**Using the SharePoint APIs**

In SharePoint 2013, you can now choose from three different APIs: the server-side object model (SSOM), the client-side object model (CSOM), and the REST API. All three APIs give you the option to build customizations for your SharePoint environment. This section will provide an overview of the three different APIs. Each API will then be used extensively in examples throughout the book. For specific detailed coverage of CSOM and REST, see Chapter 5, “Client-side programming.”

**Understanding the server-side object model**

The core server-side object model of SharePoint Foundation is loaded through an assembly named *Microsoft.SharePoint.dll*. When you reference this assembly within a Visual Studio 2012 project, you can start programming against the classes in the server-side object model, such as *SPSite, SPWeb*, and *SPList*. There are two initial requirements for a Visual Studio project that programs against the server-side object model by using the *Microsoft.SharePoint* assembly. First, the project must be configured to use .NET Framework 4 or 4.5 as its target framework. Pay extra attention if you are upgrading a SharePoint 2010 solution, because that will have been built using the .NET Framework 3.5 as its target framework. The second requirement is that your project must have a platform target setting that is compatible with a 64-bit environment, which is essential for properly loading the *Microsoft.SharePoint* assembly.

Another critical requirement for any application or component that is programmed against the server-side object model is that the application or component must be deployed and run on a SharePoint server in the farm in which you want to use the component. The deployment of applications or components that use the SharePoint server-side object model should always be done by using a SharePoint Solution or .wsp file. To deploy the solution, you will need access to at least one SharePoint server in the farm where the solution should be deployed. In most production environments, this means that you will hand off the solution and a document that describes how to deploy the solution to the administrator of the server. In your development environment, Visual Studio will usually do the deployment for you.

You can also create client applications with Visual Studio 2012 that program against the server-side object model. For example, you can create a standard console application that uses the server-side object model to access a site and the elements inside the site, such as lists and items. However, keep in mind that any client application that depends on the *Microsoft.SharePoint* assembly can be run only when launched on a server that has SharePoint installed on it and that is part of a SharePoint farm. This means that it’s not likely that you will encounter real-world scenarios that call for creating client applications that use the server-side object model. Even so, creating simple console applications that program against the *Microsoft.SharePoint*assembly in your development environment can be useful, because it gives you a quick and easy way to write and test code as you begin learning the server-side object model.

Most of the SharePoint Foundation APIs reside in *Microsoft.SharePoint.dll*. However, if you are building a custom solution, using the server-side object model you might also want to use SharePoint Server APIs and functionality. The bulk of the SharePoint Server APIs reside in *Microsoft.Office.Server.dll*; however, this isn’t the only available DLL that contains SharePoint Server APIs. For a full list of SharePoint APIs and the DLLs in which you can find them, see the MSDN page *.NET server API reference for SharePoint 2013* at [*http://msdn.microsoft.com/en-us/library/jj193058.aspx*](http://msdn.microsoft.com/en-us/library/jj193058.aspx).

Development of farm solutions has changed very little since SharePoint 2010, so this section contains links to the SharePoint 2010 SDK. To avoid confusion, keep the following points in mind at all times when using the SharePoint 2010 SDK to develop against SharePoint:

* You will see many references to "sandboxed solutions" in the SharePoint 2010 SDK. Sandboxed solutions with custom code are deprecated in SharePoint. "no code" sandboxed solutions are still viable.
* Our recommendation that farm solutions be used primarily for administrative extensions did not apply in SharePoint 2010. Therefore, many of the samples and other documentation in the SharePoint 2010 SDK may be about end-user extensions that are deployed as farm solutions.
* The terms "server-side" or "server code" in the SharePoint 2010 SDK refer to code that calls the SharePoint server object model. These terms do *not* refer to code that runs on remote web servers (that is, web servers external to the SharePoint farm). Code that calls SharePoint from remote web servers, in both SharePoint 2010 and SharePoint, always uses one of the SharePoint *client*object models. In the SharePoint 2010 SDK, such code would be called "client-side" or "client code."
* The assemblies in a farm solution in SharePoint 2010 could be deployed with Custom Access Security (CAS) policies. Such policies are ignored in SharePoint; all assemblies in farm solutions in SharePoint run with full trust.

### Packaging and deployment

The basics of packaging, installing, updating, and localizing farm solutions are explained in [Solutions Overview](http://msdn.microsoft.com/library/1983cab9-4b29-494a-a62a-0f8e83908744%28Office.15%29.aspx) and the node [Farm Solutions in SharePoint 2010](http://msdn.microsoft.com/library/845f7524-b9ff-412b-aa29-3afacda91100%28Office.15%29.aspx). Development of particular SharePoint components for inclusion in a farm solution is explained in the relevant nodes of SharePoint 2010 SDK. Most of the components in a farm solution should be encapsulated in one or more custom SharePoint Features. For information about designing and creating Features, see the [Working with Features](http://msdn.microsoft.com/library/ce5f5ce5-1429-439e-9261-2c4ba9788cc1%28Office.15%29.aspx) node of the SharePoint 2010 SDK.

### Administrative extensions

Guidance about extending the administrative functions in a SharePoint farm is in the [Windows SharePoint Services Administration](http://msdn.microsoft.com/library/cdcc1b8a-4144-446f-b471-03d4a754a8ab%28Office.15%29.aspx) node of the SharePoint 2010 SDK. There you can find explanations about extending Central Administration, creating custom Windows PowerShell cmdlets, customizing upgrades and migration, customizing backups, and customizing SharePoint event logging. One section explains how to customize the SharePoint farm health and performance measuring system. For instructions about creating a custom timer job, see [How to: Run Code on All Web Servers](http://msdn.microsoft.com/library/1bbb11b4-a342-4bed-9e7a-b8b13edd0ccc%28Office.15%29.aspx).

**Using the client-side object model**

SharePoint 2010 introduced the SharePoint Foundation client-side object model, which allows developers to use SharePoint content and objects in their client-side solutions. As a developer, you could now create a very simple solution that would be deployed into a SharePoint site or onto a user’s desktop and that could read or manage data in a SharePoint site.

In SharePoint 2010, the client-side object model was only available for SharePoint Foundation objects. In SharePoint 2013, however, the client-side object model has again been vastly improved by making a lot of the SharePoint Server objects available through the client-side object model. In SharePoint 2010 there were three client-side object models, and in SharePoint 2013 there are four. SharePoint 2013 allows you to choose between the **Managed, Silverlight, Mobile, and JavaScript object models**. Each of the four object models provides an object interface to SharePoint functionality that is based on the objects available in the *Microsoft.SharePoint* namespace. All four client-side object models also have support for at least part of the SharePoint Server 2013 functionality, but not all of them include the same SharePoint Server 2013 components.

The four client-side object models also all have their own usages. Each of the four object models presents an object interface in front of a service proxy. Developers write client-side code by using the object model, but the operations are batched and sent as a single XML request to the Client.svc service. When the XML request is received, the Client.svc service makes calls to the server-side object model on behalf of the client. The results of the server-side calls are then sent back to the calling client in the form of a JavaScript Object Notation (JSON) object.

The object model for Microsoft Silverlight can be used to build Silverlight applications, Web Parts, ASP.NET applications, apps for SharePoint and Office, and Silverlight applications for phones that use SharePoint data or SharePoint objects. A Silverlight application is compiled into an .xap file that can pretty much be stored anywhere. Examples of where .xap files can be deployed are a client computer, the file system of a SharePoint server, a list in a SharePoint library, and an external (web) server. The Silverlight client-side object model is contained in assemblies in the LAYOUTS\ClientBin folder. The following DLLs are available:

* Microsoft.SharePoint.Client.Silverlight.dll
* Microsoft.SharePoint.Client.Silverlight.Runtime.dll
* Microsoft.SharePoint.Client.DocumentManagement.Silverlight.dll
* Microsoft.SharePoint.Client.Publishing.Silverlight.dll
* Microsoft.SharePoint.Client.Search.Applications.Silverlight.dll
* Microsoft.SharePoint.Client.Search.Silverlight.dll
* Microsoft.SharePoint.Client.Taxonomy.Silverlight.dll
* Microsoft.SharePoint.Client.UserProfiles.Silverlight.dll
* Microsoft.SharePoint.Client.WorkflowServices.Silverlight.dll
* Microsoft.Office.Client.Policy.Silverlight.dll
* Microsoft.Office.Client.TranslationServices.Silverlight.dll

The Mobile object model can be used to create applications that run on Windows Phones. The Mobile client-side object model is a special version of the Silverlight client-side object model. The Mobile object model contains most of the same functionality as the Silverlight object model. A couple of areas are missing, but when you are creating a Windows Phone application using the Mobile object model you can use the REST APIs to access these areas. The Mobile client-side object model also contains some functionality that is specific to phones, such as APIs that enable a phone app to register for notifications from the Microsoft Push Notification Service. The Mobile client-side object model can be found in the same folder as the Silverlight client-side object model, in the LAYOUTS\ClientBin folder. The DLLs that are available for the Mobile client-side object model are:

* Microsoft.SharePoint.Client.Phone.dll
* Microsoft.SharePoint.Client.Phone.Runtime.dll
* Microsoft.SharePoint.Client.DocumentManagement.Phone.dll
* Microsoft.SharePoint.Client.Publishing.Phone.dll
* Microsoft.SharePoint.Client.Taxonomy.Phone.dll
* Microsoft.SharePoint.Client.UserProfiles.Phone.dll
* Microsoft.Office.Client.Policy.Phone.dll
* Microsoft.Office.Client.TranslationServices.Phone.dll

The Managed object model can be used to create .NET applications that run on Windows operating systems that aren’t phones or SharePoint servers. This means that the Managed object model can be used to create applications that run on client computers, or on Windows web servers not running SharePoint. The Managed object model can be found in the ISAPI folder and is contained in the following DLLs:

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll
* Microsoft.SharePoint.Client.ServerRuntime.dll
* Microsoft.SharePoint.Client.DocumentManagement.dll
* Microsoft.SharePoint.Client.Publishing.dll
* Microsoft.SharePoint.Client.Search.Applications.dll
* Microsoft.SharePoint.Client.Search.dll
* Microsoft.SharePoint.Client.Taxonomy.dll
* Microsoft.SharePoint.Client.UserProfiles.dll
* Microsoft.SharePoint.Client.WorkflowServices.dll
* Microsoft.Office.Client.Education.dll
* Microsoft.Office.Client.Policy.dll
* Microsoft.Office.Client.TranslationServices.dll
* Microsoft.Office.SharePoint.ClientExtensions.dll

The last client-side object model is the JavaScript object model. The JavaScript object model can be used in inline script or in separate .js files. Using the JavaScript client-side object model is an excellent way to add custom SharePoint code to a SharePoint-hosted app. The JavaScript object model is different from the other three in that it is not contained in a set of DLLs. Instead, it is contained in a JavaScript library, inside of .js files. The many .js files that make up the JavaScript client-side object model are located in the LAYOUTS folder. The core SharePoint functionality can be found in SP.js and in SP.Core.js.

Though the four client-side object models don’t contain exactly the same functionality, Microsoft has taken great care to ensure that the four models return objects that behave similarly. This means that if you know how to write code against one of the models, you can easily port that code to either of the other three models. Table 2-2 shows some of the main objects supported by each model alongside the related object from the server-side model.

Table 2-2 Equivalent objects in the server and client models

| **Server model** | **Managed model** | **Silverlight model** | **Mobile model** | **JavaScript model** |
| --- | --- | --- | --- | --- |
| SPContext | ClientContext | ClientContext | ClientContext | ClientContext |
| SPSite | Site | Site | Site | Site |
| SPWeb | Web | Web | Web | Web |
| SPList | List | List | List | List |
| SPListItem | ListItem | ListItem | ListItem | ListItem |
| SPField | Field | Field | Field | Field |

As in the standard code you write against the server-side object model, code written for client object models requires a starting point in the form of a context object. The context object provides an entry point into the associated API that can be used to gain access to other objects. When you have access to the objects, you can interact with the scalar properties of the object freely (for example, *Name, Title, Url*, and so on). Listing 2-4 shows how to create a context in each of the models and return an object representing a site collection. After the site collection object is returned, the *Url* property is examined. Code for the server model is included for comparison.

Listing 2-4 Creating contexts

//Server Object Model

SPSite siteCollection = SPContext.Current.Site;

string url = siteCollection.Url;

//Managed Client Object Model

using (ClientContext ctx = new ClientContext("http://intranet.wingtiptoys.com"))

{

 Site siteCollection = ctx.Site;

 ctx.Load(siteCollection);

 ctx.ExecuteQuery();

 string url = siteCollection.Url;

}

//Silverlight Client Object Model

using (ClientContext ctx =

 new ClientContext("http://intranet.wingtiptoys.com"))

{

 Site siteCollection = ctx.Site;

 ctx.Load(siteCollection);

 ctx.ExecuteQuery();

 string url = siteCollection.Url;

}

//Mobile Client Object Model

using (ClientContext ctx =

 new ClientContext("http://intranet.wingtiptoys.com"))

{

 Site siteCollection = ctx.Site;

 ctx.Load(siteCollection);

 ctx.ExecuteQuery();

 string url = siteCollection.Url;

}

//JavaScript Client Object Model

var siteCollection;

function getSiteCollection

{

 var ctx = new SP.ClientContext("/");

 siteCollection = ctx.get\_site;

 ctx.load(site);

 ctx.executeQueryAsync(success, failure);

}

function success {

 string url = siteCollection.get\_url;

}

function failure {

 alert("Failure!");

}

The *ClientContext* class in the Managed, Silverlight, and Mobile object models inherits from the *ClientContextRuntime* class. By using the *ClientContext* class, you can get a valid run-time context by passing in the URL of a site. In addition, this class provides several members that are needed to access data and invoke methods on the server.

The *SP.ClientContext* class in the JavaScript client object model inherits from the *SP.ClientContextRuntime*class and provides equivalent functionality to the *ClientContext* class found in the Managed, Silverlight, and Mobile client object models. As with the Managed and Silverlight models, you can get a run-time context in the JavaScript model by using the *SP.ClientContext* class and passing a URL. Unlike the other client object models, however, the JavaScript model also allows you to get a run-time context to the current site by using a constructor with no arguments, so the example above could be rewritten as simply *var ctx = new SP.ClientContext*.

All four client-side object models only communicate with the SharePoint server when the code calls the *ExecuteQuery* or *ExecuteQueryAsync* method. This is to prevent the object models from making too many calls to the SharePoint server and from affecting the SharePoint server’s health by querying the server too much. This means that when you are writing your code, you have to really think about when the statements that you are writing actually have to be executed on the server. You will want to minimize traffic to the server, but you will need to communicate with the server if you want to request data from, or send data into, the SharePoint environment.

The *ExecuteQuery* method creates an XML request and passes it to the Client.svc service. The client then waits synchronously while the batch is executed and the JSON results are returned. The *ExecuteQueryAsync*method, which is used in the Silverlight and Mobile client object models, sends the XML request to the server, but it returns immediately. Designated success and failure callback methods receive notification when the batch operation is complete.

The JavaScript model works like the Managed and Silverlight models by loading operations and executing batches. In the case of the JavaScript model, however, all batch executions are accomplished asynchronously. This means that you must call the *ExecuteQueryAsync* method and pass in the name of functions that will receive success and failure callbacks, as shown earlier in Listing 2-4.

**Using the REST APIs**

The most lightweight option for performing relatively simple operations on data in SharePoint lists and sites is to use the REST capabilities that are built into SharePoint 2013. The SharePoint 2013 implementation of a REST web service uses the Open Data Protocol (OData) to perform CRUD operations on data in SharePoint. Using REST allows your code to interact with SharePoint by using standard HTTP requests and responses. Table 2-3 shows the mapping between HTTP verbs and data operations.

Table 2-3 Mapping between HTTP verbs and data operations

| **HTTP verb** | **Data operation** |
| --- | --- |
| *GET* | Retrieve |
| *POST* | Create |
| *PUT* | Update all fields |
| *DELETE* | Delete |
| *MERGE* | Update specified fields |

The Client.svc web service handles the HTTP request and serves a response in either Atom or JSON format.

To access any object on a site by using a RESTful call, the URL you should use will start with the following construction:

http://<server>/<site>/\_api

To access an actual object within the site you simply add the object to the URL:

//Access a site collection

http://<server>/<site>/\_api/site

//Access a specific site

http://<server>/<site>/\_api/web

//Access a list in a specific site

http://<server>/<site>/\_api/web/lists('GUID')

You can use the querystring syntax to specify parameters for the methods that you call by using a RESTful HTTP request:

//Apply a "blank" site site definition to a SharePoint site

http://<server>/<site>/\_api/web/applyWebTemplate?template="STS#1"

The query strings can become rather complex, but because of that the queries that can be performed are rather powerful as well. You can select, sort, page, filter, and expand data by using a RESTful query. The filtering allows both numeric and string comparisons as well as date and time comparisons. The next example of a RESTful query requests the FirstName, LastName, and PhoneNumber columns from a list with a specific GUID and filters the items by items where the FirstName starts with an *a*:

http://<server>/<site>/\_api/web/lists('GUID')/items?$select=FirstName,LastName,

PhoneNumber$filter=startWith(FirstName, a)

# **Transform farm solutions to the SharePoint Add-in model**

If you have extended your SharePoint environment by using farm solutions, and you want to migrate your extensions to the SharePoint Add-in model to make your transition to SharePoint Online easier, you need to transform your farm solutions to the SharePoint Add-in model.

Transforming your farm solutions to the SharePoint Add-in model involves analyzing your existing extensions, designing and developing your new SharePoint Add-in, and then testing and deploying your add-in in your production environment.

This article describes the process and best practices to use when you transform your farm solutions to the SharePoint Add-in model.

## Planning the transformation process

When you transform your farm solutions to the SharePoint Add-in model, you want to ensure that the impact on your users is minimal. Carefully analyze your current farm solutions, and then design your new SharePoint Add-in to meet the needs of your organization. We recommend the following process to ensure a successful transformation.

### Readiness

Learn about:

* The SharePoint Add-in model, different kinds of add-ins, and hosting options. For more information, see [SharePoint Add-ins](https://docs.microsoft.com/en-us/sharepoint/dev/sp-add-ins/sharepoint-add-ins).
* Remote access technologies for accessing your on-premises data.

### Solution assessment

Analyze the functional and business requirements by:

* Identifying deployed farm solutions in your current environment. Consider using third-party tools to help identify deployed extensions. Perform a detailed analysis of each farm solution identified.
* Reviewing requirements with your users. Consider asking your users to demonstrate how they use the existing farm solutions to perform their daily work.
* Identifying, documenting, and designing new functionality to include in the new SharePoint Add-in. Consider reviewing your list of new feature requests from your users for additional ideas.
* Identifying unused features, and agreeing with your users to omit this functionality from the new SharePoint Add-in.
* For each farm solution, determining whether to replace it with a SharePoint Add-in. Some solutions, such as SharePoint administration extensions, cannot be duplicated in the SharePoint Add-in model. For more information, see [SharePoint Application Lifecycle Management](https://docs.microsoft.com/en-us/sharepoint/dev/general-development/sharepoint-server-application-lifecycle-management) and [SharePoint Add-ins compared with SharePoint solutions](https://docs.microsoft.com/en-us/sharepoint/dev/general-development/sharepoint-add-ins-compared-with-sharepoint-solutions).

### Solution planning

Design the new application by using the SharePoint Add-in model based on:

* The requirements gathered in the **Solution assessment** step.
* Your analysis of the existing code. During your code analysis, consider identifying portions of the code that can be dropped (for example, the code is no longer being used, or the requirements have changed).

### Develop and test the SharePoint Add-in model version of your application

This is usually the most time-consuming step in the transformation process.

### Deploy your new add-in

Depending on your requirements, you might decide to keep the farm solutions running in parallel to the new SharePoint Add-in, or you may retract the farm solution and only allow users to use the new SharePoint Add-in. In either scenario, ensure that your deployment is stable, and send appropriate communication to your users.

If your content in existing site collections depended on your farm solutions (for example, if content was created by using a content type), before you fully retract the farm solution, you need to transform your existing content to use your new SharePoint Add-in model solution. Ensure that you allow enough time to complete this task because it can be time-consuming and difficult.

## Transformation approaches to deploy your new SharePoint Add-in

After you finish development and unit testing of your new SharePoint Add-in, start transforming your farm solution to the new SharePoint Add-in by using one of the transformation approaches listed in the following table.

| **Transformation approach** | **Description** | **Advantages** | **Disadvantages** |
| --- | --- | --- | --- |
| In-place | Deploy your new SharePoint Add-in into your existing SharePoint environment.You must ensure that your site is using the new SharePoint Add-in before retracting the farm solution. | * Less overall user impact.
* Fewer resources needed because you are using your existing SharePoint environment.
* No need for third-party tools.
* Minimal site downtime.
* Upgrade one site collection at a time, rather than upgrading the entire farm all at once.
* URLs do not change.
 | * Difficult to track completion progress of all affected assets on a site.
* Increased chance of creating orphans (when an asset points to a file on the file system that does not exist, this is referred to as an orphan).
 |
| Swing or content migration | Extract your content from your existing site collections where your farm solutions are currently deployed, and deploy the content in a new site collection that uses the new SharePoint Add-in.When you migrate content to SharePoint Online, this process is normally used. | * Clean SharePoint environment with no previous farm solution dependencies.
* The new site collection is isolated from your production environment. Release the updated site collection when ready.

. | * Requires third-party tools to help with the content migration.
* Requires an additional SharePoint environment.
* Site downtime required.
* URLs might change if you keep both sites running in parallel for a period of time.
 |

## Best practices for specific farm solutions

Apply the following best practices when transforming specific solutions.

### Page layouts and master pages

Custom page layouts and master pages might exist on publishing sites or team sites with the publishing features turned on.

To replace page layouts and master pages:

1. Upload the new page layout or master page to your site. Upload new master pages and page layouts to your site collection either manually or by using remote APIs. Remote APIs include the client-side object model (CSOM) or REST. This ensures that the master pages and page layouts do not have dependencies on a farm solution.
2. Configure your site to use the new page layouts and master pages.
3. Retract the previous version of the page layouts and master pages.

### Web parts and controls

To replace web parts and controls:

1. Scan all your existing pages to determine which pages have web parts.
2. (Optional) Review out-of-the-box web parts to determine if any can replace your custom web part.
3. Replace existing web parts with app part instances or by using other techniques (such as embedded JavaScript in pages or page layouts) to achieve the same functionality.
4. Use embedded JavaScript to manipulate UI elements.

**Note**

To replace your existing web parts with app parts, you need to:

* Enable sideloading of add-ins in your Office 365 subscription. Consult with your Office 365 administrator.
* Use CSOM to enable sideloading of add-ins on your site. For more information, see the Core.SideLoading code sample.
* Install your app part on your site.
* Disable sideloading of add-ins on your site.
* Disable sideloading of add-ins on your Office 365 subscription. Consult with your Office 365 administrator.

### Page manipulation

You might need to implement page manipulation during your custom site provisioning process. The [Provisioning.Pages](https://github.com/SharePoint/PnP/tree/master/Samples/Provisioning.Pages) code sample shows page manipulation techniques, including creating a wiki page, adding HTML content to the page, creating a promoted links list, creating pages with different layouts, adding out-of-the-box web parts to the page, and removing the page.

### Site columns, list definitions, and content types

If your site columns, list definitions, and content types were created using the Feature Framework elements, which were deployed using farm solutions, you must use the swing or content migration transformation approach. This does not apply to Feature Framework elements deployed by using sandbox solutions. To use the content migration transformation approach, you must use third-party tools to remove the farm solution dependencies.

### Modules or Feature Framework

Modules use pointers to files, which means that the files are not customized and are deployed on the file system. If your farm solutions use modules, customize the files by deploying alternate versions of the same files to the content database, scan and update your solutions to point to the new files stored in the content database, and then retract the farm solution that pointed to files stored on the file system.

### Site templates and web templates

You should focus on transforming Feature Framework elements deployed by the site template or web template. For example, ensure that the default.aspx page of the site is not replaced when retracting the farm solution.

### Timer jobs

If you are using SharePoint Online, you cannot create and manage timer jobs. Instead, you can create a console application that uses Windows Task Scheduler or an [Azure WebJob](https://docs.microsoft.com/en-us/azure/app-service/web-sites-create-web-jobs) to schedule and run the console application remotely.

When creating a custom timer job, determine whether you need to use a specific account or an OAuth-based app-only token. The [Core.TimerJobs.Samples](https://github.com/SharePoint/PnP/tree/master/Solutions/Core.TimerJobs.Samples) code sample shows how to create your own custom timer job.

**Note**

If your timer job uses server-side code, you must redesign your timer job to use the CSOM or another method.

## In this section

| **Article** | **Shows you how to** |
| --- | --- |
| [Replace content types and site columns](https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/replace-sharepoint-content-types-and-site-columns) | Use CSOM to replace SharePoint content types and site columns, add site columns to new content types, and replace the content types with new content types. |
| [Replace files deployed using modules](https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/replace-files-deployed-using-modules-in-sharepoint-farm-solutions) | Replace files, like master pages and page layouts in SharePoint, that were deployed using modules in farm solutions by uploading and updating references to use new files. |
| [Replace lists created from list definitions](https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/replace-sharepoint-lists-created-from-list-definitions) | Replace lists and libraries that were created by using list definitions in SharePoint. |
| [Replace web parts](https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/replace-sharepoint-web-parts-with-add-in-parts) | Use the transformation process to replace web parts with add-in parts by using the SharePoint client object model (CSOM). |