Forms jQuery Plugin

Using the jQuery plugin requires a compatible version of jQuery (one is included), the jquery.onlineformhelpers.js, and the loading code in plugins.js.

- Allow User to Add/Remove Rows in a Form
- Auto-populate Current User Directory Data
- Auto-populate Today's Date
- Automatically Resize Fields to Fit the Full Available Width
- Conditionally Display Content Based on Form Selections
- Disable Content and Fields Based on Form Selections
- Look Up NetID* by Name
- Make a URL Field Clickable in Future Sections
- Perform Numeric Calculations on Form Fields
- Pre-populate Fields with Values from Other Form Fields
- Replace Default File Upload Field with a Field that Matches Bootstrap Fields
- Require Fields Prior to Submitting a Form
- Require Fields Only When They Are Visible and Active
- Set Field Values on Form Submission

Allow User to Add/Remove Rows in a Form

Some forms may have parts where the amount of data entered is variable based on user. For example, a form allowing students to report AP subjects and scores would need to expand to accommodate additional tests. To enable this in your form, follow these steps:

1. Create a container with the addremoverows-container class. You can have multiple containers but they must not be nested
2. The node that should be replicated must have the addremoverows-template class and must be in the container. There may be only one of these in the container
3. The container must have a field to store the data with the addremoverows-data class (preferably hidden). There may be only one of these in the container
4. You may include an add button with the class addremoverows-add. If this button is not present the initial row will not be generated. Omitting the add button is only useful for a future section to pull data from a previous section (covered below)
5. The template may have a delete button with the class addremoverows-remove. Omitting the delete button prevents rows from being deleted
6. Optionally you can have an error message outside the container with the class addremoverows-noscript-CONTAINER_ID that will display if JavaScript is off. If JavaScript is off the entire container will be hidden

Sample Code

```
Adding the type="button" attribute to a button prevents it from causing the form to submit when clicked
```

```
Adding the tabindex=-1 attribute to a button prevents it from being highlighted as the user tabs through the form
```
Pulling Data from Another Expandable Section

You may want to have two sections in your form that draw from the same data. For example, in a student section a student may petition to apply several study abroad credits towards distribution requirements. In the approval section the adviser may mark yes/no on each of these credits. In this case the "credits" table is being replicated in two places so the data populating the adviser's view should be driven from the student's submission. Note: the individual fields that form the rows can vary, only fields with identical names will be populated.

To pull data from another section add the `addremoverows-datasource-XXXXXX` class to the hidden data field in the second section where `XXXXXX` is the id of the first section's data field.

Sample Code for First Section, the Source Data

Note: the `id="Manage_Colors_Data"` attribute was added to the hidden data field.

```html
<div id="Manage_Colors_Container" class="addremoverows-container">
    <div class="addremoverows-template">
        <input type="text" name="Color">
        <button class="addremoverows-remove type="button" tabindex="-1">Remove</button>
    </div>
    <button class="addremoverows-add type="button" tabindex="-1">Add</button>
    <input type="hidden" id="Manage_Colors_Data" name="Color_Data" class="addremoverows-data">
</div>
<div class="addremoverows-noscript-Manage_Colors_Container">
    <p>You must have JavaScript enabled in your browser to manage colors.</p>
</div>
```

Sample Code for Second Section Pulling Data from First

Note: this form omits the add and remove buttons and include a new checkbox for each color.

```html
<div id="Manage_Colors_Container" class="addremoverows-container">
    <div class="addremoverows-template">
        <input type="text" name="Color">
        <button class="addremoverows-remove type="button" tabindex="-1">Remove</button>
    </div>
    <button class="addremoverows-add type="button" tabindex="-1">Add</button>
    <input type="hidden" id="Manage_Colors_Data" name="Color_Data" class="addremoverows-data">
</div>
<div class="addremoverows-noscript-Manage_Colors_Container">
    <p>You must have JavaScript enabled in your browser to manage colors.</p>
</div>
```
Event Hooks for Adding/Removing Rows

You may want to execute your own custom code upon various events of the add/remove rows helped. The following hooks are supported:

- **runBeforeInit**: runs before the module initializes itself. This will be the first event triggered on page load.
- **runAfterInit**: runs after initialization finishes. Add and populate events may trigger before this is called.
- **runBeforeAdd**: runs when the user clicks the add button, before the new row is created.
- **runAfterAdd**: runs when the user clicks the add button, after the new row is created.
- **runBeforePopulate**: runs before fields are populated with data when the form gets rendered, if saved data exists.
- **runAfterPopulate**: runs after fields are populated with data when the form gets rendered, if saved data exists.
- **runBeforeRemove**: runs when the user clicks the remove button, before the row is removed.
- **runAfterRemove**: runs when the user clicks the remove button, after the row is removed.
- **runBeforeSubmit**: runs when the user submits the form, before the data is packed in to the hidden data field.
- **runAfterSubmit**: runs when the user submits the form, after the data is packed in to the hidden data field.

To hook in to an event you should override that function when the helper is applied to the document, before it is initialized. To do this you will need to put a script tag in your head between `jquery.onlineformhelpers.js` and `plugins.js`. In this example we merely print a message to the console before the helper is initialized:

```javascript
$.fn.onlineformhelpers.initPluginsFunctions.addremoverows = function(selector) {
  $(selector).addremoverows({
    'runBeforeInit': function(){
      console.log('Hello, World');
      // More of your code here...
    }
  });
};
```

Auto-populate Current User Directory Data

To auto-populate information from the LDAP directory about the current user, add a class of `form-autofill-currentuser-XXXXXX` to the containing input field where XXXXXX is a field the plugin has access to. This will only occur on enabled (not disabled) form fields, i.e., the currently active form section.

The online forms system presently has access to the following fields:

- `displayname`
- `nuStudentGivenName`
- `nuStudentSn`
- `nuMiddleName`
- `nuStudentCurrentAddress`
- uid (NetID)
- mail (email)
- nuStudentPermanentAddress
- nuStudentCurrentPhone
- nuStudentNumber (emplid)

1 for nuMiddleName, nuMiddleName_initial will only return the first character of that string.

2 for nuStudentCurrentAddress, LDAP returns a field delimited by dollar signs. There three suffixes you may add to this field to make it more useful:

- form-autofill-currentuser-nuStudentCurrentAddress_newlines, which converts dollars to new lines
- form-autofill-currentuser-nuStudentCurrentAddress_tabs, which converts dollars to tabs
- _street, _city, _state, _zip, or _country to get only that part of the string

Auto-populate Today's Date

To auto-populate today's date in a field, add a class of "form-autodate" to the containing input field. This will place an American shorthand date in most fields (M/D/YYYY) but will place a timestamp-style date (YYYY-MM-DD) in inputs with type=date (to conform to the HTML 5 spec for this type, since display is overridden by the browser). This will only occur on enabled (not disabled) form fields, i.e., the currently active form section.

This will not update if the user's local data changes after the field is populated.

Automatically Resize Fields to Fit the Full Available Width

To automatically resize fields to expand to the full available width of the parent node, add a class of "form-autosize" to the containing input field. This plugin will resize the input as the window is resized.

Conditionally Display Content Based on Form Selections

To conditionally display a node of HTML based on the value of another form input, follow these steps:

1. Decide what field will trigger the display. That will be our "target" and it must have a unique id attribute.
2. Decide what node or nodes will display based on what value or values. These will be the "result" nodes.
3. Add a "showwhen-target-XXXXXX" class to the result node/s, where XXXXXX is the id of the target.
4. If a given result node should be displayed for a given value of that target, create a showwhen-value-XXXXXX class for each valid value, where XXXXXX is a value that should cause the result to show. Alternatively, you can use showwhen-optiontext-XXXXXX to trigger based on the option text rather than the value, where XXXXXX is the text of an option node with spaces converted to underscores.

Additional Options:
- You can also use showwhen-valueempty or showwhen-optionemptytextempty if it should display when the value or option text is empty.
- You can use showwhen-never if content should never be displayed.

Conditionally Hide Content Based on Form Selections

The same process can be used to conditionally hide contents using "hidewhen-target-XXXXXX" and "hidewhen-value-XXXXXX" classes.
- Content will be visible when the page loads. To make things hidden by default use the "hidewhen-starthidden" class.
- Use "hidewhen-always" to have content always hidden and "hidewhen-ifjs" to hide content only when JavaScript is enabled.

Disable Content and Fields Based on Form Selections

To conditionally disable form fields or nodes of HTML based on the value of another form input, follow these steps:

1. Decide what field will trigger the disabled/enabled fields. That will be our "target" and it must have a unique id attribute.
2. Decide what node or nodes will display based on what value or values. These will be the "result" nodes.
3. Add a "disablewhen-target-XXXXXX" class to the result node/s, where XXXXXX is the id of the target.
4. If a given result node should be disabled for a given value of that target, create a disablewhen-value-XXXXXX class for each valid value, where XXXXXX is a value that should cause the result to be disabled. Alternatively, you can use disablewhen-optiontext-XXXXXX to trigger based on the option text rather than the value, where XXXXXX is the text of an option node with spaces converted to underscores.

Fields disabled in this manner are given the disabled attribute. If an element that isn't a form field is disabled it will be given a disabled CSS class. The style sheet provided by forms will make the contents light gray. Form fields that are descendants of a disabled HTML node will be given the
disabled attribute.

Additional Options:

- You can also use disablewhen-valueempty or disablewhen-optiontextempty if it should be disabled when the value or option text is empty.
- Use the disablewhen-always class if a field or node should always be disabled.
- You can use the disablewhen-onload class if a field should begin in a disabled state when the page loads.

Look Up NetID* by Name

* And other directory elements

Form sections can be assigned by NetID requiring routers to manually look up NetIDs in the Online Directory. This feature allows routers to type a name and have a lookup performed that displays results in a drop down (similar to Facebook and many search features). To implement this:

1. Create a container with the class typeahead-directory-container
2. Apply the typeahead-result-field-uid class to the field containing the NetID. This field may be hidden. A best practice is to leave this field visible since some NetIDs may be hidden and the search only displays public results
3. Apply the typeahead-search class to the field that the user should type in to for results
4. Optionally apply the typeahead-ifsupported class to any elements that should not appear if the typeahead is not supported in the user’s browser (e.g., JavaScript is disabled). Typically this class is applied to the search field or its parent and any instructions

In step 2 you may use any typeahead-result-field-XXXXXX where XXXXXX is a field returned by an anonymous bind

Note: this plugin relies on Bootstrap.

Sample Code

```html
<div class="row-fluid typeahead-directory-container">
 <div class="span6 typeahead-ifsupported">
  <label>Search by Name</label>
  <input type="text" name="Lookup_Approver_NetID" class="form-autosize typeahead-search" autocomplete="off">
 </div>
 <div class="span2">
  <label>Approver NetID</label>
  <input type="text" name="formcycle-assign-to-netid-Approver_Signature" id="Approver_Route_Netid" class="form-autosize typeahead-result-field-uid">
 </div>
 <div class="span4">
  <label>Approver Email</label>
  <input type="text" name="Approver_Email" class="form-autosize typeahead-result-field-mail">
 </div>
</div>
```

Search for the approver's NetID by entering their name in the "Lookup by Name" field below.

Download Example of Look Up NetID by Name

Make a URL Field Clickable in Future Sections

Some forms have users enter a URL that future sections may want to review as part of the workflow process. Rather than requiring the user to copy and paste the link, you can apply the "clickable" class to the URL field to make it open a new window/tab. This field will only be active when it is in a "readonly" state (i.e., when that field is not the active form section).
Perform Numeric Calculations on Form Fields

To perform numeric calculations based on data entered in form fields and display the result in another form field, follow these steps:

1. Decide what field will be display or store the result. This will be the total field.
2. Decide what fields will be used to calculate the result. These will be our value fields. Each of these fields must have a unique id.
3. Add a "formmath-basevalue-XXXXXX" class to the total field, where XXXXXX is the id of the value field that should be the initial value of the calculation.
4. Add one of the four following classes to the total field, depending on what mathematical operation you wish to perform on that initial value, where XXXXXX is the id of the value field:
   - formmath-add-XXXXXX
   - formmath-subtract-XXXXXX
   - formmath-multiply-XXXXXX
   - formmath-divide-XXXXXX

Calculations are performed in this order:

1. Base value is set.
2. That value is multiplied by all the formmath-multiply-XXXXXX fields.
3. That value is divided by all the formmath-divide-XXXXXX fields.
4. That value is added with all the formmath-add-XXXXXX fields.
5. That value is subtracted by all the formmath-subtract-XXXXXX fields.

In other words, order of operations are followed. If you wanted to make a calculation that did not follow the order of operations (i.e., like the use of parentheses in math) you could use an intermediate hidden field to hold the temporary value then base your final total field on this temporary field.

Additional Options:

- Add a formmath-decimalplaces-NN class to your total field where NN is the number of decimal places you wish to show. By default two decimal places are displayed when fractions are needed, but no decimal places will be shown if the result is a round number.

Pre-populate Fields with Values from Other Form Fields

To pre-populate a field with contents of another field (e.g., to pre-populate a later section with contents from an earlier section) follow these steps:

1. Decide what field to pull the contents from. This field must have a unique id attribute.
2. Decide which fields should be pre-populated with these contents.
3. Add a "populatefrom-id-XXXXXX" class to the fields you want pre-populated, where XXXXXX is the id of the field you are pulling the contents from.

Note: if the field to be populated is disabled or readonly, the pre-populating will not take place. This action only runs on page load, it does not continuously update while the form is being manipulated.

Replace Default File Upload Field with a Field that Matches Bootstrap Fields

Bootstrap does not style input fields of type file, instead relying on the browser's default implementation. This can cause form fields that don't match the style of the rest of the form. To replace the browser default download field with a text box and "attach" button that matches Bootstrap styles, simply apply the "fancyupload" class to the input field. Note that if you want autosizing to occur you should apply the "form-autosize" class as well.
Require Fields Prior to Submitting a Form

By default fields with the required class are only forced to be required on the backend when the form is submitted. To have required values be enforced by the browser in “approve” actions, add the “onsubmit-checkrequired” class to your form section.

```html
<form id="Submitter" class="form-section visiblefromall onsubmit-checkrequired">
 <!-- ... -->
</form>
```

If you use the required attribute instead of the required class the browser will force the field to have a value on all submissions including saving as a draft, rejecting, and returning a section. Using the required class only forces fields to have a value on the approve action. In most instances this makes more sense.

Require Fields Only When They Are Visible and Active

Sometimes you want to require a field but only when it is visible and active. Just using the require class won’t work because that always requires the field to have a value. Here are a few examples where that doesn’t work:

- When you Conditionally Display Content Based on Form Selections and want to require a field only shown under certain circumstances
- When you Disable Content and Fields Based on Form Selections and don’t want to require a field that was disabled
- When you Allow User to Add/Remove Rows in a Form and a new row is added with required fields

Using the "required-whenactive" will only require a field when it is both visible and editable and only on “approve” actions. Note this is only enforce on the front-end so this can be circumvented. For most browsers the built-in required attribute will be used to enforce this. On browsers that don’t support this attribute (Safari and IE8) the plugin will generate error messages in the same style as if they were being enforced on the backend (a red box at the top of the form listing the missing fields).

Sample Code
Set Field Values on Form Submission

In this section "hidden (not visible)" means an input where type="hidden" or any field that is not visible at the time of submit (e.g., display:none).

Set a Hidden (not visible) Field to the Value of Another Field

Apply class "setonsubmit-ifhidden-FIELD_ID" where FIELD_ID is the ID (not the name) of the field you want to take the value from.

Clear the Value of a Hidden (not visible) Field

To clear the value on a field-by-field basis, apply the "setonsubmit-ifhidden_clear" class to each field. To apply it to all hidden fields in a given form use the "setonsubmit-clearhidden" field.

Disable a Hidden (not visible) Field

To disable a hidden field on submission, use the "setonsubmit-ifhidden_disable" class. There isn't much difference between this and clearing the value. Clearing the value will still submit the field as the key with no value. Disabling the field will not transmit it whatsoever.

Clear the Value of Any Field on a "Return" Action

Sometimes you may want to clear the value of a field when the form is being returned (e.g., the date and signature of an approval section). To do this on a field-by-field basis, apply the "setonsubmit-ifreturn_clear" class to each field. To clear all fields in a given section, apply "setonsubmit-ifreturn_clear" to the <form> itself.