# Office AMS: Show apps in a dialog

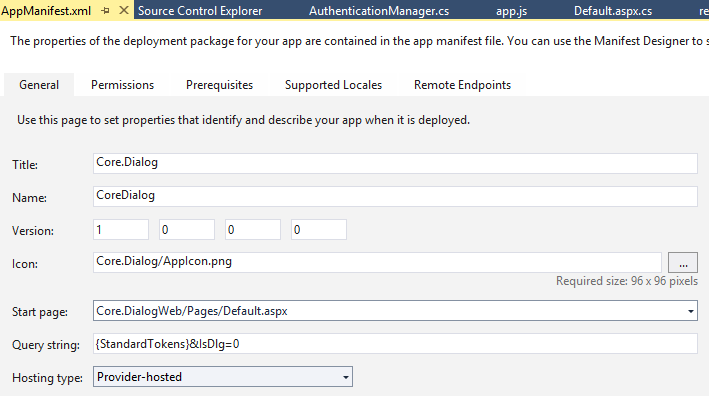
|  |  |
| --- | --- |
| Summary: | Applies to: |
| This scenario shows how you can show an app inside a dialog | * Office 365 Multi-Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Core.Dialog, version 1.0 |
| Author: | Bert Jansen (Microsoft) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 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.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# General comments

This sample scenario uses the Office AMS core library to inject links for opening a SharePoint app inside a dialog. We’ll show how to open the app inside a dialog from a custom action and how to do the same from a link on a SharePoint wiki page. The app that will be shown in a dialog is the same app that you’ll use to setup the demo, meaning that you’ll be able to experience how one and the same app can be used in a full page immersive experience (using the chromecontrol) and in a modal dialog experience. Some special attention has been given to the button click handling: the same OK and Cancel buttons behave differently when the app is shown in a dialog or as a full page immersive experience. Finally the app shows how you can use JSOM to obtain data from the host web regardless of the whether the app is shown in a dialog (=uses iframe) or not. Next chapters provide more details on this.

# IsDlg URL parameter

To specify whether the app is shown in a dialog or not we’ve foreseen an additional URL parameter named IsDlg. If this one has a value of 1 then this is an indication that the app is shown in a dialog, value 0 indicates the default full page experience. This IsDlg parameter is added as additional query string:



# Scenario 1: Insert a custom action to open the App in a dialog from the site settings menu

This scenario uses the Office AMS core method “AddCustomAction” to insert a custom action to the site actions menu of the hosting web. In order to open the app in a dialog it uses JavaScript instead of a static url for the url value of the custom action. In the JavaScript we make use of the SharePoint [SP.UI.ModalDialog.showModalDialog](http://msdn.microsoft.com/en-us/library/office/ff410058(v=office.14).aspx) class to show a modal dialog.

StringBuilder modelDialogScript = new StringBuilder(10);

modelDialogScript.Append("javascript:var dlg=SP.UI.ModalDialog.showModalDialog({url: '");

modelDialogScript.Append(String.Format("{0}", SetIsDlg("1")));

modelDialogScript.Append("', dialogReturnValueCallback:function(res, val) {} });");

//Create a custom action

CustomActionEntity customAction = new CustomActionEntity()

{

Title = "Office AMS Dialog sample",

Description = "Shows how to launch an app inside a dialog",

Location = "Microsoft.SharePoint.StandardMenu",

Group = "SiteActions",

Sequence = 10000,

Url = modelDialogScript.ToString(),

};

//Add the custom action to the site

cc.Web.AddCustomAction(customAction);

See [here](http://msdn.microsoft.com/en-us/library/office/bb802730(v=office.15).aspx) for more information on the custom action settings.

# Scenario 2: Insert a script editor web part to open the app in a dialog from a site wiki page

Here we use the Office AMS Core page and web part manipulation methods to create a new page and add a configured script editor web part to it.

string scenario1Page = String.Format("scenario1-{0}.aspx", DateTime.Now.Ticks);

string scenario1PageUrl = cc.Web.AddWikiPage("Site Pages", scenario1Page);

cc.Web.AddLayoutToWikiPage("SitePages", WikiPageLayout.OneColumn, scenario1Page);

WebPartEntity scriptEditorWp = new WebPartEntity();

scriptEditorWp.WebPartXml = ScriptEditorWebPart();

scriptEditorWp.WebPartIndex = 1;

scriptEditorWp.WebPartTitle = "Script editor test";

cc.Web.AddWebPartToWikiPage("SitePages", scriptEditorWp, scenario1Page, 1, 1, false);

# Ensuring that your JSOM code works, even when the app is shown in a dialog

When an app is running inside another app domain and it needs data from the host then we’re dealing with a cross domain call. To realize this one needs to use the ProxyWebRequestExecutorFactory class as shown below. This technique allows the app to make the cross domain call, regardless of whether the app is loaded as a dialog or not.

context = new SP.ClientContext(appWebUrl);

factory = new SP.ProxyWebRequestExecutorFactory(appWebUrl);

context.set\_webRequestExecutorFactory(factory);

appContextSite = new SP.AppContextSite(context, spHostUrl);

this.web = appContextSite.get\_web();

See [here](http://msdn.microsoft.com/en-us/library/office/fp179927(v=office.15).aspx) for more information on cross domain calls.