Miva Merchant

MMButton Guide

Miva Merchant 9

MM9-MISC-305-1.0 3/27/15

Scope

This document describes how to create an **MMButton** in Miva Merchant by defining a JavaScript class. Experience with JavaScript is presumed.

Creating an MMButton

The basic setup of an **MMButton** is as follows:

```
...
var button;


button          = new MMButton( element_parent );
button.SetText( 'Cancel' );
button.SetImage( 'cancel' );
/* where 'cancel' is a pre-defined value in the MivaIconMap function. At this
   point, only pre-defined values are allowed (aka, no custom images)*/
button.SetHoverText( 'Stop the save process' );
button.SetOnClickHandler( function( e ) { alert( 'Fooled you! You can\'t stop
 it.' ); } );
...
```

Note: You can choose to set only image or only text.

There are a number of other **Set**/**Get** functions that allow finer control over your **MMButton**. Following is a list of all current modification functions:

- **button.SetText(text);**
- **button.GetText();**
- **button.SetImage(image);**
- **button.GetImage();**
- **button.SetProcessingText(text);**
- **button.GetProcessingText();**
- **button.SetProcessingImage(image);**
- **button.GetProcessingImage();**
- **button.SetHoverText(text);**
- **button.GetHoverText();**
- **button.SetClassName(classname);**
- **button.GetClassName();**
- **button.SetAllowMiddleClick(true | false);**
- **button.SetAllowRightClick(true | false);**
- **button.SetOnClickHandler(function(e) { ; });**

By default, the processing text when a button is put into **processing** mode is “Processing”, but this text can be overridden in circumstances where it makes sense to do so.

By default, middle-click and right-click functionality are disabled.

Controlling an MMButton

The following list of functions is used to control the **MMButton**. **SimulateClick**, for example, is used to trigger a click event on the button if you need to do so without user input.

button.SimulateClick();	Triggers a click on the button
button.Enable();	Allows interactivity with the button
button.Disable();	Disables the button (cannot click)
button.RemoveFromParent();	Removes the button element from the DOM
button.Show();	Makes the button visible in the DOM if it is not already
button.Hide();	Hides the button from view by setting the element's display property to ‘none’
button.ShowImage();	Makes the button image visible if a button image is present
button.HideImage();	Hides the button image
button.ShowText();	Shows the button text if text is present
button.HideText();	Hides the button text
button.SetFocus();	Brings DOM focus to the button element
button.RemoveFocus();	Removes DOM focus from the button element
button.SetPocessing_Start();	Starts the **processing** mode. When processing mode is enabled, the button text/image will be converted to the processing mode. This is useful when you have a list of buttons and you want to show which button was clicked.
button.SetProcessing_End();	Ends the **processing** mode

Custom MMButton (DeriveFrom)

MMButton is like any other JavaScript class and is capable of being overridden (like **MMBatchList**). It should be rare that this is needed, but in certain cases it might be a requirement. This is how we achieved the **MMUploadButton** (see *MMUploadButton* on page 6).

Styling an MMButton

There are a number of different CSS styles that are available for settings styles. If the **MMButton** classnames are unchanged, the element/classname structure is as follows:

```
a - "mm9_button" - "active", "focus", and "disabled" are applied to this
    element when the necessary circumstances are met
    span - "mm9_button_image mm9_mivaicon"
    span - "mm9_button_text"
```

When a button is pressed, it simultaneously receives focus (if the button didn't already have focus) and has the **active** state assigned. The **active** state will go away once the **mouseup** event is called. Focus will remain until another element in the DOM takes focus away from the button element.

If the button has been disabled, the root level element for the button will receive a "disabled" class name appended. Best practice in styling the button is to maintain the existing class name and simply append a custom class name that you would use to override/add style declarations to.

Example:

```
button.SetClassName( button.GetClassName() + ' my_button_class' );
```

If you wanted to override the button's text color, you could then do something like:

```
<style type="text/css">
        .my_button_class .mm9_button_text
        {
            color: #1eabbd;
        }
    </style>
```

MMUploadButton

MMUploadButton has a slightly different setup than a normal **MMButton**. Following is an example of how to setup and use an MMUploadButton:

```
...
var button;


button          = new MMUploadButton( element_parent, /* Support multiple file
 selection in browsers where this is an option */ true );
button.SetText( 'Upload Image' );
button.SetImage( 'upload' );
 /* where 'cancel' is a pre-defined value in the MivaIconMap function. At this
 point, only pre-defined values are allowed (aka, no custom images) */
button.SetHoverText( 'Upload an image to your account' );
button.SetOnChangeHandler( function( button, file_input ) { Process_Upload(
 button, file_input ) ); } );


/* Also available:
    button.SetMultiple( true ); // Set the "allow mutliple file selection"
                                // flag on the button
    button.GetFileInput();      // Returns the <input type="file" />
                                // DOM element
    button.Reset()              // Resets the button to the default state (
                                // you cause this to wipe the button/input
                                // field clean after your upload is complete)
*/
...

function Process_Upload( button, file_input )
{
    var i, i_len, progressbar, progressdialog;


    progressdialog              = new ProgressDialog( 'Uploading Image(s)' );
    progressdialog.onhide       = function() { self.button_upload.Reset(); };


    // Check to see if our file input has/supports multiple file uploads
    if ( file_input.files && ( window.FormData || ( window.FileReader &&
     XMLHttpRequest.sendAsBinary ) ) )
    {
        for ( i = 0, i_len = file_input.files.length; i < i_len; i++ )
        {
            Process_File( progressdialog, progressbar, file_input.files[ i ]
             );
        }
    }
```

```
    else
    {
        progressbar                = progressdialog.ProgressBar_Append( true );
        progressbar.oncomplete     = function( response )
        {
            if ( !response.success )
            {
                return Modal_Alert( response.error_message );
            }

            progressdialog.ProgressBar_Remove( progressbar );
        }

        Image_Upload( file_input, null, progressbar );
    }

    /* Reset the file input so we can use it again. This function is found in
     admin/ui.js */
    ClearFileInputValue( file_input );
}

function Process_File( progressdialog, progressbar, file )
{
    progressbar                 = progressdialog.ProgressBar_Append( true );
    progressbar.oncomplete      = function( response )
    {
        if ( !response.success )
        {
            return Modal_Alert( response.error_message );
        }

        progressdialog.ProgressBar_Remove( progressbar );
    }

    Image_Upload( null, file, progressbar );
}

function Image_Upload( file_input, file_object, progress_object )
{
    return AJAX_Call_WithFile( progress_object, 'admin', 'Image_Upload', null,
     'Image', file_input, file_object );
}
```