



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. |

# [Corporate events reference app](https://github.com/OfficeDev/PnP/tree/dev/Solutions/BusinessApps.CorporateEventsApp)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample app for SharePoint provides a composite reference implementation for creating and working with line of business (LoB) entities in the context of a provider-hosted app. | This sample shows you how to implement an MVC ASP.NET web application that interacts with SharePoint as a data store for LoB entities. It also shows you how to implement multiple pieces of a complex business task with a single provider-hosted app. | The app implements a centralized corporate events management system consisting of SharePoint entities (lists and content types). For each new content type, it creates corresponding LoB entities in an MVC ASP.NET web application. Components of the web application run as remotely hosted app parts within the SharePoint interface and also as pages running entirely on the remote web host. The app overrides the default welcome page for your SharePoint site so that it can present a custom interface on the site home page that allows users to interact with the app. |

**Related samples**:

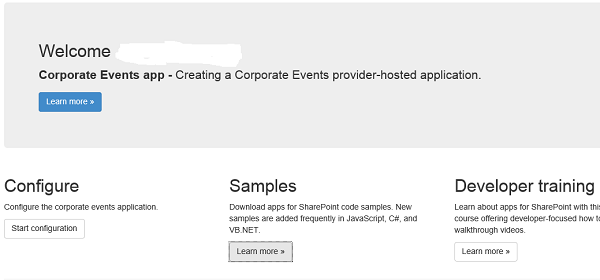
[Core.ModifyPages](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.ModifyPages)

[Provisioning.Pages](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Provisioning.Pages)

[Core.DevPnPCore](https://github.com/OfficeDev/PnP/tree/dev/OfficeDevPnP.Core)

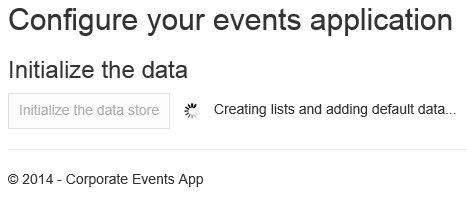
When you first launch the app, the app’s Home page welcomes you by your user name and presents you with a button for configuring the sample. It also points you to a number of resources for further investigation (Figure 1).

**Figure 1. Corporate events sample launch page**



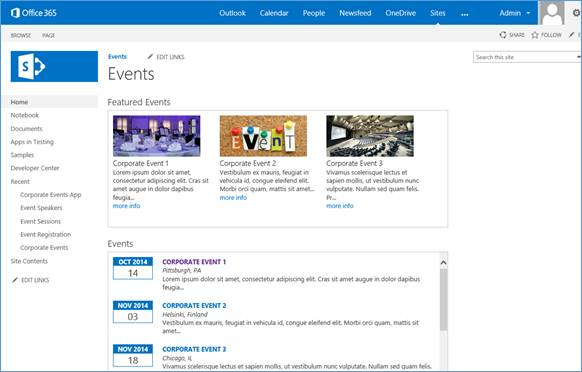
When you click on the **Start configuration** button at left, you go to the **Configuration** page, which contains the **Initialize the data store** button. When you click on the button, the sample deploys the SharePoint entities and sample data that support the sample (Figure 2).

**Figure 2. Corporate events sample deploys the SharePoint entities and data**



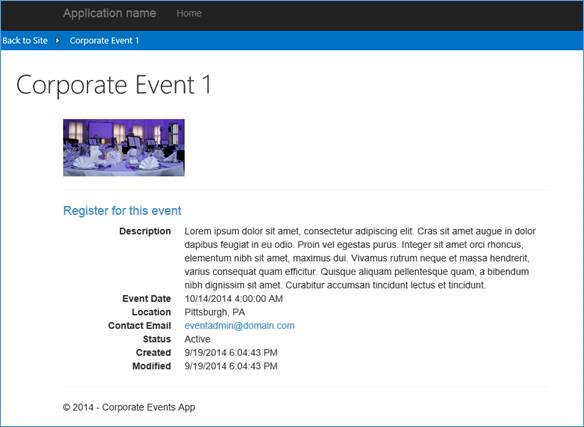
After you’ve initialized the data store, you can go back to your site to see a new welcome page (the EventsHome.aspx page), which is populated by two web parts that have been deployed by the app (Figure 3). In the left column, you’ll see the four new lists (Event Speakers, Event Sessions, Event Registration, and Corporate Events) that the app has installed. The Corporate Events list will be populated by sample data.

**Figure 3. Corporate events sample deploys new SharePoint welcome page**

[](https://camo.githubusercontent.com/cc83734131e94a4b85e1fce1ca4a72682b3a0e80/687474703a2f2f692e696d6775722e636f6d2f6e774842566e372e706e67)

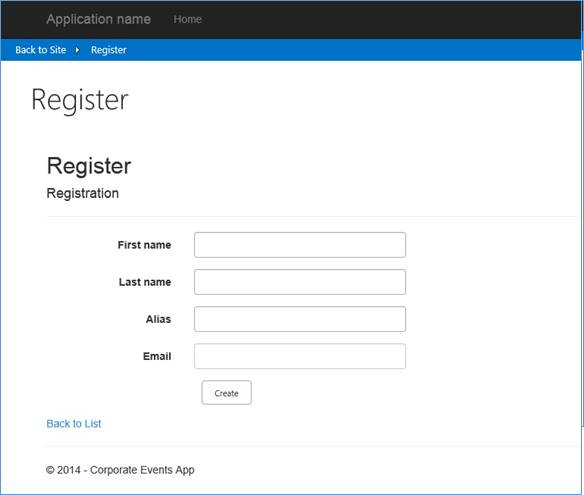
Each web part contains links to each of the displayed events, where you can see the event details. The event details page runs separately on the remote host (Figure 4). Its UI allows you to return to the SharePoint site, and also to register yourself for the event.

**Figure 4. Corporate events sample event details display**

[](https://camo.githubusercontent.com/5b9080f1a63c24f22382f6f515ea74ea51a883d4/687474703a2f2f692e696d6775722e636f6d2f5741784473434c2e706e67)

The registration page also runs separately on the remote host, and also contains a link back to the SharePoint host site (Figure 5). When you finish registering for the event, your name will appear on the newly installed **Event Registration** list.

**Figure 5. Corporate events sample event registration display**

[](https://camo.githubusercontent.com/62c905a1ee4d6078f9ce4d30e3b802a5de87a262/687474703a2f2f692e696d6775722e636f6d2f644d41615a62362e706e67)

The Models\DataInitializer.cs file contains the code that runs when you click on this button. The code in this file creates and deploys four new SharePoint lists, along with four corresponding content types:

* Corporate Events
* Event Registration
* Event Speakers
* Event Sessions

The code in this file uses a method similar to the one in the [Core.ModifyPages](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.ModifyPages) sample to add a custom page to the site.

// create default wiki page

web.AddWikiPage("Site Pages", "EventsHome.aspx");

**AddWikiPage** is an extension method from the [Core.DevPnPCore](https://github.com/OfficeDev/PnP/tree/dev/OfficeDevPnP.Core) project to add a new page to the site. This new page also becomes the new **WelcomePage** for the site. It also prepares to add the web parts to this page.

var welcomePage = "SitePages/EventsHome.aspx";

var serverRelativeUrl = UrlUtility.Combine(web.ServerRelativeUrl, welcomePage);

File webPartPage = web.GetFileByServerRelativeUrl(serverRelativeUrl);

if (webPartPage == null) {

return;

}

web.Context.Load(webPartPage);

web.Context.Load(webPartPage.ListItemAllFields);

web.Context.Load(web.RootFolder);

web.Context.ExecuteQuery();

web.RootFolder.WelcomePage = welcomePage;

web.RootFolder.Update();

web.Context.ExecuteQuery();

This file also defines the XML for both web parts that are displayed on the new welcome page and then adds each one to the page. The following snippets show how this works for the **Featured** Events web part.

**Define web part XML:**

var webPart1 = new WebPartEntity(){

WebPartXml = @"<webParts>

<webPart xmlns='http://schemas.microsoft.com/WebPart/v3'>

<metaData>

<type name='Microsoft.SharePoint.WebPartPages.ClientWebPart, Microsoft.SharePoint, Version=16.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c' />

<importErrorMessage>Cannot import this Web Part.</importErrorMessage>

</metaData>

<data>

<properties>

<property name='Description' type='string'>Displays featured events</property>

<property name='FeatureId' type='System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'>3a6d7f41-2de8-4e69-b4b4-0325bd56b32c</property>

<property name='Title' type='string'>Featured Events</property>

<property name='ProductWebId' type='System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'>12ae648f-27db-4a97-9c63-37155d3ace1e</property>

<property name='WebPartName' type='string'>FeaturedEvents</property>

<property name='ProductId' type='System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'>3a6d7f41-2de8-4e69-b4b4-0325bd56b32b</property>

<property name='ChromeState' type='chromestate'>Normal</property>

</properties>

</data>

</webPart>

</webParts>",

WebPartIndex = 0,

WebPartTitle = "Featured Events",

WebPartZone = "Rich Content"

};

**Add the web parts to the page:**

var limitedWebPartManager = webPartPage.GetLimitedWebPartManager(Microsoft.SharePoint.Client.WebParts.PersonalizationScope.Shared);

web.Context.Load(limitedWebPartManager.WebParts);

web.Context.ExecuteQuery();

for (var i = 0; i < limitedWebPartManager.WebParts.Count; i++) {

limitedWebPartManager.WebParts[i].DeleteWebPart();

}

web.Context.ExecuteQuery();

var oWebPartDefinition1 = limitedWebPartManager.ImportWebPart(webPart1.WebPartXml);

var oWebPartDefinition2 = limitedWebPartManager.ImportWebPart(webPart2.WebPartXml);

var wpdNew1 = limitedWebPartManager.AddWebPart(oWebPartDefinition1.WebPart, webPart1.WebPartZone, webPart1.WebPartIndex);

var wpdNew2 = limitedWebPartManager.AddWebPart(oWebPartDefinition2.WebPart, webPart2.WebPartZone, webPart2.WebPartIndex);

web.Context.Load(wpdNew1);

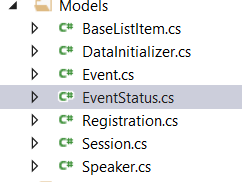
web.Context.Load(wpdNew2);

web.Context.ExecuteQuery();

In the Models directory of your web project (Figure 6), you’ll notice that this MVC ASP.NET web application contains four class names that correspond with the lists and content types that the app has installed to SharePoint: Event.cs (Corporate Events), Registration.cs (Event Registration), Session.cs (Event Sessions), and Speaker.cs (Event Speakers). These four classes and their corresponding SharePoint content types together constitute the four LoB entities used in this app.

The EventStatus.cs file defines an enum for setting the event status (Active, Cancelled, Expired). The BaseListItem.cs file defines the abstract class from which the four LoB classes inherit.

**Figure 6. MVC models correspond with the new SharePoint content types**



The DataInitializer.cs adds sample data for the **Corporate Events** list by creating sample **Event** objects that correspond with the **Corporate Events** content type and that the app adds to the **Corporate Events** list. When you register for an event, the app creates a **Registration** object that corresponds with the **Event Registration** content type and that the app adds to the **Event Registration** list. The sample has not yet fully implemented the Session and Speaker objects, so the app currently doesn’t work with those objects.

The following table shows which properties need to be implemented by the classes that inherit from the **BaseListItem** abstract class.

|  |  |
| --- | --- |
| **Member** | **Description** |
| ContentTypeName | Gets the content type that is associated with the item. If null, the default library content type will be assigned to the item when you save it. |
| FieldInternalNames | A list of field names that can be cached to improve performance when used for checking field data prior to save. |
| ListTitle | Gets the title of the list (case sensitive). |

The following table shows which methods need to be implemented by the classes that inherit from the **BaseListItem** abstract class. These methods return parameters should be set to [blittable types](http://msdn.microsoft.com/en-us/library/75dwhxf7(v=vs.110).aspx) so that they can be used on multiple platforms.

|  |  |
| --- | --- |
| **Method** | **Description** |
| ReadProperties(ListItem) | Reads properties from the **ListItem** object using the **BaseGet** and **BaseGetEnum** methods and assigns them to properties of the subclass. |
| SetProperties(ListItem) | Sets properties on the **ListItem** object using the **BaseSet** and **BaseSetTaxonomyField** methods of the abstract class. |

The following table shows the helper methods from the **BaseListItemClass** that the subclasses need to use in order to implement the **ReadProperties** and **SetProperties** methods.

|  |  |
| --- | --- |
| **Helper Method** | **Description** |
| BaseGet(ListItem item, string internalName) | Gets the property defined by the **internalName** parameter from the **ListItem** and returns them of generic type **T**. |
| BaseSet(ListItem item, string internalName, object value) | Sets the **ListItem** property defined by the **internalName** parameter. |
| BaseSetTaxonomyField(ListItem item, string internalName, string label, Guid termId) | Sets the **ListItem** taxonomy field defined by the **internalName** and **termId** parameters. |
| BaseGetEnum(ListItem item, string internalName, T defaultValue) | Gets the value of the enum property defined by the **internalName** parameter. Returns the value of the **defaultValue** parameter if the property is not set. |

The Event.cs file contains these implementations of the **ReadProperties** and **SetProperties** methods.

**ReadProperties:**

protected override void ReadProperties(ListItem item) {

RegisteredEventId = BaseGet<string>(item, FIELD\_REGISTERED\_EVENT\_ID);

Description = BaseGet<string>(item, FIELD\_DESCRIPTION);

Category = BaseGet<string>(item, FIELD\_CATEGORY);

EventDate = BaseGet<DateTime?>(item, FIELD\_DATE);

Location = BaseGet<string>(item, FIELD\_LOCATION);

ContactEmail = BaseGet<string>(item, FIELD\_CONTACT\_EMAIL);

Status = BaseGetEnum<EventStatus>(item, FIELD\_STATUS);

var imageUrl = BaseGet<FieldUrlValue>(item, FIELD\_IMAGE\_URL);

if (imageUrl != null)

ImageUrl = imageUrl.Url;

}

SetProperties:

protected override void SetProperties(ListItem item) {

BaseSet(item, FIELD\_REGISTERED\_EVENT\_ID, RegisteredEventId);

BaseSet(item, FIELD\_DESCRIPTION, Description);

BaseSet(item, FIELD\_CATEGORY, Category);

BaseSet(item, FIELD\_DATE, EventDate);

BaseSet(item, FIELD\_LOCATION, Location);

BaseSet(item, FIELD\_CONTACT\_EMAIL, ContactEmail);

BaseSet(item, FIELD\_STATUS, Status.ToEnumDescription());

BaseSet(item, FIELD\_IMAGE\_URL, ImageUrl);

}

This is how the underlying BaseGet and BaseSet methods are defined in BaseListItem.cs.

**BaseGet:**

protected T BaseGet<T>(ListItem item, string internalName){

var field = \_fields[internalName.ToLowerInvariant()];

var value = item[field.InternalName];

return (T)value;

}

**BaseSet:**

protected void BaseSet(ListItem item, string internalName, object value) {

if (\_fields.ContainsKey(internalName)) {

var field = \_fields[internalName.ToLowerInvariant()];

if (field is FieldUrl && value is string) {

var urlValue = new FieldUrlValue() {

Url = value.ToString()

};

value = urlValue;

}

}

item[internalName] = value;

}

The **BaseListItem** class also contains a **Save** method that is used to save each LoB entity that the app creates and manipulates. This method loads the list and determines if the current item has an ID that is greater than 0. If the ID is not greater than 0, it assumes that it’s not valid and creates a new list item. It uses the **SetProperties** method to set properties on the **ListItem** and then sets the properties on the subclass by using the **ReadProperties** method.

public void Save(Web web) {

var context = web.Context;

var list = web.GetListByTitle(ListTitle);

if (!IsNew && Id > 0) {

ListItem = list.GetItemById(Id);

}

else {

var listItemCreationInfo = new ListItemCreationInformation();

ListItem = list.AddItem(listItemCreationInfo);

}

// ensure that the fields have been loaded

EnsureFieldsRetrieved(ListItem);

// set the properties on the list item

SetProperties(ListItem);

BaseSet(ListItem, TITLE, Title);

// use if you want to override the created/modified date

//BaseSet(ListItem, CREATED, Created);

//BaseSet(ListItem, MODIFIED, Modified);

ListItem.Update();

if (!string.IsNullOrEmpty(ContentTypeName)) {

var contentType = list.GetContentTypeByName(ContentTypeName);

if (contentType != null)

BaseSet(ListItem, "ContentTypeId", contentType.Id.StringValue);

}

ListItem.Update();

// Execute the batch

context.ExecuteQuery();

// reload the properties

ListItem.RefreshLoad();

UpdateBaseProperties(ListItem);

ReadProperties(ListItem);

}