Coach View Life Cycle

Coach views are reusable sets of user interfaces that users use to interact with a business object or service. Coach views can consist of one or more other **coach views, data bindings, layout instructions, and behaviors.**

* Views have predefined event handlers that perform callback functions when an event is detected. You can add your own code to these event handlers in the Behavior properties of the view.

* When a coach runs, the views that it contains have a lifecycle that they follow independently. The lifecycle consists of a number of stages with their associated event handlers.

* Composite views are views that contain other views. When a composite view enters a stage, its sub-views enter the same stage. Within a stage, however, the event handlers for the composite view and the event handlers for its sub-views are called in a specific order.

* When a Coach is run, the **load()** function is called first to perform the initialization process. After the initialization is complete, the **view()** function is called, which performs any required logic before the view is rendered. The **unload()** function is called at the end to clean up resources before the view is completed.

Life cycle of event handlers

1	load()	
2	view()	
3	change()	collaboration()
4	unload()	

1 When the coach is run, the **HTML is generated first** and then **the load() function is called** to perform initialization logic, such as defining variables. **The load() function is only called once.**

* For composite views, during the **load()** call, each **subview's load() is invoked before the composite view's load()**. **Similarly for view() calls.** For **unload()**, the order is reversed; **the parent view's unload() method gets called before the child view's unload() method**.

2 After initialization is completed, the **view() function is called before the user sees the view.** The **view()** function typically performs logic to determine what a user sees when the view is rendered. For example, you can show or hide labels depending on the visibility setting.

3 The **change() function reacts to changes in binding or configuration data**. If the **change()** function is not implemented, the **view()** function is called.

* The **collaboration(event) function** is called if it is implemented, when multiple people work on the same view at the same time. If **collaboration(event)** function is not implemented, the default **collaboration coach view logic which is to outline the coach view DOM node takes effect.**

• **validation()** The validate(event) function is called when a boundary event occurs and its **Fire Validation property is set to Before**. The event is an error event or clear event.
The validate(event) function uses the error event to decorate the Coach View to show that it has non-valid data. It uses the clear event to remove the decoration.

4 The **unload() function is called when the coach view is removed from the coach.** For example, when deleting a coach view inside a table row, or the user clicks away from the page. The **unload()** function performs any required cleanup, and recursively invokes the unload() function on all subviews.
*** The parent view's unload() method gets called before the child view's unload() method. The unload() function is only called once.**

Types Of Event Handlers and Usage:

1. Load event Handler
* The load function performs initialization logic before the view loads.

Usage:
The load function is the initial logic that runs as soon as the view is launched. The load method of the view is only called once during the lifecycle of the view. Use load to perform initialization such as defining variables.

Parameters
The load event handler does not take any parameters.

2. View event Handler
* This view function performs logic such as populating the view with data before the user can see the view.

Usage
Use the view function to perform logic before the view is rendered. For example, you can show or hide labels depending on the visibility setting.

Parameters
* The view function does not take any parameters.
* By default, if visibility is not set for the view, it will inherit its parent's visibility. If you want to have your own view logic in addition to the default logic of the view() method, you can invoke the super-class view logic inside your view() function by using the

Syntax: this.constructor.prototype.view.call(this);

3. Change event Handler

The change function is called when there is a change in binding or configuration data.

Usage
* Use this function to react to changes to binding or configuration data.

* If this function is not implemented, the view() function is called instead.
The event object
The function takes a single event object that indicates what field the user changed and its new value.

Properties of the event object

Property	Type	Description
type	String	Valid values are "binding","config", undefined, or null. An undefined or null value means"binding".
property	String	The property that was changed
oldVal		For simple types, the previous value
newVal		new value
insertedInto	Integer	For arrays, the index where the object was added to the list
removedFrom	Integer	For arrays, the index where the object was removed from the list

4. Collabaration Event

The collaboration function overrides the default UI feedback behavior when multiple people work on the same view at the same time. In addition, any custom collaboration event fired during a collaboration session is delivered to the collaboration function, as the default behavior does not handle any custom collaboration events.

Usage
In most cases you do not need to use the collaboration event handler, as the default behavior is sufficient. If this event handler is not defined, by default, the DOM element is highlighted. Use this function only when you need additional code to propagate changes made by a user to another user's browser or if you want to code your own feedback behavior for the user interface.
RestrictionThe collaboration function is not available for Coaches that are running outside of Process Portal.

* To include the default collaboration behavior in your own collaboration(event) logic, you can invoke the super-class collaboration logic inside your collaboration(event) function by using the **syntax: this.constructor.prototype.collaboration.apply(this, [event]);**
To fire a custom collaboration event, use **this.context.triggerCollaboration().**

5. Validate event handler

The validate function handles validation errors that are caused by the data in the Coach View.

Usage
The validate function contains the logic to display indicators or some other notifier that there is a problem with the data. The framework starts with the validate function in **parent Coach Views before it calls the validate function in their children Coach Views.**

* The validate event handler is for when you want to provide custom error visualizations and behavior.
For example, the Text stock control changes color and displays an error icon when it contains non-valid data. The error icon has hover help that displays the message that is associated with the error condition.
* The validate function takes a single event object. The type of the **event object is error or clear.**

* An **error event** means that a validation service or script has detected one or more errors and that the Coach View must handle the errors to display an appropriate visualization. A **clear event** means that the Coach View must remove the visualization that resulted from an earlier error event.
* An **error event** can have two lists of error objects. One list is named **errors** and the other is named **errors_subscription**.

* Coach Views that are bound to a business object automatically receive **errors** for themselves. They also automatically receive errors for all descendant Coach Views that are bound to the same business object.

Properties of the error objects in the errors list

Property	Type	Description
binding	String	Contains the path to the data binding relative to the data binding of the current Coach View.
message	String	Contains the localized message that describes the error.
view_dom_ids[]	String[List]	Contains the list of DOM IDs of the Coach Views that receive the same error message. The list can include the current Coach View and any of its descendant Coach Views.

* Coach Views that call **subscribeValidation()** receive certain errors in the **errors_subscription** list. These errors occur in descendant Coach Views that are bound to a different business object than the current Coach View.
* Coach Views that do not have a data binding, such as the Tab stock control, must call subscribeValidation() to receive errors.

Properties of the error objects in the errors_subscription list

Property	Type	Description
messages	String[List]	Contains a list of localized error messages
view_dom_id	String	Contains the DOM ID of the Coach View with the non-valid data.

6. Unload Event Handler

* The **unload function performs cleanup after the view has completed.**
* The **unload** function is only called once during the lifecycle of the view.

Usage

Use the **unload function to clean up resources before the view is removed.**

* The binding handle is an example of such a resource. The binding handle is returned when **bindAll() or bind()** is invoked. You can release the binding in the unload event handler by call in **handle.unbind()**.

For example, you have MyTableView in which users can select and deselect rows. You register listeners in load event handler of MyTableView using the following code:

this.connectHandles = [];

this.connectHandles.push(dojo.connect(..., "onSelected",...));
this.connectHandles.push(dojo.connect(..., "onDeselected",...));
In the unload event handler for MyTableView you , unregister the listeners:

```
Array.forEach(this.connectHandles, function(handle) {
  dojo.disconnect(handle);
});
```

Parameters
The **unload** function does not take any parameters.

For more Information
https://www.ibm.com/support/knowledgecenter/SSFTDH_8.5.0/com.ibm.wbpm.wle.editor.doc/develop/topics/reventhandler.html