Preparing SharePoint solutions for migration to apps for SharePoint

Preparing SharePoint solutions for migration to apps for SharePoint

Microsoft Corporation
June 2013

Applies to: SharePoint 2010 | SharePoint 2013

Summary: Using information in this white paper, SharePoint developers can easily transform new or existing SharePoint solutions into apps for SharePoint in SharePoint 2013. IT professionals also can use this information to host, scale, and manage solutions and apps more seamlessly.

Contents

Executive summary

Microsoft SharePoint 2013 introduces the power, flexibility, and scalability of a multi-tier app model. It includes new client and server APIs and easy integration into cloud computing models from on-premises deployments. A broader range of developer skill sets now can be used to develop SharePoint extensions, and new deployment options also are available. Branching out from the solutions in SharePoint 2010, SharePoint 2013 has both *apps* and *classic solutions*. Both are collectively called *extensions* in SharePoint 2013. Apps are units of specific functionality that extend the capabilities of SharePoint to fulfill business or user requirements; they are lightweight and easy to use. Classic solutions are the same as the solutions (both farm and sandboxed) in earlier versions of SharePoint.

SharePoint 2013 has introduced many architectural changes to improve existing SharePoint 2010 features. Although these changes provide benefits for developers building extensions on SharePoint 2013, they may pose challenges for developers looking to transform an existing SharePoint 2010 solution into an app for SharePoint. Thus, to retain investments in code written in SharePoint 2010, developers should build SharePoint 2010 solutions in a way that allows them to be quickly and easily converted into apps in SharePoint 2013 when required.

Who should read this paper?

This white paper is intended for SharePoint developers who want to build SharePoint 2010 classic solutions in a way that allows them to be easily transformed and deployed as apps in SharePoint 2013. The paper also can benefit IT professionals looking for simpler ways to host, scale, and manage apps and solutions. Topics covered include:

- Understanding the SharePoint 2013 architecture and app model.
- Building extensions on SharePoint 2010 with scenarios and considerations for SharePoint 2013.

Why read this paper?

This paper provides guidance for building or updating solutions in SharePoint 2010 so that they can be easily converted into apps for SharePoint. It also includes examples with sample code for building extensions compatible with SharePoint 2013.

With SharePoint 2013, developers can transform SharePoint 2010 solutions (classic solutions) into apps for SharePoint based on the new app model. This ability can help address certain challenges with previous versions of SharePoint, such as:

- Instability of farm environment: The SharePoint farm environment risked becoming unstable due to running custom code or custom operations.
- Scalability of solutions: To scale a solution in previous versions of SharePoint, developers had to consider multiple factors related to the environment, including downtime, cost, and high availability/disaster recovery (HADR) changes.

SharePoint 2013 provides developers and IT professionals with new features and other improvements to overcome these challenges. With enhancements such as the app model, developers can build platform-independent code and use their existing skills and tools to write code and apps without an in-depth knowledge of SharePoint 2013. The SharePoint 2013 app model provides multiple hosting options outside of the SharePoint environment, with the code being executed on either the client side or another non-SharePoint server in the cloud. In addition, IT professionals can easily scale apps without drastically affecting or expanding the SharePoint environment. Finally, with the SharePoint 2013 app model, developers have multiple options for integrating their apps, such as through services or by developing customized solutions.

Introduction

Apps for SharePoint are self-contained pieces of functionality that extend the capabilities of a SharePoint website. Apps are lightweight and easy to use, and they can be targeted to solve specific user needs. Apps can include SharePoint components such as lists, workflows, and site pages; they also can support the surfacing of remote web applications and remote data. You can easily install and uninstall apps because they have fewer or no dependencies on other software, apart from the platform on which they are deployed. In addition, it is easy to deliver and share your apps through new delivery models in SharePoint 2013, such as the Office Store and the app catalog. Apps in SharePoint 2013 have some distinct benefits over classic solutions, including the ability to:

- Maximize flexibility when building future upgrades.
- Maximize the use of existing non-SharePoint development skill sets.
- Integrate cloud-based resources in smoother and more flexible ways.
- Use different authentication schemes for the apps than for the users who are running the apps.
- Use cross-platform standards such as HTML, Representational State Transfer (REST), JavaScript, and OAuth.
- Use the SharePoint cross-domain JavaScript library to access SharePoint data.

Application migration requires modifying the code base of an application so that the functionalities provided by the APIs of the existing system are replicated in the new target environment. Solutions built on SharePoint 2010 can be easily migrated to SharePoint 2013 without major changes in application code. Apps in SharePoint 2013 support client object model (CSOM) and REST APIs, but not the SharePoint 2010 server object model. By developing SharePoint 2010 solutions using CSOM and REST APIs, you can eliminate the requirement of rewriting compatible code for running the solutions as apps for SharePoint.

There are two basic approaches to migrating applications from one platform to another:

- All-at-once: This approach, sometimes called the *big bang approach*, is where you rewrite your entire application all at once to the new platform. This has the benefit of cost

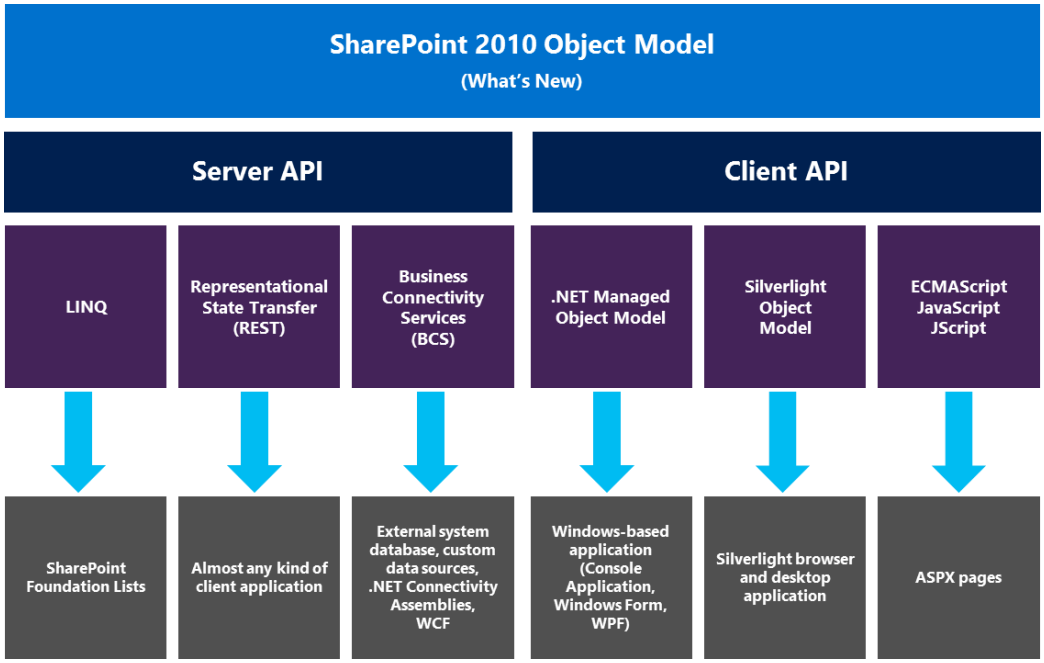
savings, but introduces risk because it requires extensive regression testing to make sure that no business requirements are lost in the migration effort. This is a good approach when you have comprehensive knowledge of the requirements of the application and can afford the risk of downtime if the deployment does not go as planned.

- Iterative: This approach migrates an application in multiple phases. Here you migrate the application slowly by rewriting parts of it over time. This approach reduces the overall risk of a catastrophic failure because it is simpler to test and manage the smaller pieces. In preparing SharePoint solutions for migration to apps, this approach also facilitates loosely coupling a SharePoint solution for risk-averse migration. The downside to this approach is that it typically costs more because you write code and spend resources specifically for the migration process. This approach is best for when an application is not comprehensively known or cannot afford much downtime.

SharePoint 2010 development model

In SharePoint 2010, you can use a number of object models to access a server. Managed client object models are based on server object models. Using the server object model in SharePoint 2010, you can write programs to access SharePoint lists and document libraries, site collections, sites, and other resources (Figure 1).

Figure 1. Overview of the SharePoint 2010 object model



Client object models

The client object models use the same or similar programming concepts as the server object model in SharePoint 2010. The CSOMs can be accessed through .NET code, Microsoft Silverlight code, or JavaScript. The new CSOMs share structural design traits, such as object

model hierarchy, object identity, data retrieval semantics, client context, infrastructural client objects, collections, and exception handling.

The CSOMs are consistent with the Microsoft SharePoint Foundation server object model, so if you are familiar with the server API, you can learn to use the client .NET-managed, Silverlight, or JavaScript API. Although the CSOMs do not have one-to-one parity with the server object model, they generally have parity with each other. So, when you learn one subset of the client object model, you also have learned most of the other subsets. Whenever possible, the new object models borrow asynchronous paradigms from the .NET Framework—in particular, ADO.NET.

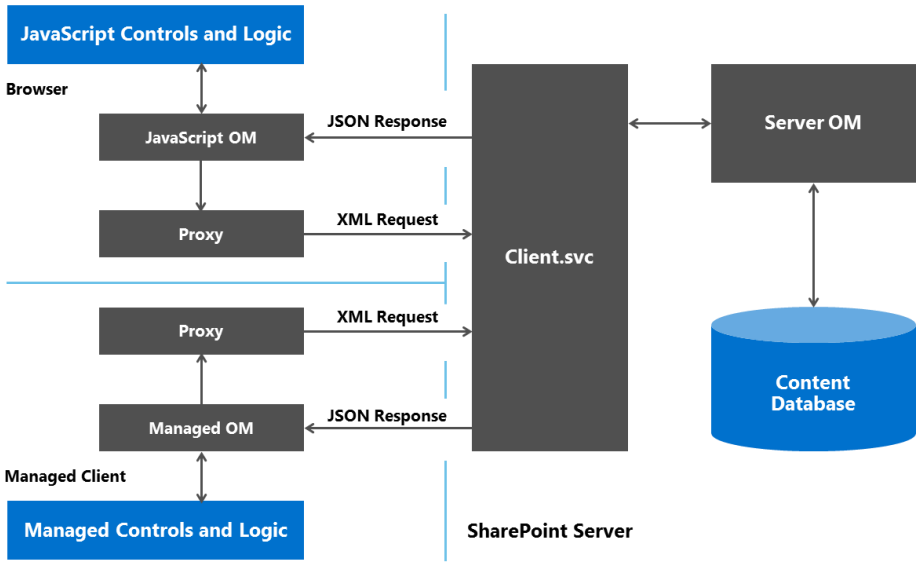
Silverlight client object model

SharePoint 2010 supports the implementation of the Silverlight client object model in two contexts: within a Silverlight Web Part and within the Silverlight Cross-Domain Data Access system (through which a Silverlight application interacts with SharePoint Foundation 2010 data). A third possibility—modifying client access cross-domain policy on the server—opens security risks and is not supported in SharePoint Foundation 2010.

ECMAScript (JavaScript) client object model

The JavaScript client object model lets you work with objects without deploying the code on the server (Figure 2). Being able to work with objects on the client side is useful when deploying a sandboxed solution or working with the SharePoint Server ribbon. Due to the nature of JavaScript, all code is executed asynchronously and relies on callback functions to work with the objects inside SharePoint Foundation 2010. JavaScript can work only with objects in the current context. That is, there is no ability to do cross-site scripting or access objects outside of the current context.

Figure 2. JavaScript client object model mechanics



WCF Data Services

SharePoint 2010 introduces a new REST-based web service using Windows Communication Foundation (WCF) Data Services. Using the WCF entry point, applications can reach across the network to query and update items in a SharePoint list. Although the LINQ to SharePoint provider introduces a new approach for writing the code to access SharePoint list data, it can be used only by the code that actually runs on the front-end web server. When your code is running from across the network (a desktop application, for example), you can achieve many of the same benefits of the LINQ to SharePoint provider by using SharePoint 2010 support for REST-based web services that access SharePoint list items.

SharePoint 2013 development model

SharePoint 2013 is a versatile development platform for building apps and solutions with varying scopes to address diverse business and user needs. SharePoint 2013 introduces the *Cloud App Model* for creating apps. Apps are self-contained pieces of functionality that extend the capabilities of a SharePoint website. An app may include SharePoint components such as lists, workflows, and site pages, but it also can surface remote web applications and remote data in SharePoint.

An app has few or no dependencies on other software on the device or platform where it is installed, other than what is built into the platform. This characteristic enables apps to be installed simply and uninstalled cleanly. Apps have no custom code that runs on the SharePoint servers. Instead, all custom logic moves "up" to the cloud or "down" to the client computers. In other words, apps for SharePoint are platform independent and can be stored and executed from any other non-SharePoint location in the cloud or any other server.

In addition, SharePoint 2013 introduces an innovative delivery model for apps for SharePoint that includes components like the SharePoint Store and the app catalog.

Cloud App Model

The Cloud App Model in SharePoint 2013 supports a tiered architecture in which the business logic, data, and user interface (UI) of the app can be distributed into separate components. You can take advantage of tools that are designed specifically for the development of a particular tier instead of using general-purpose tools. For example, you can have an app whose presentation logic is in HTML and JavaScript running on a client whose business logic is in Microsoft .NET running in Windows Azure and whose data is stored in a Windows Azure SQL Database. Or you can have an app that is written in PHP, has its data stored in a SharePoint list, and runs on a remote host accessing list data. There are many other options in this flexible model.

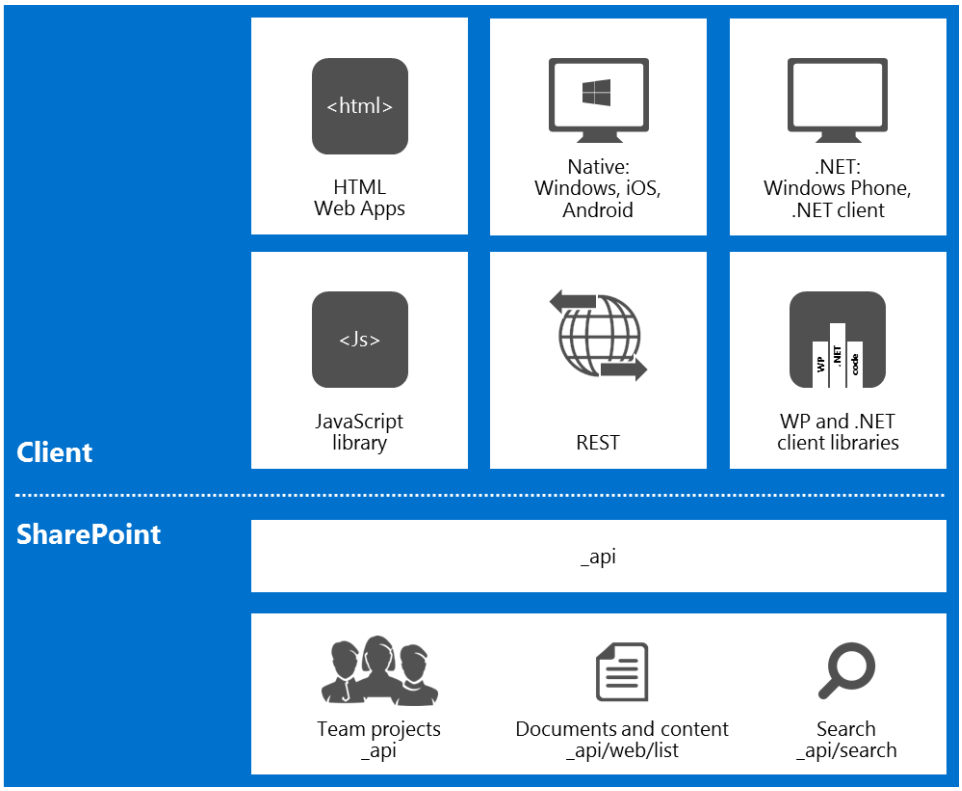
You can connect an app with almost any internal or public web service, take advantage of the new OAuth 2.0 support in SharePoint, and use the REST and client APIs (JavaScript and .NET)

to integrate and connect the app with SharePoint. SharePoint 2013 also provides a wide variety of features—such as search, workflow, social networking, taxonomy, user profiles, and Business Connectivity Services (BCS)—to make the apps even more robust.

Enhancements in SharePoint 2013

SharePoint 2013 supports a new cloud-based architecture and app-driven development framework. From lower-level SharePoint APIs to connectivity to social media integration, SharePoint 2013 is designed and executed to support a rich application development experience (Figure 3). In addition to the use of REST endpoints for web services, there is a broad API for both server and client development. Remote event receivers are now supported, in addition to client-side rendering.

Figure 3. Robust application development in SharePoint 2013



REST service

SharePoint 2013 introduces a Representational State Transfer service that is similar to the way SharePoint client object models work with the SharePoint server. You can interact remotely with SharePoint data by using any technology that supports HTTP web requests. This means that you can perform Create, Read, Update, and Delete (CRUD) operations from your apps, solutions, and client applications using standard REST web technologies.

SharePoint 2013 client object model

Custom logic in apps for SharePoint is always distributed "down" to the client or "up" to the cloud (or "over" to a server outside the SharePoint farm). In all of these distribution models, one of the CSOMs or REST/OData endpoints must be used. SharePoint 2013 has three CSOMs for managed code: .NET, Silverlight, and mobile. The SharePoint CSOMs enable a client application to make batch requests to a front-end web server running SharePoint products and technologies to perform operations on the core platform.

Client-side rendering

Client-side rendering provides a mechanism that you can use to produce your own output for a set of controls hosted on a SharePoint page. This mechanism lets you use well-known technologies, such as HTML and JavaScript, to define the rendering logic of custom field types. In client-side rendering, you can specify your own JavaScript resources and host them in the data storage options available to the farm solution, such as the **_layouts** folder or document library.

Compatibility modes

To minimize the impact on users of upgrading to a new version of the software, SharePoint 2013 supports new site collection provisioning modes. This means users can choose which version of SharePoint a site collection should be provisioned in when it is created.

Note: The scope of this paper is to show migration options from SharePoint 2010 solutions to apps for SharePoint 2013.

For more information on migrating your SharePoint 2010 environment to SharePoint 2013, download the IT Professional Reviewer's Guide (http://download.microsoft.com/download/0/9/D/09D05E53-08B6-44CE-B658-DCA670D5E2BB/SharePoint%20Server%202013%20Preview%20IT%20Pro%20Reviewer%27s%20Guide.pdf).

Developing extensions: Key considerations for SharePoint 2013

Whether using full-trust or sandboxed solutions, you should consider the following guidance when designing SharePoint 2010 extensions that will migrate more readily to apps for SharePoint:

- Think of SharePoint as a service provider as opposed to using the SharePoint runtime directly.
- The client object models are extended in SharePoint 2013. CSOMs now support search, taxonomy, and workflow. Therefore, you have to make minor changes to the CSOM code when transforming extensions to SharePoint 2013.

- You may have to add code for authentication because additional authentication and authorization mechanisms are supported in SharePoint 2013.
- Separate out the dependencies in your SharePoint solution and abstract them as REST services. (For example, rather than consuming the list APIs directly for general storage, use a REST service that abstracts the data source to provide flexibility in choosing local or cloud-based storage solutions in the future.)
- Change the application security model to be loosely coupled from the SharePoint security model.

Ensuring readiness: Approaches for migration from SharePoint solutions to apps for SharePoint

SharePoint provides multiple ways to develop applications for the SharePoint platform. One of the most significant additions, specifically to SharePoint 2010, was enhanced client-side functionality. SharePoint 2013 builds on these models for application development, which can facilitate a smoother transition from solutions to apps.

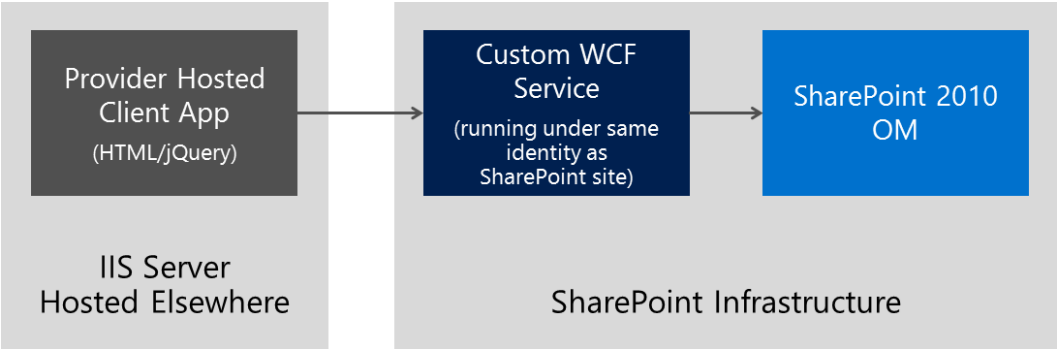
Client object models: CSOMs provide client-side applications with access to a subset of the SharePoint server object model, including core objects such as site collections, sites, lists, and list items. You can design client applications to access SharePoint content without installing code on the server. The CSOMs consist of multiple APIs, including the ECMAScript (JavaScript), .NET-managed, and Silverlight client object models.

- The ECMAScript and Silverlight client object models provide a smaller subset of functionality. The focus is to enhance the user experience, as these object models minimize the time it takes Silverlight applications or JavaScript functions running in a webpage to load the files required for operation.
- The .NET-managed client object model provides a larger subset of functionality for stand-alone client applications.

SharePoint 2010 CSOM code can run as-is on apps in SharePoint 2013. However, if you are using authentication mechanisms (claims-based, SAML, or forms-based) in a SharePoint 2010 classic solution, you may need to add code in your migrated app to suit the SharePoint 2013 authentication mechanisms.

Custom REST WCF Service: Using this approach, you can create a REST URI scheme on SharePoint 2010 that is similar to the one available on SharePoint 2013 (Figure 4). This generalizes the code to call SharePoint services and helps to save effort while transforming your SharePoint solution to an app for SharePoint.

Figure 4. Creating a custom REST URI scheme on SharePoint 2010



To create a REST URI scheme:

1. Create a web application using a preferred web technology, such as ASP.NET or PHP.
2. Create a service and host it within Microsoft SharePoint Server. (The service contains the server-side code that communicates with SharePoint. It has REST endpoints that can be consumed by any application.)
3. Use the URL scheme http://[DOMAIN]/_api/SERVICENAME for the REST endpoints. (This makes converting to SharePoint 2013 easier because its REST service uses a similar URL scheme.)

The rest of this guidance discusses using custom WCF REST endpoints and CSOM for refactoring your SharePoint solution to be as loosely coupled as possible. This will allow you to more seamlessly migrate your SharePoint solution to an app for SharePoint at a later time. The guidance is offered through a series of common scenarios for key workloads. The scenarios include sample code for building extensions compatible with SharePoint 2013. The workloads discussed are List, Web, Site, Search, Social, Workflow, and Content Management.

Preparing the environment and infrastructure

To help ensure a more seamless refactoring of your SharePoint solution, the environment and infrastructure must be properly set up. This section provides guidance for setting up the custom REST API web application environment as well as the CSOM environment.

To use the custom REST API and set up the web application environment

1. Create a web application using a preferred web technology, such as ASP.NET.
2. Create a WCF service and host it within SharePoint Server. (The service contains the server-side code that communicates with SharePoint. It has REST endpoints that can be consumed by any application.)

3. Add the Operation Contracts for WCF Service:
   - Create an interface and decorate it with the **ServiceContract** attribute.
   - Decorate methods in the interface with the **OperationContract** attribute. For example:

```
[ServiceContract()]
public interface <InterfaceName>
{
    [OperationContract]
    [WebInvoke(Method = "POST", BodyStyle = WebMessageBodyStyle.Wrapped,
RequestFormat = WebMessageFormat.Json, ResponseFormat =
WebMessageFormat.Json, UriTemplate = "/_api/web/lists/<functionname>")]
    Returntype FunctionName(Parameters);
}
```

4. Add the Data Contract for WCF Service:
   - Create a class and apply the **DataContract** attribute to it.
   - Define the members (properties, fields, or events), and apply the DataMember (http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datamemberattribute.aspx) attribute to each of them. For example:

```
[DataContract]
    public class <ClassName>
    {
        [DataMember(Name = "Title")]
        Public string Title;
}
```

5. Use the URL scheme http://[DOMAIN]/_api/SERVICENAME for the REST endpoints. (This makes converting to SharePoint 2013 easier because its REST service uses a similar URL scheme.)
6. Use the *System.ServiceModel.Activation.WebServiceHostFactory* in the .svc file.

Note: For more details on creating the REST API using WCF, see the WCF REST Programming Model Overview (http://msdn.microsoft.com/en-us/library/bb412172(v=vs.90).aspx).

To set up the CSOM for a .NET Web application

1. Create a new web project in Microsoft Visual Studio.
2. Add the following DLLs for CSOM for .NET in the application:
   - Microsoft.SharePoint.Client.Runtime.dll

- Microsoft.SharePoint.Client.dll

  Note: These DLLs are located in <SharePoint installation drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI.

To set up the JavaScript CSOM for a .NET Web application

1. To use the JavaScript CSOM, create a new web project in your development environment.
2. Add an entry like this in the Web Part ASCX control:

```
<SharePoint:ScriptLink Name="SP.js" runat="server" OnDemand="true"
Localizable="false" />
```

3. To ensure the JavaScript code runs after sp.js finishes loading, add the following code:

```
ExecuteOrDelayUntilScriptLoaded([YourJavaScriptFunctionName], "sp.js");
```

Note: For more information about implementing the managed client object model, see Implementing the Client-Side Object Model (http://msdn.microsoft.com/en-IN/library/ff521585(v=office.14).aspx).

For more information about using JavaScript to interact with SharePoint, see How to: Enable ECMA Client Object Model IntelliSense in Visual Studio 2010. (http://msdn.microsoft.com/en-us/library/ff798328.aspx).

List

Scenario: Create a webpage that displays the list of active tickets in a help desk application.

Using the custom REST API

This section discusses how to perform CRUD operations on a SharePoint list using a REST-based WCF service. The service methods have the server object model code to access SharePoint APIs. The URLs for hosting the WCF service on SharePoint are structured similarly to the calls for the SharePoint 2013 REST API (http://<site url>/_api/web/lists). This helps to ensure a smooth migration to SharePoint 2013.

In this scenario, IT personnel use the help desk application to get a list of all active service requests that need to be addressed and to add new service requests.

Task 1 – Create and implement the interface in the WCF REST service

Create the Operation Contracts and Data Contract for CRUD operations on the SharePoint list in the WCF REST service interface class (IContosoService.cs), as discussed in the Using the custom REST API and setting up the web application environment section.

1. Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
[WebGet(UriTemplate = "/_api/web/lists/getbytitle('{listName}')/items",
        RequestFormat = WebMessageFormat.Json,
        ResponseFormat = WebMessageFormat.Json)]
    ListResponseWrapper GetByTitle(string listName);

[OperationContract]
[WebInvoke(Method = "POST", UriTemplate = "/_api/web/lists/getbytitle('{ListName}')/additem",
        ResponseFormat = WebMessageFormat.Json,
        RequestFormat = WebMessageFormat.Json,
        BodyStyle = WebMessageBodyStyle.WrappedRequest
        )]
    string AddListItem(string ListName, string Title, string Description);
```

2. Add the following Data Contracts in the interface:

```
[DataContract]
  public class ListResponseWrapper
  {
    [DataMember]
    public ListResponse d;
  }

  [DataContract]
  public class ListResponse
  {
    [DataMember]
    public List<ListResult> results;
  }

  [DataContract]
  public class ListResponseMetaData
  {
    [DataMember]
    public int id { get; set; }
```

```
    [DataMember]
    public string type { get; set; }

    [DataMember]
    public Uri uri { get; set; }
}

[DataContract]
public class ListResult
{
    [DataMember(Name = "__metadata", Order = 1)]
    public ListResponseMetaData metadata;

    [DataMember(Order = 2)]
    public string Title;

    [DataMember(Order = 3)]
    public DateTime created;

    [DataMember(Order = 4)]
    public string Description;
}
```

Task 2 – Create the service methods

- Implement the interface in the **ContosoService.cs** file, and add the methods to perform the following operations:
  - Get active tickets from **HelpDeskList**
  - Add a Ticket

```
public ListResponseWrapper GetByTitle(string listName)
      {
         var listResults = new List<ListResult>();
         SPSecurity.RunWithElevatedPrivileges(delegate()
         {
            using (SPSite site = new SPSite(sharepointSiteUrl))
            {
               SPWeb web = site.OpenWeb();
               web.AllowUnsafeUpdates = true;
               SPList listByTitle = web.Lists[listName];
```

```
            foreach (SPListItem item in listByTitle.Items)
            {
                listResults.Add(new ListResult
                {
                    metadata = new ListResponseMetaData()
                    {
                        id = item.ID,
                        type = item.GetType().FullName
                    },
                    Title = item.Title,
                    created = DateTime.Parse(item["Created"].ToString()),
                    Description = item["Description"].ToString()
                });
            }
            web.AllowUnsafeUpdates = false;
        }
    });
    var response = new ListResponseWrapper()
    {
        d = new ListResponse()
        {
            results = listResults
        }
    };
    return response;
}

public string AddListItem(string ListName, string Title, string Description)
{
    SPSecurity.RunWithElevatedPrivileges(delegate()
    {
        using (SPSite site = new SPSite(sharepointSiteUrl))
        {
            using (SPWeb web = site.OpenWeb())
            {
                SPList ticketList = web.Lists.TryGetList(ListName);
                web.AllowUnsafeUpdates = true;
                if (ticketList != null)
                {
                    SPListItem ticket = ticketList.Items.Add();
                    ticket["Title"] = Title;
                    ticket["Description"] = Description;
                    ticket.Update();
                }
```

```
                    web.AllowUnsafeUpdates = false;
                }
            }
        });
        return "Success Message";
    }
```

Task 3 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, add the following generic **Get Response** method. This calls the WCF service APIs.

```
private string GetResponse(string requestUrl, string requestMethod,     string requestBod
y = "")
    {
        WebRequest request = HttpWebRequest.Create(requestUrl);
        request.Method = requestMethod;

        Stream dataStream = null;
        if (!string.IsNullOrEmpty(requestBody))
        {
            byte[] byteArray = Encoding.UTF8.GetBytes(requestBody);
            request.ContentType = "application/json";
            request.ContentLength = byteArray.Length;
            dataStream = request.GetRequestStream();
            dataStream.Write(byteArray, 0, byteArray.Length);
            dataStream.Close();
        }
        string responseFromServer = string.Empty;
        using (WebResponse response = request.GetResponse())
        {
            dataStream = response.GetResponseStream();
            var reader = new StreamReader(dataStream);
            responseFromServer = reader.ReadToEnd();
            reader.Close();
            dataStream.Close();
        }
        return responseFromServer;
    }
```

2. Consider these sample snippets to **Get Active Tickets** and **Add a new Ticket**:

```
protected void btnGetAllTickets_Click(object sender, EventArgs e)
        {
            string listName = "TicketsListName";
            string requestUrl = baseUrl + string.Format("web/lists/getbytitle('{0}')/items", listN
ame);
            var response = GetResponse(requestUrl, "GET");
            if (!string.IsNullOrEmpty(response))
            {
                var serializer = new JavaScriptSerializer();
                var listResponse = serializer.Deserialize<ListResponseWrapper>(response);
                var listResults = listResponse.d.results;
            }
        }

        protected void btnAddNewTicket_Click(object sender, EventArgs e)
        {
            string listName = "TicketsListName";
            string requestUrl = baseUrl + string.Format("web/lists/getbytitle('{0}')/additem", li
stName);
            var serializer = new JavaScriptSerializer();
            var requestBody = serializer.Serialize(new
            {
                Title = txtTicketTitle.Text,
                Description = txtTicketDescription.Text
            });
            var response = GetResponse(requestUrl, "POST", requestBody);
        }
```

Using the client object model

This section discusses how to perform CRUD operations on a SharePoint list using CSOM. First, the DLL for the CSOM is added, as discussed in the Setting up the CSOM environment section. The following is the sample code.

```
public string GetAllActiveItems(string siteUrl, string listTitle)
      {
            string data = string.Empty;
            ClientContext context = new ClientContext(siteUrl);
            Web web = context.Web;
            List listByTitle = context.Web.Lists.GetByTitle(listTitle);
            CamlQuery camlQuery = new CamlQuery();
            camlQuery.ViewXml = @"<View><Query><Where><Eq><FieldRef
Name='IsActive'/><Value Type='Text'>True</Value></Eq></Where></Query></View>";
            ListItemCollection items = listByTitle.GetItems(camlQuery);
            context.Load(items);
            context.ExecuteQuery();
            for (int count = 0; count < items.Count; count++)
            {
               ListItem item = items[count];
               data += (item["Title"].ToString()) + " : " +
                  (item["Description"].ToString()) + " : " +
                  (item["IsActive"].ToString()) + "<BR/>";
            }
         return data;
      }
```

Note: This is not the complete CSOM code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Web

Scenario: Change the welcome page of the help desk application to enable a customized landing page experience for multiple websites in your organization.

Using the custom REST API

This section discusses how to perform web-related operations for changing the welcome page of a SharePoint website by using a REST-based WCF service. The service methods have the server object model code to access SharePoint APIs. The URLs for hosting the WCF service on SharePoint are structured similarly to the calls for the SharePoint 2013 REST API (http://<site url>/_api/../). This helps to ensure a smooth migration to SharePoint 2013.

In this scenario, IT personnel create a custom welcome page for the help desk application and apply it to the help desk site, giving it a new look.

Task 1 – Create and implement the interface in the WCF REST service

Create the Operation Contracts to change the welcome page of the site in the WCF REST service interface class (IContosoService.cs), as discussed in the Using the custom REST API and setting up the web application environment section.

- Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
    [WebInvoke(Method = "POST", UriTemplate = "/_api/web/RootFolder/Update",
      ResponseFormat = WebMessageFormat.Json,
      RequestFormat = WebMessageFormat.Json,
      BodyStyle = WebMessageBodyStyle.WrappedRequest
      )]
    string ChangeWelcomePage(string SiteName, string PageUrl);
```

Task 2 – Create the service methods

1. Implement the interface in the ContosoService.cs file, and add the following code to it:

```
public string ChangeWelcomePage(string SiteName, string PageUrl)
    {
      SPSecurity.RunWithElevatedPrivileges(delegate()
      {
        using (SPSite site = new SPSite(sharepointSiteUrl))
        {
          SPWeb web = site.OpenWeb(SiteName);
          web.AllowUnsafeUpdates = true;
          SPFolder folder = web.RootFolder;
          folder.WelcomePage = PageUrl;
          folder.Update();
          web.Update();
          web.AllowUnsafeUpdates = false;
        }
      });
      return "Success Message";
    }
```

2. Deploy the project.

Task 3 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, create the Get Response method, as shown earlier. This calls the WCF service APIs.

2. Use the following code snippet to call the WCF REST service to change the welcome page:

```
string requestUrl = baseUrl + "web/RootFolder/Update";
var serializer = new JavaScriptSerializer();
var requestBody = serializer.Serialize(new
{
    SiteName = "HelpDeskSiteName",
    PageUrl = "SitePages/NewHomePageName.aspx"
});
var response = GetResponse(requestUrl, "POST", requestBody);
```

Using the client object model

This section discusses how to change the welcome page of a SharePoint site using CSOM. First, the DLL for the CSOM is added, as discussed in the Setting up the CSOM environment section. The following is the sample code.

```
public string ChangeWelcomePage(string siteUrl, string pagepath)
    {
        ClientContext context = new ClientContext(siteUrl);
        var web = context.Web;
        Folder spfolder = web.RootFolder;
        spfolder.WelcomePage = pagepath;
        spfolder.Update();
        web.Update();
        context.ExecuteQuery();
        return "Successful";
    }
```

Site

Scenario: Easily change permissions for the help desk site to automate adding a new employee.

Using the custom REST API

This section discusses how to perform site-related operations on a SharePoint site using a REST-based WCF service. The service methods have the server object model code to access SharePoint APIs. The URLs for hosting the WCF service on SharePoint are structured similarly

to the calls for the SharePoint 2013 REST API (http://<site url>/_api/../). This helps to ensure a smooth migration to SharePoint 2013.

In this scenario, the administrator of the help desk application assigns permissions to users, depending on their privileges on the site.

Task 1 – Create and implement the interface in the WCF REST service

Create the Operation Contracts to designate a permission level and assign it in the WCF REST service interface class (**IContosoService.cs**), as discussed in the Using the custom REST API and setting up the web application environment section.

- Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
    [WebInvoke(Method = "POST", UriTemplate = "/_api/web/RoleAssignments/add",
      ResponseFormat = WebMessageFormat.Json,
      RequestFormat = WebMessageFormat.Json,
      BodyStyle = WebMessageBodyStyle.WrappedRequest
      )]
    string AssignPermissions(string UserName, string SiteName,  string
PermissionLevelName);
```

Task 2 – Create the service methods

- Implement the interface in the **ContosoService.cs** file, and add the following code to it:

```
public string AssignPermissions(string UserName, string SiteName,  string PermissionLeve
lName)
    {
      SPSecurity.RunWithElevatedPrivileges(delegate()
      {
        using (SPSite siteCollection = new SPSite(sharepointSiteUrl))
        {
          using (SPWeb web = siteCollection.OpenWeb(SiteName))
          {
            web.AllowUnsafeUpdates = true;
            SPUserCollection SpUsers = web.SiteUsers;
            SPUser sPUser = SpUsers[UserName];
            SPRoleAssignment SpRoleAss = new SPRoleAssignment((SPPrincipal)sPU
ser);
            web.BreakRoleInheritance(true);
            SpRoleAss.RoleDefinitionBindings.Add(web.RoleDefinitions[PermissionLev
```

```
elName]);
                    web.RoleAssignments.Add(SpRoleAss);
                    web.AllowUnsafeUpdates = false;
                }
            }
        });
        return "Success Message";
    }
```

Task 3 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, create the Get Response method, as shown earlier. This calls the WCF service APIs.
2. Consider the following code snippets that use the WCF REST services:

```
        string requestUrl = baseUrl + "web/RoleAssignments/add";
        var serializer = new JavaScriptSerializer();
        var requestBody = serializer.Serialize(new
        {
            UserName = txtUserName.Text,
            SiteName = "HelpDeskSiteName",
            PermissionLevelName = txtPermissionLevelName.Text
        });
        var response = GetResponse(requestUrl, "POST", requestBody);
```

Using the client object model

This section discusses how to assign rights to site users using CSOM. First, the DLL for the CSOM is added, as discussed in the Setting up the CSOM environment section. The following is the sample code.

```
public string CreatePermissionLevel(string siteUrl, string permissionlevelName)
  {
     ClientContext clientContext = new ClientContext(siteUrl);
        Site site = clientContext.Site;
        Web web = clientContext.Web;
        BasePermissions permissions = new BasePermissions();
        permissions.Set(PermissionKind.ApplyStyleSheets);
        permissions.Set(PermissionKind.ManageLists);
```

```
            // Create a new role definition.
            RoleDefinitionCreationInformation roleDefinitionCreationInformation = new
RoleDefinitionCreationInformation();
            roleDefinitionCreationInformation.Name = permissionlevelName;
            roleDefinitionCreationInformation.Description = "This Permission level has custom
set of rights defined within it"; ;
            roleDefinitionCreationInformation.BasePermissions = permissions;
            RoleDefinition roleDefinition =
site.RootWeb.RoleDefinitions.Add(roleDefinitionCreationInformation);
            RoleDefinitionBindingCollection roleDefinitionBindingCollection =
               new RoleDefinitionBindingCollection(clientContext);
            roleDefinitionBindingCollection.Add(roleDefinition);
            clientContext.ExecuteQuery();
            return "Successful";
    }
```

Note: This is not the complete CSOM code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Search

Scenario: Create a Web Part that searches help desk supporting documents for a specific set of properties.

Using the custom REST API

This section discusses how to perform search-related operations on a SharePoint site using a REST-based WCF service. Specifically, it covers how to create a custom search scope and search help desk-related documents by entering a keyword. The service methods have the server object model code to access SharePoint APIs. The URLs for hosting the WCF service on SharePoint are structured similarly to the calls for the SharePoint 2013 REST API (http://<site url>/_api/../). This helps to ensure a smooth migration to SharePoint 2013.

In this scenario, users of the help desk application can search for all help desk-related documents, and in the search results, they can retrieve only the necessary properties related to the documents, such as Title, Path, and Hit Highlighted Summary.

Task 1 – Create and implement the interface in the WCF REST service

Create the Operation Contracts and Data Contracts to search documents related to the help desk in the WCF REST service interface class (IContosoService.cs), as discussed in the Using the custom REST API and setting up the web application environment section.

1. Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
[WebGet(UriTemplate = "/_api/search/query/querytext/'{SearchText}'",
        RequestFormat = WebMessageFormat.Json,
        ResponseFormat = WebMessageFormat.Json)]
    SearchResponseWrapper Search(string SearchText);
```

2. Add the following DataContracts in the interface:

```
[DataContract]
  public class SearchResponseWrapper
  {
    [DataMember]
    public SearchResponse d;
  }

  [DataContract]
  public class SearchResponse
  {
    [DataMember]
    public SearchResult query;
  }

  [DataContract]
  public class SearchResponseMetaData
  {
    [DataMember]
    public string type { get; set; }
  }

  [DataContract]
  public class SearchPrimaryQueryResult
  {
    [DataMember]
    public SearchRelevantResults RelevantResults { get; set; }
  }
```

```
[DataContract]
public class SearchRelevantResults
{
    [DataMember]
    public SearchTable Table { get; set; }
}

[DataContract]
public class SearchTable
{
    [DataMember]
    public SearchTableRows Rows { get; set; }
}

[DataContract]
public class SearchTableRows
{
    [DataMember]
    public List<SearchTableRowsResult> results { get; set; }
}

[DataContract]
public class SearchTableRowsResult
{
    [DataMember]
    public string Value { get; set; }
}

[DataContract]
public class SearchResult
{
    [DataMember(Name = "__metadata", Order = 1)]
    public SearchResponseMetaData metadata;
    [DataMember(Order = 2)]
    public SearchPrimaryQueryResult PrimaryQueryResult;
}
```

Task 2 – Create the service methods

Note: You need to add references to the following DLLs from ProgramFiles/CommonFiles/Microsoft Shared/WebServerExtensions/14/ISAPI):

- Microsoft.Office.Server.dll
- Microsoft.Office.Server.Search.dll

- Implement the interface in the ContosoService.cs file and add the following code to it:

```
public SearchResponseWrapper Search(string searchText)
        {
            var searchRows = new List<SearchTableRowsResult>();
            using (SPSite site = new SPSite(sharepointSiteUrl))
            {
                SPWeb web = site.OpenWeb();
                web.AllowUnsafeUpdates = true;
                SearchServiceApplicationProxy SSAProxy = (SearchServiceApplicationProxy)Se
archServiceApplicationProxy.
                GetProxy(SPServiceContext.GetContext(new SPSite(sharepointSiteUrl)));
                KeywordQuery keywordQuery = new KeywordQuery(SSAProxy);
                keywordQuery.ResultsProvider = Microsoft.Office.Server.Search.Query.SearchP
rovider.SharepointSearch;
                keywordQuery.QueryText = searchText + " +isDocument:1";
                keywordQuery.HiddenConstraints = "site:\"" + sharepointSiteUrl + "\"";
                keywordQuery.KeywordInclusion = KeywordInclusion.AllKeywords;
                keywordQuery.ResultTypes |= ResultType.RelevantResults;
                keywordQuery.SelectProperties.Add("Title");
                keywordQuery.SelectProperties.Add("Path");
                keywordQuery.SelectProperties.Add("HitHighlightedSummary");
                ResultTableCollection searchResults = keywordQuery.Execute();
                if (searchResults.Exists(ResultType.RelevantResults))
                {
                    ResultTable searchResultTable = searchResults[ResultType.RelevantResults];
                    DataTable result = new DataTable();
                    result.TableName = "SearchResults";
                    result.Load(searchResultTable, LoadOption.OverwriteChanges);
                    foreach (DataRow resultRow in result.Rows)
                    {
                        SearchTableRowsResult searchResultValue = new SearchTableRowsResult
```

```
()
                {
                    Value = resultRow.ItemArray[0].ToString(),
                };
                searchRows.Add(searchResultValue);
            }
        }
        web.AllowUnsafeUpdates = false;
    }

    var response = new SearchResponseWrapper()
    {
        d = new SearchResponse()
        {
            query = new SearchResult()
            {
                metadata = new SearchResponseMetaData() { },
                PrimaryQueryResult = new SearchPrimaryQueryResult()
                {
                    RelevantResults = new SearchRelevantResults()
                    {
                        Table = new SearchTable()
                        {
                            Rows = new SearchTableRows()
                            {
                                results = searchRows
                            }
                        }
                    }
                }
            }
        }
    };
    return response;
}
```

Task 3 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, create the Get Response method, as shown earlier. This calls the WCF service APIs.
2. Consider the following code snippets that use the WCF REST services:

```
        string searchText = txtSearch.Text;
        string requestUrl = baseUrl + string.Format("search/query/querytext/'{0}'", search
Text);
        var response = GetResponse(requestUrl, "GET");
        if (!string.IsNullOrEmpty(response))
        {
            var serializer = new JavaScriptSerializer();
            var searchResults = serializer.Deserialize<SearchResponseWrapper>(response
);
            var results = searchResults.d.query.PrimaryQueryResult.RelevantResults.Table.R
ows.results;
        }
```

Using the Search service in JavaScript

In this scenario, users can search using keywords in JavaScript. SharePoint 2010 provides the **Search.asmx** service to perform this operation. The following is the sample code.

```
function Search() {
    var soapxml = CreateEnvelopeForSearchQuery([keyword]);
    $.ajax({
        url: "http://[Your Server Name]/_vti_bin/search.asmx",
        type: "POST",
        dataType: "xml",
        data: soapxml,
        complete: processResult,
        contentType: "text/xml; charset=\"utf-8\""
    });
    function CreateEnvelopeForSearchQuery(query)
    {
        var queryXML = "<QueryPacket
xmlns='urn:Microsoft.Search.Query'><Query><Context>  <QueryText language='en-US'
type='STRING'>" + query + "</QueryText></Context></Query></QueryPacket>";
        var soapXml = "<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-
```

```
instance' xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'> \
<soap:Body> \
<Query xmlns='urn:Microsoft.Search'> \
<queryXml>" + escapeHTML(queryXML) + "</queryXml> \
</Query> \
</soap:Body> \
</soap:Envelope>";
        return soapXml;
    };
    function escapeHTML(str) {
        return str.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/>/g, '&gt;');
    }
    function processResult(xData, status) {
        // Process your result here.
    }
};
```

Note: This is not the complete JavaScript code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Note: While this JavaScript code may work in SharePoint 2013, it is recommended that you use the REST API approach for Search in SharePoint 2013 (http://msdn.microsoft.com/en-us/library/jj163876.aspx).

Social

Scenario: Integrate a newsfeed into the help desk site.

In this scenario, users of the help desk application can view their MySite activities or events directly in the web application.

Using the custom REST API

This section discusses how to perform operations related to social feeds on a SharePoint site using a REST-based WCF service. The service methods have the server object model code to access SharePoint APIs. The URLs for hosting the WCF service on SharePoint are structured similarly to the calls for the SharePoint 2013 REST API (http://<site url>/_api/../). This helps to ensure a smooth migration to SharePoint 2013.

Task 1 – Create and implement the interface in the WCF REST service

Create the Operation Contracts to view activity feeds from MySite in the WCF REST service interface class (IContosoService.cs), as discussed in the Using the custom REST API and setting up the web application environment section.

1. Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
[WebInvoke(Method = "POST",
          UriTemplate = "/_api/socialfeed/my/feed",
          RequestFormat = WebMessageFormat.Json,
          ResponseFormat = WebMessageFormat.Json,
          BodyStyle = WebMessageBodyStyle.WrappedRequest)]
      SocialFeedResponseWrapper GetMyActivities(string UserName);
```

2. Add the following DataContracts in the interface:

```
[DataContract]
   public class SocialFeedResponseWrapper
   {
      [DataMember]
      public SocialResponse d;
   }

   [DataContract]
   public class SocialResponse
   {
      [DataMember(Name = "__metadata", Order = 1)]
      public SocialResponseMetaData metadata;

      [DataMember]
      public SocialResult SocialFeed;
   }

   [DataContract]
   public class SocialResponseMetaData
   {
      [DataMember]
```

```
    public int id { get; set; }

    [DataMember]
    public string type { get; set; }

    [DataMember]
    public Uri uri { get; set; }
}

[DataContract]
public class SocialResult
{
    [DataMember]
    public List<SocialThread> Threads;
}

[DataContract]
public class SocialThread
{
    [DataMember]
    public SocialThreadActor Actors { get; set; }

    public SocialThreadRootPost RootPost { get; set; }
}

[DataContract]
public class SocialThreadActor
{
    [DataMember]
    public List<SocialThreadActorResult> results { get; set; }
}

[DataContract]
public class SocialThreadActorResult
{
    [DataMember]
```

```
        public string Name { get; set; }
    }

    [DataContract]
    public class SocialThreadRootPost
    {
        [DataMember]
        public string Text { get; set; }
    }
```

Task 2 – Create the service methods

Note: You need to add a reference to the following DLL from ProgramFiles/CommonFiles/Microsoft Shared/WebServerExtensions/14/ISAPI): Microsoft.Office.Server.UserProfiles.dll.

- Implement the interface in the ContosoService.cs file and add the following code to it.

```
public SocialFeedResponseWrapper GetMyActivities(string CurrentUserName)
        {
            var socialThreads = new List<SocialThread>();
            SPSecurity.RunWithElevatedPrivileges(delegate()
            {
                using (var sitecollection = new SPSite(sharepointSiteUrl))
                {
                    var currentcontext = SPServiceContext.GetContext(sitecollection);
                    var userprofmanager = new UserProfileManager(currentcontext);
                    var currentuser = userprofmanager.GetUserProfile(CurrentUserName);
                    ActivityManager activitymanager = new ActivityManager(currentuser, curre
ntcontext);
                    ActivityEventsCollection eventscollection = activitymanager.GetActivitiesByM
e();

                    foreach (ActivityEvent activity in eventscollection)
                    {
                        if (activity.LinksList != null)
                        {

                            socialThreads.Add(new SocialThread()
                            {
                                Actors = new SocialThreadActor()
```

```
                        {
                            results = new List<SocialThreadActorResult>(){
                                    new SocialThreadActorResult(){ Name = activity.Publisher
.Name}
                            }
                        },
                        RootPost = new SocialThreadRootPost()
                        {
                            Text = activity.Value,
                        }
                    });
                }

            }
        }
    });

    var response = new SocialFeedResponseWrapper()
    {
        d = new SocialResponse()
        {
            metadata = new SocialResponseMetaData() { },
            SocialFeed = new SocialResult()
            {
                Threads = socialThreads
            }
        }
    };
    return response;
}
```

Task 3 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, create the **Get Response** method, as shown earlier. This calls the WCF service APIs.

2. Consider the following code snippets that use the WCF REST services:

```
string requestUrl = baseUrl + "socialfeed/my/feed";
            var serializer = new JavaScriptSerializer();
            var requestBody = serializer.Serialize(new
            {
                UserName = HttpContext.Current.User.Identity.Name
            });
            var response = GetResponse(requestUrl, "POST", requestBody);
            if (!string.IsNullOrEmpty(response))
            {
                var feedResults = serializer.Deserialize<SocialFeedResponseWrapper>(respon
se);
                var results = feedResults.d.SocialFeed.Threads;
            }
```

Using the SocialDataService in JavaScript

In this scenario, users want to add comments on MySite using the **SocialDataService.asmx** service. The following is the sample code for **Add a Comment**:

```
function AddComment() {
        var soapxml = CreateEnvelopeForAddingComment([url], [SampleComment],
[isHighPriority], [Title]);
    $.ajax({
        url: "http:/[Your Server Name]/_vti_bin/socialdataservice.asmx",
        type: "POST",
        dataType: "xml",
        data: soapxml,
        complete: processResult,
        contentType: "text/xml; charset=\"utf-8\""
    });

    function CreateEnvelopeForAddingComment(url, comment, isHighPriority, title) {
        var soapXml = "<soap:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">" +
          "<soap:Body>" +
            "<AddComment
```

```
xmlns=\"http://microsoft.com/webservices/SharePointPortalServer/SocialDataService\">" +
            " <url>" + url + "</url>" +
             "<comment>" + comment + "</comment>" +
             "<isHighPriority>" + isHighPriority + "</isHighPriority>" +
             "<title>" + title + "</title>" +
           "</AddComment>" +
         "</soap:Body>" +
       "</soap:Envelope>"
       return soapXml;
   };
   function processResult(xData, status)
   {
      // Process your result here.
   }
};
```

Note: This is not the complete JavaScript code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Note: While this JavaScript code may work in SharePoint 2013, it is recommended that you use the REST API approach for social operations in SharePoint 2013 (http://msdn.microsoft.com/en-us/library/dn155789.aspx).

Workflow

This section discusses major changes in the SharePoint 2013 workflow engine from SharePoint 2010. It also explores SharePoint workflow interop, showing how this feature allows SharePoint 2013 to call SharePoint 2010 workflows within it.

SharePoint 2013 workflow enhancements

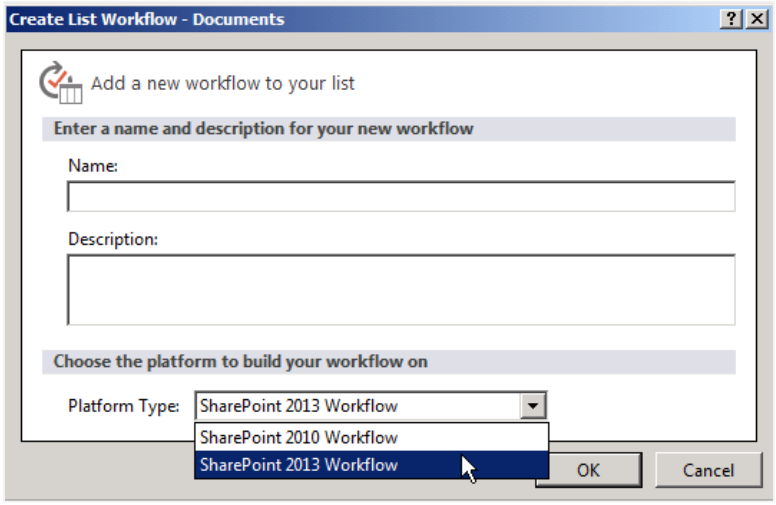
The workflow framework in SharePoint 2013 is significantly changed from previous versions. SharePoint 2013 workflows are powered by Windows Workflow Foundation (WF) 4, which was substantially redesigned. SharePoint 2010 used WF found in the .NET Framework 3.5 SP1.

Workflows built for SharePoint 2010 were tightly coupled with and designed for on-premises deployments. This meant that coded workflows could not run within the sandbox or SharePoint Online. All workflow data also was retained within the site collection's content database where the workflow ran.

Perhaps the most prominent feature of the new workflow infrastructure is the introduction of Windows Azure as the workflow execution host. The workflow execution engine now lives outside of SharePoint in Windows Azure. The workflow service provided by Workflow Manager is decoupled from SharePoint and no longer runs in the SharePoint farm; rather, it runs in Microsoft data centers. Workflow Manager is an installable product and can be hosted in an on-premises SharePoint farm. SharePoint instructs Workflow Manager to execute a workflow, and the two products communicate with each other through standard protocols: WCF services over port 80 (HTTP)/443 (HTTPS).

These workflows are primarily declarative but can also be extended with custom code, similar to how you can create custom activities and actions in SharePoint 2010 workflows for use in SharePoint Designer 2010. However, you must install custom-coded workflows separately as farm solutions in SharePoint 2013. Note that when you create a workflow in SharePoint Designer 2013, you have the option of choosing the platform on which you wish to build it (Figure 5).

Figure 5. Choosing a platform for building a workflow



- SharePoint 2010 workflow: These workflows are based on the SharePoint 2010 model using .NET Framework 3.5 SP1 Windows Workflow Foundation. They execute the same way they did in SharePoint 2010, meaning they run within the same SharePoint processes. They cannot take advantage of any improvements offered by Windows Azure.
- SharePoint 2013 workflow: This option is available when SharePoint 2013 is connected to a configured instance of Workflow Manager. These workflows execute within Workflow Manager instead of SharePoint processes.

SharePoint workflow interop

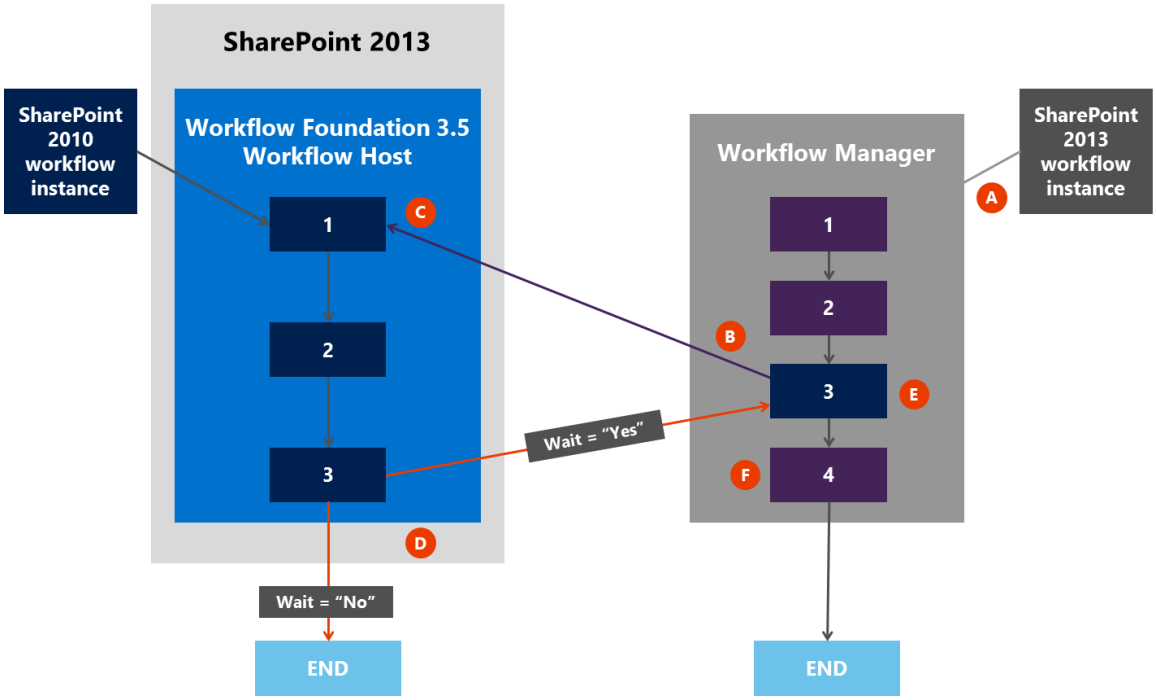
SharePoint workflow interop enables SharePoint 2010 workflows (which are based on WF 3) to be called from SharePoint 2013 workflows (which are based on WF 4). This important

feature lets you reuse existing workflow capabilities and call on workflow activities that are not integrated into SharePoint 2013.

For example, say you have legacy SharePoint 2010 workflows that you want to reuse on the SharePoint 2013 platform. Or perhaps you are creating new SharePoint 2013 workflows and need to invoke activities available only on the SharePoint 2010 platform. SharePoint workflow interop can resolve these issues.

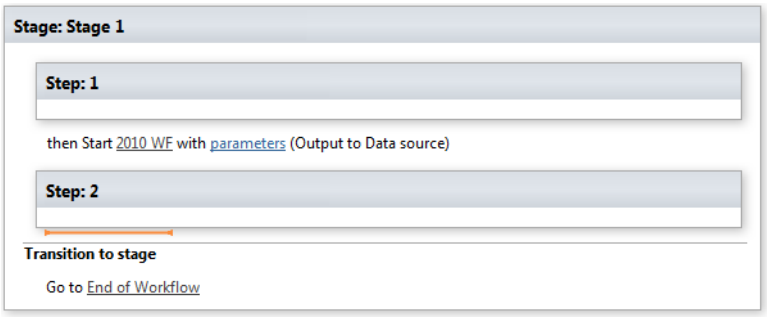
The Workflow Foundation 4 execution engine in SharePoint 2013 is hosted in Workflow Manager, which runs as an external service; however, SharePoint 2013 still contains the legacy workflow host used to process SharePoint 2010 workflows. SharePoint workflow interop negotiates the two execution environments (Figure 6).

Figure 6. Overview of SharePoint workflow interop



- As shown at point A, an instance of a SharePoint 2013 workflow starts to run in the Workflow Foundation 4-based Workflow Manager. Note that Workflow Manager is not in SharePoint; instead, it runs as an external service.
- At point B, you reach a place in the SharePoint 2013 workflow (step 3 in Workflow Manager) where you want to invoke a SharePoint 2010 workflow. Using the Microsoft Visual Studio 2012 workflow designer, you can do this by implementing the Start 2010 WF activity (Figure 7). From the perspective of the SharePoint object model, this is accomplished using the Start Workflow method on the Workflow Interop Service class.

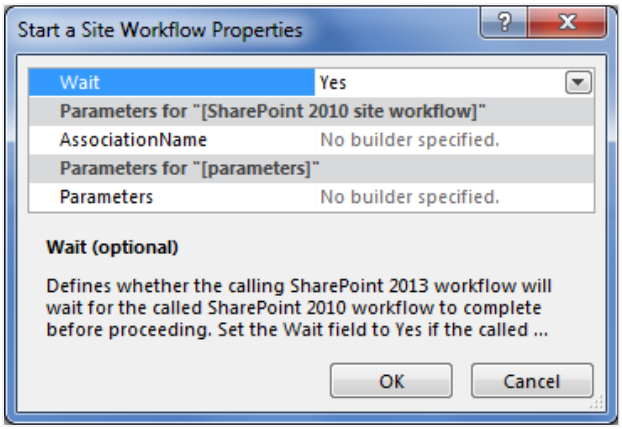
Figure 7. Stage tile for starting a SharePoint 2013 workflow



- At point C, the SharePoint 2010 workflow begins executing in the Workflow Foundation 3.5 workflow host inside SharePoint, but an important consideration arises: In some scenarios, you may want the SharePoint 2013 workflow to wait for the SharePoint 2010 workflow to finish executing (and perhaps return some data) before continuing to execute the SharePoint 2013 workflow. In other scenarios, this may not be necessary, and both workflows can run independently in parallel.

  To control this behavior, the **Workflow Interop** class, which controls executing workflows in the Workflow Foundation 3.5 workflow host, provides a **Wait** property (Figure 8). Setting this Boolean property to **Yes** (in the properties dialog box) or to **true** (in the **Wait** property) causes the SharePoint 2013 workflow to pause until the SharePoint 2010 workflow finishes executing and returns a **completed** message.

  Figure 8. Setting the Wait property to pause a SharePoint 2013 workflow



- At point D, the practical effect of selecting **Yes** or **No** in the properties dialog box (or **true** or **false** in the **Wait** property) is depicted. If **Wait** is **true**, the SharePoint 2010 workflow passes a WorkflowCompleted (http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.workflowservices.workflowinteropeventreceiver.workflowcompleted(v=office.15).aspx) event (and, optionally, returns data as a DynamicValue (http://msdn.microsoft.com/en-us/library/jj193446.aspx) property). Of course, if **Wait** is **false**, the SharePoint 2010 workflow executes and terminates normally.

- The step at point **E** is only relevant if the invocation of the SharePoint 2010 workflow is specified as **Wait=true**. In this case, the SharePoint 2013 workflow receives the **Workflow Completed** event and restarts its execution at the point where it left off.
- At point **F**, the SharePoint 2013 workflow completes its execution and terminates normally. If **Wait=false**, the SharePoint 2013 workflow executes and terminates independently of the SharePoint 2010 workflow.

Scenario: Route a help desk service request to IT personnel on the basis of the request's priority.

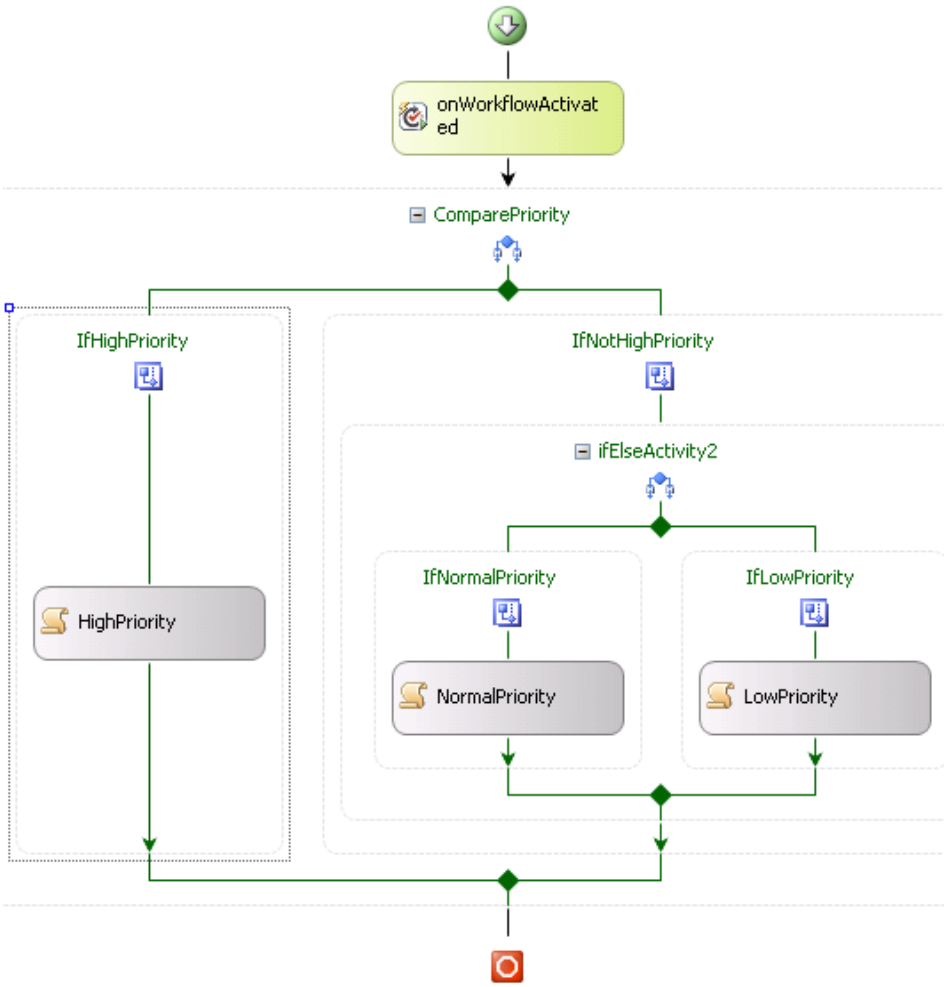
Using the custom REST API

This section discusses how to create a simple workflow in SharePoint 2010 using Microsoft Visual Studio 2012, and how to associate the workflow with a SharePoint list using a web service. In the help desk application, the workflow is associated with a Service Request list that has **Priority** and **AssignedTo** fields. Based on the priority of the service request (high, normal, or low), the request is assigned to different IT personnel.

Task 1 – Create a SharePoint 2010 workflow in Visual Studio

1. In a SharePoint project, add a Site workflow and add activities to it (Figure 9).

Figure 9. Adding activities to a Site workflow



As shown, the **If Else** activity has been added to test the value of the **Priority** field. On the basis of that value, the code activities **High Priority**, **Normal Priority**, and **Low Priority** are generated.

2. Add a property for the **Priority** field in the **workflow.cs** file, as shown in this example.

```
string priority;
private void onWorkflowActivated1_Invoked(object sender,
ExternalDataEventArgs e)
    {
        priority = workflowProperties.Item["Priority"].ToString();
    }
```

3. Generate handlers for the code activities: Right-click a code activity and click **Generate Handler**. Then add the following code to the **.cs** file of the workflow.

```
private void HighPriority_ExecuteCode(object sender, EventArgs e)
```

```
        {
            string loginName = "CORP\\ITAdmin";
            SPUser userToAssign = workflowProperties.Web.SiteUsers[loginName];
            SPListItem itemToUpdate = workflowProperties.Item;
            itemToUpdate["Assigned To"] = userToAssign.ID.ToString() + ";#" +
userToAssign.Name;
            itemToUpdate.Update();
        }
private void NormalPriority_ExecuteCode(object sender, EventArgs e)
        {
            string loginName = "CORP\\lisa";
            SPUser userToAssign = workflowProperties.Web.SiteUsers[loginName];
            SPListItem itemToUpdate = workflowProperties.Item;
            itemToUpdate["Assigned To"] = userToAssign.ID.ToString() + ";#" +
userToAssign.Name;
            itemToUpdate.Update();
        }
private void LowPriority_ExecuteCode(object sender, EventArgs e)
        {
            string loginName = "CORP\\John";
            SPUser userToAssign = workflowProperties.Web.SiteUsers[loginName];
            SPListItem itemToUpdate = workflowProperties.Item;
            itemToUpdate["Assigned To"] = userToAssign.ID.ToString() + ";#" +
userToAssign.Name;
            itemToUpdate.Update();
        }
```

Task 2 – Create and implement the interface in the WCF REST service

Create the Operation Contracts to associate the workflow to a list in the WCF REST service interface class (**IContosoService.cs**), as discussed in the Using the custom REST API and setting up the web application environment section.

- Consider the following Operation Contracts in the Service Contract:

```
[OperationContract]
[WebInvoke(Method = "POST",
    UriTemplate = "/_api/web/lists/getbytitle('{ListName}')/workflowassociations/add",
    RequestFormat = WebMessageFormat.Json,
    ResponseFormat = WebMessageFormat.Json,
```

```
        BodyStyle = WebMessageBodyStyle.WrappedRequest)]
string AssociateWorkflow(string AssociationListName, string TaskListName, string History
ListName, string WorkflowName
);
```

Task 3 – Create the service methods

- Implement the interface in the ContosoService.cs file, and add the following code to it.

```
public string AssociateWorkflow(string AssociationListName, string TaskListName, string
HistoryListName, string WorkflowName)
        {
            SPSecurity.RunWithElevatedPrivileges(delegate()
            {
                using (SPSite spSite = new SPSite(sharepointSiteUrl))
                {
                    using (SPWeb spWeb = spSite.OpenWeb())
                    {
                        var workflowTemplate = spWeb.WorkflowTemplates.GetTemplateByNam
e(WorkflowName, System.Globalization.CultureInfo.CurrentCulture);
                        var taskList = spWeb.Lists[TaskListName];
                        var historyList = spWeb.Lists[HistoryListName];
                        var workflowAssociation = SPWorkflowAssociation.CreateListAssociation(
workflowTemplate, WorkflowName, taskList, historyList);
                        workflowAssociation.AutoStartChange = true;
                        workflowAssociation.AutoStartCreate = true;
                        workflowAssociation.AllowManual = true;
                        spWeb.AllowUnsafeUpdates = true;

                        var associatedList = spWeb.Lists[AssociationListName];
                        associatedList.WorkflowAssociations.Add(workflowAssociation);
                        associatedList.Update();

                        spWeb.AllowUnsafeUpdates = false;
                    }
                }
            });
            return "Success Message";
        }
```

Task 4 – Consume the WCF service in a web application

1. To consume the service in a web application hosted in a different domain, create the **Get Response** method, as shown earlier. This calls the WCF service APIs.
2. Consider the following code snippets that use the WCF REST services:

```
string listName = "List Name Here";
string requestUrl = baseUrl + string.Format("web/lists/getbytitle('{0}')/workflowassociations/add", listName);
var serializer = new JavaScriptSerializer();
var requestBody = serializer.Serialize(new
        {
            WorkflowName = "Workflow Name",
            TaskListName = listName ,
            AssociationListName = "Association Name",
            HistoryListName = "History List Name"
        });
var response = GetResponse(requestUrl, "POST", requestBody);
```

Using the client object model

In the help desk application, the workflow is associated with a Service Request list that has **Priority** and **AssignedTo** fields. Service requests are assigned to different IT personnel based on their priority (high, normal, or low). The following is sample code for associating the workflow to the SharePoint list using the CSOM.

```
 public string CreateListworkflowAssociaton(string siteUrl, string listTitle, string workflowTemplateName,
      string historyListName, string taskListName, string workflowName)
      {
         try
         {
            ClientContext clientContext = new ClientContext(siteUrl);
            Site site = clientContext.Site;
            Web web = clientContext.Web;

            // Create historyList and task List.
            ListCreationInformation ListCreationInformationHistory = new ListCreationInformation();
            ListCreationInformationHistory.Title = historyListName;
```

```
            ListCreationInformationHistory.TemplateType =
(int)ListTemplateType.WorkflowHistory;
            web.Lists.Add(ListCreationInformationHistory);
            ListCreationInformation ListCreationInformationTask = new
ListCreationInformation();
            ListCreationInformationTask.Title = taskListName;
            ListCreationInformationTask.TemplateType = (int)(ListTemplateType.Tasks);
            web.Lists.Add(ListCreationInformationTask);
            clientContext.ExecuteQuery();
            clientContext.Load(web.Lists);
            List listbyTitle = web.Lists.GetByTitle(listTitle);
            WorkflowTemplate workflowTemplate =
web.WorkflowTemplates.GetByName(workflowTemplateName);
            clientContext.ExecuteQuery();
            WorkflowAssociationCreationInformation workflowAssociationCreationInformation
= new WorkflowAssociationCreationInformation();
            workflowAssociationCreationInformation.HistoryList =
web.Lists.GetByTitle(historyListName);
            workflowAssociationCreationInformation.TaskList =
web.Lists.GetByTitle(taskListName);
            workflowAssociationCreationInformation.Template = workflowTemplate;
            workflowAssociationCreationInformation.Name = workflowName;
            WorkflowAssociation workflowAssociation =
listbyTitle.WorkflowAssociations.Add(workflowAssociationCreationInformation);
            workflowAssociation.AutoStartCreate = true;
            workflowAssociation.AutoStartChange = false;
            workflowAssociation.AllowManual = false;
            workflowAssociation.Enabled = true;
            workflowAssociation.Update();
            clientContext.Load(workflowAssociation);
            clientContext.ExecuteQuery();
            return "Successful";
        }
        catch (Exception ex)
        {
            return ex.Message;
        }
```

```
    }
```

Note: This is not the complete CSOM code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Content management

Scenario: Get SharePoint taxonomy to tag help desk tickets and optimize searching.

Using the TaxonomyClientService in JavaScript

This section discusses how to fetch taxonomy terms using taxonomyclientservice.asmx. This web service enables a client to interact with the managed metadata TermStore object.

The following sample code generates the SOAP envelopes and then uses the AJAX call from JavaScript to get the terms within the term set.

```
function GetTerm() {
    var soapxml = CreateSoapEnvelope([Your TermStore Guid], 1033, [Your TermSet Guid);
    $.ajax({
        url: "http://[your Server Name]/_vti_bin/taxonomyclientservice.asmx",
        type: "POST",
        dataType: "xml",
        data: soapxml,
        complete: processResultData,
        contentType: "text/xml; charset=\"utf-8\""
    });
    function CreateSoapEnvelope(sspIdGuid, lcidGuid, termSetIdGuid) {
        var soapXml = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
        "<soap:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
        "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" " +
        "xmlns:soap=\"http://www.w3.org/2003/05/soap-envelope\">" +
        "<soap:Body>" +
        "<GetChildTermsInTermSet
xmlns=\"http://schemas.microsoft.com/sharepoint/taxonomy/soap/\">" +
        "<sspId>" + sspIdGuid + "</sspId>" +
        "<lcid>" + lcidGuid + "</lcid>" +
        "<termSetId>" + termSetIdGuid + "</termSetId>" +
        "</GetChildTermsInTermSet>" +
        "</soap:Body>" +
        "</soap:Envelope>";
```

```
        return soapXml;
    };
    function processResultData(xData, status)
    {

    // Process your result here.    }
};
```

Note: This is not the complete JavaScript code. Rather, it is a brief snippet to provide you with an approach for writing the code.

Note: While this JavaScript code may work in SharePoint 2013, it is recommended that you use the client object model APIs for taxonomy operations in SharePoint 2013 (http://msdn.microsoft.com/en-us/library/jj163949.aspx#SP15_ManagedMetadataAndNav_CSOMSupport).

Conclusion

SharePoint 2013 introduces a new multi-tier model for apps that increases their power, flexibility, and scalability. With new client and server APIs, easy integration into cloud computing models, and expanded deployment options, the architectural changes in SharePoint 2013 simplify development and installation tasks.

Developers can now build and update solutions in SharePoint 2010 in a way that allows them to be easily transformed into apps for SharePoint 2013. This largely eliminates the time and effort required for migration and avoids common conflicts associated with upgrading product versions. Organizations can continue to benefit from investments in SharePoint 2010, while still capitalizing on the enhanced features of SharePoint 2013.

The scenarios and sample code in this paper help to explain how to convert SharePoint 2010 solutions into apps for SharePoint 2013 and how these conversions can improve the efficiency and reach of specific workloads.

Additional resources

Developing applications for SharePoint 2010
http://msdn.microsoft.com/en-us/library/ff770300.aspx

Apps for SharePoint overview
http://msdn.microsoft.com/en-us/library/fp179930.aspx

SharePoint 2013 overview
http://office.microsoft.com/en-us/sharepoint/sharepoint-2013-overview-collaboration-software-features-FX103789323.aspx

Development best practices for SharePoint 2010
http://social.technet.microsoft.com/wiki/contents/articles/8666.sharepoint-2010-best-practices.aspx#Development

Development best practices for SharePoint 2013
http://social.technet.microsoft.com/wiki/contents/articles/12438.sharepoint-2013-best-practices.aspx#Development

To comment on this paper or request more documentation on these developer features, contact Office and SharePoint Developer Documentation at Doc This (docthis@microsoft.com).