Customizing Behavior

You can add custom behaviors by adding JavaScript to your form/flow. For example, you may want to change the look of your form's submit button when the user hovers the mouse over the button. It is possible to associate a custom JavaScript handler to any form control. See Custom JavaScript Examples for sample code.

> Although JavaScript will accomplish the task, it is important to note that your scripts are not subjected to formal testing by the frevvo quality assurance team. Choosing an approach that is part of the Live Forms product like Business Rules is a better choice than adding JavaScript since released versions undergo rigorous quality assurance testing. Customizations using JavaScript should only be used under special situations and if there is no other way to customize the form to the requirements.
>
> For example, let's say you have a message control in your form that contains markup to pull a custom JavaScript. The URL for the script contains the Live Forms home directory and revision number - (http://localhost:8080/frevvo/js-24487/libs/script.js). An upgrade of Live Forms to new release or revision version could potentially stop the form from working.
>
> If you choose to add JavaScript, we strongly recommend that you are familiar with a JavaScript debugger / web development tool such as the Firebug extension for Firefox.

On this page:

Adding JavaScript to your Form

You have the choice of two different approaches:

1. Method 1 - Add JavaScript to the custom.js file then upload it into your application.
2. Method 2 - Add a Message Control that contains your JavaScript.

Method 1

Upload your custom JavaScript to Live Forms via the Scripts tab on the left menu inside your application. Follow these steps:

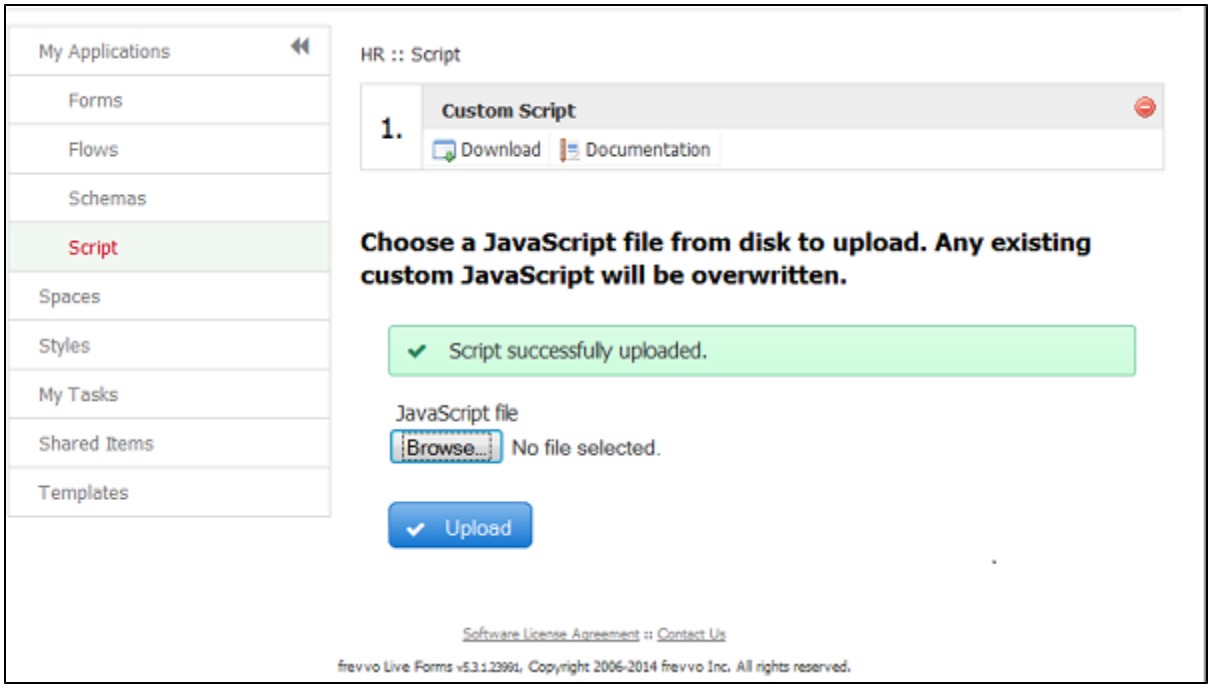
1. Your custom JavaScript must contain the comment // frevvo custom JavaScript as the first line of the file or the upload will not be successful. Here is an example of code that will add dashes to a Social Security number control as the user is typing it in. See Custom JavaScript Examples for information on this sample code. Notice the JavaScript comment is the first line in the script file.

```
// frevvo custom JavaScript
```

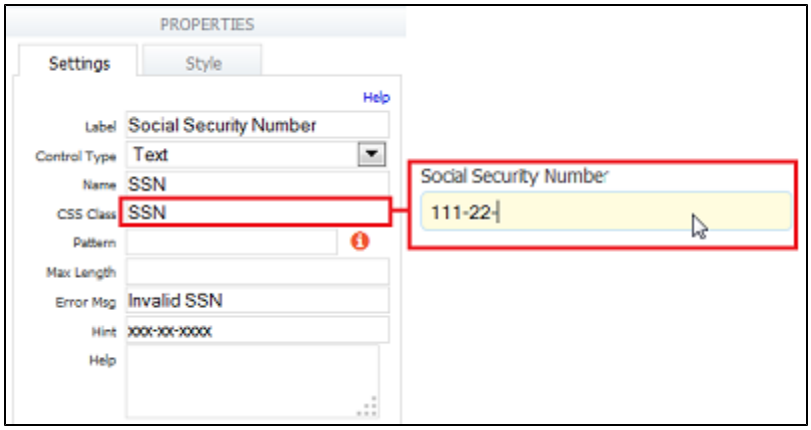
This JavaScript comment must be the first line in the script file that you upload to Live Forms.

```
var CustomEventHandlers = {
       setup: function (el) {
           if (CustomView.hasClass(el, 'SSN')) {
               FEvent.observe(el, 'keydown', this.formatSSN.bindAsObserver(this, el));
               FEvent.observe(el, 'keyup', this.formatSSN.bindAsObserver(this, el));
           }
       },
       formatSSN: function (event, element) {
           if (event.keyCode != 46 && event.keyCode != 8) {
               fldVal = $(element).value;
               var nom = fldVal.charAt(fldVal.length - 1);
               if (isNaN(nom) && nom != "-") {
                   $(element).value = fldVal.substring(0, fldVal.length - 1);
               } else {
                   if ((fldVal.length == 3) || (fldVal.length == 6)) {
                       $(element).value = fldVal + "-";
                   }
                   if (fldVal.length >= 11) {
                       $(element).value = fldVal.substring(0, fldVal.length - 1);
                   }
               }
           }
       }
   }
```

2. Login to Live Forms as a designer user. Click the 
   Edit icon for the application where you want to use the JavaScript. Click on the Script tab located on the left menu.



3. Browse your hard drive for you script file, Click Upload. Your file will be uploaded and it will display with the name Custom Script even though your script file may have another name. Be Aware that existing JavaScript files will be overwritten...
4. If you need to modify the script, you must download it, make your modifications and then upload it again. When you download the script by clicking on the 
   Download icon, it will be named custom.js.
5. Once you have uploaded the JavaScript, it is available for all forms/flows in the application. Remember to add the CSS class name to your form controls or your JavaScript may not work. Here is an image of an Employee Information form using an uploaded JavaScript to enter dashes in the Social Security Number field while the user is entering the data.



> If your JavaScript does not behave as expected after upgrading your version of Live Forms, it may be caused by a scrubbing of the custom.js file content by XSS protection. You may notice that all your code is on a single commented out line upon Inspection of the custom.js file, Upload an "unsrubbed" version of your custom.js file to solve the issue.

Method 2

Follow these steps to add JavaScript to your form using the Message control:

1. Add a message control to your form.
2. Add a <script> tag in the message property.
3. Put your JavaScript inside the script tag.

Here is an example of the script tag: put your JavaScript inside the script tag.

```
<script>
        /* <![CDATA[ */
        code goes here
        /* ]]> */
</script>
```

What can you do in a handler

You can call any JavaScript code in a handler; you can access the DOM of the page (note that you only have access to the DOM of the iframe in which the form is rendered assuming you're using a standard embedding) or call external code. In addition, you have access to the following methods:

1. CustomView.$frevvoControl(el) returns the HTML element corresponding to the Live Forms control that triggered the handler. This is typically a DIV HTML element and typically has CSS class 'f-control'
2. CustomView.getState (el) returns the state of the control as a JavaScript object. The state contains things like the label, value, hint, help etc. Use a tool such as Firebug or a different JavaScript debugger to view it in detail.
3. CustomView.getExtId (el) returns the name of the control as set in the Form Designer.
4. CustomView.getIndex (el) returns the index of the control in its enclosing repeat control if the control is repeating. If the control is not repeating, it returns -1.
5. CustomView.hasClass (el, className) returns true if the control has the indicated CSS class, false otherwise.

Custom Event Handler Example

Here is an example of a custom event handler:

```
var CustomEventHandlers = {
    setup: function (el) {
        if (CustomView.hasClass(el, 's-submit'))
            FEvent.observe(el, 'click', this.submitClicked.bindAsObserver(this,el));
        else if (CustomView.hasClass(el, 'my-class'))
            FEvent.observe(el, 'change', this.myHandler.bindAsObserver(this,el));
    },
    submitClicked: function (evt, el) {
        alert ('Submit button was clicked');
    },
    myHandler: function (evt, el) {
        alert ('My Class change handler called');
    },
    onSaveSuccess: function (submissionId) {
        alert ("Save complete. New submission id is: " + submissionId);
    },
    onSaveFailure: function () {
        alert ("Save failed");
    }
}
```

Let's look in a little more detail

1. You must have a class called CustomEventHandlers declared. Note that it is case-sensitive.
2. It must have a method called setup() which takes a single argument (called el above). When the form is loaded, Live Forms will call this setup() method for each control in the form and will pass in the control as the argument.
3. For each control we are interested in, the CustomEventHandlers class has a handler method. All handlers are functions that take two arguments: the event that triggered the handler and the control itself.
4. The example above uses a custom CSS class on the control in question to figure out if it's the control we are interested in. You can set a custom css class for any control in the Form Designer.
5. We're interested in two controls: the Submit button which already has a CSS class s-submit and a user-defined input control with custom CSS class my-class. For each control, we've associated a handler as described above. To associate a handler, call FEvent.observe (el, EVENT_NAME, handler) using the syntax above. The EVENT_NAME can be any standard event fired by your browser e.g. click (also called onClick) or change (also called onChange).
6. When the event is question is fired by the control in question, your handler will be called.
7. In addition to the event handlers, you can also provide methods that are called when the form is Saved using the Save/Load feature.
8. The onSaveSuccess() method is called when a submission is saved and the resulting submission ID is passed in.
9. The onSaveFailure() method is called when the save fails.

More Examples

You can add different event handling to your JavaScript code. This example handles click, mouseover and mouseout events to the Submit button:

```
/*
  * Custom Javascript here.
  */
   var CustomEventHandlers = {
    setup: function (el) {
        if (CustomView.hasClass(el, 's-submit'))
        {
            alert('setting up s-submit events');
            FEvent.observe(el, 'click', this.submitClicked.bindAsObserver(this,el));

            FEvent.observe(el, 'mouseover',
this.submitMouseOver.bindAsObserver(this,el));
            FEvent.observe(el, 'mouseout',
this.submitMouseOut.bindAsObserver(this,el));
        }
    },
    submitClicked: function (evt, el) {
        alert ('Submit button was clicked');
    },
    submitMouseOver: function (event, element) {
        alert ('Submit mouse over');
    },
    submitMouseOut: function (event, element) {
        alert ('Submit mouse out');
    }
}
```

Extensions for flows

In addition to the above, flows also support a few other methods.

```
var CustomFlowEventHandlers = {
    onNextClicked: function (name, id) {
        alert ("Next button clicked for activity: " + name);
    },
    onNavClicked: function (name, id) {
        alert ("Nav button clicked for activity: " + name);
    },
    onSaveSuccess: function (submissionId) {
        alert ("Save complete. New submission id is: " + submissionId);
    },
    onSaveFailure: function () {
        alert ("Save failed");
    }
}
```

As the method names indicate

1. onNextClicked() is called when the Continue button is clicked: this button might be labeled Continue or Finish or might have a custom label assigned by you. The parameters are the name and id of the current activity (step) in the flow.
2. onNavClicked() is called when the user clicks on a link in the Navigation toolbar if it is displayed. The parameters are the name and id of the activity (step) in the flow that is clicked.
3. Finally, onSaveSuccess() and onSaveFailure() for flows must be defined in the CustomFlowEventHandlers class.

It's not currently possible to directly fire a custom.js handler from a business rule. You can write a form.load rule that sets the value of a hidden

control and set a change handler for that control in your custom.js Custom Handlers.