple rows up to the length of array or number of elements associated with the data table.

This section provides you a complete code of a program in which the column tags are used inside the data table. This program will help you for the procedure of using column tag for creating columns in a table.

**Code Description:**

|  |
| --- |
| <%@ page contentType="text/html" %> <%@ taglib uri="http://[java](http://www.roseindia.net/jsf/column.shtml).sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view> <html>      <head><title>jsf h:column example</title></head>       <body>           <h:dataTable border="1">           <f:facet name="header">               <h:outputText value="This is header."/>           </f:facet>           <h:column>               <f:facet name="header">                    <h:outputText value="Student"/>               </f:facet>           </h:column>           <h:column>               <f:facet name="header">                    <h:outputText value="Marks" />               </f:facet>           </h:column>           <h:column>               <f:facet name="header">                    <h:outputText value="Percent" />               </f:facet>           </h:column>           <f:facet name="footer">                <h:outputText value="This is footer."/>           </f:facet>           </h:dataTable>      </body> </html> </f:view> |

**Rendered Output:**



**HTML Source Code:**

|  |
| --- |
| <html>  <head><title>jsf h:column example</title></head>  <body>  <table border="1">  <thead>  <tr><th colspan="3" scope="colgroup">This is header.</th></tr>  <tr>  <th scope="col">Student</th>  <th scope="col">Marks</th>  <th scope="col">Percent</th>  </tr>  </thead>  <tfoot>  <tr><td colspan="3">This is footer.</td></tr>  </tfoot>  <tbody>  </tbody>  </table>  </body>  </html> |

JSF column tag has some attribute for different purposes. These attributes are explained below:

* **rendered:** This is an attribute of the **column** tag that is optional. This attribute can hold **String** typed value. This attribute value indicates that the component should be rendered during the rendering period.
* **binding:** This attribute is also optional and take a **String** typed value. Specified value is linked with the backing bean through the attribute of the tag.
* **id:** This attribute of the **column** tag is specified only for identification of the specific tag.

JSF commandButton Tag

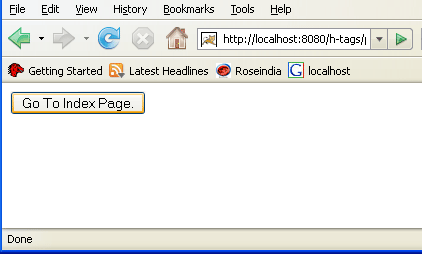
This section illustrates more about **commandButton** tag in JSF. This tag renders an HTML submit button. This button can be associated with bean. You can perform any operations at particular event by associating actionListener class for event handling. You can do any thing with JSF component by the external resources like showing message from the message bundle and handling form data after submission the form by backing bean.

This section also providing a program with complete code which display a command button inside a form. When you will click on the button, it's action attributes send a value "page1" to the faces-config.[xml file](http://www.roseindia.net/jsf/commandButton.shtml) where navigation has been made with the "page1" value. And the navigation refers the control to the Index page.

**Code Description:**

|  |
| --- |
| <%@ page contentType="text/html" %> <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view> <html>      <head><title>jsf h:commandButton example</title></head>       <body>           <h:form>                  <h:commandButton value="Go To Index Page." action="page1" />           </h:form>      </body> </html> </f:view> |

**Rendered Output:**



**HTML Source Code:**

|  |
| --- |
| <html>  <head><title>jsf h:commandButton example</title></head>  <body>  <form id="\_id0" method="post" action="/  h-tags/pages/commandButton/commandButton.jsf" enctype="application  /x-www-form-urlencoded">  <input type="submit" name="\_id0:\_id1" value="Go  To Index Page." />  <input type="hidden" name="\_id0" value="\_id0" />  </form>  </body>  </html> |

JSF commandButton tag has some attribute for different purposes. These attributes are explained below:

* **action:** This attribute holds a value for passing values from the page to the faces-config.xml file where the operation or navigation from the page to another page is decided.
* **actionListener:** The actionListener attribute of the commandButton tag is used to assign an action listener method from backing bean that performs the specific operation.
* **id:** Value of the attribute is used for identifying the component uniquely. This attribute must have a unique value in the closest container.
* **immediate:**It's value is a boolean value that indicates for the component events that should be sent to registered event listeners immediately. The immediate attribute allows you to turn off validation for a particular component.
* **rendered:** This attribute takes a boolean value that indicates for the rendering it or not in the view.
* **value:** This is the attribute holding the default value for the component whether it is specified directly or by the backing bean or any other external resources like message bundle.
* **accesskey:** This is the html attribute which specify key by pressing that key the component will be focused and accessed.
* **alt:** This is also a html attribute that is used for showing the textual description on mouse over of the component.
* **dir:** This attribute sets the value which define the component text direction. It sets the value like "LTR" means "left-to-right" and "RTL" means "right to left" direction.
* **disabled:** This attribute takes a boolean value. If the value is true then the component will be disabled otherwise the component will be enable.
* **image:** This attribute takes a relative or absolute url of the image that has to be displayed on the component. The image attribute is used for showing image on the component.
* **lang:** It sets the code for the language to be used in the markup generated by this component.
* **onblur:** This attribute indicates the event of the component. As the value of this attribute is defined as a [JavaScript](http://www.roseindia.net/jsf/commandButton.shtml) method which has to be performed when the component loses the focus.
* **onchange:** Specified JavaScript method is executed when the value has changed on losing focus after gaining focus.
* **onclick:** In the attribute, a JavaScript method is specified that is called when the component is clicked by user.
* **ondblclick:** This attribute indicates for performing the specified operation when the component is clicked two times continuously (or double click).
* **onfocus:** The JavaScript method can be called for the component when the component is focused.
* **onkeydown:** The specified JavaScript method or operation is performed when the key is pressed down over the component.
* **onkeypress:** This is the event of the component. It indicates the key press event. You can call a JavaScript method on the event of the component whatever you have mentioned.
* **onkeyup:** The specified JavaScript method is executed when key is released over this component.
* **onmousedown:** Specified JavaScript method is executed when mouse is pressed down over the component or element.
* **onmousemove:** This attribute sets the [JavaScript code](http://www.roseindia.net/jsf/commandButton.shtml) to executed when the mouse pointer is moved within the component or element.
* **onmouseout:** This attribute sets the JavaScript code to execute when the mouse pointer is moved away from the element or the component.
* **onmouseover:** This attribute sets the JavaScript code to execute when the mouse pointer is moved inside the element or the component.
* **onmouseup:** This attribute sets the JavaScript code to executed when the mouse pointer is released from the component.
* **onselect:** When you select the text contained by the component or element then the value of the attribute (JavaScript code) will be executed.
* **readonly:** This attribute sets the boolean value for making the component read-only or not.
* **style:** If you want ot add any CSS with the component then you can put the style as the value of the attribute. Added CSS will be applied on for the component.
* **styleClass:** This attribute holds the CSS class name which is defined in the external style sheet.
* **tabindex:** This attribute sets the tab index for the component. When you press the TAB key then the component will be focused after focusing all those components whose tab index is less than the component.
* **title:** This attribute holds a string value that is shown as a tool-tip text of a component or element.
* **type:** This attribute tells the component type whether it is submit type or reset etc.
* **binding:** This attribute binds the specified value with the backing bean

This section illustrates you about the JSF commandLink tag which is rendered as a anchor tag. And this tag behaves as a command button for the form submission and this tag is also used for the event handling purposes through the backing bean. This tag has text that can be labeled by some external resources like properties file from the message bundle.

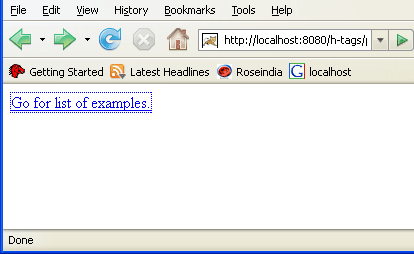
Here, you will see more about the commandLink tag of JSF how is it used in JSF [programming](http://www.roseindia.net/jsf/commandLink.shtml). There is a program with the complete code of JSF has been given for understanding the procedure of using the tag.

**Code Description:**

|  |
| --- |
| <%@ page contentType="text/html" %> <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view> <html>      <head><title>jsf h:commandLink example</title></head>       <body>           <h:form>                  <h:commandLink value="Go for list of examples." action="page2" />           </h:form>      </body> </html> </f:view> |

Here, when your run the above program, output will be seen like the following image in which the text "Go for list of examples." is looking like a hypertext. This is not only a anchor tag. This link behaves like a command button. You can perform an action at the specific event.

**Rendered Output:**



When you run the above example, your JSF tags are converted in to HTML code that is given. Following html source code is for the above written JSF code.

**HTML Source Code:**

|  |
| --- |
| <html>  <head><title>jsf h:commandLink example</title></head>  <body>  <form id="\_id0" method="post" action="/h-tags/pages/commandLink/  commandLink.jsf" enctype="application/x-www-form-urlencoded">  <a href="#" onclick="[document](http://www.roseindia.net/jsf/commandLink.shtml).forms['\_id0']['\_id0:\_idcl']  .value='\_id0:\_id1'; document.forms['\_id0'].submit(); return false;">Go  for list of examples.</a>  <input type="hidden" name="\_id0" value="\_id0" /><input  type="hidden" name="\_id0:\_idcl" />  </form>  </body>  </html> |

This JSF tag has some attributes these explained as follows:

* **accesskey:** This attribute set the key for the component through which the component can be accessed. Specified key is also used for transferring focus from one component to the component where it mentioned in with the JSF commandLink tag.
* **action:** This is also an attribute of the JSF commandLink tag. This attribute is used for handling events from the backing bean or any other resources to invoke the component when the component is activated by user. This type of action event is completed through a action method of the backing bean class. This method will either return a boolean value true or false by whom the Java [Server](http://www.roseindia.net/jsf/commandLink.shtml) Faces MVC Framework is designed. And navigations are depended on generated events and actions.
* **actionListener:** This attribute sets a method-binding expression with a backing bean. This method handles events.
* **binding:** This attribute binds values with backing bean.
* **charset:** This attribute sets the character encoding for the document that linked to by the hyperlink.
* **coords:** When you are using the link with a client-side image map, this attribute sets the position and the shape of spot on the screen.
* **dir:** This attribute set the direction of the text. The value for the attribute is accepted as "LTR" (left to right) or "RTL" (right to left).
* **hreflang:** This attribute sets the language code for the resource linked to by the hyperlink.
* **id:** This attribute sets the name for the identification of the component. It's value will be unique in the closest naming container.
* **immediate:** It's value is a boolean value that indicates for the component events that should be sent to registered event listeners immediately. The immediate attribute allows you to turn off validation for a particular component.
* **lang:** It sets the code for the language to be used in the markup generated by this component.
* **onblur:** This attribute sets [JavaScript code](http://www.roseindia.net/jsf/commandLink.shtml) when the component loses the focus.
* **ondblclick:** This attribute sets the [JavaScript](http://www.roseindia.net/jsf/commandLink.shtml) code when the component has been double-clicked over the component.
* **onfocus:** This attribute sets the JavaScript code when the component receives focus.
* **onkeydown:** This attribute sets the JavaScript code when key is pressed down over the element.
* **onkeypress:** This attribute sets the JavaScript code when key is pressed and released over the element or the component.
* **onkeyup:** This attribute sets the JavaScript code when key is released over the element or component.
* **onmousedown:** Specified JavaScript method is executed when mouse is pressed down over the component.
* **onmousemove:** This attribute sets the JavaScript code to executed when the mouse pointer is moved within the component.
* **onmouseout:** This attribute sets the JavaScript code to execute when the mouse pointer is moved away from the element.
* **onmouseover:** This attribute sets the JavaScript code to execute when the mouse pointer is moved inside the element.
* **onmouseup:** This attribute sets the JavaScript code to executed when the mouse pointer is released from the component.
* **rel:** This attribute tells you about the relationship between the current document and the document linked to by the hyperlink. It's values are the list of the link types separated by space from each other.
* **rev:** This attribute identifies a reserve link from the document linked to by the hyperlink to the current document. It's values are a list of link types separated by space from each other.
* **rendered:** This attribute takes a boolean value that indicates for the rendering it or not in the view.
* **shape:** This attribute sets the shape of the hotspot for viewing on the screen during the client side image mapping. It's some valid value is "rect" for the rectangular region, "default" for the entire region, "circle" for the circular region and "poly" for the polygonal region.
* **style:** If you want ot add any CSS with the component then you can put the style as the value of the attribute. Added CSS will be applied on for the component.
* **styleClass:** This attribute holds the CSS class name which is defined in the external style sheet.
* **tabindex:** This attribute sets the tab index for the component. When you press the TAB key then the component will be focused after focusing all those components whose tab index is less than the component.
* **target:** This attribute set the identification of a frame in which the resource has to be displayed that is linked to by the hyperlink.
* **title:** This attribute holds a string value that is shown as a tool-tip text of a component or element.
* **type:** This attribute tells the component type whether it is submit type or reset etc.
* **value:** This attribute set the display value for the component. It will be directly or any other resources like the backing bean or a message bundle. You can manage the backing bean for the value of the component or element.

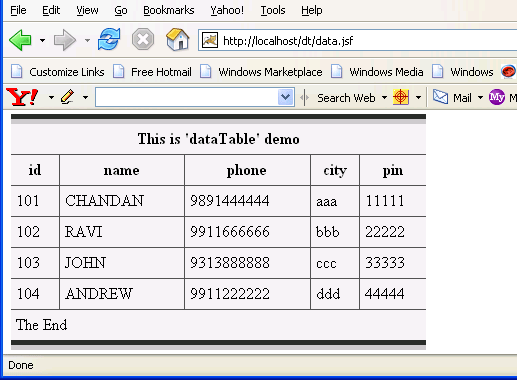
# **JSF dataTable Tag**

This tag is used to create table on the page. The component is rendered as an html <table> element. UIColumn child components are responsible for rendering columns of the table. In these columns you can put any type of component like input text box, output text, command button etc.**<h:column>** tag is used to create column. There can be many column tags within dataTable tag. You can set header and footer in this table. For this **<f:facet>** tag is used. [data table](http://www.roseindia.net/jsf/dataTable.shtml) component and its children column component can use header and footer facet.

We can associate this table element to backing bean. So we can obtain data from this backing bean and display it on the table. Association of backing bean can also be helpful for event handling purpose. Suppose we inserted command button in columns of the table then event handling can be applied here. If you want to customize the table then cascading stylesheet (CSS) can be used. This will help you to enhance the appearance of the table's headers, footer, rows, columns.

This section provides you the code to which uses this tag and some of its attributes. It uses backing bean that supplies data to the data table to be rendered to the cells of the columns of the table.  
  
**code description :**

|  |
| --- |
| <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>  <f:view><html><body> <h:form> <br><br><br> <h:dataTable id="dt1" value="#{TableBean.perInfoAll}" var="item" bgcolor="#F1F1F1" border="10" cellpadding="5" cellspacing="3" first="0" rows="4" width="50%" dir="LTR" frame="hsides" rules="all" summary="This is a JSF code to create dataTable." >  <f:facet name="header">         <h:outputText value="This is 'dataTable' demo" /> </f:facet>   <h:column>         <f:facet name="header">         <h:outputText value="id" />         </f:facet>               <h:outputText value="#{item.id}"></h:outputText> </h:column>  <h:column>         <f:facet name="header">         <h:outputText value="name"/>         </f:facet>               <h:outputText value="#{item.name}"></h:outputText> </h:column>  <h:column>         <f:facet name="header">         <h:outputText value="phone"/>         </f:facet>               <h:outputText value="#{item.phone}"></h:outputText> </h:column>  <h:column>         <f:facet name="header">         <h:outputText value="city"/>         </f:facet>              <h:outputText value="#{item.city}"></h:outputText> </h:column>  <h:column>         <f:facet name="header">         <h:outputText value="pin"/>         </f:facet>              <h:outputText value="#{item.pin}"></h:outputText> </h:column>  <f:facet name="footer">         <h:outputText value="The End" /> </f:facet>   </h:dataTable><br><br>  </h:form> </body></html></f:view> |

**Rendered Output :**

In this example we have used <h:dataTable> tag. This <h:dataTable> tag has many attributes to give form and shape to the table and getting data from backing bean's attribute to display this data to the columns of the table. **<f:facet>** tag is used to create a header and a   
footer for a dataTable component and it can create the same for columns also. So in this example we have used **facet tag in dataTabe tag** to create header and footer of the table component and in **column tag** also to create header for column. In this example, **"This is 'dataTable' demo"** is the header and **"The End"** is the footer for the table and **"id" "name" "phone" "city" "pin"** are headers for columns. In his example we have populated data to the table from bean named **"TableBean".** It contains an attribute "**perInfoAll**" that is an array of objects of type "**perInfo**" class which contains some attributes like id, name, phone, city, pin that is to be displayed on the table. In the value attribute of dataTable tag, we have specified the binding expression to bind this component to the bean (**value="#{TableBean.perInfoAll}"**) and value specified in var attribute is used to populate data to the column of the table from attributes defined in class perInfo (**value="#{item.id}"**). The code for the bean has been given below (remember to specify it in faces-cocfig.[xml file](http://www.roseindia.net/jsf/dataTable.shtml)) :

|  |
| --- |
| public class TableBean {  private perInfo[] perInfoAll = new perInfo[]{ new perInfo(101, "CHANDAN", "9891444444", "aaa", 11111), new perInfo(102, "RAVI", "9911666666", "bbb" ,22222), new perInfo(103, "JOHN", "9313888888", "ccc", 33333), new perInfo(104, "ANDREW", "9911222222", "ddd" , 44444), new perInfo(105, "SYMONDS", "9313999999", "eee", 55555), };  public perInfo[] getperInfoAll() { return perInfoAll; }  public class perInfo { int id; String name; String phone; String city; int pin;  public perInfo(int id, String name, String phone, String city, int pin) { this.id = id; this.name = name; this.phone = phone; this.city = city; this.pin= pin; }  public int getid() { return id; }  public String getname() { return name; }  public String getphone() { return phone; }  public String getcity() { return city; }  public int getpin() { return pin; }  }  } |

**HTML Source Code:**

|  |
| --- |
| <html>  <body>  <form id="\_id0" method="post" action="/dt/data.jsf;jsessionid  =B95AF7B5F7D80BCB5638B91AC8154C33" enctype="[application](http://www.roseindia.net/jsf/dataTable.shtml)/x-www-form-urlencoded">  <br><br><br>  <table id="\_id0:dt1" bgcolor="#F1F1F1" border="10"  cellpadding="5" cellspacing="3" dir="LTR" frame="hsides" rules="all"  summary="This is a JSF code to create dataTable." width="50%">  <thead>  <tr><th colspan="5" scope="colgroup">This is  'dataTable' demo</th></tr>  <tr>  <th scope="col">id</th>  <th scope="col">name</th>  <th scope="col">phone</th>  <th scope="col">city</th>  <th scope="col">pin</th>  </tr>  </thead>  <tfoot>  <tr><td colspan="5">The End</td></tr>  </tfoot>  <tbody>  <tr>  <td>101</td>  <td>CHANDAN</td>  <td>9891444444</td>  <td>aaa</td>  <td>11111</td>  </tr>  <tr>  <td>102</td>  <td>RAVI</td>  <td>9911666666</td>  <td>bbb</td>  <td>22222</td>  </tr>  <tr>  <td>103</td>  <td>JOHN</td>  <td>9313888888</td>  <td>ccc</td>  <td>33333</td>  </tr>  <tr>  <td>104</td>  <td>ANDREW</td>  <td>9911222222</td>  <td>ddd</td>  <td>44444</td>  </tr>  </tbody>  </table>  <br><br>  <input type="hidden" name="\_id0" value="\_id0" />  </form>  </body>  </html> |

This tag has some attributes. These are listed below :

* **id :** This is used to uniquely identify the table component. This must be unique within the closest parent component.
* **value :** It represents the value of the component. It represents the value over which iteration is to be done. It may be an array or any iterator object .
* **var :** This is the name of the variable created by the data table that represents the current item in the value. This attribute helps exposing the data in the rows of the table.
* **bgcolor :** This attribute is used to set the background [color](http://www.roseindia.net/jsf/dataTable.shtml) for the table.
* **border :** We can set the width of the table's border around the table.
* **cellpadding :** This sets the space between the content and the border of the cell.
* **cellspacing :** It specifies the amount of space to leave between cells.
* **first :** This is used to specify the row number of the first row from which displaying is to be started onwards. Suppose, this property is set to 3,displaying will be started from the third row of the underlying data.
* **rows :** This attribute specifies the number of rows to display. This displaying will be started from the index specified in the "first" attribute. If we set this attribute to zero then all rows will be displayed.
* **width :** This is used to set the width of the entire table. Its value is specified in %. Suppose we set it to 50% then this table will be shown in the 50% space of the width of your screen.
* **dir :** This attribute indicates the direction of the text to be displayed in the cell. It takes "LTR" (left-to-right) and "RTL" (right-to-left) values. If we don't specify this attribute then the content will be displayed in center.
* **frame :** This attribute specifyes which sides of the frame surrounding this table will be visible. This attribute can take some values shown below :

1. **none**            No side, Default Value
2. **above**          Top side only
3. **below**          Bottom side only
4. **hsides**         Top and bottom sides only
5. **vsides**         Right and left sides only
6. **lhs**               Left hand side only
7. **rhs**              Right hand side only
8. **box**             All four sides
9. **border**        All four sides

* **rules :** This attribute is used to draw lines between cells. It can take some values given below :
  1. **none**           No rules, default value
  2. **groups**        Between row groups
  3. **rows**           Between rows only
  4. **cols**            Between columns only
  5. **all**               Between all rows and columns
* **summary :** You can specify summary of the purpose of the table.
* **rendered :** It takes boolean value.This indicates whether or not this component should be rendered. Its default value is "true". If it is set to false then it prevents rendering of this component to the page.
* **captionClass :** Space separated list of CSS class or classes that will be applied to any caption generated for this table.
* **captionStyle :** It specifies CSS style or styles to be applied when this caption is rendered.
* **columnClasses :** Comma seperated list of CSS classes that will be applied to the columns of this table.
* **footerClass :** This attribute takes Space-separated list of CSS style class or classes that will be applied to aheaderter generated for this table.
* **headerClass :** This attribute takes Space-separated list of CSS style class or classes that will be   
  applied to any header generated for this table.
* **rowClasses :** It is a list of CSS classes applied to the rows of the table.These classes should be separated by comma. If we want to apply CSS class for individual rows then we can specify space separated list of CSS classes. Style classes are applied to rows in the same order that they are   
  defined. If we have two CSS classes then first class is applied to the first row and the second one is applied to the second. Then again in the third row, the first CSS is applied and so on. This process goes on till the last row of the table.
* **lang :**  It sets the base language of an element’s attributes and text i.e. the language used in the generated markup for this component.
* **styleClass :** It sets the name of CSS classor classes that is applied at the time of rendering the element.
* **title :** The title attribute is used to set the tooltip text to display for the rendered  component.Tooltip describes an element when rendered to the client.
* **binding :** It is a value binding expression that is used to link component to a property in a backing bean.
* **onclick :** It sets the JavaScript code to execute when a pointer button is clicked over this element.
* **ondblclick :** It sets the JavaScript code to execute when a pointer button is double clicked over this element.
* **onkeydown :** It sets the JavaScript code to execute when a key is pressed down over this element.
* **onkeypress :** It sets the JavaScript code to execute when a key is pressed and released over this element.
* **onkeyup :** It sets the JavaScript code to execute when a key is released over this element.
* **onmousedown :** It sets the JavaScript code to execute when a pointer button is pressed down over this element.
* **onmousemove :** It sets the JavaScript code to execute when a pointer button is moved within this element.
* **onmouseout :** It sets the JavaScript code to execute when a pointer button is moved away from this element.
* **onmouseover :** It sets the JavaScript code to execute when a pointer button is moved onto this element.
* **onmouseup :** It sets the JavaScript code to execute when a pointer button is released over this element.

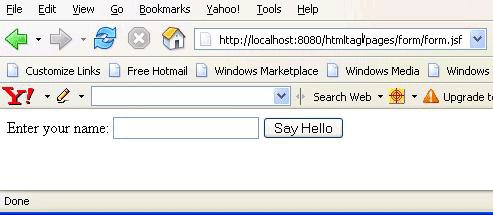
# **JSF form tag**

This tag renders html form element which contains the data that is submitted with the form. This tag uses "POST" method. The components under the particular form i.e. children of the form only are processed. "id" attribute is used to uniquely identify the form. [CSS](http://www.roseindia.net/jsf/form.shtml) can be used to make it more attractive.

**Code Description :**

|  |
| --- |
| <%@ page contentType="text/[html](http://www.roseindia.net/jsf/form.shtml)" %> <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view>   <html>     <head><title>jsf h:form example</title></head>     <body>       <h:form>         <h:outputText value="Enter your name: " /></td>         <h:inputText value="#{StoreNameBean.personName}" />         <h:commandButton action="result" value="Say Hello" />       </h:form>     </body>   </html> </f:view> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <html>    <head><title>jsf h:form example</title></head>    <body>      <form id="\_id0" method="post" action="/h-tags/pages/form/form.jsf" enctype="application/x-www-form-urlencoded"> Enter your name: </td>       <input type="text" name="\_id0:\_id2" />       <input type="submit" name="\_id0:\_id3" value="Say Hello" />       <input type="hidden" name="\_id0" value="\_id0" /></form>    </body> </html> |

This tag contains some attributes that are discussed below :

* **id :** This attribute is used to uniquely identify the element within the closest container.
* **rendered :** Its a boolean attribute. Its default value is true. It determines whether this component should be rendered or not.
* **binding :** It takes the value binding expression that is used to link the component to the property of the backing bean.
* **dir :** It is used to set the direction of the text to be displayed. It can take two values LTR(left to right) and RTL (right to left).
* **lang :** It is used to set the base language of the component when displayed.
* **style :** It is used to set the CSS style definition for the component.
* **title :** It is the [standard html](http://www.roseindia.net/jsf/form.shtml) attribute. It is used to set the tooltip text for this component.
* **styleClass :** It is used to set the CSS class for the component.
* **target :** This is used to identify the name of the frame where the response generated by the [server](http://www.roseindia.net/jsf/form.shtml) is displayed when we submit the form.
* **accept :** Its a comma separated list of content types that is handled by the server which processes this form.
* **acceptcharset :** It is used to list character encoding used for the data input by the user and that will be accepted by the user.
* **enctype :** This is used to set the content type of the data that will be submitted to the server.
* **onclick :** Script to be invoked when the element is clicked.
* **ondblclick :** It is used for Java Script code to be invoked when the element is double-clicked.
* **onmousedown :** It is used for Java Script code to be invoked when the pointing device is pressed over this element.
* **onmouseup :** It is used for Java Script code to be invoked when the pointing device is released over this element.
* **onmouseover :** It is used for Java Script code to be invoked when the pointing device is moved into this element.
* **onmousemove :** It is used for Java Script code to be invoked when the pointing device is moved while it is in this element.
* **onmouseout :** It is used for Java Script code to be invoked when the pointing device is moves out of this element.
* **onkeypress :** It is used for Java Script code to be invoked when a key is pressed over this element.
* **onkeydown :** It is used for Java Script code to be invoked when a key is pressed down over this element.
* **onkeyup :** It is used for Java Script code to be invoked when a key is released over this element.
* **onreset :** It is used for Java Script code to be invoked when form is reset.
* **onsubmit :** It is used for Java Script code to be invoked when form is submitted.

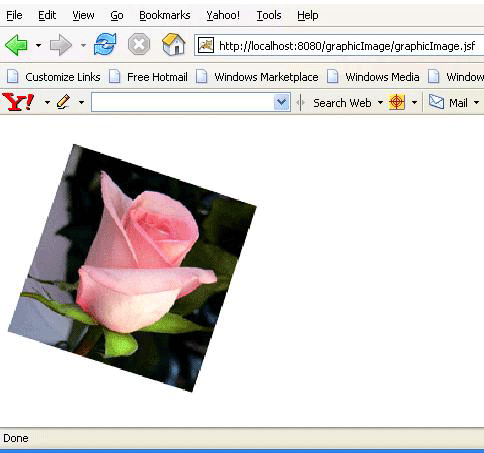
JSF graphicImage Tag

This section explains about **"graphicImage"** tag. This displays the image on the page. This tag renders an html "**img"** element. This tag renders the image file stored in the location specified in the **"value"** attribute of **"graphicImage"** tag. If this  image file is not present in the specified location then the value of the alt attribute is displayed instead of the desired image file. In this example, **value** attribute is set to the location **"/image/rose.gif"** in the [web-application](http://www.roseindia.net/jsf/graphicImage.shtml) root directory and **alt** attribute is set to "**The image could not be found.".** So this image is displayed on the page, if there is any problem in getting this image file (**"/image/rose.gif")** then the text specified in the alt attribute ("**The image could not be found.")** is displayed on the page.  We can set width, height of the image. If we want tooltip for this image to be displayed when focus comes to the image, then we can use **title** attribute. There are several attributes of this tag to give it different look and structure.

This section provides you the code that uses this tag and some of its attributes to render the image on the page.  **Code Description :**

|  |
| --- |
| <%@ taglib uri="http://java.sun.com/[jsf](http://www.roseindia.net/jsf/graphicImage.shtml)/html" prefix="h"%> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>  <f:view>           <html>               <body>                   <h:form><br>                         <h:graphicImage id="gi" alt="The image could not be found."     value="/image/rose.gif" width="250" height="250" title="This is demo for 'graphicImage' tag" ></h:graphicImage>                  </h:form>             </body>        </html> </f:view> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <html> <body>   <form id="\_id0" method="post"            action="/graphicImage/graphicImage.jsf" enctype="application/x-www-form-urlencoded"><br>       <img id="\_id0:gi" src="/graphicImage/image/rose.gif" alt="The image could not be found." height="250" title="This is demo for 'graphicImage' tag" width="250" />        <input type="hidden" name="\_id0" value="\_id0" />      </form> </body>  </html> |

All attributes of  graphicImage tag have been described  below:

* **id :** This is the identifier for the component. This must be the unique value within the closest parent component.
* **alt :** This is [standard html](http://www.roseindia.net/jsf/graphicImage.shtml) attribute. This is the alternate textual value to be displayed in the absence of the image file specified in the value attribute.
* **value :** This attribute takes context relative URL to the image. URL  is called  relative to the context path of the web-application if URL starts with '/'. This image is rendered, if present, on the page.
* **url :** Its an alias for the value attribute.
* **width :** It is used to set new width of the image over the natural width.
* **height :**It is used to set new height of the image over the natural height.
* **title :** This is standard html attribute. It sets the tooltip for the component.
* **rendered :** It takes boolean value. Its default value is "true". It indicates whether the component should be rendered or not.
* **dir :** It sets the direction of the text to be displayed. It can take two values "LTR" (Left to Right) and "RTL" (Right to Left).
* **lang :** This is a standard html attribute. It is used to describe the base language used in the markup generated for this component.
* **ismap :** This takes a boolean value. This is used to indicate whether this image is to be used as a server side image or not.
* **onclick :** It sets the [Java Script code](http://www.roseindia.net/jsf/graphicImage.shtml) to be invoked when the element is clicked.
* **ondblclick :** It sets the Java Script code to be invoked when the element is double-clicked.
* **onkeydown :** It sets the Java Script code to be invoked when a key is pressed down over this element.
* **onkeypress :** It sets the Java Script code to be invoked when a key is pressed over this element.
* **onkeyup :** It sets the Java Script code to be invoked when a key is released over this element.
* **onmousedown :** It sets the Java Script code to be invoked when the pointing device is pressed over this element.
* **onmousemove :** It sets the Java Script code to be invoked when the pointing device is moved within this element.
* **onmouseout :** It sets the Java Script code to be invoked when the pointing device is moves out of this element.
* **onmouseover :** It sets the Java Script code to be invoked when the pointing device is moved into this element.
* **onmouseup :** It sets the Java Script code to be invoked when the pointing device is released over this element.
* **style :** This is used to set the CSS style definition which will be applied to the component when it is rendered.
* **styleClass :** This is used to set the CSS class which will be applied to the component when it is rendered.
* **usemap :** This is html map element. This describes the name of client side [image map](http://www.roseindia.net/jsf/graphicImage.shtml) for which this element provides the image.
* **binding :** Its a value binding expression that is used to link the component to the backing bean property.

JSF inputHidden Tag

This section describes the inputHidden JSF tag. This tag is used to create the field that is invisible to the user. This is the field that is used to pass the variables from one page to another. It renders the [html](http://www.roseindia.net/jsf/inputHidden.shtml) **input** element with the **type** attribute set to **"hidden".**

**Code Description :**

|  |
| --- |
| <%@ taglib uri="http://[java](http://www.roseindia.net/jsf/inputHidden.shtml).sun.com/jsf/html" prefix="h"%> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>  <f:view>   <html>      <body>         <h:form><br>            <h:inputHidden id="ih" value="hv"></h:inputHidden>         </h:form>      </body>   </html> </f:view> |

**Rendered Output :**

There is **no rendered output** for this tag . This gives **invisible element** as an output.

**Html Scorce Code :**

|  |
| --- |
| <html>   <body>      <form id="\_id0" method="post" action="/inputHidden/inputHidden.jsf"   enctype="[application](http://www.roseindia.net/jsf/inputHidden.shtml)/x-www-form-urlencoded"><br> <input id="\_id0:ih" type="hidden" name="\_id0:ih" value="hv" /> <input type="hidden" name="\_id0" value="\_id0" />      </form>   </body> </html> |

There is a list of all attributes that can be used in this **inputHidden** tag:

* **id :** This is set to uniquely identify the component. So this must be the unique value within the closest container.
* **value :** This is the current value of the component.
* **binding :** This is used to set the binding expression that is used to link the component to the backing bean's property.
* **rendered :** This is the boolean attribute that is set to describe that this component should be rendered or not at the time of render response  phase.
* **required :** This is the boolean attribute. User is required to provide the value for the field or not  at the time of  submission of the form is indicated by this attribute.
* **validator :** It takes a method binding expression that represents validator method. This method is called to validate the value of the component.
* **immediate :** This is the boolean attribute. This is used to ensure that the events should be sent to the associated listener immediately. It should not be sent after validation phase.
* **converter :** It is used to register the converter instance to the component.
* **valueChangeListener :** It takes the method binding expression that notifies the value change listener method. It notifies this method when value is changed and new value is set for this component.

JSF inputSecret Tag

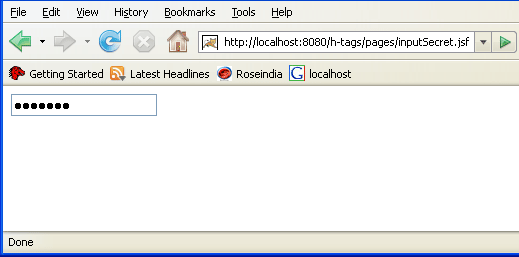
This section illustrates you more about the JSF HTML **inputSecret** tag. This tag is used to create a [text box](http://www.roseindia.net/jsf/inputSecret.shtml) for having password that is secret. This text box shows only a symbol for every enteries. You can't understand the value of the text box by seeing.

Here, you will see in the program code that the value of the **inputSecret** tag has been assigned "chandan" but by seeing it's output as given below, any one can't understand. This type of text box helps you to keep your data secret.

**Code Description:**

|  |
| --- |
| <%@ page contentType="text/html" %> <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view> <html>      <head><title>jsf h:message example</title></head>       <body>           <h:inputSecret value="chandan" />      </body> </html> </f:view> |

**Rendered Output:**



**HTML Source Code:**

Following is the html code generated by the browser when your JSF program is run.

|  |
| --- |
| <html>  <head><title>jsf h:message example</title></head>  <body>  <input type="password" name="\_id0" value="" />  </body>  </html> |

All attributes of the **inputSecret** tag are explained as follows:

* **accesskey:** This is [standard html](http://www.roseindia.net/jsf/inputSecret.shtml) attribute. It is used to set the access key for the element which is used to send the focus to the element when pressed.
* **alt:** This is used as an alternate text that is displayed when browser is not able to display the element.
* **binding:** It takes the value binding expression that is used to link the component to the property of the backing bean.
* **converter:** This specifies the converter for the component. This can be static value or EL expression.
* **dir:** It is used to set the direction of the text to be displayed. It can take two values LTR(left to right) and RTL (right to left).
* **disabled:** Its a boolean attribute. This is used to disable the element to receive focus, when it is set to true.
* **id:** This attribute is used to uniquely identify the element within the closest container.
* **immediate:** Its a boolean attribute. It is used to identify during which phase value change event should occur. If this attribute is set to true then in place of firing the event during the process validation phase, these event are sent immadiately at the end of apply request values phase.
* **lang:** It is used to set the base language of the component when displayed.
* **maxlength:** It is used to set the maximum length of character that can be input into the text field.
* **onblur:** It is used to set the java script code to execute when focus is lost from the element.
* **onchange:** It is used to set the java script code to execute when element is modified.
* **onclick:** Script to be invoked when the element is clicked.
* **ondblclick:** It is used for Java Script code to be invoked when the element is double-clicked.
* **onfocus:** It is used to set the java script code to execute when element receives the focus.
* **onkeydown:** It is used for Java Script code to be invoked when a key is pressed down over this element.
* **onkeypress:** It is used for Java Script code to be invoked when a key is pressed over this element.
* **onkeyup:** It is used for Java Script code to be invoked when a key is released over this element.
* **onmousedown:** It is used for Java Script code to be invoked when the [pointing device](http://www.roseindia.net/jsf/inputSecret.shtml) is pressed over this element.
* **onmousemove:** It is used for Java Script code to be invoked when the pointing device is moved while it is in this element.
* **onmouseout:** It is used for Java Script code to be invoked when the pointing device is moves out of this element.
* **onmouseover:** It is used for Java Script code to be invoked when the pointing device is moved into this element.
* **onmouseup:** It is used for Java Script code to be invoked when the pointing device is released over this element.
* **onselect:** It is used to set the java script code to execute when text of the component is selected by the user.
* **readonly:** Its a boolean attribute. It is used to indicate the user that [its value](http://www.roseindia.net/jsf/inputSecret.shtml) can't be modified, if it is set to true.
* **redisplay:** This is a boolean attribute. This is used to specify that the password, that have been entered previously, has to be rendered again or not. Its default value is **"false"** because we generally need not to do this for security reasons.
* **rendered:** Its a boolean attribute. Its default value is true. It determines whether this component should be rendered or not.
* **required:** Its a boolean attribute. It indicates that its value is required by the user before the submission of the form to the [server](http://www.roseindia.net/jsf/inputSecret.shtml). If it is set to true and value is not provided then an error message comes.
* **size:** It is used to set the width (in character) of the component.
* **style:** It is used to set the CSS style definition for the component.
* **styleClass:** It is used to set the CSS class for the component.
* **tabindex:** This is a standard html attribute. It is used to set the order of receiving the focus on the movement of TAB key by the user.
* **title:** It is the standard html attribute. It is used to set the tooltip text for this component.
* **validator:** It takes the method binding expression. This expression represents the validator method. This method is called at the time of validation of the component.
* **value:** This is to set the current value of the component.
* **valueChangeListener:** This also takes a method binding expression. This expression represents value change listener method. This method will be called when new value is set for this component.

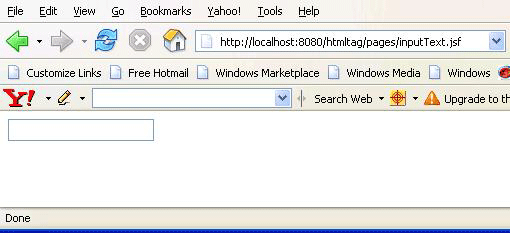
JSF inputText tag

This tag is used to create html input element which has the type **"text"**. It creates input text field where text can be entered.

**Code Description :**

|  |
| --- |
| <%@ page contentType="text/html" %> <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %> <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  <f:view>   <html>     <head><title>jsf h:inputText example</title></head>     <body>          <h:inputText />     </body>   </html> </f:view> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <html>     <head><title>jsf h:inputText example</title></head>          <body>              <input type="text" name="\_id0" />          </body> </html> |

 This tag contains some attributes that are discussed below :

* **id :** This attribute is used to uniquely identify the element within the closest container.
* **rendered :** Its a boolean attribute. Its default value is true. It determines whether this component should be rendered or not.
* **binding :** It takes the value binding expression that is used to link the component to the property of the backing bean.
* **dir :** It is used to set the direction of the text to be displayed. It can take two values LTR(left to right) and RTL (right to left).
* **lang :** It is used to set the base language of the component when displayed.
* **style :** It is used to set the CSS style definition for the component.
* **accesskey :** This is [standard html](http://www.roseindia.net/jsf/inputText.shtml) attribute. It is used to set the access key for the element which is used to send the focus to the element when pressed.
* **alt :** This is used as an alternate text that is displayed when [browser](http://www.roseindia.net/jsf/inputText.shtml) is not able to display the element.
* **converter :** This specifies the converter for the component. This can be static value or EL expression .
* **disabled :** Its a boolean attribute. This is used to disable the element to receive focus, when it is set to true.
* **immediate :** Its a boolean attribute. It is used to identify during which phase value change event should occur. If this attribute is set to true then in place of firing the event during the process validation phase, these event are sent immadiately at the end of apply request values phase.
* **maxlength :** It is used to set the maximum length of character that can be input into the text field.
* **onblur :** It is used to set the java script code to execute when focus is lost from the element.
* **onchange** : It is used to set the java script code to execute when element is modified.
* **size :** It is used to set  the width (in character) of the component.
* **title :** It is the standard html attribute. It is used to set the tooltip text for this component.
* **styleClass :** It is used to set the CSS class for the component.
* **onclick :** Script to be invoked when the element is clicked.
* **ondblclick :** It is used for Java Script code to be invoked when the element is double-clicked.
* **onmousedown :** It is used for Java Script code to be invoked when the [pointing device](http://www.roseindia.net/jsf/inputText.shtml) is pressed over this element.
* **onmouseup :** It is used for Java Script code to be invoked when the pointing device is released over this element.
* **onmouseover :** It is used for Java Script code to be invoked when the pointing device is moved into this element.
* **onmousemove :** It is used for Java Script code to be invoked when the pointing device is moved while it is in this element.
* **onmouseout :** It is used for Java Script code to be invoked when the pointing device is moves out of this element.
* **onkeypress :** It is used for Java Script code to be invoked when a key is pressed over this element.
* **onfocus :** It is used to set the java script code to execute when element receives the  focus.
* **onkeydown :** It is used for Java Script code to be invoked when a key is pressed down over this element.
* **onkeyup :** It is used for Java Script code to be invoked when a key is released over this element.
* **onselect :** It is used to set the java script code to execute when text of the component is selected by the user.
* **readonly :** Its a boolean attribute. It is used to indicate the user that [its value](http://www.roseindia.net/jsf/inputText.shtml) can't be modified, if it is set to true.
* **required :** Its a boolean attribute. It indicates that its value is required by the user before the submission of the form to the [server](http://www.roseindia.net/jsf/inputText.shtml). If it is set to true and value is not provided then an error message comes.
* **tabindex :** This is a standard html attribute. It is used to set the order of receiving the focus on the movement of TAB key by the user.
* **validator :** It takes the method binding expression. This expression represents the validator method. This method is called at the time of validation of the component.
* **value :** This is to set the current value of the component.
* **valueChangeListener :** This also takes a method binding expression. This expression represents value change listener method. This method will be called when new value is set for this component.

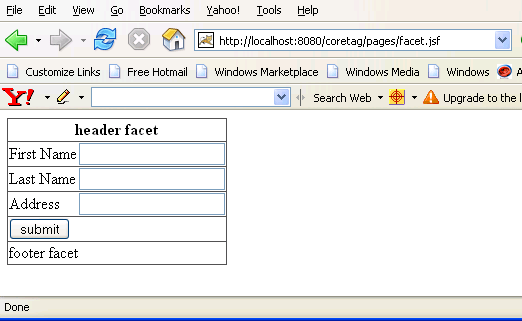
JSF facet Tag

This tag is used to add a facet to the component means this tag is used to add its child as a facet of the closest parent component. With the help of this tag we can add header and footer facet to the container component like panelGroup.

**Code Description :**

|  |
| --- |
| <%@ taglib uri="http://java.sun.com/[jsf](http://www.roseindia.net/jsf/facet.shtml)/html" prefix="h"%> <%@ taglib uri="http://[java](http://www.roseindia.net/jsf/facet.shtml).sun.com/jsf/core" prefix="f"%>  <f:view> <html> <body> <h:form> <h:panelGrid columns="2" border="1" rules="rows" title="This is facet tag demo"> <f:facet name="header"> <h:outputText value="header facet"/> </f:facet> <h:outputText value="First Name"/> <h:inputText/> <h:outputText value="Last Name"/> <h:inputText/> <h:outputText value="Address"/> <h:inputText/> <h:commandButton value="submit"/>  <f:facet name="footer">  <h:outputText value="footer facet" /> </f:facet> </h:panelGrid>  </h:form> </body> </html> </f:view> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <html> <body> <form id="\_id0" method="post" action="/coretag/pages/facet.jsf" enctype="[application](http://www.roseindia.net/jsf/facet.shtml)/x-www-form-urlencoded"> <table border="1" rules="rows" title="This is facet tag demo"> <thead> <tr><th colspan="2" scope="colgroup">header facet</th></tr> </thead>  <tfoot> <tr><td colspan="2">footer facet</td></tr> </tfoot> <tbody> <tr> <td>First Name</td> <td><[input type](http://www.roseindia.net/jsf/facet.shtml)="text" name="\_id0:\_id4" /></td> </tr> <tr> <td>Last Name</td> <td><input type="text" name="\_id0:\_id6" /></td> </tr> <tr> <td>Address</td>  <td><input type="text" name="\_id0:\_id8" /></td> </tr> <tr> <td><input type="submit" name="\_id0:\_id9" value="submit" /></td> </tr> </tbody> </table>  <input type="hidden" name="\_id0" value="\_id0" /></form> </body> </html> |

This tag contains one attribute :  
  
**name :** This is the required attribute and is used to set the name of the facet. "header" and "footer" values can be used for this attribute

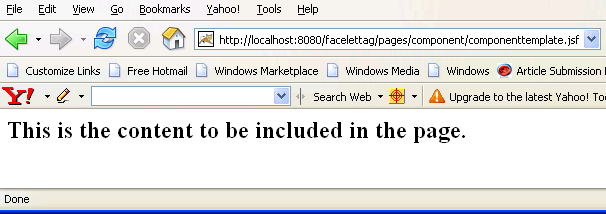
# **Facelet component Tag**

This tag is used to add a new component into the JSF component tree as children of UI component instance. This tag shows its behavior like composition tag. The difference is that the component tag inserts a new UIcomponent instance in the component tree and this instance is the root of all its child components or fragments. The content outside of the tag is ignored as it happens with composition tag.

**Code Description :    
  
comptemplate.xhtml :**

|  |
| --- |
| <![DOCTYPE html](http://www.roseindia.net/jsf/component.shtml) PUBLIC "-//W3C//[DTD XHTML](http://www.roseindia.net/jsf/component.shtml) 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://[java](http://www.roseindia.net/jsf/component.shtml).sun.com/jsf/facelets"> <head> <title>facelet example </title> </head> <body>  Content above component tag will not be rendered. <ui:component > <h2>This is the content to be included in the page.</h2> </ui:component > Content below component tag will not be rendered. </body> </html> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <h2>This is the content to be included in the page.</h2> |

 This tag contains two attributes :

**id :** This attribute assigns unique identity to the component. Its not a required attribute. So if you don't give id then facelet provides it automatically. **binding :** This attribute is used to bind the component to the backing bean property.

Facelet composition Tag

This is a templating tag and is used for the wrapping the content that can be included in any other facelet. This tag provides some useful features. Any content outside of this tag is left to be rendered. You can  include normal html content  in your page but Facelet will render only content that is within this tag i.e. composition tag. This tag takes one attribute named **"template"**. This attribute is set to the path of the template where the content of this tag will be included.

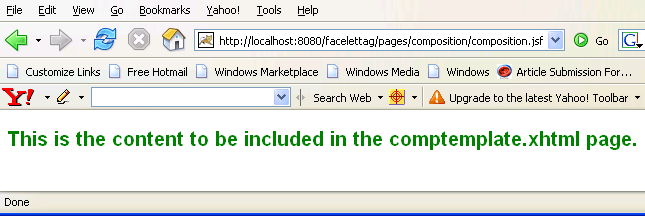
**Code Description :   
  
composition.xhtml :** In the code below we have taken template attribute which indicates the template to which the content inside this composition tag will be rendered. Here we have written some content outside of the composition tag, that content will not be rendered. In the comptemplate.xhtml we have used **insert** tag to include the content inside the composition tag to the comptemplate.xhtml page.

|  |
| --- |
| <![DOCTYPE html](http://www.roseindia.net/jsf/composition.shtml) PUBLIC "-//W3C//[DTD XHTML](http://www.roseindia.net/jsf/composition.shtml) 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml"          xmlns:ui="http://java.sun.com/[jsf](http://www.roseindia.net/jsf/composition.shtml)/facelets">    <body>         Content above composition tag will **not** **be rendered**.         <**ui:composition** **template**="/pages/composition/**comptemplate.xhtml**">              <h2>This is the content to **be included** in the comptemplate.xhtml page.</h2>         </ui:composition>         Content below composition tag will **not** **be rendered**.    </body> </html> |

**comptemplate.xhtml :**

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml"          xmlns:ui="http://[java](http://www.roseindia.net/jsf/composition.shtml).sun.com/jsf/facelets">     <head>         <title>facelet example </title>             <link href="../../style/CSS.css" rel="stylesheet" type="text/css"/>     </head>     <body>          <ui:insert />     </body> </html> |

**Rendered Output :**



**Html Source Code :**

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" > <html xmlns="http://www.w3.org/1999/xhtml">    <head>        <title>facelet example </title>             <link href="../../style/CSS.css" rel="stylesheet" type="text/css" />       </head>       <body>            <h2>This is the content to be included in the comptemplate.xhtml page.</h2>       </body> </html> |

 This tag contains only one attribute :

**template :** This attribute is set to the template where the content inside this tag will be included.

Performance increase by adding these entries in web.xml.

<context-param>

<description>

State saving method: "client" or "server" (= default) See

JSF Specification 2.5.3

</description>

<param-name>javax.faces.STATE\_SAVING\_METHOD</param-name>

<param-value>server</param-value>

</context-param>

<context-param>

<description>

Only applicable if state saving method is "server" (=

default). Defines the amount (default = 20) of the latest

views are stored in session.

</description>

<param-name>

org.apache.myfaces.NUMBER\_OF\_VIEWS\_IN\_SESSION

</param-name>

<param-value>20</param-value>

</context-param>

<context-param>

<description>

Only applicable if state saving method is "server" (=

default). If true (default) the state will be serialized to

a byte stream before it is written to the session. If false

the state will not be serialized to a byte stream.

</description>

<param-name>

org.apache.myfaces.SERIALIZE\_STATE\_IN\_SESSION

</param-name>

<param-value>true</param-value>

</context-param>

<context-param>

<description>

Only applicable if state saving method is "server" (=

default) and if

org.apache.myfaces.SERIALIZE\_STATE\_IN\_SESSION is true (=

default) If true (default) the serialized state will be

compressed before it is written to the session. If false the

state will not be compressed.

</description>

<param-name>

org.apache.myfaces.COMPRESS\_STATE\_IN\_SESSION

</param-name>

<param-value>true</param-value>

</context-param>