**Process Definition**

**Hierarchy of Process Definition**

Process definition should have the following hierarchy.

1. Process definitions have the following attributes like **name**, **description** and **initialize script**.
2. Process definitions have n number steps.
3. Step have the following attributes like **Group Name, Short Description , Long Description** and **Form Html**. **Group Name** will be shown the left hand side and **Short Description** on the right hand side of the process UI. **Form Name** in the step should refer a page UI component, and it will load when the step is opened and the **Long Description** will be shown the top of the screen.
4. Each step has n numbers of different or same message types.
5. Each message type have the following attributes like **Type** - Message Type, **Type Name** – Group the all message type under same type name e.g. Diagnostic Tests, **Form Html** – html form which will be entry point of the step, **Title** – display title, **Category** – Service, Drug etc. If we defined a same message type more than once under the same step it should have different message form.
6. There is an **Order** option to all steps and message types based on that the UI should render.

**Process API**

How to create a Process Instance

**FactoryClass API** provides a method called **createProcessDefinitionInstance** it will return the instance of process definition based on the JSON string of process definition as a parameter to the method and also available methods like **createStep**, **createMessageType**, **createMessage** etc.

**ProcessDefinition, Step, MessageType, Message API** provides to access the each steps, message types and messages.

Eg. var steps[]= processDefinition. getSteps();

var step = processDefinition.getStep(“StepName”);

var messageTypes[]= step.getMessageTypes();

var messageType = step. getMessageTypeByType (“PRPA\_MT201000HT03”);

var messageType = step.getMessageTypeByTypeName(“Diagnostic Tests”);

var messages[] = messageType.getMessages();

var message=messageType.getMessage(“100”);

**Page UI Component and Events**

**Form Name** in the step should refer a page UI component with suffix “Page”

For e.g. Form name is defined as ‘**PatientRegistration’** under the stepin process definition , then UI component name should be ‘**PatientRegistrationPage.js’**

This page will act as subclass of the **Page.**

**Page** is the base class which has the following events.

**init**

This event initializes the page, in this method we can initialize variables and execute any initialize script etc.

**pageBeforeLoad**

This event will trigger before the page load.

**pageAfterLoad**

This event will trigger after the page load.

**pageResized**

This event will trigger when the page is resized.

**addCompleteHandler**

This event will trigger when UI form renders. E.g. add a new form or load an existing form.

**removeCompleteHandler**

This event will trigger when remove a form from the UI.

**finishCompleteHandler**

This event will trigger once finished or saved the process data.

The above mentioned all events should be trigger to the sub class page like ‘**PatientRegistrationPage.js’.**

The page object can access through the following method.

**var** page = appController.getComponent("RenderingEngine")

.getChildComponent("Form").getPage();

**Form** (The naming convention is wrong, we should change to ‘PageEntry’ )

This is entry point of a page and it will decide the step page and trigger the page init event.

**UIGenerator**

This class will support to generate the UI based on the design, which means it’s tightly coupled with UI.

But the steps which followed in this class can be refered to generate new UI.

Step1. **Provide the placeholder.**

Place holder is nothing but where to render the component

Step2. **Create or retrieve the html data and place to placeholder.**

Html data is the form which will finally render to the UI

Step3. **Create ‘Message’ object.**

Message object can create using FactoryClass API.

Step4. **Create Message document object.**

Message document object can create using datalayer API.

Step5. **Create ‘MessageScript’ object which used to update/modify xml doc.**

‘MessageScript’ object can create using message document object.

‘MessageScript’ API provides initialize, fillData , fillParticipants , complete methods etc which will update the message document.

Step6. **Create ‘MessageAndBinderUI’ object.**

‘MessageAndBinderUI’ object can create using message document object.

‘MessageAndBinderUI’ API provides loadDataOntoForm method which fill the message in to related components.

Step7. **Create ‘FormScript’ object.**

‘FormScript’ object can create using message object.

‘FormScript’ which provides the following API like initialize, onLoad and unLoad.

Refer the API docs for further implementations.