4. Partial Rendering and Processing

PrimeFaces provides a partial rendering and view processing feature based on standard JSF 2 APIs to enable choosing what to process in JSF lifecycle and what to render in the end with ajax.

4.1 Partial Rendering

In addition to components like autoComplete, datatable, slider with built-in ajax capabilities, PrimeFaces also provides a generic PPR (Partial Page Rendering) mechanism to update JSF components with ajax. Several components are equipped with the common PPR attributes (e.g. update, process, onstart, oncomplete).

4.1.1 Infrastructure

PrimeFaces Ajax Framework is based on standard server side APIs of JSF 2. There are no additional artifacts like custom AjaxViewRoot, AjaxStateManager, AjaxViewHandler, Servlet Filters, HtmlParsers, PhaseListeners and so on. PrimeFaces aims to keep it clean, fast and lightweight.

On client side rather than using client side API implementations of JSF implementations like Mojarra and MyFaces, PrimeFaces scripts are based on the most popular javascript library; jQuery which is far more tested, stable regarding ajax, dom handling, dom tree traversing than a JSF implementations scripts.

4.1.2 Using IDs

Getting Started

When using PPR you need to specify which component(s) to update with ajax. If the component that triggers PPR request is at the same namingcontainer (eg. form) with the component(s) it renders, you can use the server ids directly. In this section although we’ll be using commandButton, same applies to every component that’s capable of PPR such as commandLink, poll, remoteCommand and etc.

```xml
<h:form>
    <p:commandButton update="display" />
    <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

PrependId

Setting prependId setting of a form has no effect on how PPR is used.
ClientId

It is also possible to define the client id of the component to update.

```html
<h:form id="myform">
    <p:commandButton update="myform:display" />
    <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

Different NamingContainers

If your page has different naming containers (e.g. two forms), you also need to add the container id to search expression so that PPR can handle requests that are triggered inside a namingcontainer that updates another namingcontainer. Following is the suggested way using separator char as a prefix, note that this uses same search algorithm as standard JSF 2 implementation;

```html
<h:form id="form1">
    <p:commandButton update=":form2:display" />
</h:form>
<h:form id="form2">
    <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

Please read **findComponent** algorithm described in link below used by both JSF core and PrimeFaces to fully understand how component referencing works.


JSF h:form, datatable, composite components are naming containers, in addition tabView, accordionPanel, dataTable, dataGrid, dataList, carousel, galleria, ring, sheet and subTable are PrimeFaces component that implement NamingContainer.
Multiple Components

Multiple Components to update can be specified with providing a list of ids separated by a comma, whitespace or even both.

```html
<h:form>
    <p:commandButton update="display1,display2" />
    <p:commandButton update="display1 display2" />
    <h:outputText id="display1" value="#{bean.value1}"/>
    <h:outputText id="display2" value="#{bean.value2}"/>
</h:form>
```

Keywords

There are a couple of reserved keywords which serve as helpers.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@this</td>
<td>Component that triggers the PPR is updated</td>
</tr>
<tr>
<td>@parent</td>
<td>Parent of the PPR trigger is updated.</td>
</tr>
<tr>
<td>@form</td>
<td>Encapsulating form of the PPR trigger is updated</td>
</tr>
<tr>
<td>@none</td>
<td>PPR does not change the DOM with ajax response.</td>
</tr>
<tr>
<td>@all</td>
<td>Whole document is updated as in non-ajax requests.</td>
</tr>
</tbody>
</table>

Example below updates the whole form.

```html
<h:form>
    <p:commandButton update="@form" />
    <h:outputText value="#{bean.value}"/>
</h:form>
```

Keywords can also be used together with explicit ids, so update="@form, display" is also supported.

4.1.3 Notifying Users

ajaxStatus is the component to notify the users about the status of global ajax requests. See the ajaxStatus section to get more information about the component.
Global vs Non-Global

By default ajax requests are global, meaning if there is an ajaxStatus component present on page, it is triggered.

If you want to do a "silent" request not to trigger ajaxStatus instead, set global to false. An example with commandButton would be:

```xml
<p:commandButton value="Silent" global="false" />
<p:commandButton value="Notify" global="true" />
```

4.1.4 Bits&Pieces

PrimeFaces Ajax Javascript API

See the javascript section to learn more about the PrimeFaces Javascript Ajax API.
4.2 Partial Processing

In Partial Page Rendering, only specified components are rendered, similarly in Partial Processing only defined components are processed. Processing means executing Apply Request Values, Process Validations, Update Model and Invoke Application JSF lifecycle phases only on defined components.

This feature is a simple but powerful enough to do group validations, avoiding validating unwanted components, eliminating need of using immediate and many more use cases. Various components such as commandButton, commandLink are equipped with process attribute, in examples we’ll be using commandButton.

4.2.1 Partial Validation

A common use case of partial process is doing partial validations, suppose you have a simple contact form with two dropdown components for selecting city and suburb, also there’s an inputText which is required. When city is selected, related suburbs of the selected city is populated in suburb dropdown.

```html
<h:form>
  <h:selectOneMenu id="cities" value="#{bean.city}"
    <f:selectItems value="#{bean.cityChoices}"/>
    <p:ajax listener="#{bean.populateSuburbs}" update="suburbs"
      process="@all"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="suburbs" value="#{bean.suburb}"
    <f:selectItems value="#{bean.suburbChoices}"/>
  </h:selectOneMenu>
  <h:inputText value="#{bean.email}" required="true"/>
</h:form>
```

When the city dropdown is changed an ajax request is sent to execute populateSuburbs method which populates suburbChoices and finally update the suburbs dropdown. Problem is populateSuburbs method will not be executed as lifecycle will stop after process validations phase to jump render response as email input is not provided. Reason is p:ajax has @all as the value stating to process every component on page but there is no need to process the inputText.

The solution is to define what to process in p:ajax. As we’re just making a city change request, only processing that should happen is cities dropdown.
<h:form>
<h:selectOneMenu id="cities" value="#{bean.city}" class="fixed"/>
    <f:selectItems value="#{bean.cityChoices}" />
    <p:ajax actionListener="#{bean.populateSuburbs}" event="change" update="suburbs" process="@this" />
</h:selectOneMenu>

<h:selectOneMenu id="suburbs" value="#{bean.suburb}" class="fixed"/>
    <f:selectItems value="#{bean.suburbChoices}" />
</h:selectOneMenu>
<h:inputText value="#{bean.email}" required="true"/>
</h:form>

That is it, now populateSuburbs method will be called and suburbs list will be populated. Note that
default value for process option is @this already for p:ajax as stated in AjaxBehavior
documentation, it is explicitly defined here to give a better understanding of how partial processing
works.

4.2.2 Keywords

Just like updates, partial processing also supports keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@this</td>
<td>Component that triggers the PPR is processed.</td>
</tr>
<tr>
<td>@parent</td>
<td>Parent of the PPR trigger is processed.</td>
</tr>
<tr>
<td>@form</td>
<td>Encapsulating form of the PPR trigger is processed</td>
</tr>
<tr>
<td>@none</td>
<td>No component is processed, useful to revert changes to form.</td>
</tr>
<tr>
<td>@all</td>
<td>Whole component tree is processed just like a regular request.</td>
</tr>
</tbody>
</table>

Important point to note is, when a component is specified to process partially, children of this
component is processed as well. So for example if you specify a panel, all children of that panel
would be processed in addition to the panel itself.

<p:commandButton process="panel" />
<p:panel id="panel">
    //Children
</p:panel>

4.2.3 Using Ids

Partial Process uses the same technique applied in PPR to specify component identifiers to process.
See section 5.1.2 for more information about how to define ids in process specification using
commas and whitespaces.
4.3 PFS (PrimeFaces Selectors)

PFS integrates jQuery Selector API with JSF component referencing model, so for partial update and process, referencing JSF components can be done using jQuery Selector API instead of regular JSF model which is based on UIComponent.findComponent. Here are some examples;

Update all forms

```html
update="@((form)"
```

Update first form

```html
update="@((form:first)"
```

Update all components that has styleClass named mystyle

```html
update="@(.mystyle)"
```

Update and process all inputs

```html
update="@(:input)" process="@(:input)"
```

Update all datatables

```html
update="@(.ui-datatable)"
```

Process input components inside any panel and update all panels

```html
process="@(.ui-panel :input)" update="@(.ui-panel)"
```

Process input components but not select components

```html
process="@(:input:not(select))"
```

Update input components that are disabled

```html
update="@(:input:disabled)"
```

PFS can be used with regular component referencing as well;

```html
update="compId :form:compId @(:input)"
```
PFS provides an alternative, flexible, grouping based approach to reference components to partially process and update. There is less CPU server load compared to regular referencing because JSF component tree is not traversed on server side to find a component and figure out the client id as PFS is implemented on client side by looking at dom tree. Another advantage is avoiding naming container limitations, just remember the times you’ve faced with cannot find component exception since the component you are looking for is in a different naming container like a form or a datatable. PFS can help you out in tricky situations by following jQuery’s “write less do more” style.

For full reference of jQuery selector api, see;

http://api.jquery.com/category/selectors/
4.4 PartialSubmit

Core JSF Ajax implementation and PrimeFaces serializes the whole form to build the post data in ajax requests so the same data is posted just like in a non-ajax request. This has a downside in large views where you only need to process/execute a minor part of the view. Assume you have a form with 100 input fields, there is an input field with ajaxbehavior attached processing only itself (@this) and then updates another field onblur. Although only a particular input field is processed, whole form data will be posted with the unnecessary information that would be ignored during server side processing but consume resources.

PrimeFaces provides partialSubmit feature to reduce the network traffic and computing on client side. When partialSubmit is enabled, only data of components that will be partially processed on the server side are serialized. By default partialSubmit is disabled and you can enable it globally using a context parameter.

```xml
<context-param>
  <param-name>primefaces.SUBMIT</param-name>
  <param-value>partial</param-value>
</context-param>
```

Components like buttons and behaviors like p:ajax are equipped with partialSubmit option so you can override the global setting per component.

```xml
<p:commandButton value="Submit" partialSubmit="true|false" />
```