Advanced Forms API Plugin

Appian

Overview

- The main components described in this package include:
  - Call Appian expressions (executed server side) via **AJAX** calls in JavaScript.
  - Dynamically **change dropdown values** based upon changes in the form at runtime. This feature allows you to call any expression (OOB, Expression Rules, Query Rules and Custom Java Expressions) to perform this action.
  - Display **documents inline** (such as PDF files) from within the form.
  - Attach **autocomplete** behavior to text inputs, with the source data coming from other fields or server side expression functions, expression rules and query rules.
  - Include **reusable form sections** from other forms

How to Deploy

- This plugin is a completely self contained OSGi plugin
- You do not need to reference any custom JavaScript files in your form through the limited one JS file input.
- The necessary JavaScript is contained in the plugin itself.
- To use these additional functions, simply apply a new onload expression to your form and put this at the top:

```
importScript("/plugins/servlet/FormsExt.js");
```

- This allows you to leverage your own JS library on the Form without having to merge with the FormsExt.js base.
- Alternatively, include the importScript line above at the top of the JS file that you included in your forms.
- If you want to see the contents of FormsExt.js you can view it inside the OSGi plugin.

Appian

AJAX Calls from Forms

- FormAPI.evaluateServerSideExpression
  - Using this feature, any server side expression can be called from a form, albeit asynchronously.
  - This allows you to reference Query Rules, Expression Rules and Constants directly in forms and use these for client side field validation.
- Usage:

```
FormAPI.evaluateServerSideExpression([callback_function], [serverside_expression]);
```

- The server side expression can be a string or a JavaScript array with the function name as the first item in the array and the parameters as the remaining items.
- Example (string expression):

```
FormAPI.evaluateServerSideExpression(function(fn){
    alert("the users first name is "+fn);
},"=user(\"john.doe\",\"firstName\")");
```

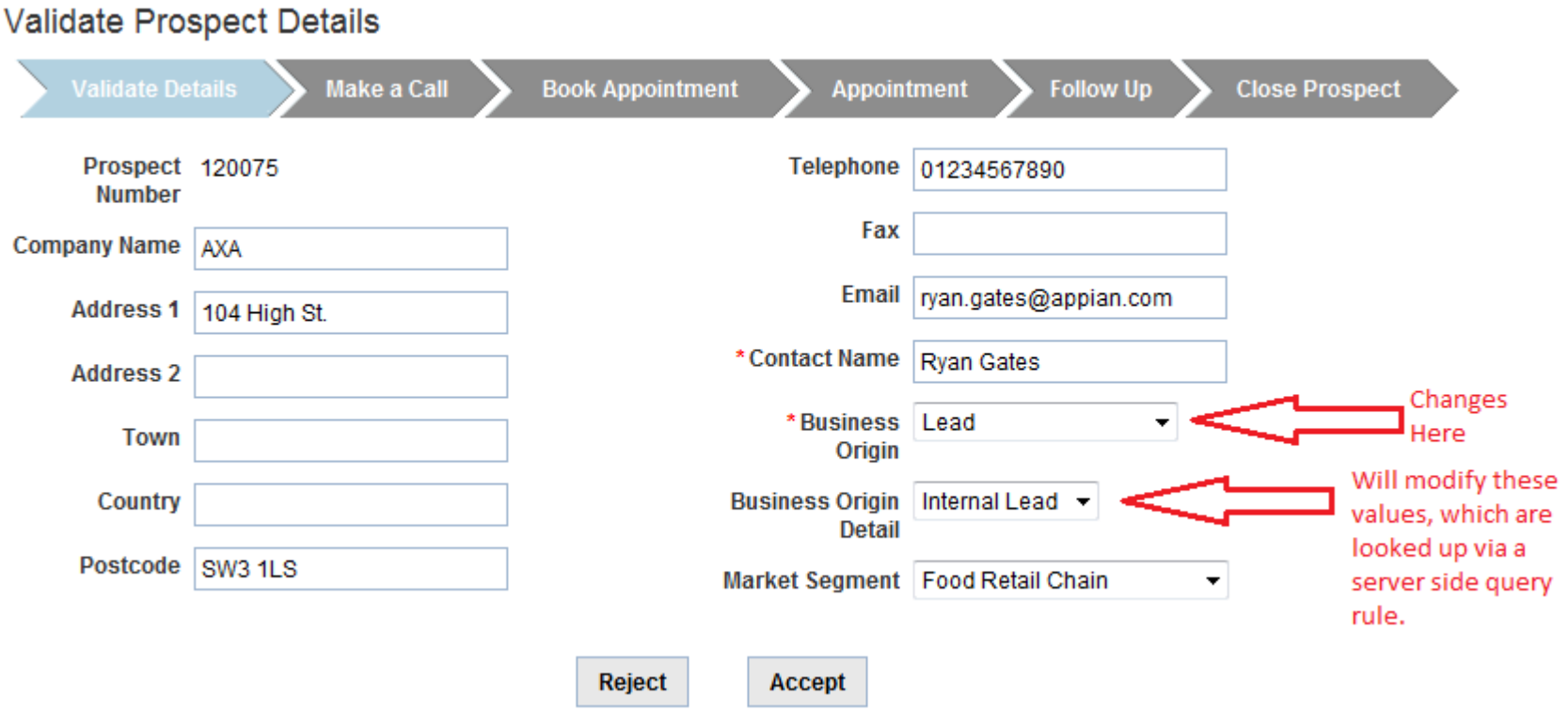
- Example (separated function name and parameters list) NEW:

```
FormAPI.evaluateServerSideExpression(function(fn){
    alert("the users first name is "+fn);
},["user", "john.doe", "firstName"])
```


Appian

Populate Dropdowns Dynamically

- This extension allows you to change a dropdown value on changes from another field.



Populate Dropdowns Dynamically

- FormAPI.populateDropdown
  - This requires a function return that has a list of variables that will populate a dropdown (like a query rule)
- Usage:

```
FormAPI.populateDropdown([dropdown_id], [expression],
  [id_attr], [display_attr]);
```

- Example:

```
FormAPI.populateDropdown("dropdown6",
  ["GetBusinessOriginDetailsByOriginCode",
  FormAPI.getValue("dropdown5").id], "code", "description");
```

- In this example, we are using the value from dropdown5 to populate the values in dropdown6. The query rule GetBusinessOriginDetailsBy OriginCode will return a CDT of the following structure:

```
[{code:1,description:"External"},{code:2,description:"Internal"}]
```

- The CDT attributes can be specified in the last two parameters of the function.
- CDTs are no longer required, you can return simple types, but the id and value in the dropdown will be the same. When doing this do not pass in a id_attr or display_attr.

Appian

Populate Dropdowns Dynamically

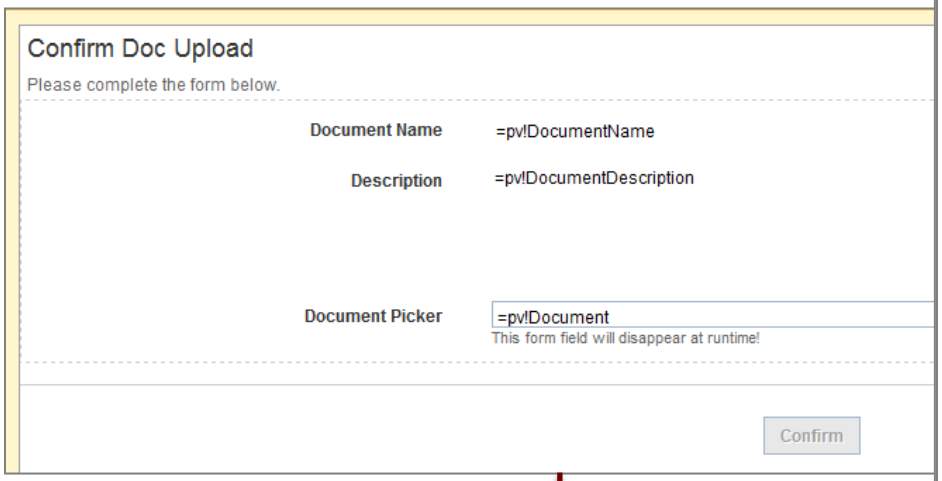
- FormAPI.updateDropdownValues
  - This is a convenience method to change the values of a dropdown, and is used by the populateDropdown function described early.
  - Use this function if you already have the values and do not need to execute something on the server.
- Usage:

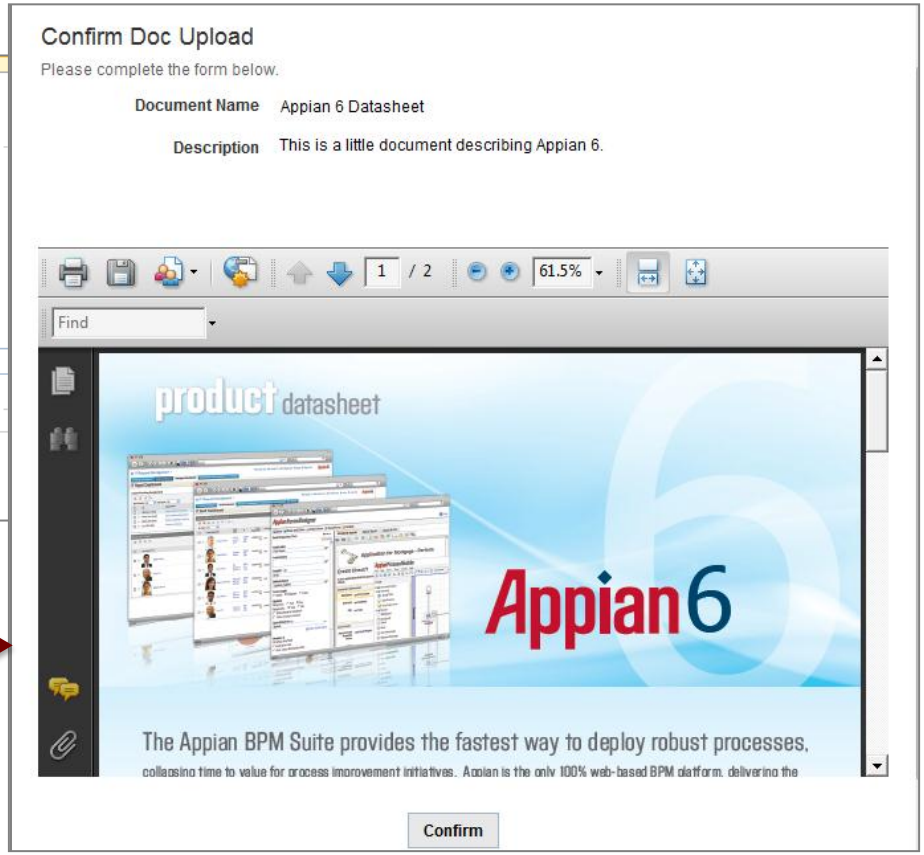
  FormAPI. updateDropdownValues([field_id], [ids],[display_values]);
- Example:

  FormAPI. updateDropdownValues("dropdown6",[1,2],["Internal", "External"]);

Display Document Inline

- This extension allows you to convert a Document Picker input at design time into a inline Document object at runtime.





Display Document Inline

- FormAPI.displayDocumentInline
- Usage:

  FormAPI.displayDocumentInline([doc_picker_id]);
- Example:

  FormAPI.displayDocumentInline("pdf")
- In this example, we are converting the form input with id 'pdf' into an inline document field.
- Considerations:
  - Put this on the load function of the form
  - Make sure the form input has a default value – this is used to show the document inline.
  - You must have the appropriate plugin to display the file (i.e. Acrobat for PDF, QuickTime for MOV etc.)

Autocomplete

- FormAPI.attachAutocompleteFromExpression
  - Using this function, you can attach an autocomplete control to a text field. This can populate the values from the server.
- Usage:

  FormAPI.attachAutocompleteFromExpression(fieldid, expr, id_attr, display_attr, multiple);
- Parameters:
  - fieldId: The field id of the form text input to be modified
  - expr: can be a string or a JavaScript array with the function name as the first item in the array and the parameters as the remaining items. Can return a simple or complex type.
  - id_attr: if the expression returns a CDT, this is the id property of the CDT (optional)
  - display_attr: If you are using a CDT, then you need to specify what property of the CDT has the display value. Not used if the expression returns a list of primitive values (string, int, etc).
  - multiple: allow you to pick more than one item.
- The autocomplete field will allow you to look up items from the back end, and these can be simple strings (i.e. constants) or CDTs with properties.
- If you have another field (i.e. text or hidden) that follows the naming convention fieldid_id, this will be populated with the id of the item selected.
  - Example, if you have a customer array and each customer object has an id and name field. If the primary field id is "customer" you could create a hidden field called "customer_id" which would store the id of the selected customer. The customer field will store the customer's name.

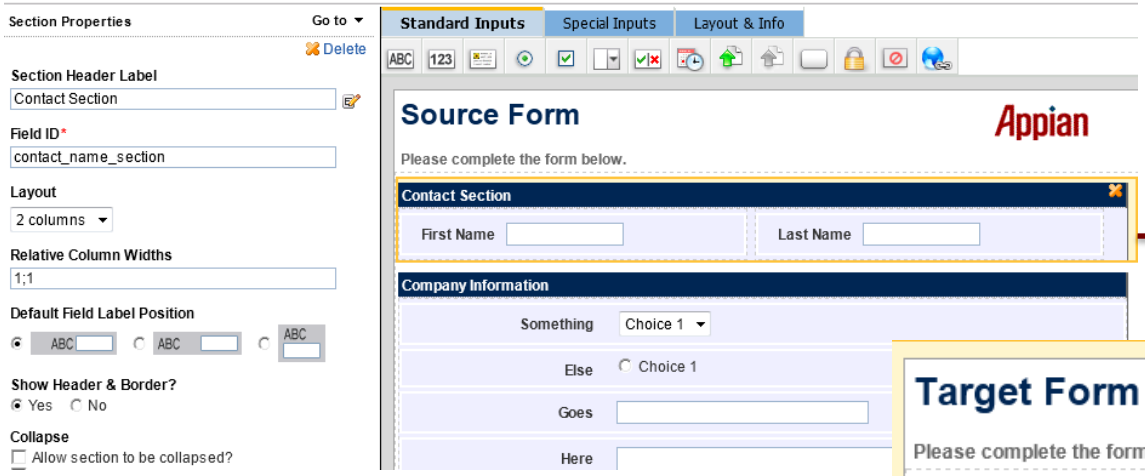
Autocomplete

- FormAPI.attachAutocompleteFromField
  - Using this function, you can attach an autocomplete control to a text field. This can populate the values from another form input. The source field must have a semicolon separated list of items.
- Usage:

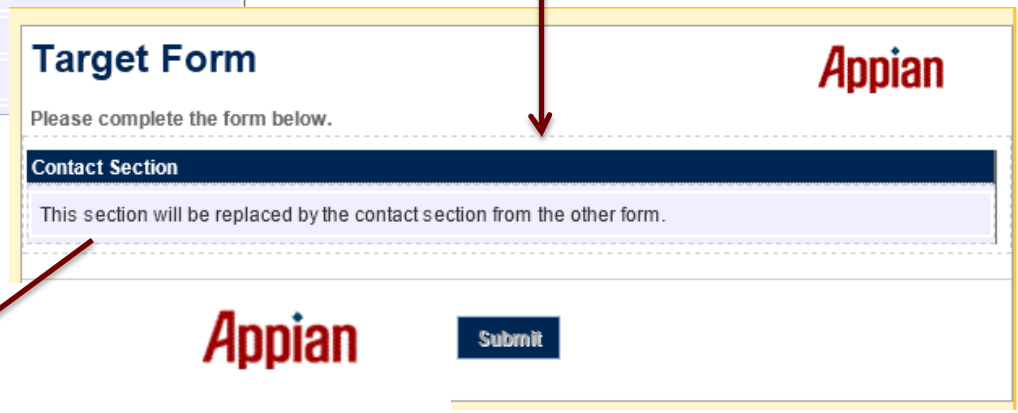
  FormAPI.attachAutocompleteFromField(fieldid, optionsListFieldId, multiple);
- Parameters:
  - fieldId: The field id of the form text input to be modified
  - Optionslistfieldid: the source field (likely a hidden item).
  - multiple: allow you to pick more than one item.

Reusable Form Sections

This extension allows you to reuse form sections from other forms



The section above in the source form will replace this section in the target form



At runtime you can see the second form has the same layout and form inputs as specified in the first form

Appian

Reusable Form Sections

- FormAPI.loadSection
  - Using this function, you reuse a section from another form within the current form. A section in the source form is used to replace a section in the current form. Any items in the section to be replaced will removed.
- Usage:

  FormAPI.loadSection (exportedFormId, sectionFrom, sectionToReplace);
- Parameters:
  - exportedFormId: Must resolve to an Appian Form File (.aff) file in the document management system. This file is exported from another form. Can be a hardcoded id, such as 216, or an expression, such as "=cons!MyFormFileConstant". The later is preferred for obvious reasons.
  - sectionFrom: The field id of the section in the source AFF file.
  - sectionTo: The field id of the section in the current form that will be completely replaced.
- The form inputs included in this section should be mapped to ACPs in the current form of the same name and type. Therefore you must create ACPs locally in the form that will then be mapped to the imported form section.

Reusable Form Sections Example

- If the source form has a section with a first name and last name input in a section called **contact_name_section**. The first name field is called **first_name** and maps to an ACP called **firstName**. The last name field is called **last_name** and maps to an ACP called **lastName**
- If the new form that wishes to reuse this section, you should create an empty section called "contact_name_section" (this name doesn't have to be the same name, but might be convenient).
- In the new form or task you need to create two ACPs called **firstName** and **lastName**, and delete the automatically created form inputs.
- Onload of the new form will look something like this:

```
importScript("FormsExt.js");
FormAPI.loadSection("cons!formsectiona","contact_name_section","contact_name_section");
```

- When the form is run, the loadSection Javascript will create the section based upon what is stored in the AFF file and map the form inputs to the ACPs specified in the new task.

Appian

Set Fields as Readonly

- FormAPI.setReadonly
  - Using this function, you can make a field readonly or make a readonly field editable.
- Usage:

  FormAPI.setReadonly(fieldId, readonly);
- Parameters:
  - fieldId: The field id of the form input
  - readonly: boolean

Set Fields as Required

- FormAPI.setRequired
  - Using this function, you can make a field required or make a required field not required.
- Usage:

  FormAPI.setRequired(fieldId, required);
- Parameters:
  - fieldId: The field id of the form input
  - required: boolean

Set Fields as Disabled

- FormAPI.setDisabled
  - Using this function, you can make a field disabled or make a disabled field editable.
- Usage:

  FormAPI.setDisabled(fieldId, disabled);
- Parameters:
  - fieldId: The field id of the form input
  - disabled: boolean