**The JSF lifecycle: an overview**

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

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

The six phases show the order in which JSF typically processes a form GUI. The list shows the phases in their likely order of execution with event processing at each phase, but the JSF lifecycle is hardly set in stone. You can change the order of execution by skipping phases or leaving the lifecycle altogether. For example, if an invalid request value were copied to a component, the current view would be redisplayed, and some of the phases might not execute. In this case, you could issue a FacesContext.responseComplete method invocation to redirect the user to a different page, then use the request dispatcher (retrieved from the request object in the FacesContext) to forward to an appropriate Web resource. Alternately, you could call FacesContext.renderResponse to re-render the original view. (See the sample application below for details.)

The point is to let the lifecycle structure your development efforts without feeling completely tied to it. You can alter the lifecycle when needed without fear of breaking your application. In most cases, you'll find that the JSF lifecycle is worth adhering to because it's quite logical. Forms have to be validated before any application logic can be executed, and field data has to be converted before being validated. Sticking to the lifecycle frees you up to think about the details of validation and conversion, rather than the phases of the request process itself. It's also important to note that other Web frameworks have similar lifecycles; they just don't tend to be as well advertised.

**Focusing your efforts**

Some developers using JSF may never write a component or extend the framework, while others may focus on just those tasks. While the JSF lifecycle will be the same for almost any project, you'll likely tap into it at different stages based on your role in the project. If you're concentrating more on the overall application development, you'll likely be concerned with the middle phases of the request processing lifecycle:

* Apply requests values
* Process validations
* Update model values
* Invoke application

If you're concentrating on JSF component development, you'll probably focus on the first and last phases of the lifecycle:

* Restore view
* Render response

In the sections that follow, I'll walk you through every phase of the JSF request processing lifecycle, including event handling and validation. Once you have a basic understanding of each phase, I'll introduce a sample application that shows how they all come together. Before we get started, take a look at Figure 1, a diagram of the JSF lifecycle.

**Phase 1: Restore view**

In the first phase of the JSF lifecycle -- *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.

The JSF framework controller uses the view ID to look up the components for the current view. If the view doesn't already exist, the JSF controller creates it. If the view already exists, the JSF controller uses it. The view contains all the GUI components.

This phase of the lifecycle presents three view instances: new view, initial view, and postback, with each one being handled differently. In the case of a *new view*, JSF builds the view of the Faces page and wires the event handlers and validators to the components. The view is saved in aFacesContext 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; viewRoot contains all the JSF components for the current view ID.

In the case of an *initial view* (the first time a page is loaded), JSF creates an empty view. The empty view will be populated as the user causes events to occur. From an initial view, JSF advances directly to the render response phase.

In the case of a *postback* (the user returns to a page she has previously accessed), the view corresponding to the page already exists, so it needs only to be restored. In this case, JSF uses the existing view's state information to reconstruct its state. The next phase after a postback is apply request values.

**Phase 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. Component values are typically retrieved from the request parameters, although they can also be retrieved from cookies or headers.

If a component's immediate event handling property is *not* set to true, the values are just converted. So if the field is bound to an Integerproperty, the value is converted to an Integer. If the value conversion fails, an error message is generated and queued in the FacesContext, where it will be displayed during the render response phase, along with any validation errors.

If a component's immediate event handling property *is* set to true, the values are converted to the proper type and validated. The converted value is then stored in the component. If the value conversion or value validation fails, an error message is generated and queued in the FacesContext, where it will be displayed during the render response phase, along with any other validation errors.

**Immediate event handling**

The immediate event handling property of JSF is used to handle events that normally don't necessitate validating the entire form. For example, say an employee form has a radio button denoting whether an employee is a manager. When the employee selects the Manager option, the application populates a list of perks for managers. Because the radio button is used only to populate the list and does not require the user to fill out the entire form, you do not need to validate the form in its entirety. In this case, you would use immediate event handling. See [Immediate event handling](http://www.ibm.com/developerworks/library/j-jsf2/#immediate) for further details on this subject.

**Phase 3: Process validation**

The first event handling of the lifecycle takes place after the apply request values phase. At this stage, each component will have its values validated against the application's validation rules. The validation rules can be pre-defined (shipped with JSF) or defined by the developer. Values entered by the user are compared to the validation rules. If an entered value is invalid, an error message is added to FacesContext, and the component is marked invalid. If a component is marked invalid, JSF advances to the render response phase, which will display the current view with the validation error messages. If there are no validation errors, JSF advances to the update model values phase.

**Phase 4: Update model values**

The fourth phase of the JSF application lifecycle -- *update model values* -- updates the actual values of the server-side model -- namely, by updating the properties of your backing beans (also known as managed beans). Only bean properties that are bound to a component's value will be updated. Notice that this phase happens after validation, so you can be sure that the values copied to your bean's properties are valid (at least at the form-field level; they may still be invalid at the business-rule level).

**Phase 5: Invoke application**

At the fifth phase of the lifecycle -- *invoke application* -- the JSF controller invokes the application to handle Form submissions. The component values will have been converted, validated, and applied to the model objects, so you can now use them to execute the application's business logic.

At this phase, you also get to specify the next logical view for a given sequence or number of possible sequences. You do this by defining a specific outcome for a successful form submission and returning that outcome. For example: *on successful outcome, move the user to the next page*. For this navigation to work, you will have to create a mapping to the *successful outcome* as a navigation rule in the faces-config.xml file. Once the navigation occurs, you move to the final phase of the lifecycle.

**Phase 6: Render response**

In the sixth phase of the lifecycle -- *render response* -- you display the view with all of its components in their current state.