This document outlines how the requirements of Deal Registration Form are met by the framework.

1. **Show-hide of fields on the basis of selection of radio list item (US Federal Government):** This can be modeled using the RadioListFormField.dependencies. For each FormFieldVisualDependency instance in RadioListFormField.dependencies, where the FormFieldVisualDependency.formFieldValue points to the “US Federal Government” item, the FormFieldVisualDependency.dependentObjectId will point to a FormField or a FormSection. The FormFieldVisualDependency.isShow will decide whether to set visibility = true or false.
2. **Controls being populated based on the value of Location dropdown:** The following shows how a typical AJAX call with data bind is set up:

Assume that the formContextData contains a key called locations which points to a List<LocationInfo>. The LocationInfo contains properties like legalName, dbaName, contactName, locationId, ceId etc. Here is how it will be set up.

* 1. The Location Selector dropdown will have a postback registered on the onchange event i.e. on the PostbackEvents.ONCHANGE
  2. When the user selects some value in the drop-down, the onchange event fires and an AJAX request is made to the <http\_root>/dynamicForms/maintainForm
  3. The ELScript registered for the dropdown in FormField.maintenanceScripts[PostbackEvents.ONCHANGE] will be run. This script can be the following:

**//Get the form field id which caused postback**

**var locationDropdownId = dynamicFormsContext.postbackSrcId;**

**//Get the Form Field by id**

**var locationDropDown = dynamicFormsContext.form.getFormFieldById(locationDropdownId);**

**//Get its value i.e. the selection in dropdown**

**var locationDropDownVal = locationDropDown.value;**

**//Load some service from the Spring context**

**var someService = dynamicFormsContext.getSpringContext().getBean(“some\_service\_bean”);**

**//Call some hypothetical getLocation method using the location drop-down value**

**var locationObject = someService.getLocation(locationDropDownVal);**

**//Return the locationObject. If script.putInContext is true, it will be put in the //dynamicFormsContext.formContextData**

**return locationObject;**

* 1. Assume that the above script result is placed in context with key ‘aLocation’. Now, if the FormField.binding values for the Legal Name field is dynamicFormsContext.formContextData[‘aLocation’].legalName, then the value of the locationObject.getLegalName will be bound to the ‘Legal Name’ textbox.

**Note** that the above script could be written as a one-liner. But it would be bulky and error prone. This is why JEXL is preferred over Spring EL (because it supports scripts).

1. **Search Location popup:** The Location popup is a FormSection with isPopup as true. The popup can be opened (made visible) using a FormFieldVisualDependency on a HyperlinkFormField or a ButtonFormField. The dependentObjectId would be the id of the popup.

The AJAX call on the Search button can be modeled in the same way as shown in section 2. The difference being that the postback will be registered on the onclick event (via PostbackEvent.ONCLICK) of the Search button.

1. **Opportunity and Lead Brand Opportunity relationship:** This is modeled simply by the FormFieldValueDependency class, which models the dependencies between a source form field value with one or more form field values on a target FormField.

The DynamicFormsTransformService will be responsible for converting this dependency into appropriate JavaScript code which will filter the values in the target dropdown/radio-list/checkbox-list based on the selection in the source dropdown/radio-list/checkbox-list.

1. **Auto-complete dropdown:** This is done on the client side using the [FilteringSelect Dijit control](http://livedocs.dojotoolkit.org/dijit/form/FilteringSelect).
2. **Automatic help text on typing:** This can again be done by an AJAX call. Here the postback will be registered on the onkeypress event (via PostbackEvent.ONKEYPRESS) of the text area. The hover help can be a FormSection which is initially hidden. The EL script for this event will do the following:
   1. Call some service to get the help based on what is currently in the text area.
   2. Bind the help text to the form section which will act as hover help.
   3. Make the form section visible.
3. **Address Grid:** The address grid will be modeled using the GridFormField. The GridFormField uses the [Dijit DataGrid](http://dojotoolkit.org/reference-guide/1.7/dojox/grid/DataGrid.html) to display itself. The GridFormField.value will be an array of custom objects. It is assumed that there will be a custom class (let’s say Address) which will contain data for one row.

The GridFormField.columns is a map where the key is the “name of the column” to be used on the grid and the value is the name of the property of the custom class to which it will be bound.

1. **Attachment Form:** The attachment form will be modeled using the UploadFormField. The UploadFormField uses a [dojox.form.FileUploader](http://dojotoolkit.org/reference-guide/1.7/dijit/form/Button.html) to display itself.
2. **Submit and Save Draft Button –** It will just be ButtonFormField with isSubmit = true. It uses [Dijit Button](dijit.form.button) to display itself. The Dijit form can be setup to call form.validate() method to validate all the fields in the form before submitting. This way it will prevent form submission if all fields are not validated. This code will be written by default by the DynamicFormsTransformService. The submit will post to /dynamicForms/submitForm and the save as draft will post to /dynamicForms/saveFormDraft
3. **Parent and Child Fields:** This is modeled using the FormField.child. In this case, there will be JavaScript written which will automatically populate the child field based on value of parent field (on the blur event of the parent field). The checkbox to enable/disable this functionality can set the FormField.isEnabled property to true/false.