



Composite business apps in SharePoint 2013 and SharePoint Online solution pack

Microsoft Corporation

November 2014

**Applies to:** SharePoint 2013 and SharePoint Online

**Summary:** This solution pack includes code and documents that demonstrate and describe techniques for creating composite business apps in apps for SharePoint 2013 and SharePoint Online.

©2014 Microsoft Corporation. All rights reserved.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

# Contents

This solution pack refers to apps that are tightly integrated with your business processes and other line of business technologies (databases, web services, etc.) as composite business apps. These apps generally need to encapsulate a number of relatively complex interactions, both with users and with other technologies, and they require some planning related to high level design.

The composite business apps solution pack focusses on the following building blocks that you may need to integrated into your apps and into your overall approach to the SharePoint 2013 app model:

* Hosting approaches (SharePoint-hosted vs. provider-hosted)
* Migration of InfoPath forms capabilities to the app model
* Data storage models
* Departmental apps that integrate with complex business processes. This solution pack uses an event planning app as a reference implementation
* **Workflows with custom web service calls**

This first module of the solution pack covers the hosting, InfoPath, and data storage building blocks. The second module discusses departmental apps, and the third module focusses on workflows.

Table 1 describes these three modules.

**Table 1. Composite business apps in SharePoint 2013 and SharePoint Online solution pack modules**

|  |  |  |
| --- | --- | --- |
| **Module** | **Name** | **Description** |
| 1 | Hosting options, forms, and data | Describes appropriate use cases for SharePoint-hosted apps and provider-hosted apps. Provides guidance for moving InfoPath form capabilities to SharePoint 2013, and compares data storage options for the app model. |
| 2 | Departmental apps | Describes a reference implementation of a corporate events app that demonstrates several ways to integrate apps for SharePoint into your business operations. |
| **3** | **Workflows** | **Describes three code samples that demonstrate how to deploy a SharePoint 2013 workflow to the host web from an app for SharePoint and two approaches for calling web services from workflows.** |

# Workflows that call custom web services

The SharePoint 2013 app model allows you to create and deploy workflows that run on either the app web or the host web. These workflows can interact with the remotely hosted portions of provider-hosted apps and call remote web services that contain important business data in one of two ways:

* The workflow passes query information to the remotely hosted portion of the app. The remote web application then calls the web service and passes the information back to SharePoint.
* The workflow queries the web service by using the SharePoint 2013 web proxy. The workflow passes the results of the query to the remotely hosted portion of the app, which then passes the information to SharePoint.

The information retrieved from the web service can be stored in SharePoint lists. In the first two examples discussed in this module, the workflows and the lists are deployed to the app web when the app installs. The third example contains a basic shell of a workflow, along with instructions for deploying the workflow to the host web and associating it with a list on the host web. The instructions for that sample are a workaround that won’t be necessary when Visual Studio tools for deploying workflows to the host web become available.

## [Call custom web services from a workflow](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallCustomService)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample demonstrates how to design a provider-hosted app so that it queries a web service via the remotely hosted web application that deploys with the app. | The design of this sample is useful when you want all of the interactions with the web service to be handled by the remotely hosted portion of your provider-hosted app. | The sample starts a workflow from a remote web application. This workflow passes query information submitted by the user to the remote web application, which then uses that information to construct a query to the Northwind OData web service. The query returns the product suppliers for a given country. After it receives that information, the remote web application updates a product suppliers list that the app has deployed to the app web. |

**Related samples**:

[Workflow.AssociateToHostWeb](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.AssociateToHostWeb)

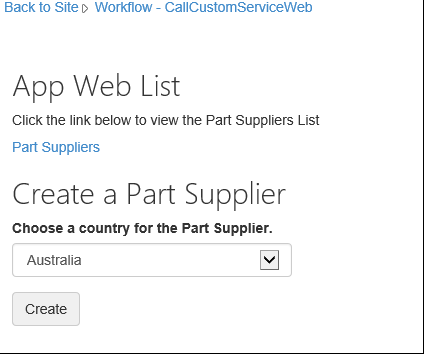
**Alternative approaches:**

[Workflow.CallServiceUpdateSPViaProxy](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallServiceUpdateSPViaProxy)

The [Workflow.CallCustomService](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallCustomService) project page contains instructions for deploying this app. You can also deploy and test with F5 debugging in Visual Studio if you follow the instructions in this [Debugging SharePoint 2013 workflows using Visual Studio 2013](http://blogs.msdn.com/b/officeapps/archive/2013/10/30/debugging-sharepoint-2013-workflows-using-visual-studio-2013.aspx) blog post.

This app’s launch page presents you with a drop-down menu that allows you to pick a country for which you want to create a product suppliers list (Figure 1).

**Figure 1. Launch page of custom web service workflow app**



That button calls a method in the Controllers\PartSuppliersController.cs file that creates a new entry in the Part Suppliers list on the app web. The **Create** method in that file calls the **Add** method that is defined in the Services\PartSuppliersService.cs file.

**Create:**

public ActionResult Create(string country, string spHostUrl)

{

var spContext = SharePointContextProvider.Current.GetSharePointContext(HttpContext);

using (var clientContext = spContext.CreateUserClientContextForSPAppWeb())

{

var service = new PartSuppliersService(clientContext);

var id = service.GetIdByCountry(country);

if (id == null)

{

id = service.Add(country);

TempData["Message"] = "Part Supplier Successfully Created!";

}

else

TempData["ErrorMessage"] = string.Format("Failed to Create The Part Supplier: There's already a Part Supplier who's country is {0}.", country);

return RedirectToAction("Details", new { id = id.Value, SPHostUrl = spHostUrl });

}

}

**Add:**

public int Add(string country)

{

var item = list.AddItem(new ListItemCreationInformation());

item["Country"] = country;

item.Update();

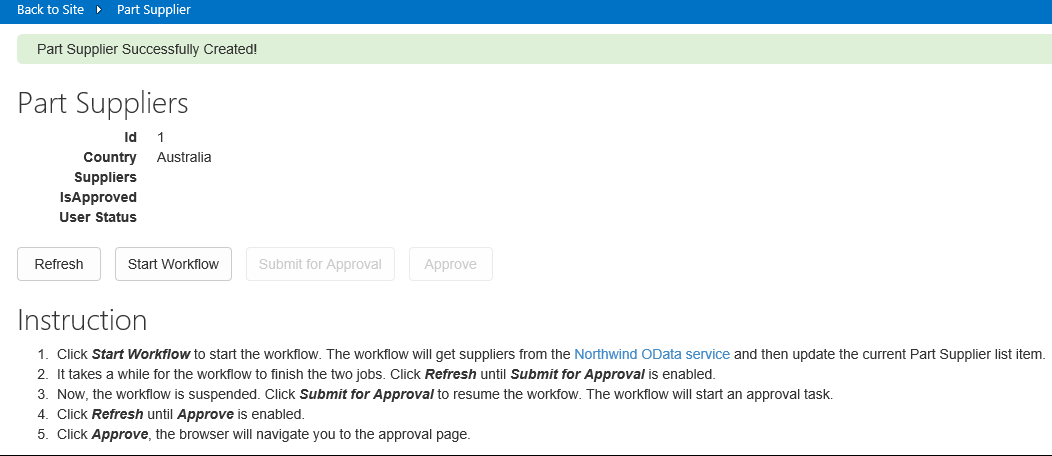
clientContext.ExecuteQuery();

return item.Id;

}

After creating that new list item, the app presents a button that starts the approval workflow (Figure 2).

**Figure 2. Start workflow button in the custom web service workflow app**



Clicking on the **Start Workflow** button triggers the **StartWorkflow** method in the Controllers\PartSuppliersController.cs file. This method packages the app web URL, the web service URL (for your remotely hosted web application, not for the Northwind web service), and the context token values and passes them to the **StartWorkflow** method in the Services\PartSuppliersService.cs file. The **PartSuppliersService** will need the context token to interact with SharePoint.

public ActionResult StartWorkflow(int id, Guid workflowSubscriptionId, string spHostUrl)

{

var spContext = SharePointContextProvider.Current.GetSharePointContext(HttpContext) as SharePointAcsContext;

var webServiceUrl = Url.RouteUrl("DefaultApi", new { httproute = "", controller = "Data" }, Request.Url.Scheme);

var payload = new Dictionary<string, object>

{

{ "appWebUrl", spContext.SPAppWebUrl.ToString() },

{ "webServiceUrl", webServiceUrl },

{ "contextToken", spContext.ContextToken }

};

using (var clientContext = spContext.CreateUserClientContextForSPAppWeb())

{

var service = new PartSuppliersService(clientContext);

service.StartWorkflow(workflowSubscriptionId, id, payload);

}

TempData["Message"] = "Workflow Successfully Started!";

return RedirectToAction("Details", new { id = id, SPHostUrl = spHostUrl });

}

The **StartWorkflow** method in the PartSuppliersService.cs file creates a workflow instance and passes the three values (**appWebUrl**, **webServiceUrl**, **contextToken**) stored in the payload variable to the workflow.

{

var workflowServicesManager = new WorkflowServicesManager(clientContext, clientContext.Web);

var subscriptionService = workflowServicesManager.GetWorkflowSubscriptionService();

var subscription = subscriptionService.GetSubscription(subscriptionId);

var instanceService = workflowServicesManager.GetWorkflowInstanceService();

instanceService.StartWorkflowOnListItem(subscription, itemId, payload);

clientContext.ExecuteQuery();

}

After the workflow starts, it makes a POST HTTP request to the remotely hosted web application. This request tells the web application to update the suppliers list with the suppliers for the country that the user has just added. The Controllers\DataController.cs file contains a POST method that receives the contents of this request.

public void Post([FromBody]string country)

{

var supplierNames = GetSupplierNames(country);

UpdateSuppliers(country, supplierNames);

}

The **GetSupplierNames** method constructs and executes a LINQ query to the Northwind OData web service for all of the suppliers associated with the selected country. The **UpdateSuppliers** method then updates the **Suppliers** field of the newly added list item. The code that handles these operations is in the Controllers\DataController.cs file.

**Query Northwind:**

private string[] GetSupplierNames(string country)

{

Uri uri = new Uri("http://services.odata.org/V3/Northwind/Northwind.svc");

var entities = new NorthwindEntities(uri);

var names = entities.Suppliers

.Where(s => s.Country == country)

.AsEnumerable()

.Select(s => s.CompanyName)

.ToArray();

return names;

}

**Update suppliers list:**

private void UpdateSuppliers(string country, string[] supplierNames)

{

var request = HttpContext.Current.Request;

var authority = request.Url.Authority;

var spAppWebUrl = request.Headers["SPAppWebUrl"];

var contextToken = request.Headers["SPContextToken"];

using (var clientContext = TokenHelper.GetClientContextWithContextToken(

spAppWebUrl, contextToken, authority))

{

var service = new PartSuppliersService(clientContext);

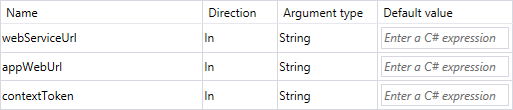
service.UpdateSuppliers(country, supplierNames);

}

}

If you look at the design view of the workflow.xaml file in the Approve Suppliers directory of the app project, you’ll see (by clicking on the **Arguments** tab at the bottom left of the design view) that the workflow stores the three values in the payload variable that has been passed to it as workflow arguments (Figure 3).

**Figure 3. Payload arguments passed to the workflow as workflow arguments**

[](https://camo.githubusercontent.com/b335b9014d96cbaff76fc38e881cf62b2e10d40e/687474703a2f2f692e696d6775722e636f6d2f797673516231592e706e67)

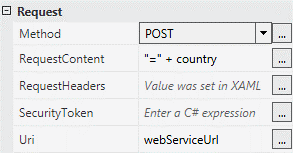
The **HttpSend** activity occurs before workflow approval. This activity sends the POST query to your remote web application that triggers the call to the Northwind web service and then the list item update (with the suppliers list). This activity is configured to send the request to the webServiceUrl value that was passed as a workflow argument (Figure 4).

**Figure 4. HttpSend activity webServiceUrl**

[https://camo.githubusercontent.com/aedfb1cd8ce2aa0cad1e0232a30f6edeaf160f6f/687474703a2f2f692e696d6775722e636f6d2f75434a3668454a2e706e67](https://camo.githubusercontent.com/aedfb1cd8ce2aa0cad1e0232a30f6edeaf160f6f/687474703a2f2f692e696d6775722e636f6d2f75434a3668454a2e706e67)

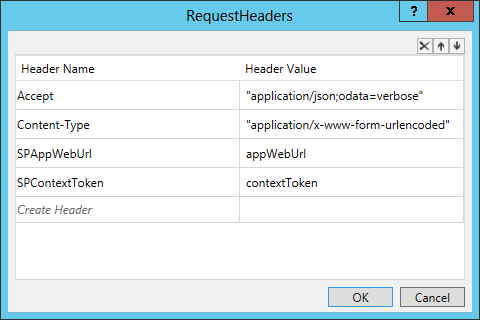
The POST request also passes the country value that is stored in the list item on which the workflow is operating (Figure 5).

**Figure 5. HttpSend activity properties**

[](https://camo.githubusercontent.com/b2b93d2e9d4fd9b652ffb02e51334e8203a752af/687474703a2f2f692e696d6775722e636f6d2f494741684147732e706e67)

The workflow sends the **appWebUrl** and **contextToken** values to the web application through the request headers (Figure 6). The headers also set the content types for sending and accepting requests.

**Figure 6. HttpSend activity request headers**

[](https://camo.githubusercontent.com/d0707cfafacb09fd5a937832a76c1f894530169b/687474703a2f2f692e696d6775722e636f6d2f5a70327062366e2e706e67)

If the workflow is approved, it changes the value of the **isApproved** field of the list item to **true**.

## [Call custom web service from a workflow and update SharePoint via the SharePoint web proxy](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallServiceUpdateSPViaProxy)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample demonstrates how to design a provider-hosted app so that it queries a web service and then passes that information to a SharePoint list via the SharePoint 2013 web proxy. | The design of this sample is useful when you want to encapsulate all of the interactions with a web service so that they are handled directly by the workflow. Using the web proxy makes it easier to update the remote web application logic without having to update the workflow instance. If you’re not using the proxy and you have to update the logic in your web application, you’ll have to remove the existing workflow instances and then redeploy the app. **For this reason, we recommend this design when you need to call a remote web service.** | The sample starts a workflow from a remote web application. This workflow passes query information submitted by the user to the Northwind OData web service. The query returns the product suppliers for a given country. After it receives the web service response, the workflow passes the information from the response to the remote web application. The remote web application then updates a product suppliers list that the app has deployed to the app web. |

**Related samples**:

[Workflow.AssociateToHostWeb](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.AssociateToHostWeb)

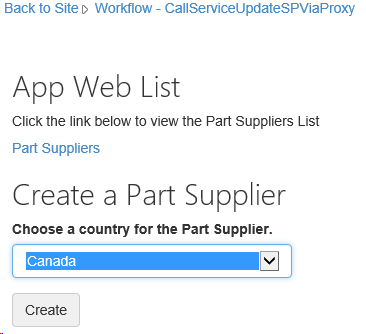
**Alternative approaches:**

[Workflow.CallCustomService](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallCustomService)

The [Workflow.CallCustomServiceUpdateViaProxy](https://github.com/OfficeDev/PnP/tree/master/Samples/Workflow.CallServiceUpdateSPViaProxy) project page contains instructions for deploying this app. You can also deploy and test with F5 debugging in Visual Studio if you follow the instructions in this [Debugging SharePoint 2013 workflows using Visual Studio 2013](http://blogs.msdn.com/b/officeapps/archive/2013/10/30/debugging-sharepoint-2013-workflows-using-visual-studio-2013.aspx) blog post.

This app’s launch page presents you with a drop-down menu that allows you to pick a country for which you want to create a product suppliers list (Figure 7).

**Figure 7. Launch page of custom web service and update with proxy workflow app**



That button calls a method in the Controllers\PartSuppliersController.cs file that creates a new entry in the Part Suppliers list on the app web. The **Create** method in that file calls the **Add** method that is defined in the Services\PartSuppliersService.cs file.

**Create:**

public ActionResult Create(string country, string spHostUrl)

{

var spContext = SharePointContextProvider.Current.GetSharePointContext(HttpContext);

using (var clientContext = spContext.CreateUserClientContextForSPAppWeb())

{

var service = new PartSuppliersService(clientContext);

var id = service.GetIdByCountry(country);

if (id == null)

{

id = service.Add(country);

TempData["Message"] = "Part Supplier Successfully Created!";

}

else

TempData["ErrorMessage"] = string.Format("Failed to Create The Part Supplier: There's already a Part Supplier who's country is {0}.", country);

return RedirectToAction("Details", new { id = id.Value, SPHostUrl = spHostUrl });

}

}

**Add:**

public int Add(string country)

{

var item = list.AddItem(new ListItemCreationInformation());

item["Country"] = country;

item.Update();

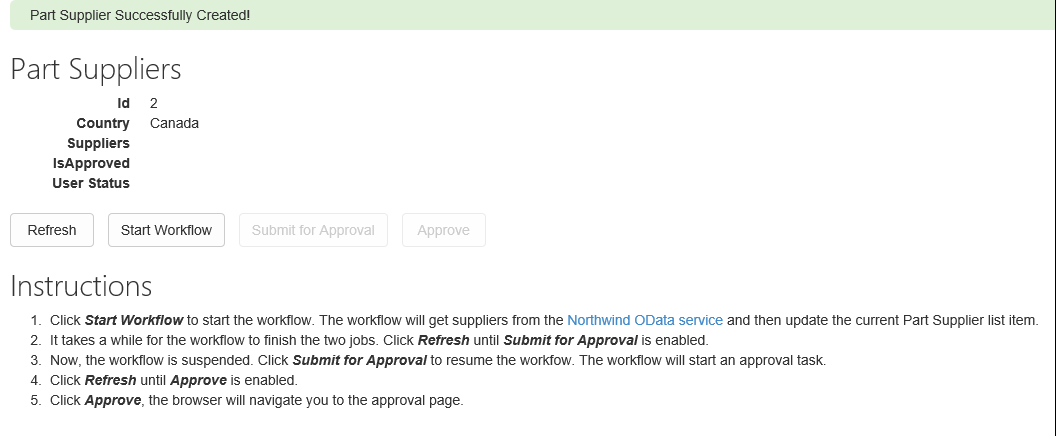
clientContext.ExecuteQuery();

return item.Id;

}

After creating that new list item, the app presents a button that starts the approval workflow (Figure 8).

**Figure 8. Start workflow button in the custom web service and update with proxy workflow app**



Clicking on the **Start Workflow** button triggers the **StartWorkflow** method in the Controllers\PartSuppliersController.cs file. This method packages the app web URL and the web service URL (for your remotely hosted web application, not for the Northwind web service) and passes them to the **StartWorkflow** method in the Services\PartSuppliersService.cs file. The workflow is going to communicate with the remote web application via the web proxy, and the web proxy will add the access token in a request header. This is why the workflow doesn’t pass a context token to the **StartWorkflow** method in this sample.

public ActionResult StartWorkflow(int id, Guid workflowSubscriptionId, string spHostUrl)

{

var spContext = SharePointContextProvider.Current.GetSharePointContext(HttpContext);

var webServiceUrl = Url.RouteUrl("DefaultApi", new { httproute = "", controller = "Data" }, Request.Url.Scheme);

var payload = new Dictionary<string, object>

{

{ "appWebUrl", spContext.SPAppWebUrl.ToString() },

{ "webServiceUrl", webServiceUrl }

};

using (var clientContext = spContext.CreateUserClientContextForSPAppWeb())

{

var service = new PartSuppliersService(clientContext);

service.StartWorkflow(workflowSubscriptionId, id, payload);

}

TempData["Message"] = "Workflow Successfully Started!";

return RedirectToAction("Details", new { id = id, SPHostUrl = spHostUrl });

}

The **StartWorkflow** method in the PartSuppliersService.cs file creates a workflow instance and passes the two values (**appWebUrl** ad **webServiceUrl**) stored in the payload variable to the workflow.

public void StartWorkflow(Guid subscriptionId, int itemId, Dictionary<string, object> payload)

{

var workflowServicesManager = new WorkflowServicesManager(clientContext, clientContext.Web);

var subscriptionService = workflowServicesManager.GetWorkflowSubscriptionService();

var subscription = subscriptionService.GetSubscription(subscriptionId);

var instanceService = workflowServicesManager.GetWorkflowInstanceService();

instanceService.StartWorkflowOnListItem(subscription, itemId, payload);

clientContext.ExecuteQuery();

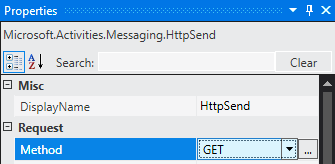
}

After the workflow starts, and before it is approved, the workflow makes a query to the Northwind web service to retrieve the suppliers list for the country that you’ve selected. It does this via an HTTP send activity that sends an OData query to this endpoint:

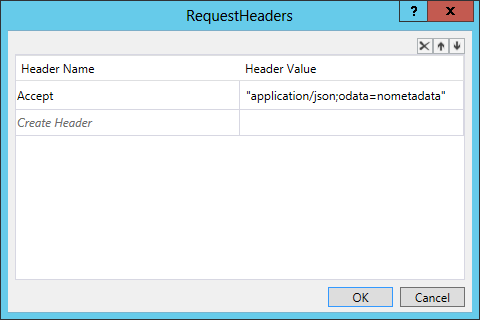
"http://services.odata.org/V3/Northwind/Northwind.svc/Suppliers/?$filter=Country eq '" + country.Replace("'", "''") + "'&$select=CompanyName"

The **HttpSend** activity should be configured as a GET request with an Accept header that specifies JSON with no metadata: application/json;odata=nometadata (Figures 9 and 10).

**Figure 9. HttpSend activity configured as GET request**

[](https://camo.githubusercontent.com/de81db877c7592dc93bb2515c776086bdd3727db/687474703a2f2f692e696d6775722e636f6d2f65784f36646e352e706e67)

**Figure 10. HttpSend activity RequestHeaders**

[](https://camo.githubusercontent.com/0f18684b95224e75d6b6203ee09fed128fca3403/687474703a2f2f692e696d6775722e636f6d2f55694a76674e4c2e706e67)

If the user has selected Canada for the new supplier list item, the JSON-formatted response will look like this:

{

value: [

{

CompanyName: "Ma Maison"

},

{

CompanyName: "Forêts d'érables"

}

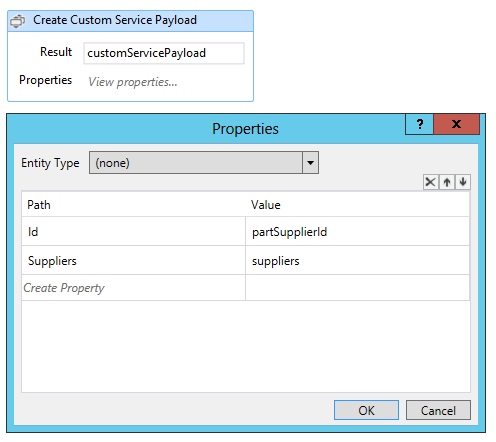
]

}

After the workflow starts, it makes a POST HTTP request containing the suppliers list to the remotely hosted web application via the SharePoint 2013 web proxy. It does this via an **HttpSend** activity that queries the web proxy URL: appWebUrl + "/\_api/SP.WebProxy.invoke".

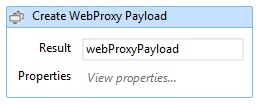
The workflow passes the supplier list that it has received from the Northwind service by building and passing a custom service payload. The Create Custom Service Payload activity properties contain the supplier list and the id for the supplier country (Figure 11).

**Figure 11. Create Custom Service Payload activity**

[](https://camo.githubusercontent.com/0417c6ab0cc0a9eb6848194895e558c02aeb930b/687474703a2f2f692e696d6775722e636f6d2f554b6c574b504f2e706e67)

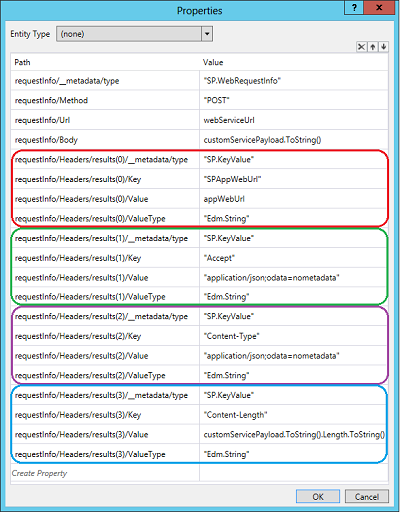
The Create WebProxy Payload activity constructs a payload that passes the contents of this payload to the web proxy URL (Figure 12).

**Figure 12. Create WebProxy Payload activity**

[](https://camo.githubusercontent.com/0d4a8bbdd0c1d0f2dac21115e2388a17bf9b9d96/687474703a2f2f692e696d6775722e636f6d2f6f775036444f552e706e67)

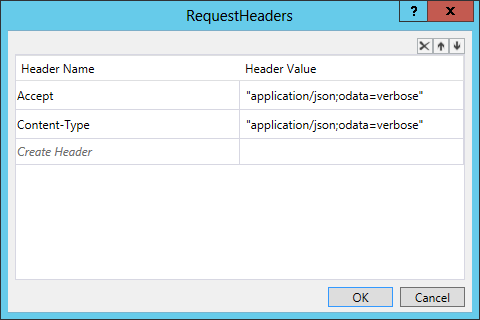
The properties for this activity specify the app web url, the POST request content length and type, and the request acceptance type via request headers (Figure 13).

**Figure 13. Create WebProxy Payload activity properties**



Once the workflow has constructed the payload and the request, it passes the request to the web proxy via an **HttpSend** activity that’s configured as a POST request to the web proxy URL. The request headers specify JSON-formatted OData for in the **Content-Type** and **Accept** headers (Figure 14).

**Figure 14. HttpSend activity properties for the web proxy request**

[](https://camo.githubusercontent.com/d656477c5c1f24bbb4310a48e482939bdb46fc8f/687474703a2f2f692e696d6775722e636f6d2f54776e374c644f2e706e67)

The **Post** method inside the Controllers\DataController.cs file accepts the contents of the request that the workflow sends through the web proxy. The **Post** method in the previous sample called a method for retrieving the supplier list from Northwind as well as one for updating the corresponding SharePoint supplier list. Since the workflow in this sample has already queried the Northwind service, this version of the method needs only to update the SharePoint list. It also passes the app web URL and the access token (which is passed by the web proxy) to the **UpdateSuppliers** method in the Services\PartSuppliersService.cs file.

public void Post(UpdatePartSupplierModel model)

{

var request = HttpContext.Current.Request;

var authority = request.Url.Authority;

var spAppWebUrl = request.Headers["SPAppWebUrl"];

var accessToken = request.Headers["X-SP-AccessToken"];

using (var clientContext = TokenHelper.GetClientContextWithContextToken(spAppWebUrl, accessToken, authority))

{

var service = new PartSuppliersService(clientContext);

service.UpdateSuppliers(model.Id, model.Suppliers.Select(s => s.CompanyName));

}

}

The **UpdateSuppliers** method in the PartSuppliers.cs file updates the Suppliers field of the newly created list item.

public void UpdateSuppliers(int id, IEnumerable<string> supplierNames)

{

var item = list.GetItemById(id);

clientContext.Load(item);

clientContext.ExecuteQuery();

string commaSeparatedList = String.Join(",", supplierNames);

item["Suppliers"] = commaSeparatedList;

item.Update();

clientContext.ExecuteQuery();

}

If the workflow is approved, it changes the value of the **isApproved** field of the list item to **true**.

## [Associating a workflow to the host web](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.AssociateToHostWeb)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample demonstrates how to deploy a workflow to the host web and associate it with a list on the host web. | The tools in Visual Studio 2013 for creating workflows make it easy to deploy workflows and the lists with which they are associated to the app web. The instructions for this sample show you how to create a workflow in Visual Studio, deploy it to the host web, and associate it with a list on the host web. You won’t need to refer to this sample after the Visual Studio tools are updated to make this work-around unnecessary. | The sample contains a simple workflow that can be associated with any list. The instructions for deploying this workflow show you how to work around the current limitations of the Visual Studio workflow tools by packaging the app, opening it up and editing a configuration file, and then repackaging it manually before deploying it to the host web. |

**Related samples**:

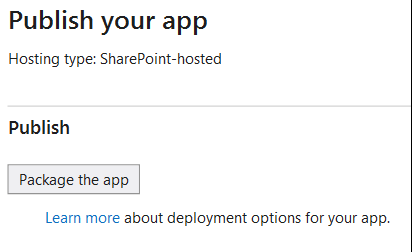
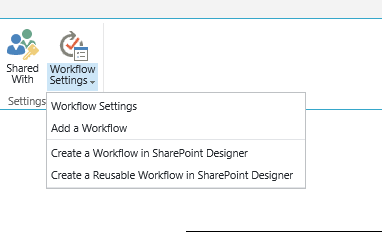
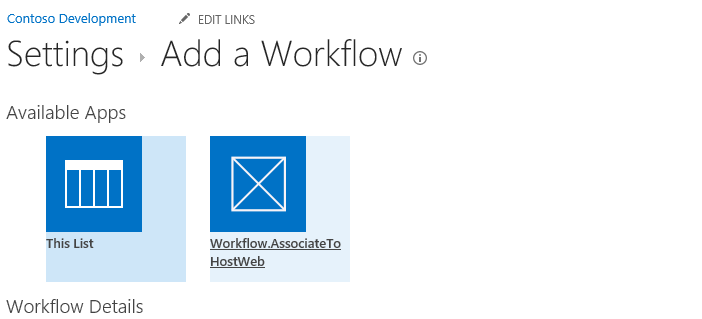
[Workflow.CallCustomService](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallCustomService)

[Workflow.CallServiceUpdateSPViaProxy](https://github.com/OfficeDev/PnP/tree/dev/Samples/Workflow.CallServiceUpdateSPViaProxy)

When you open this project in Visual Studio, you’ll see that it is a simple, generic workflow that is designed so that it can work with any SharePoint list. Other than the workflow task list, it doesn’t deploy any list with which it can be associated.

You can use this sample to learn how to deploy a SharePoint 2013 workflow to the host web and associate it with any list on the host web. This currently is not something that you can do directly in Visual Studio 2013. When the Visual Studio workflow tools are updated in the future, you’ll no longer need to use this work-around.

Follow these steps to deploy the workflow to the host web:

1. Right-click the **Workflow.AssociateToHostWeb** app project in the project explorer. Select Publish. You’ll see a window that contains a **Package the app** button (Figure 15).  
   **Figure 15. Publish your app window with Package the app button.**  
   
2. Select the Package the app button. Visual Studio will create a Workflow.AssociateToHostWeb.app file in the bin\Debug\app.publish\1.0.0.0 directory of your solution. This .app file is a type of zip file.
3. Change the extension of the Workflow.AssociateToHostWeb.app to .zip. Extract the contents.
4. Inside the extracted directory, open the WorkflowManifest.xml file, which will be empty. Add this XML to that file:  
   <SPIntegratedWorkflow xmlns="http://schemas.microsoft.com/sharepoint/2014/app/integratedworkflow">  
   <IntegratedApp>true</IntegratedApp>  
   </SPIntegratedWorkflow>
5. Save the WorkflowManifest.xml file. Select all of the files inside the extracted folder. Right-click the selection, and then select **Send to -> Compressed (zipped) folder**.
6. Reset the extension of your new .zip file to .app. You should now have a new Workflow.AssociateToHostWeb.app package that contains the updated WorkflowManifest.xml file.
7. Add the app to your app catalog.
8. Install the app to your host site.
9. Go to a list on your host site, and then select the **List** editing option at the top left of the page. You’ll see a **Workflow Settings** drop-down menu (Figure 16). Select **Add a Workflow** from the menu.  
   **Figure 16. Workflow settings for a list**  
   
10. You’ll see a selection option that will look something like Figure 17. Select the **Workflow.AssociateToHostWeb** app from the list of available options.  
    **Figure 17. Add a Workflow setting**  
    [](https://camo.githubusercontent.com/35c6dd7fb861479d306642024cf9f71bd1ca411b/687474703a2f2f692e696d6775722e636f6d2f74554144785a392e706e67)

Now you’ve deployed the workflow to the host web and associated it with a list on the host web. You can trigger a workflow manually, or you can update the workflow in Visual Studio so that it is triggered in other ways.