# Office AMS: APP PART PROPERTY UI OVERRIDE

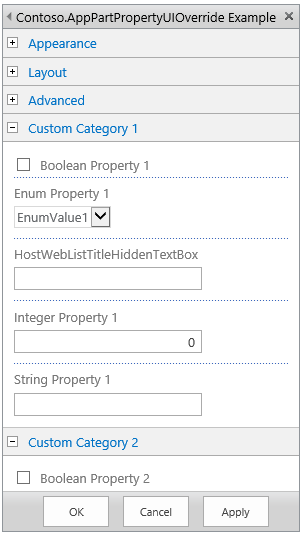
|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample shows how one can use JavaScript to alter the user interface of custom App Part properties. | * Office 365 Multi Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Core.AppPartPropertyUIOverride, version 1.0 |
| Author: | Alex Randall, Microsoft |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# THE PROBLEM: APP PART PROPERTIES

Custom App Parts are an easy way to allow end users to add functionality to their SharePoint pages. While they are extremely powerful, there are some limitations that developers may encounter.

One limitation is that there’s no explicit way to control the user interface of the App Part properties UI that SharePoint generates other than the display name, category, description, and data type. Another limiting factor is that there are only four datatypes available: boolean, enum, integer, string.

Example App Part property user interface generated by SharePoint:



Notice:

* the categories (Custom Category 1 and Custom Category 2) are at the bottom of the App Part properties UI (and are not expanded by default)
* there are no instructions/examples for the end user for each property
* there’s no other UI elements available other than checkbox, dropdown, or textbox

Example Elements.xml file that was used:

<?xml version="1.0" encoding="utf-8"?>

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

<ClientWebPart Name="ContosoAppPartPropertyUIOverride" Title="Contoso.AppPartPropertyUIOverride Example" Description="An example App Part that shows the techniques required to override the App Part property UI." DefaultWidth="500" DefaultHeight="200">

<!-- Content element identifies the location of the page that will render inside the client web part

Properties are referenced on the query string using the pattern \_propertyName\_

Example: Src="~appWebUrl/Pages/ClientWebPart1.aspx?Property1=\_property1\_" -->

<Content Type="html" Src="~remoteAppUrl/Pages/ContosoAppPartPropertyUIOverride.aspx?{StandardTokens}&amp;BooleanProperty1=\_BooleanProperty1\_&amp;EnumProperty1=\_EnumProperty1\_&amp;IntegerProperty1=\_IntegerProperty1\_&amp;StringProperty1=\_StringProperty1\_&amp;HostWebListTitleHiddenTextBox=\_HostWebListTitleHiddenTextBox\_&amp;BooleanProperty2=\_BooleanProperty2\_" />

<!-- Define properties in the Properties element.

Remember to put Property Name on the Src attribute of the Content element above. -->

<Properties>

<Property

Name="BooleanProperty1"

Type="boolean"

WebDisplayName="Boolean Property 1"

WebDescription="If this checkbox is checked, it means true."

WebCategory="Custom Category 1"

DefaultValue="false"

RequiresDesignerPermission="true"

WebBrowsable="true" />

<Property

Name="EnumProperty1"

Type="enum"

WebDisplayName="Enum Property 1"

WebDescription="Select a value in the drop down list."

WebCategory="Custom Category 1"

DefaultValue="EnumValue1"

RequiresDesignerPermission="true"

WebBrowsable="true" >

<EnumItems>

<EnumItem Value="EnumValue1" WebDisplayName="EnumValue1"/>

<EnumItem Value="EnumValue2" WebDisplayName="EnumValue2"/>

</EnumItems>

</Property>

<Property

Name="IntegerProperty1"

Type="int"

WebDisplayName="Integer Property 1"

WebDescription="Type a number into this field."

WebCategory="Custom Category 1"

DefaultValue="0"

RequiresDesignerPermission="true"

WebBrowsable="true" />

<Property

Name="StringProperty1"

Type="string"

WebDisplayName="String Property 1"

WebDescription="Type text into this text box."

WebCategory="Custom Category 1"

DefaultValue=""

RequiresDesignerPermission="true"

WebBrowsable="true" />

<Property

Name="HostWebListTitleHiddenTextBox"

Type="string"

WebDisplayName="HostWebListTitleHiddenTextBox"

WebDescription="This property is hidden via JavaScript at runtime, we are using it for storage and ability to transfer data to Remote Web Page via QueryString only."

WebCategory="Custom Category 1"

DefaultValue=""

RequiresDesignerPermission="true"

WebBrowsable="true" />

<Property

Name="BooleanProperty2"

Type="boolean"

WebDisplayName="Boolean Property 2"

WebDescription="Description for Boolean Property 2"

WebCategory="Custom Category 2"

DefaultValue="false"

RequiresDesignerPermission="true"

WebBrowsable="true" />

</Properties>

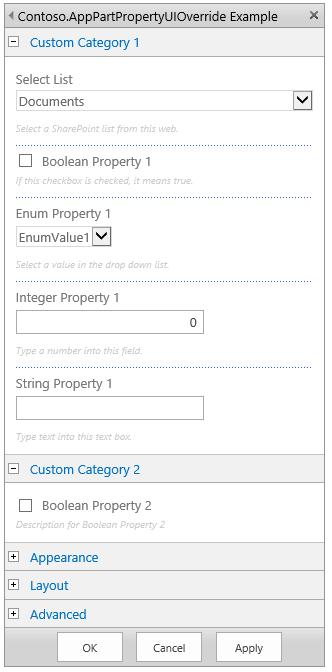
</ClientWebPart>

</Elements>

# AN EXAMPLE SOLUTION: JAVASCRIPT INJECTION TO CHANGE APP PART PROPERTY UI

An example solution is to use the JavaScript injection pattern (on the Host Web) to change the App Part property UI after it’s rendered by SharePoint.

Example App Part with custom properties UI modified by JavaScript at runtime:



Notice in the example that used JavaScript to override the default App Property user interface:

* The custom categories (Custom Category 1 and Custom Category 2) are now at the top
* The first custom category (Custom Category 1) is automatically opened on page load
* New html has been injected into the page at the top (Select List) with a dropdown that contains all of the lists on this host web
* The custom property: “HostWebListTitleHiddenTextBox” is hidden visually from the App Part Property UI (but still there behind the scenes so we can read/write from it and use it for storage
* There are instructions underneath each property informing the user what to do

## APPROACH OVERVIEW

The overall approach to the pattern is the following:

1. A small set of JavaScript logic is included on all pages of the SharePoint web and “waits” until:
   1. an App Part property pane is open and
   2. the specified category is present
2. Once detected, then it automatically detects and loads additional libraries (jQuery, SP.JS) including a specialized JavaScript file that changes the UI of the App Part properties UI.

The example uses a JavaScript injection pattern to deploy JavaScript files to the host web and wire one of them up using a UserCustomAction so it’s included on all pages in the host web.

On app uninstall, it cleans up after itself so that host web is clean.

Also note, that in the example, there are some helper libraries and functions that are provided to make the task of overriding the App Part property UI with JavaScript easier.

## DETAILS

**APP EVENT RECEIVER**

In the **AppEventReceiver.svc.cs** in the Provider Host app for SharePoint , the files are provision to the host web (and/or retracted). A set of helper libraries are provided help make wiring up the JavaScript files easy as well as jQuery (it will auto detect if jQuery is present or not, if not, it will load it).

Items of interested are highlighted in yellow below:

/// <summary>

/// Handles app events that occur after the app is installed or upgraded, or when app is being uninstalled.

/// </summary>

/// <param name="properties">Holds information about the app event.</param>

/// <returns>Holds information returned from the app event.</returns>

public SPRemoteEventResult ProcessEvent(SPRemoteEventProperties properties)

{

SPRemoteEventResult result = new SPRemoteEventResult();

using (ClientContext clientContext = TokenHelper.CreateAppEventClientContext(properties, useAppWeb: false))

{

if (clientContext != null)

{

clientContext.Load(clientContext.Web);

clientContext.ExecuteQuery();

// setup helper classes

HostWebManager hostWebManager = new HostWebManager("ContosoAppPartPropertyUIOverride", clientContext);

// now see which app event we are in

switch (properties.EventType)

{

case SPRemoteEventType.AppInstalled:

// the app for SharePoint was installed

// setup helper class

AppPartPropertyUIOverrider overrider = new AppPartPropertyUIOverrider(hostWebManager, properties, "jquery-2.1.0.min.js");

// do the actual App Part property UI overrides

overrider.OverrideAppPartPropertyUI("Custom Category 1", "Contoso.OverrideExample.js");

break;

case SPRemoteEventType.AppUninstalling:

// the app for SharePoint is uninstalling

// uninstall all app-specific assets that were deployed (global assets are left for safety)

hostWebManager.UninstallAssets();

break;

}

}

}

return result;

}

Notes:

* The custom category that is being looked for in an App Part on app pages is: **“Custom Category 1”** in this example. This category it’s searching on needs to be unique across all App Parts.
* The specialized custom JavaScript file to load if the custom category is found is: **Contoso.OverrideExample.js**

**SPECIALIZED CUSTOM JAVASCRIPT FILE**

The **Contoso.OverrideExample.js** file is a specially crafted JavaScript file that utilizes jQuery, the **Contoso.AppPartPropertyUIOverride.js** helper library (more on that below) and SP.JS (the SharePoint JavaScript Client Side Object Model (CSOM) to manipulate the App Part property UI at runtime.

The overall sequence of this example JavaScript file is:

* Move your custom categories to the top
* Expand the top most category
* Hide a property
* Create a dropdown list using jQuery/html
* Use the SharePoint JavaScript Client Side Object Model (CSOM) to get the titles of all lists on the host web
* Add them to the dropdown
* Render tool tips as instructions
* Declare we are finished (the App Part property UI is shown at this point).

Here is the example code:

/\*! Contoso.OverrideExample.js

\*

\* Example JavaScript code that changes the user interface of the example

\* App Part via JavaScript at runtime by using the example

\* Contoso.AppPartPropertyUIOverride JavaScript library

\*

\*/

// NOTE: the following four lines of course ensure that everything's in it's

// isolated module and loaded when SharePoint Minimal Download Strategy (MDS)

// is activated or not activated

(function () {

"use strict";

window.startConstosoAppPartPropertyUIOverride = function () {

(function ($, overrider) {

// at this point of JavaScript code execution, the following

// three JavaScript libraries have been automatically loaded and

// are ready for use:

//

// jQuery

// sp.js (SharePoint 2013 JavaScript Client Side Object Model (CSOM))

// Contoso.AppPartPropertyUIOverride.js

//

// also, the static Contoso.AppPartPropertyUIOverride runtime module has

// been assigned to the "overrider" variable for ease of use

overrider.moveCategoryToTop("Custom Category 2");

overrider.moveCategoryToTop("Custom Category 1");

overrider.expandCategory("Custom Category 1");

overrider.hideProperty("HostWebListTitleHiddenTextBox", "Custom Category 1");

// create new custom dropdown list in property UI at runtime

var listsDropDownJQueryWrapper = $(

overrider.createNewContentAtTop({

category:"Custom Category 1",

optionalName:"Select List",

optionalToolTip:"Select a SharePoint list from this web."

})

.html("<select id=\"contosoSelectSPList\"></select>")[0]

).find("#contosoSelectSPList");

// now use the SharePoint JavaScript Client Side Object (CSOM) model

// to get the titles of all lists in this host web

var clientContext = new SP.ClientContext();

var hostWeb = clientContext.get\_web();

var lists = hostWeb.get\_lists();

clientContext.load(lists, 'Include(Title)');

clientContext.executeQueryAsync(

function () {

// query is done

// loop through list titles and construct html to output

var listEnumerator = lists.getEnumerator();

var list = null;

var html = [];

while (listEnumerator.moveNext()) {

list = listEnumerator.get\_current();

// build the html string for a select list (dropdown) value

html.push("<option>");

html.push(list.get\_title());

html.push("</option>");

}

// inject the html to the drop down

listsDropDownJQueryWrapper.html(html.join(""));

// now set the current value of the dropdown

// based on what is in the hidden property

listsDropDownJQueryWrapper.val(overrider.getValue("HostWebListTitleHiddenTextBox", "Custom Category 1"));

// wire up an event handler on the drop down

// so that when it's changed,

// we automatically write the value to the hidden text box

listsDropDownJQueryWrapper.change(function () {

// dropdown changed by end user

// write value to hidden text box

overrider.setValue("HostWebListTitleHiddenTextBox", listsDropDownJQueryWrapper.val(), "Custom Category 1");

});

// render the tool tips as instructions

overrider.renderToolTipsAsInstructions("Custom Category 1");

// tell the AppPartPropertyUIOverride framework that we are done

// overriding the App Part property UI and to show the property pane now

overrider.finished();

});

}(jQuery, Contoso.AppPartPropertyUIOverride))

};

}())

// Register this JavaScript file for SharePoint 2013's SharePoint Minimal Download Strategy (MDS) if possible

RegisterModuleInit("Contoso.OverrideExample.js", startConstosoAppPartPropertyUIOverride); //MDS registration

startConstosoAppPartPropertyUIOverride(); //non MDS run

if (typeof (Sys) != "undefined" && Boolean(Sys) && Boolean(Sys.Application)) {

Sys.Application.notifyScriptLoaded();

}

if (typeof (NotifyScriptLoadedAndExecuteWaitingJobs) == "function") {

NotifyScriptLoadedAndExecuteWaitingJobs("Contoso.OverrideExample.js");

}

This was just an example, however using this helper library and (even other libraries such as jQueryUI), the possibilities are endless as to what you can inject into your App Part property UI at runtime via JavaScript.

## EXAMPLE JAVASCRIPT LIBRARY DOCUMENTATION

What follows is the JavaScript documentation for the example **Contoso.AppPartPropertyUIOverride.js** library that makes it easy to change the user interface of custom App Part properties.

**moveCategoryToTop(category)**

Moves the specified category to the top of the App Part property pane UI.

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLE:

overrider.moveCategoryToTop("Custom Category 1");

**expandCategory(category)**

Expands the specified category and closes all others in the App Part property pane UI.

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLE:

overrider.expandCategory("Custom Category 1");

**hideProperty(name, category)**

Hides the specified property in the App Part property UI.

*name:* (string) the display name of the property. Example: “My Property 1”

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLE:

overrider.hideProperty("HostWebListTitleHiddenTextBox", "Custom Category 1");

**createNewContentAtBottom(settings)**

Creates a new html content area in the specified App Part Property UI category bottom and returns a jQuery object that wraps the new content area created.

*settings:* (Object) A JavaScript object that contains the required and optional settings for this operation.

*category:*  (string) the display name of the category. Example: “Custom Category 1”

*optionalName:*  (string) the optional display name of the property to create.

Example: “My Property 1”

*optionalToolTip:*  (string) the optional tool tip of the property to create.

Example: “My Tool Tip 1”

*outputSeparator:*  (boolean) true to output the dotted separator line after this property; false

to not output it. Default is true

EXAMPLES:

// EXAMPLE 1: create new content area with no property name and dotted line at bottom of the "Custom Category 1" and inject html into it using jQuery

overrider.createNewContentAtBottom({ category: "Custom Category 1" }).html("This is html injected inside the content area");

// EXAMPLE 2: create new content area with property name, tooltip, and no dotted line at bottom of the "Custom Category 1" and inject html into it using jQuery

overrider.createNewContentAtBottom({

category: "Custom Category 1",

optionalName: "Custom Property 1",

optionalToolTip: "Custom Property 1 Tool Tip",

outputSeparator: false

}).html("This is html injected inside the content area");

**createNewContentAtTop(settings)**

Creates a new html content area in the specified App Part Property UI category top and returns a jQuery object that wraps the new content area created.

*settings:* (Object) A JavaScript object that contains the required and optional settings for this operation.

*category:*  (string) the display name of the category. Example: “Custom Category 1”

*optionalName:*  (string) the optional display name of the property to create.

Example: “My Property 1”

*optionalToolTip:*  (string) the optional tool tip of the property to create.

Example: “My Tool Tip 1”

*outputSeparator:*  (boolean) true to output the dotted separator line after this property; false

to not output it. Default is true

EXAMPLES:

// EXAMPLE 1: create new content area with no property name and dotted line at top of the "Custom Category 1" and inject html into it using jQuery

overrider.createNewContentAtTop({ category: "Custom Category 1" }).html("This is html injected inside the content area");

// EXAMPLE 2: create new content area with property name, tooltip, and no dotted line at top of the "Custom Category 1" and inject html into it using jQuery

overrider.createNewContentAtTop({

category: "Custom Category 1",

optionalName: "Custom Property 1",

optionalToolTip: "Custom Property 1 Tool Tip",

outputSeparator: false

}).html("This is html injected inside the content area");

**getValue(name, category)**

Gets the current value of the declared string, number, enum, or boolean property that's already rendered in the App Part property UI.

*name:* (string) the display name of the property. Example: “My Property 1”

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLE:

var value = overrider.getValue("My Property 1", "Custom Category 1");

**setValue(name, value, category)**

Sets the current value of the declared string, number, enum, or boolean property that's already rendered in the App Part property UI.

*name:* (string) the display name of the property. Example: “My Property 1”

*value:* (any) the value to set. Example: “The value”

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLES:

// EXAMPLE 1: set the value of a string property

overrider.setValue("My Property 1", "Hello", "Custom Category 1");

// EXAMPLE 2: set the value of an integer property

overrider.setValue("My Property 1", 342, "Custom Category 1");

// EXAMPLE 3: set the value of a boolean property

overrider.setValue("My Property 1", true, "Custom Category 1");

// EXAMPLE 4: set the value of an enum property

overrider.setValue("My Property 1", "EnumValue1", "Custom Category 1");

**renderToolTipsAsInstructions(category)**

Renders tool tips as html instruction text below each property in the specified category.

*category:* (string) the display name of the category. Example: “Custom Category 1”

EXAMPLE:

overrider.renderToolTipsAsInstructions("Custom Category 1");

**finished()**

Tells the App Part property UI framework you are done with overriding the App Part property UI and to show the property pane again.

EXAMPLE:

overrider.finished();