Mobile SDK for Xamarin - Web API 1.0

All the official Sitecore documentation.

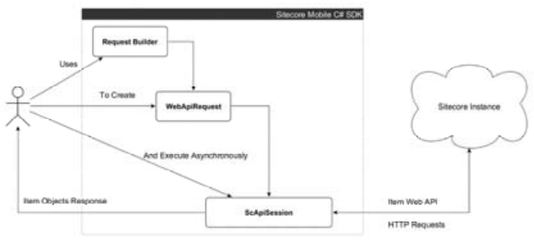
sitecore®
Own the experience™

Accessing Sitecore content

The Sitecore Mobile SDK serves as an interface that connects the Sitecore Item Web API service with an application that lets users work with Sitecore items and their fields. Although the Mobile SDK uses a RESTful request-response model, it lets developers work at a higher level of abstraction than HTTP requests and JSON responses.

To retrieve Sitecore content, you build an Item Web API request object, send the request using the session object, and wait for the response.

The diagram represents the process of accessing Sitecore content using the Mobile SDK:



To process requests asynchronously, the Mobile SDK uses the `async` and `await` features of the .NET 4.5 Runtime. This enables the processing of all the requests using the same operation flow, which consists of the following steps:

- Authenticating
- Constructing an HTTP request
- Networking
- Parsing the response data

The Mobile SDK does not cache the content downloaded from a Sitecore instance. You should implement your own persistence logic for your application.

Send feedback about the documentation to docsite@sitecore.net.

Building item requests

When you have configured a request with all the necessary parameters, you can build the request. Invoke the `Build()` method to get the request object.

```
var request =
ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
  .Database("master")
  .Language("fr")
  .AddFieldsToRead("Name")
  .AddFieldsToRead("Photo")
  .AddScope(ScopeType.Self)
  .AddScope(ScopeType.Parent)
  .Payload(PayloadType.Content)
  .Build();
```

When you build a request, you must observe the following rules:

- Define the item identification parameter first. Define optional parameters in any order.

  Correct requests:

  ```
  ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("master")
    .Language("fr")
    .Build();
  ```

  Or:

  ```
  ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Language("fr")
    .Database("master")
    .Build();
  ```

  An incorrect request:

  ```
  var request = ItemWebApiRequestBuilder.Database("master")
    .ReadItemsRequestWithPath("/sitecore/content/home")
    .Build();
  // This will not compile
  ```

- The `Add…` methods accumulate values, so consider the invocation order.

  For example, the following request retrieves the *Home* item and its two fields: the Name field and the Photo field.

  ```
  ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .AddFieldsToRead("Name")
    .AddFieldsToRead("Photo")
    .Build();
  ```

- Invoke methods that don't accumulate values only once.

  A correct request:

  ```
  ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("web")
    .Language("en")
    .Build();
  ```

  An incorrect request that throws *InvalidOperationException*:

  ```
  ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
    .Database("master")
    .Language("fr")
    .Database("web")
    .Language("en")
    .Build();
  ```

Send feedback about the documentation to docsite@sitecore.net.

Creating items using the Mobile SDK

The Mobile SDK lets you create new Sitecore items via the API.

To create a new item, you must pass a request that specifies item parameters to the `ISitecoreWebApiSession.CreateItemAsync()` method.

A sample item creation request:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithId("{item-id-GUID}")
  .ItemTemplate(“Sample/Sample Item”)
  .ItemName(“Name of new item”)
  .Build();
var createResponse = await session.CreateItemAsync(request);
```

When you construct a request to create a Sitecore item, you must:

- Set the item location.
- Set the item name.
- Set the item template.
- (Optionally) Set the item content.

You must specify the following parameters to create an item:

- Item name
- Item template
- Parent item

Optionally, you can also specify the following item parameters:

- Database
- Language
- Item fields

Setting the item location

To create a new item in Sitecore, you must specify its location in the content tree.

To set the item location, define its parent item using one of the following methods:

- Define the parent item by ID: `ItemWebApiRequestBuilder.CreateItemRequestWithParentParentId()`

  ```
  var request =
  ItemWebApiRequestBuilder.CreateItemRequestWithParentParentId("{item-id-GUID}")
                                        .ItemTemplatePath(“Sample/Sample Item”)
                                        .ItemName(“Name of new item”)
                                        .Build();
  ```

- Define the parent item by path: `ItemWebApiRequestBuilder.CreateItemRequestWithParentParentPath()`

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentParentPath("/path/to/parent")
                        .ItemTemplatePath(“Sample/Sample Item”)
                        .ItemName(“Name of new item”)
                        .Build();
```

The Mobile SDK validates the ID and path of the parent item using the same corresponding parameters as the item retrieval request.

In addition to the parent item, you can specify the low-level storage parameters, such as, the item language and the database to store the item in.

Setting the item name

In order for the content of items to be accessible, each Sitecore item must have a name.

To specify the name of the item, use the `ItemName()` method.

Property	Details
Mobile SDK method	`ItemName(string)`
Item Web API parameter	`name`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters. You cannot use the following symbols in the item name:
  - A period “.”
  - A dash “-“
  - A slash “/”

  To use the item name in both queries and XPATH expressions, you must conform to the value validation rules. Using forbidden symbols in the item name causes an exception, for example:

  ```
  ItemWebApiRequestBuilder.CreateItemRequestWithPath(parentPath)
                          .ItemName(“/item/sub-path”) // throws ArgumentException
  ```

The item name must be unique within a given folder. If you specify an existing name, the Item Web API increments the item name automatically.

Setting the item template

The item template defines the list of available fields of the item.

To set the template that you want new items to be based on, specify one of the following parameters:

- Template path
- Template ID
- Branch ID

Important

These parameters are mutually exclusive. To construct a single request, you must only use one of them. If you use more than one parameter in a single request, you get a compiler error.

Specifying the template path

To identify the item template by path, use the `ItemTemplatePath()` method.

Property	Details
Mobile SDK method	`ItemTemplatePath(string)`
Item Web API parameter	`template`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must not begin with a forward slash and must contain at least one symbol besides whitespace characters.

Sitecore stores all the templates in the */sitecore/templates* folder. Always use a template path that is relative to this template root folder and do not begin the path with a leading forward slash.

A sample request:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
```

```
                                      .ItemTemplatePath(“Sample/Sample Item”)
                                      .ItemName(“Name of new item”)
                                      .Build();
```

Specifying the template ID

To identify the item template by its ID, use the `ItemTemplateId()` method.

Property	Details
Mobile SDK method	`ItemTemplateId(string)`
Item Web API parameter	`template`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol enclosed in braces.

A sample request:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                                      .ItemTemplateId(“{GUID-ITEM-ID}”)
                                      .ItemName(“Name of new item”)
                                      .Build();
```

Specifying the branch ID

To identify the item template by its branch ID, use the `BranchId()` method.

Property	Details
Mobile SDK method	`ItemTemplateId(string)`
Item Web API parameter	`template`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol enclosed in braces.

A sample request:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                                      .BranchId(“{GUID-ITEM-ID}”)
                                      .ItemName(“Name of new item”)
                                      .Build();
```

Setting the item content

To compile a list of field names and raw values for the item that you create, use the `AddFieldsRawValuesByNameToSet()` method in the item creation request.

Property	Usage guidelines

Mobile SDK method	`AddFieldsRawValuesByNameToSet(string key, string rawValue)`
	`AddFieldsRawValuesByNameToSet(IDictionary)`

Item Web API parameter	The request body

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values.
- You cannot submit duplicate values.
- Both the key and the value must contain at least one symbol besides whitespace characters.

To submit key-value pairs for the item fields, use one of the following approaches:

- Submit values one at a time:

```
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                  .ItemTemplatePath(“Sample/Sample Item”)
                  .ItemName(“new item”)
                  .AddFieldsRawValuesByNameToSet("Title", “Device”)
                  .AddFieldsRawValuesByNameToSet("Text", “Smartphone”)
                  .Build();
```

- Append a dictionary that contains the values:

```
// set up fields beforehand
var fields = new Dictionary<string, string>();
fields.Add(“Title”, “Device”);
fields.Add(“Text”, “Smartphone”);
//
var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)
                  .ItemTemplatePath(“Sample/Sample Item”)
                  .ItemName(“new item”)
                  .AddFieldsRawValuesByNameToSet(fields)
                  .Build();
```

Send feedback about the documentation to docsite@sitecore.net.

Delete an item using the Mobile SDK

The Mobile SDK lets you delete items in the Sitecore content tree from a .NET application via the API.

To delete an item:

1. Using one of the following request interfaces, identify the item that you want to delete:
   - `IDeleteItemsByIdRequest` – identify the item by ID.
   - `IDeleteItemsByPathRequest` – identify the item by path.
   - `IDeleteItemsByQueryRequest` – identify the item by Sitecore query or Fast query.
2. If you want to remove the item from a particular database, use the `sc_database` parameter.

   The following request removes all the versions of the item in all the languages from the Master database:

   ```
   var request = ItemWebApiRequestBuilder.DeleteItemRequestWithPath(itemPath)
                                         .Database(“master”)
                                         .Build();
   ```

   Do not use the `language` parameter to add or delete a language version of an item because the Item Web API does not support this functionality.
3. If you want to remove the parent or children items, use the `scope` parameter:

   ```
   var request = ItemWebApiRequestBuilder.DeleteItemRequestWithSitecoreQuery(queryString)
                        .AddScope(ScopeType.Self, ScopeType.Children)
                        .Build();
   ```
4. Pass the request to the session:

   ```
   var request = ItemWebApiRequestBuilder.DeleteItemRequestWithPath(itemPath)
                                         .Database(“master”)
                                         .Build();
   ScItemsResponse response = await session.DeleteItemAsync(request);
   ```

Send feedback about the documentation to docsite@sitecore.net.

Executing requests

When you request Sitecore items, all the requests are executed asynchronously by the *ISitecoreWebApiSession* interface instance. The SDK contains the corresponding class, but it should not be instantiated directly.

ISitecoreWebApiSession provides three overloads for the `ReadItemAsync()` method to process the requests based on one of the item identification parameters - *IReadItemsByIdRequest*, *IReadItemsByPathRequest* and *IReadItemsByQueryRequest*. For example:

```
ScItemsResponse response = await session.ReadItemAsync(request);
```

If you cannot use the `async` keyword with your method, use the `Task.WaitAll()` method instead:

```
var readHomeItemTask = session.ReadItemAsync(request);
```

```
Task.WaitAll(readHomeItemTask);
```

```
ScItemsResponse items = readHomeItemTask.Result;
```

To create, edit or remove items, use the following session methods:

- `CreateItemAsync` – for creating items.
- `UpdateItemAsync` – for editing items.
- `DeleteItemAsync` – for removing items.

These methods have similar overloads for requests based on different item identification parameters.

Identifying Sitecore items

You can use the Mobile SDK to create, retrieve, delete, and update items in the Sitecore instance. To request a specific item, you should identify it by one of the following parameters:

- Item ID
- Item Path
- Sitecore Query or Fast Query

You submit the item identification parameters as string objects, so you can submit any values. However, to be correctly processed by the Item Web API service, they must conform to a set of validation rules.

When these item identifications parameters are not sufficient to identify the item precisely, you can specify the item source.

The Mobile SDK does not let you request an item by more than one parameter at a time, therefore, there are three separate request objects for retrieving items:

- IReadItemsByIdRequest
- IReadItemsByPathRequest
- IReadItemsByQueryRequest

You can also use the ID, path, and query parameters when you generate requests to create, remove, and update items.

All requests are immutable read-only interfaces. To generate these requests, use the `ItemWebApiRequestBuilder` class:

```
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithId(id).Build();
```

```
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(path).Build();
```

```
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithSitecoreQuery(query).Build();
```

Validating the Item Identification Parameters

The Item Web API service cannot process certain values of the item parameters, for example, it cannot read an item that you do not have access to or locate an item by an incorrectly formatted parameter. To prevent errors, the Mobile SDK validates the submitted item parameters before passing them to the Item Web API service. It performs the validation on the client whenever possible. If the validation fails, the SDK throws an exception.

The following validation rules are applied to item identification parameters:

- Item ID – the string should contain a GUID, which is a 128-bit number separated by hyphens and enclosed in curly braces:

  ```
  const string itemId = "{3D6658D8-QQQQ-QQQQ-B3E2-D050FABCF4E1}";
  ```

  The braces and four hyphens are required. If you omit the braces, the request builder throws the *ArgumentException*. If you pass a string that is null, empty, or consists only of whitespace characters, the request builder throws an exception. In case of other errors, the request is sent to the server, which returns an appropriate response or an error.

- Item Path – the absolute path to the item, beginning with a forward slash:

  ```
  const string itemPath = "/sitecore/content/Home";
  ```

  The leading forward slash is required. If you omit the slash, the request builder throws an *ArgumentException*. If you pass a string that is null, empty, or consists only of whitespace characters, the request builder throws an exception.

- Sitecore Queries - the query syntax is not validated on the client. The request builder only verifies that the value is not a null, empty, or consisting only of whitespace characters.

Specifying the item source

The item identifications parameters such as the item ID or the item path may be not sufficient to identify the item precisely. The content you retrieve based on these parameters may differ depending on the database in which Sitecore stores the item as well as its language and numeric version.

To specify the item source, use the following item request configuration parameters:

- `sc_database` – to specify a database.
- `language` – to specify a language version.
- `sc_itemversion` – to specify a numeric version.

You can configure the item source in the following ways:

- Per request on the client – the item source parameters are defined in the item request. The values from the request are a top priority.

- Per session on the client – if the request doesn't contain the item source parameters, the SDK uses the values from the *ISitecoreWebApiSession.DefaultSource* property of the current session.
- On the server – if neither the request nor the session contains the item source parameters, the SDK does not pass any values to the server. In this case, the server uses its own default values configured by the administrator to define the item source.

The item source is an important parameter that affects both item retrieval and caching. Therefore, you should specify the item source explicitly to always control where the content comes from.

However, to ensure that you retrieve the latest version of the item, you can omit the `sc_itemversion` parameter.

Send feedback about the documentation to docsite@sitecore.net.

Reading new or updated items

When you create or update an item using the Mobile SDK, you can start using the newly created or updated item immediately, without additional Item Web API calls.

When you configure an item request, you can specify the list of item fields that you want to retrieve using one of these methods:

- The `AddFieldsToRead()` method:

  ```
  var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath(parentItemPath)

                    .ItemTemplatePath("Sample/Sample Item")

                    .ItemName(expectedItem.DisplayName)

                    .AddFieldsRawValuesByNameToSet("Title", "test")

                    .AddFieldsRawValuesByNameToSet("Text", "test")

                    .AddFieldsToRead("Text", "Title")

                    .Build();
  ```

```
var request = ItemWebApiRequestBuilder.UpdateItemRequestWithPath(parentItemPath)

                  .AddFieldsRawValuesByNameToSet("Title", "test")

                  .AddFieldsRawValuesByNameToSet("Text", "test")

                  .AddFieldsToRead("Text", "Title")

                  .Build();
```

- The `Payload()` method:

  ```
  var request = ItemWebApiRequestBuilder.CreateItemRequestWithParentPath (parentItemPath)

                    .ItemTemplatePath ("Sample/Sample Item")

                    .ItemName(expectedItem.DisplayName)

                    .AddFieldsRawValuesByNameToSet("Title", "test")

                    .AddFieldsRawValuesByNameToSet("Text", "test")

                    .Payload(PayloadType.Content)

                    .Build();
  ```

```
var request = ItemWebApiRequestBuilder.UpdateItemRequestWithPath(parentItemPath)

                  .AddFieldsRawValuesByNameToSet("Title", "test")

                  .Payload(PayloadType.Content)

                  .Build();
```

Send feedback about the documentation to docsite@sitecore.net.

Retrieve a Sitecore item

You can use the Mobile SDK to read your Sitecore items and use them in client .NET applications.

To retrieve a Sitecore item:

1. Identify the item that you want to retrieve using one of the following request interfaces:
   - `IReadItemsByIdRequest` – to identify the item by ID.
   - `IReadItemsByPathRequest` – to identify the item by path.
   - `IReadItemsByQueryRequest` – to identify the item by Sitecore query or Fast query.
2. You can specify the following optional parameters for the requested item:
   - Database
   - Language
   - Fields to read
   - Scope
   - Payload
3. Pass a request that specifies item parameters to the `ISitecoreWebApiSession.ReadItemAsync()` method.

   ```
   var request =
   ```

```
    ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
      .Database("master")
      .Language("fr")
      .Build();
    ScItemsResponse response = await session.ReadItemAsync(request);
```

Send feedback about the documentation to docsite@sitecore.net.

Retrieving media content using the Mobile SDK

To retrieve the media content using the Mobile SDK, you should construct and pass a request to a corresponding session building method.

```
// constructing a request
string mediaPath = "/Images/green_mineraly1";
var request = ItemWebApiRequestBuilder.DownloadResourceRequestWithMediaPath(mediaPath).Build();
// processing the request
using ( Stream response = await this.session.DownloadMediaResourceAsync(request) )
{
  // working with the media resource stream
}
```

Classes that process images differ between platforms, for example:

Platform	Class name
iOS	`UIImage`
Android	`Bitmap`
Windows Phone	`BitmapImage`

Also, the requested media content may be not an image. For these reasons, the user receives the media content stream after executing the request.

Note

To avoid memory and resource leaks, you should wrap both the stream and the image class in a `using` block.

Setting Media Path

When you request a media resource, the first parameter that you should pass to the request builder is the resource location.

The request builder accepts the following media paths:

- The path relative to the Media Library root folder, which is the raw value of the Image field:

  `"/Images/green_mineraly1"`

- The absolute path of a media item in the Content Editor:

  `"/Sitecore/Media Library/Images/green_mineraly1"`

- The media URL fragment copied from a web browser:

  `"~/media/Images/green_mineraly1.ashx"`

  This fragment must begin with the media hook that has been specified at the time of session initialization.

You can use an absolute path or a media URL fragment for prototyping. However, the safest approach is to use a relative media path.

Send feedback about the documentation to docsite@sitecore.net.

The connection endpoint parameters

When you start configuring a session, you should use the connection endpoint parameters to define a host, authentication credentials, and target site.

The Mobile SDK provides the following connection endpoint parameters:

- Instance URL
- Credentials
- Site

- Web API version

The Instance URL parameter

The `Instance URL` parameter defines a session host – so it is the first parameter that you use to invoke a session initializer.

Property	Details
Session builder API	`AnonymousSessionWithHost(string)` `AuthenticatedSessionWithHost(string)`
Sample value	`http://my-site.com:80`
Parameter position	First in a sequence

Usage guidelines:

- This parameter is required for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations do not compile.
- The value must contain at least one symbol besides whitespace characters.
- If you do not specify the URL scheme in the `Instance URL` parameter, the default value is `http`. For example, `my-site.com` is resolved as http://my-site.com.

The Credentials parameter

To restrict the access to your content, you should set up user permissions in the Security Editor and pass the user name and password to an authenticated session using the `Credentials` parameter:

```
var session =
SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
         .Credentials(loginAndPasswordProvider)
    .BuildSession();
```

To build an anonymous session, skip the `Credentials` parameter:

```
var session =
SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)
.BuildSession();
```

Property	Details
Session builder API	`Credentials(IWebApiCredentials)`
Sample value	n/a
Parameter position	Immediately after the `Instance URL` parameter

Usage guidelines:

- This parameter is required for an authenticated session.
- You cannot invoke this parameter several times in a session – multiple invocations do not compile.
- The value must contain at least one symbol besides whitespace characters.

You can enter the user login in the following forms:

- The full form – the user login containing the domain information, for example, `sitecore/admin`.
- The short form – the user login without the domain information, for example, `admin`. In this case, you should specify the site that the user logs in to.

Sitecore Mobile SDK accepts the credentials via the `IWebApiCredentials` interface. You can either use the Sitecore password provider or implement a custom credentials provider.

The Site parameter

The `Site` parameter lets you retrieve data from the different sites running on your Sitecore instance:

```
SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(“http://my-host.com”)
  .Credentials(loginAndPasswordProvider) //contains username and password
  .Site("/sitecore/shell") //assume the user is registered at the “sitecore” domain
  .BuildSession();
```

Property	Details
Session builder API	`Site(string)`
Sample value	`"/sitecore/shell"`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must begin with a forward slash.

The `Site` parameter defines the instance configuration, which affects the following settings:

- Valid database values
- Valid language values
- Default user domain if not specified in the login string
- Default database if not specified either in the session or in the request
- Default language if not specified either in the session or in the request

The Web API Version parameter

Use the `WebApiVersion` parameter to only target a legacy version of the service:

```
var session =

SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)

                            .WebApiVersion("v1")

                            .BuildSession();
```

Property	Details
Session builder API	`WebApiVersion(string)`
Sample value	`"v1"`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

You should always use the latest version of both the Mobile SDK and the Item Web API service, so you should not have to set this parameter explicitly.

Note

So far, all the existing releases of the Item Web API have version 1 and use the "v1" value.

Send feedback about the documentation to docsite@sitecore.net.

The HTML rendering request

To display the content of an item without using native controls, you can retrieve its HTML representation and display it in a web browser control.

To get HTML rendering, use the `ISitecoreWebApiSession.ReadRenderingHtmlAsync()` method:

```
var request = ItemWebApiRequestBuilder.RenderingHtmlRequestWithSourceAndRenderingId(sourceGUID, renderingGUID)

            .Build();
```

To retrieve the rendering, you must specify the values of the following fields:

- Source ID
- Rendering ID

This allows you to get an instance of the `Stream` class to read a string of rendered HTML.

The following table lists the optional parameters that you can specify in a rendering request and the corresponding methods to use:

Parameter	Method	Details

The database of the source item and the rendering	`SourceAndRenderingDatabase()`	The methods work similarly to the `Database()` and `Language()` methods for other request types. The difference is that in rendering requests, you specify the database and the language version for both the source item and its rendering.
The language version of the source item and the rendering	`SourceAndRenderingLanguage()`	
The version of the source item	`SourceVersion()`	The method works similarly to the `Version()` method for other request types.
The key and value pair of the rendering parameter	`AddRenderingParameterNameValue()`	The method enables you to add HTML rendering parameters to rendering requests.

A sample request:

```
var request = ItemWebApiRequestBuilder.RenderingHtmlRequestWithSourceAndRenderingId("{SOURCE-ITEM-ID}","{RENDERING-ID}")

            .SourceAndRenderingDatabase("web")

            .SourceAndRenderingLanguage("en")

            .SourceVersion(1)

            .AddRenderingParameterNameValue("parameter1", "value1")

            .Build();
```

If the item with the specified parameters is absent, you retrieve the item with default parameters.

HTML rendering parameters

To add HTML rendering parameters, use the `AddRenderingParameterNameValue()` method.

Property	Details

Mobile SDK method name	`AddRenderingParameterNameValue(string parameterName, string parameterValue)`
	`AddRenderingParameterNameValue(IDictionary<string, string> parametersValuesByName)`

Item Web API parameter	`parameterName = parameterValue`

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values.
- You cannot submit duplicate values.
- Both the key and the value must contain at least one symbol besides whitespace characters.

To submit key-value pairs for rendering parameters, use one of the following approaches:

```
var request = ItemWebApiRequestBuilder.RenderingHtmlRequestWithSourceAndRenderingId(sourceGUID, renderingGUID)

            .AddRenderingParameterNameValue("parameter1", "value1")

            .AddRenderingParameterNameValue("parameter2", "value2")

            .Build();
```

Or:

```
var parameters = new Dictionary<string, string>(); //specify parameters

parameters.Add("parameter1", "vapue1");

parameters.Add("parameter2", "value2");

var request = ItemWebApiRequestBuilder.RenderingHtmlRequestWithSourceAndRenderingId(sourceGUID, renderingGUID)

            .AddRenderingParameterNameValue(parameters) //set parameters

            .Build();
```

The item request configuration parameters

Sitecore Mobile SDK includes functionality to create, retrieve, delete, and update items in the Sitecore instance. The request builder syntax is identical for all the types of item requests. However, the scope of applicable methods varies depending on the operation type. This topic outlines the usage guidelines for the item request methods and parameters.

The Mobile SDK provides a set of methods to define the following properties:

- Database
- Language
- Version
- Payload
- Fields
- Scope
- Paging

The Database (sc_database) method

To configure the database to get items from, use the `Database()` method.

Property	Details
Mobile SDK method	`Database(string)`
Item Web API parameter	`sc_database`

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The Language (language) method

To define the item language, use the `Language()` method.

Property	Details
Mobile SDK method	`Language(string)`
Item Web API parameter	`language`

Usage guidelines:

- The `language` parameter is not applicable to item deletion requests.
- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The Version (sc_itemversion) method

To define the item version, use the `Version()` method. To retrieve the latest version of an item, do not invoke this method.

Property	Details
Mobile SDK method	`Version(int?)`
Item Web API parameter	`sc_itemversion`

Usage guidelines:

- Do not use the `sc_itemversion` parameter to:
  - Resolve a set of items by a query.
  - Add or delete an item or a particular item version.
- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- You can only use null or integer numbers as the value.
- Pass a positive number to get a specific version of an item.

The Payload (payload) method

To define the field list that comes with the server response, use the `Payload()` method.

Property	Details
Mobile SDK method	`Payload(PayloadType)`
Item Web API parameter	`payload`

Usage guidelines:

- The `payload` parameter is not applicable to item creation and deletion requests.
- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- You can only use the `PayloadType` enumerators as the value.

Alternatively, you can list the fields to be read explicitly.

The AddFieldsToRead (fields) method

To explicitly set the item fields to be retrieved, use the `AddFieldsToRead()` method.

Property	Details

Mobile SDK methods	`AddFieldsToRead(string)`
	 `AddFieldsToRead(IList<string>)`

Item Web API parameter	`fields`

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values in invocation order.
- The value must contain at least one symbol besides whitespace characters.
- Field names cannot be null or empty.

The AddScope (scope) method

To retrieve additional content from parent or child items, use the `AddScope()` method.

Property	Details

Mobile SDK methods	`AddScope(ScopeType)`
	 `AddScope(IEnumerable<ScopeType>)`

Item Web API parameter	`scope`

Usage guidelines:

- This is an optional parameter.
- You can invoke this method several times in a request – multiple invocations accumulate values in invocation order.
- You cannot submit duplicate values.
- The value must contain at least one symbol besides whitespace characters.
- Field names cannot be null or empty.

The scope can take the following parameters in any combination:

- `ScopeType.Self`
- `ScopeType.Parent`
- `ScopeType.Children`

The order of the scope parameters in the request object affects the order in which the server returns items.

The content from the scope can either replace that of the item matching the search predicate or query, or it can be appended to the response.

The `scope` parameter is not applicable to item creation requests.

Note

You should not use the scope parameter for query-based requests because it can be difficult to differentiate between items that match the query and items that are in the scope.

The ItemsPerPage (perPage) and PageNumber (page) methods

If the request returns a large dataset, for example, a news feed, a photo stream, or search results, loading the whole content may take a long time and cause memory warnings. To avoid this, the Sitecore Item Web API service and the Mobile SDK enable you to load data as a series of chunks.

To split a server response into chunks, use the `ItemsPerPage()` and `PageNumber()` methods.

Property	Details
Mobile SDK method	`ItemsPerPage(int)`

Item Web API parameter	perPage

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must be a positive number or you get the *ArgumentException* exception.

Property	Details
Mobile SDK method	`PageNumber(int)`
Item Web API parameter	`page`
Value range	0 to TotalItemsCount/ItemsPerPage + 1

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must be a positive number or zero or you get the *ArgumentException* exception.

You must either specify both parameters or omit both of them, otherwise the code does not compile.

```
string query = "/sitecore/media library/images/*";
var request = ItemWebApiRequestBuilder.ReadItemsRequestWithSitecoreQuery(query)
  .AddScope(ScopeType.Self)
  .PageNumber(0)
  .ItemsPerPage(2)
  .Build();
var response = await this.session.ReadItemAsync(request);
```

The paging options are only available for reading requests because other operations on items do not require partial execution.

Send feedback about the documentation to docsite@sitecore.net.

The item source parameters

Because Sitecore stores items in different databases and languages, you may need to construct a session that requests items in a particular language from a particular database. To do that, use the `DefaultDatabase` and `DefaultLanguage` parameters:

```
var session = SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                          .DefaultDatabase("web")
                                          .DefaultLanguage("en")
                                          .BuildSession();
```

The Default Database Parameter

Property	Details
Session builder API	`DefaultDatabase(string)`
Sample value	`“master”`, `“web”`, `“core”`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The Default Language Parameter

Property	Details

Session builder API	`DefaultLanguage(string)`
Sample value	`“en”`, `“da”`, `“fr”`, `“ger”`, `“ja”`, `“cn”`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

Because the builder is order insensitive and both the `DefaultDatabase` and the `DefaultLanguage` parameters are optional, you can specify either both of them, or one of them, or none, for example:

```
var session = SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)

                                          .DefaultDatabase("web")

                                          .BuildSession();
```

Or:

```
var session = SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)

                                          .DefaultLanguage("en")

                                          .BuildSession();
```

Or:

```
var session = SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)

                                          .BuildSession();
```

You can override the default source settings by resetting them in the request object.

Send feedback about the documentation to docsite@sitecore.net.

The media download options

Many mobile applications have a master-detail interface, which displays a master list and the details for the currently selected item.

For media items, this means that when the user selects an item preview from the master list, the higher-quality media is populated in the detail area. If the application downloads, resizes, and reuses original media for every operation, it suffers from lack of memory and it can crash. The correct approach is to load small images for the master view and larger ones for the detailed view.

Sitecore lets you scale images to different sizes on the server before sending the image to an application. When you request a media item, you can apply parameters to modify the output, for example:

```
var options = new MediaOptionsBuilder().Set

                                       .MaxWidth(1920)

                                       .MaxHeight(1080)

                                       .Width(1024)

                                       .Height(768)

                                       .BackgroundColor("red")

                                       .DisableMediaCache(false)

                                       .AllowStrech(false)

                                       .Scale(1.0f)

                                       .DisplayAsThumbnail(false)

                                       .Build();
```

Unlike in item requests, you should pass the image properties to media requests as a complete object.

Note

The API does not support image resizing in Sitecore 8.x.

You can also specify the media item source parameters. You should pass them directly in the request builder because they apply to all media content.

```
foreach (string mediaPath in myArtworkCollection)

{

var request = ItemWebApiRequestBuilder.DownloadResourceRequestWithMediaPath(mediaPath)

                                          .Database("master")

                                          .Language("fr")

                                          .Version(1)
```

```
                                             .DownloadOptions(options)
                                             .Build();
Stream response = await session.DownloadMediaResourceAsync(request);
// working
}
```

The Mobile SDK supports the dynamic properties for media content available in the `sc:image` tag and media URLs.

Dynamic property	Media query parameter	Example Value	MediaOptionsBuilder method	Value validation
Width (in pixels)	w	[0]	Width(int)	Negative values cause *ArgumentException*
Height (in pixels)	h	[0]	Height(int)	Negative values cause *ArgumentException*
Maximum width in pixels	mw	[0]	MaxWidth(int)	Negative values cause *ArgumentException*
Maximum height in pixels	mh	[0]	MaxHeight(int)	Negative values cause *ArgumentException*
Background color	bc	[black]	BackgroundColor(string)	Must contain at least one symbol besides whitespace characters
Disable media cache	dmc	[0]	DisableMediaCache(boolean)	n/a
Allow stretch	as	[0]	AllowStretch(boolean)	n/a
Scale by floating point number	sc	[0] (0.25 = 25%)	Scale(float)	Negative values cause *ArgumentException*
Thumbnail	thn	[0] (= false; 1 = true)	DisplayAsThumbnail(boolean)	n/a

The Mobile SDK supports the following media item source parameters:

Source parameter	Media query parameter	Value	MediaOptionsBuilder method	Value validation
Language	la	[en]	Language(string)	Must contain at least one symbol besides whitespace characters
Version	vs	[1]	Version(int?)	Must be a positive integer number
Database name	db	[web]	Database(string)	Must contain at least one symbol besides whitespace characters

Send feedback about the documentation to docsite@sitecore.net.

The media library configuration parameters

You can use the Mobile SDK to download files from the media library of a Sitecore instance. You can set the media library parameters in the server configuration file or redefine them on the client during the session initialization.

The Mobile SDK provides the following media library configuration parameters:

- The Media Library Root parameter
- The Media Prefix parameter
- The Default Media Resource Extension parameter
- The Media Resizing Strategy parameter

The Media Library Root parameter

To request a media item, the API can use either the full item path or a media path that is relative to the media library root folder. To specify the root folder, use the `MediaLibraryRoot` parameter.

Property	Details
Session builder API	`MediaLibraryRoot(string)`
Sample value	`“/sitecore/media library”`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must begin with a forward slash.

At the time of session initialization, you can change the root folder of the media library that is specified on the server:

```
SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                             .MediaLibraryRoot("/sitecore/media library")
                             .BuildSession();
```

The Media Prefix parameter

The `MediaPrefix` parameter defines the hook that the Mobile SDK uses to load media files.

Note

The `MediaPrefix` parameter is not applicable if you are using Sitecore 7.5 or later.

Property	Details
Session builder API	`MediaPrefix(string)`
Sample value	`“~/media/”`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

The default media hook is `~/media`. You can configure the hook at the time of session creation:

```
SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                             .MediaPrefix("~/media/")
                             .BuildSession();
```

The Default Media Resource Extension parameter

The `DefaultMediaResourceExtension` parameter specifies the extension to use in media request URLs.

Note

The `DefaultMediaResourceExtension` parameter is not applicable if you are using Sitecore 7.5 or later.

Property	Details
Session builder API	`DefaultMediaResourceExtension(string)`
Sample value	`“ashx”`
Parameter position	Any position after the `Instance URL` and `Credentials` parameters

Usage guidelines:

- This parameter is optional for all session types.
- You cannot invoke this parameter several times in a session – multiple invocations cause *InvalidOperationException*.
- The value must not contain a dot before the extension name and must contain at least one symbol besides whitespace characters.

The default media resource extension is `ashx`. You can change this value at the time of session creation:

```
SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                              .DefaultMediaResourceExtension("ashx")
                              .BuildSession();
```

The Media Resizing Strategy parameter

Sitecore lets you scale the same image to different sizes on the server before sending the image to an application.

Depending on the Sitecore version, you should use one of the following image resizing options:

- Plain

  From Sitecore 6.6 to 7.2, Sitecore resizes an image by the media hook in the URL.

  ```
  SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                                .MediaResizingStrategy(DownloadStrategy.Plain)
                                .BuildReadonlySession();
  ```

- Hashed

  In Sitecore XP 7.5, image URLs must have a hash value appended on the query string. The Item Web API provides the hash.

  ```
  SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                                .MediaResizingStrategy(DownloadStrategy.Hashed)
                                .BuildReadonlySession();
  ```

Note

The `MediaResizingStrategy` parameter is not applicable if you are using Sitecore 8.x.

Send feedback about the documentation to docsite@sitecore.net.

The session objects and session parameters

To send Item Web API requests, you need a session object. Sitecore Mobile SDK only provides a public interface for the session. To create a session object, use the `SitecoreWebApiSessionBuilder` class API.

Session objects vary in permission types:

Permission type	Session objects
Security permissions	• Anonymous session • Authenticated session
Operation permissions	• ReadOnly session • ReadWrite session

A session object stores the following session configuration parameters:

Parameter type	Session configuration parameters
The connection endpoint parameters	• Instance URL • Credentials • Site of the Sitecore instance • Web API version
The item source parameters	• Default database • Default language
The Media Library configuration parameters	• Media Library root item • Media prefix • Default extension for media resources • Media resizing strategy

You can pass the parameters one at a time in any order. For example:

```
var session =
SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(“http://my-host.com”)
                            .Credentials(loginAndPasswordProvider)
                            .Site("/sitecore/shell")
                            .WebApiVersion("v1")
                            .DefaultDatabase("web")
                            .DefaultLanguage("en")
                            .MediaLibraryRoot("/sitecore/media library")
                            .MediaPrefix("~/media/")
                            .DefaultMediaResourceExtension("ashx")
                            .MediaResizingStrategy(DownloadStrategy.Plain)
                            .BuildSession();
```

Building sessions

Once you have set all the session configuration parameters, you can invoke a builder method to create the session.

The Mobile SDK provides the following builder methods:

- BuildReadonlySession()

  Use the `BuildReadonlySession()` method to create a read-only session to display the content stored in Sitecore without providing the ability to modify it. When you use this builder method, you must not invoke methods that change the content, or your code does not compile.

  ```
  SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                              .BuildReadonlySession();
  ```

- BuildSession()

  If users need to send data back to the Sitecore instance, use the `BuildSession()` method to create a standard session to allow both read and write access.

  ```
  SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(“http://my-host.com”)
                              .BuildSession();
  ```

Send feedback about the documentation to docsite@sitecore.net.

Update an item using the Mobile SDK

The Sitecore Mobile SDK includes the content editing API that lets you send new raw values for items to the Sitecore instance and edit the content in different databases and languages.

To update an item:

1. Identify the item that you want to update using one of the following request interfaces:
   - `IUpdateItemByIdRequest` – to identify the item by ID.
   - `IUpdateItemByPathRequest` – to identify the item by path.
2. If you want the update to affect a particular language, version, or database, specify the item source:
   - To update the item in a particular database, use the `sc_database` parameter.

   ```
   var request = ItemWebApiRequestBuilder.UpdateItemRequestWithId(“{item-id-GUID}”)
                                         .Database(“master”)
                                         .Build();
   ```

   - To pass a new value to a particular language version of the item, specify the `language` parameter.

   ```
   var request = ItemWebApiRequestBuilder.UpdateItemRequestWithId(“{item-id-GUID}”)
                                  .Language("en")
                                  .AddFieldsRawValuesByNameToSet("Text", “Hello!”)
                                  .Build();
   var request = ItemWebApiRequestBuilder.UpdateItemRequestWithId(“{item-id-GUID}”)
                                  .Language("de")
                                  .AddFieldsRawValuesByNameToSet("Text", “Guten tag”)
                                  .Build();
   ```

If you do not specify a language, the default language version is updated with the new value.

To update a specific version of the item, use the `sc_itemversion` parameter.

```
var request = ItemWebApiRequestBuilder.UpdateItemRequestWithId(“{item-id-GUID}”)
                                  .Language("en")
```

```
                                    .Version(1)
                                    .AddFieldsRawValuesByNameToSet("Text", “Hello!”)
                                    .Build();
```

If you do not specify a version, the latest version of the item is updated with the new value.

Note

The Item Web API does not support creating a new version of the item. You can only modify one of the existing versions.

1. Submit the new field values, using the `AddFieldsRawValuesByNameToSet()` method.

   ```
   var fields = new Dictionary<string, string>();
   fields.Add(“Title”, “Device”);
   fields.Add(“Text”, “Smartphone”);
   var request = ItemWebApiRequestBuilder.UpdateItemRequestWithId(“{item-id-GUID}”)
                    .AddFieldsRawValuesByNameToSet(fields)
                    .AddFieldsRawValuesByNameToSet("Manufacturer", “Unknown”)
                    .Build();
   ```

Send feedback about the documentation to docsite@sitecore.net.

Uploading media content

You can upload media content to the Media Library through the API using the Sitecore Mobile software development kit (SDK).

To upload a new media file and to create a new media item, you must pass a request that specifies item parameters to the `ISitecoreWebApiSession.UploadMediaResourceAsync()` method.

You must specify the following parameters to create a media item:

- Item name
- File name
- Resource data
- Parent item path or parent GUID

For example:

```
var request = ItemWebApiRequestBuilder.UploadResourceRequestWithParentId({GUID-ITEM-ID})
            .ItemDataStream(stream)
            .ItemName("MyMediaItem")
            .FileName("myPhoto.jpg")
            .Build();
var response = await session.UploadMediaResourceAsync(request);
```

Optionally, you can also specify the following parameters:

- Database
- Item template
- Content type

For example:

```
var request = ItemWebApiRequestBuilder.UploadResourceRequestWithParentPath(parentItemPath)
            .ItemDataStream(stream)
            .ContentType("image/jpg")
            .ItemName("MyMediaItem")
            .FileName("myPhoto.jpg")
            .Database("web")
            .ItemTemplatePath("/system/media/unversioned/jpeg")
            .Build();
var response = await session.UploadMediaResourceAsync(request);
```

The file name, resource data, and content type are specific media item parameters. The rest of the parameters are similar to those for items.

Setting the file name

To upload a media file, you must specify the file name in the Content-Disposition HTTP request header.

To specify the file name, use the `FileName()` method.

Property	**Details**

Mobile SDK method name	`FileName(string)`
HTTP request header field name	Content-Disposition
Sample value	`myPhoto.jpg`

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain:
  - At least one symbol besides whitespace characters.
  - A filename extension.

A sample request:

```
var request = ItemWebApiRequestBuilder.UploadResourceRequestWithParentId("{GUID-ITEM-ID}")
             .FileName("myPhoto.jpg")
```

Setting the resource data

To pass the data of the resource that you want to upload, use the `ItemDataStream()` method.

Property	Details
Mobile SDK method name	`ItemDataStream(Stream)`
Item Web API parameter	The request body

Usage guidelines:

- This is a required parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain a valid instance of the `Stream` class to read a sequence of bytes and send it to the server.

A sample request:

```
string parentItemPath = "/sitecore/media library/MyImages";
var request = ItemWebApiRequestBuilder.UploadResourceRequestWithParentPath(parentItemPath)             .ItemDataStream(stream)
```

Setting the content type

To specify the content type value of the HTTP request, use the `ContentType()` method.

Property	Details
Mobile SDK method	`ContentType(string)`
HTTP request header	Content-Type
Sample value	`image/jpeg`

Usage guidelines:

- This is an optional parameter.
- You cannot invoke this method several times in a request – multiple invocations cause *InvalidOperationException*.
- The value must contain at least one symbol besides whitespace characters.

A sample request:

```
var request = ItemWebApiRequestBuilder.UploadResourceRequestWithParentId("{GUID-ITEM-ID}")             .ContentType("image/jpeg")
```

Send feedback about the documentation to docsite@sitecore.net.

Exceptions and error handling

Understanding how and when the Mobile SDK throws exceptions is important when building high-quality applications.

This topic outlines some of the typical errors in the Mobile SDK and their causes:

- Request construction errors
- Request execution errors

Request construction errors

When you create a request, if you get a validation error, the Item Web API request will not be constructed and you cannot execute it.

The most common causes of request validation errors are:

- Passing an unexpected `null` parameter causes *ArgumentNullException*.
- Passing invalid structures, empty strings, or strings that do not contain anything except whitespace characters causes *ArgumentException*.
- Setting a write-once parameter for the second time causes *InvalidOperationException*.

These exceptions are not wrapped and have no inner exceptions to inspect.

Request execution errors

In the Mobile SDK, the messages generated by the .NET framework are grouped according to the request processing flow, which helps you create informative alert messages and provides the necessary debugging information.

The SDK provides the following set of predefined exceptions:

- *ProcessUserRequestException* – unable to build an HTTP request from the user's request.
- *RsaHandshakeException* – unable to encrypt the user data.
- *LoadDataFromNetworkException* – TCP or HTTP protocol physical connection errors.
- *ParserException* – unable to handle server response properly.
- *WebApiJsonErrorException* – a valid error response returned by the server.

The `SitecoreMobileSdkException` class is a common base class of all the exceptions.

The following sample highlights the exceptions that your application should handle and shows how to order them correctly:

```
try

{

  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)

     .Payload(this.currentPayloadType)

     .AddFieldsToRead(this.fieldNameTextField.Text)

     .Build();

  ScItemsResponse response = await session.ReadItemAsync(request);

// Process response

}

catch(ProcessUserRequestException)

{

// HTTP request construction failed

}

catch(RsaHandshakeException)

{

// Unable to encrypt user's data

}

catch(LoadDataFromNetworkException)

{

// Connection error

}

catch(ParserException)

{

// Server response is not valid

}

catch(WebApiJsonErrorException)

{

// Server has returned a valid response that contains an error

}

catch(SitecoreMobileSdkException)

{

// Some Item Web API response processing error has occurred.

// This code should not be reached in this example.

}

catch(Exception)
```

```
{
// Some exception has been thrown.
// This code should not be reached in this example.
}
```

You can access the original error object in the `Exception.InnerException` property, and the error stack trace in the `Exception.StackTrace` property.

```
try
{
  ScApiSession session = this.instanceSettings.GetSession();
  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)
     .Payload(this.currentPayloadType)
     .AddFieldsToRead(this.fieldNameTextField.Text)
     .Build();
     ScItemsResponse response = await session.ReadItemAsync(request);
// Process response
}
catch(Exception e)
{
  Debug.WriteLine(e.Message);
  Exception originalError =  e.InnerException;
  Debug.WriteLine(originalError .Message); // access original error
  Debug.WriteLine(originalError .StackTrace);
}
```

To avoid having redundant code, you should catch only `SitecoreMobileSdkException` and handle it in a separate class using the `Type.Equals()` method.

```
try
{
  ScApiSession session = this.instanceSettings.GetSession();
  var request = ItemWebApiRequestBuilder.ReadItemsRequestWithPath(this.ItemPathField.Text)
     .Payload(this.currentPayloadType)
     .AddFieldsToRead(this.fieldNameTextField.Text)
     .Build();
     ScItemsResponse response = await session.ReadItemAsync(request);
// Process response
}
catch(SitecoreMobileSdkException ex)
{
  this.MyErrorProcessor.ShowAlertForSitecoreMobileSdkError(ex);
}
catch(Exception ex)
{
  this.MyErrorProcessor.ShowAlertForUnexpectedError(ex);
}
```

Note

The Sitecore Mobile SDK does not provide error classes that show alerts because they may vary depending on the target platform and may require customization.

Send feedback about the documentation to docsite@sitecore.net.

Install the Mobile SDK from the Xamarin Component Store

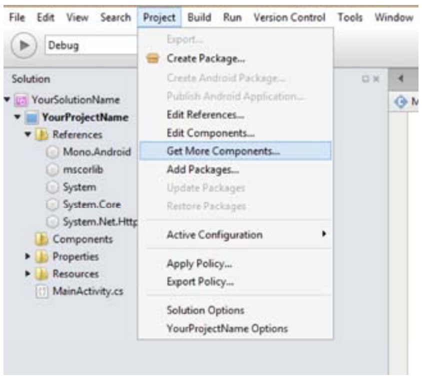
This topic describes how to include the Mobile SDK component from the Xamarin Component Store in a project using Xamarin Studio. The instructions apply to both Android and iOS platform.

Note

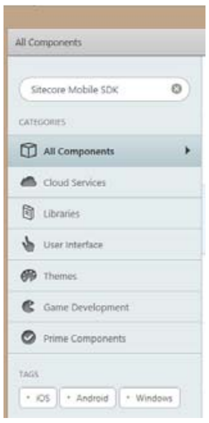
To install components from the Xamarin Component Store, you must first connect Xamarin Studio or Visual Studio to your Xamarin account.

To install the Sitecore Mobile SDK from the Xamarin Component Store in Xamarin Studio:

1. On the toolbar, click Project and then click Get More Components.



2. In the All Components dialog box, in the search field, enter the `Sitecore Mobile SDK` ID.



3. In the search results, select *Sitecore Mobile SDK for Xamarin* and click Add to App.

When the *Sitecore Mobile SDK for Xamarin* package is downloaded, you can use the Mobile SDK in your project.

Send feedback about the documentation to docsite@sitecore.net.

Requirements for the Sitecore Mobile SDK

This topic lists the supported platforms and outlines the client and server system requirements to install and use the Sitecore Mobile SDK.

Supported platforms

You can use the Mobile SDK to develop Sitecore client applications for any of these supported platforms:

- Xamarin iOS
- Xamarin Android
- Windows 7 or later
- Windows Phone 8

Client Requirements

Before installing the Mobile SDK, you must have the following applications installed and running:

- .NET framework 4.5 or later
- NuGet 2.8.5 or later
- Visual Studio 2012 or later

For iOS and Android application development using the Xamarin platform, you also need:

- Xamarin Studio 5.5.4 or later installed
- A Xamarin license

Note

Sitecore does not supply any form of license for Xamarin. For more information about the Xamarin platform and licensing, visit xamarin.com.

Server Requirements

To install and use the Mobile SDK, your server must meet the following requirements:

- Sitecore CMS 6.6 SP1 or later.

- Sitecore Item Web API module v1.2 or later installed on the Sitecore instance.

  The following URL should return the Item Web API encryption public key in the `xml` format when it is requested from the device or emulator where you run the application:*http://<host>/-/item/v1/-/actions/getpublickey*

  Note

  The Item Web API module has been bundled as part of Sitecore 7.1 and later versions. Therefore, if you are using Sitecore 7.1 or later, you don’t need to download and install the module manually.

Send feedback about the documentation to docsite@sitecore.net.

Resource management for the Mobile SDK

Because of memory and CPU limitations, mobile applications must use resources efficiently. This topic describes the resource management techniques that you can apply to optimize your resource allocation when using the Sitecore Mobile SDK:

- Resource management for Mobile SDK classes
- Resource management for media content

Resource management for Mobile SDK classes

Most classes and interfaces in the Mobile SDK are pure, managed objects. However, the session holds references to native resources, for example, a reference to the `HttpClient` instance, which uses a native socket. The credentials provider can access a native, platform-specific keychain.

To properly release unmanaged resources, the session implements the `IDisposable` interface, which requires that:

- If you use the session as a local variable, you must declare and instantiate it in a `using` statement, which calls the `Dispose` method in the correct way and causes the session object to go out of scope as soon as the method is called:

```
using  (var session =
    SitecoreWebApiSessionBuilder.AnonymousSessionWithHost(instanceUrl)
                                .Site("/sitecore/shell")
                                .DefaultDatabase("web")
                                .DefaultLanguage("en")
                                .MediaLibraryRoot("/sitecore/media library")
                                .MediaPrefix("~/media/")
                                .DefaultMediaResourceExtension("ashx")
                                .BuildReadonlySession())
{
    // Send some requests
}
```

- If you store the session as an instance variable of your class, you must implement the `IDisposable` interface and dispose the session object explicitly:

```
void ReleaseResources()
{
  if (null != this.session)
  {
    this.session.Dispose();
    this.session = null;
  }
}
public virtual void Dispose()
{
  this.ReleaseResources();
  GC.SuppressFinalize(this);
}
~MyClassThatUsesWebApiSession()
{
}
```

To ensure that your application disposes `HttpClient` and credentials properly, wrap both the session and credentials in a `using` block. Otherwise, the application may crash with a memory warning or with the *Too many open files* exception.

The session owns the credentials provided on the initialization and attempts to make a copy of them. If your implementation of the password provider does not support copying, do not wrap it in a `using` block.

Resource management for media content

To store media content, you need additional traffic and memory resources. Therefore, you should release unmanaged resources as soon as possible, by properly disposing of all the instances of the `IDisposable` interface and of the following classes of native platforms:

- `Stream` and its subclasses
- `NSData` (iOS)
- `UIImage` (iOS)
- `Bitmap` (Android)

For example:

```
string path = “/images/SitecoreLogo.png”;

var request = ItemWebApiRequestBuilder.DownloadResourceRequestWithMediaPath(path).Build();

byte[] data = null;

// getting the network stream from a session

using (Stream response = await session.DownloadMediaResourceAsync(request))

using (MemoryStream responseInMemory = new MemoryStream())

{

  // copying contents to memory of filesystem

  await response.CopyToAsync(responseInMemory);

  // Now we can convert the byte array to NSData

  data = responseInMemory.ToArray();

}

using ( NSData imageData = NSData.FromArray(data) )

using ( UIImage image = new UIImage(imageData) )

{

  // no need managing the ImageView field

  // since this.ImageView.Image creates a

  // new C# object on each call

  this.ImageView.Image = image;

}
```

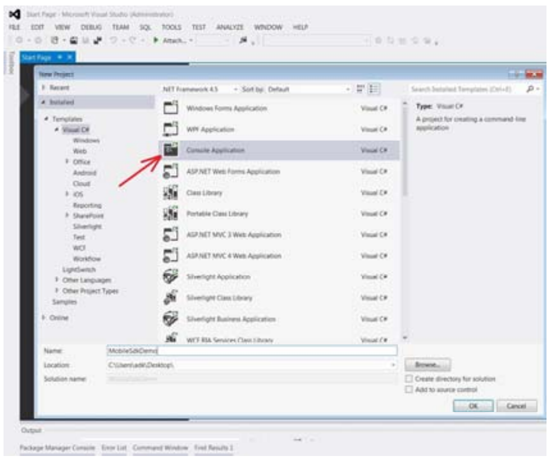
Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build a console application in Visual Studio

Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project, either in your application or in the class library. This topic describes the NuGet-based setup for Microsoft Visual Studio.

To build a console application using the Sitecore Mobile SDK in Microsoft Visual Studio:

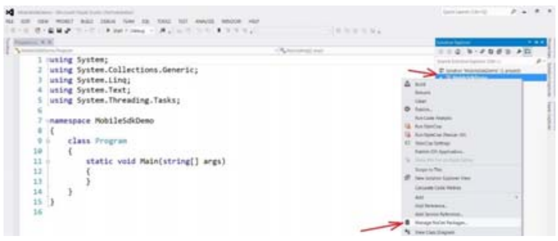
1. Create a project in Visual Studio. You can create a desktop application, mobile application, Silverlight application, and others. This example describes how to create a console application.



Type: Visual C#
A project for creating a command-line application

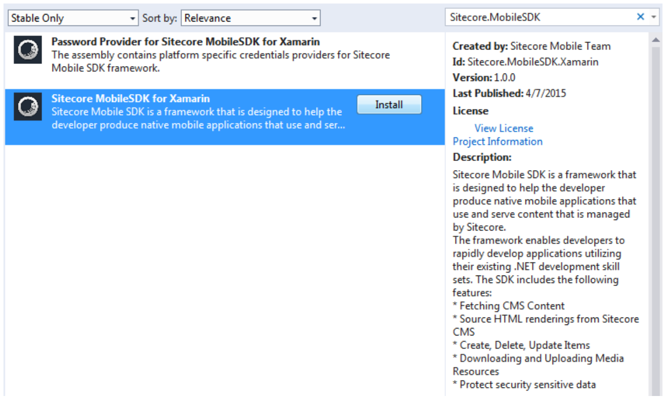
Name: MobileSdkDemo
Location: C:\Users\adk\Desktop\
Solution name:

2. To install NuGet packages, in the Solution Explorer window, right-click the project node and click Manage NuGet Packages.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MobileSdkDemo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

3. In the Manage NuGet Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
4. In the search results, select *Sitecore MobileSDK for Xamarin* and click Install. Accept the license agreements when prompted.



Password Provider for Sitecore MobileSDK for Xamarin
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

Sitecore MobileSDK for Xamarin
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and ser...

Created by: Sitecore Mobile Team
Id: Sitecore.MobileSDK.Xamarin
Version: 1.0.0
Last Published: 4/7/2015
License
View License
Project Information
Description:
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and serve content that is managed by Sitecore.
The framework enables developers to rapidly develop applications utilizing their existing .NET development skill sets. The SDK includes the following features:
* Fetching CMS Content
* Source HTML renderings from Sitecore CMS
* Create, Delete, Update Items
* Downloading and Uploading Media Resources
* Protect security sensitive data

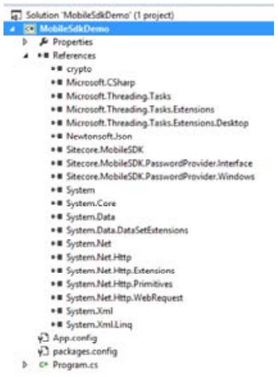
5. In the search results, select *Password Provider for Sitecore MobileSDK for Xamarin* and click Install.

   Note

   The Sitecore Password Provider package is only available for Windows Desktop, iOS, and Android platforms.

   For Windows Phone platform, you should implement a custom password provider.

   When the NuGet package is installed, you can see the updated project references in the Solution Explorer.



```
Solution 'MobileSdkDemo' (1 project)
  MobileSdkDemo
    Properties
    References
      crypto
      Microsoft.CSharp
      Microsoft.Threading.Tasks
      Microsoft.Threading.Tasks.Extensions
      Microsoft.Threading.Tasks.Extensions.Desktop
      Newtonsoft.Json
      Sitecore.MobileSDK
      Sitecore.MobileSDK.PasswordProvider.Interface
      Sitecore.MobileSDK.PasswordProvider.Windows
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net
      System.Net.Http
      System.Net.Http.Extensions
      System.Net.Http.Primitives
      System.Net.Http.WebRequest
      System.Xml
      System.Xml.Linq
    App.config
    packages.config
    Program.cs
```

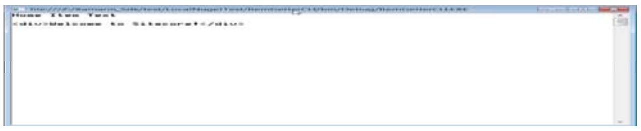
6. In the Visual Studio Code Editor, add the following code to the `Main()` function of your application:

```
private static void Main(string[] args)
{
  string instanceUrl = "http://my.site.com";
  using (var demoCredentials = new SecureStringPasswordProvider("username", "password"))
  using
  (
    var session =
      SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
                                  .Credentials(demoCredentials)
                                  .Site("/sitecore/shell")
                                  .DefaultDatabase("web")
                                  .DefaultLanguage("en")
                                  .MediaLibraryRoot("/sitecore/media library")
                                  .MediaPrefix("~/media/")
                                  .DefaultMediaResourceExtension("ashx")
                                  .BuildReadonlySession())
    {
      var request =
        ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                                .AddScope(ScopeType.Self)
                                .Payload(PayloadType.Content)
                                .Build();
      var readHomeItemTask = session.ReadItemAsync(request);
      // cannot use "await" in main
      Task.WaitAll(readHomeItemTask);
      ScItemsResponse items = readHomeItemTask.Result;
      string fieldText = items[0]["Text"].RawValue;
      Console.BackgroundColor = ConsoleColor.White;
      Console.ForegroundColor = ConsoleColor.Black;
      Console.Clear();
      Console.WriteLine("Home Item Text");
      Console.WriteLine();
      Console.WriteLine(fieldText);
      Console.ReadKey();
    }
  }
```

7. Add the following namespaces to your project to ensure that the code compiles properly:

```
using System;
using System.IO;
using System.Threading.Tasks;
using Sitecore.MobileSDK.API;
using Sitecore.MobileSDK.API.Items;
using Sitecore.MobileSDK.API.Request.Parameters;
using Sitecore.MobileSDK.PasswordProvider.Windows;
```

A sample output of the application:



```
Home Item Text
<div>Welcome to Sitecore!</div>
```

Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build an Android application in Visual Studio

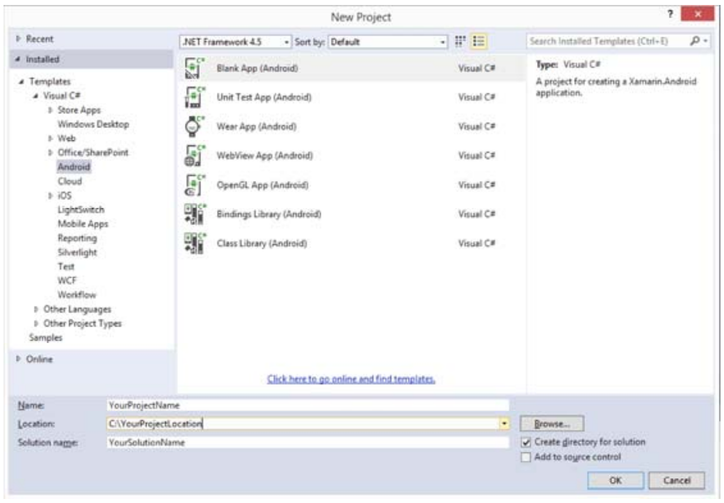
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Mobile SDK using NuGet and create an Android application in Microsoft Visual Studio.

Note

If you use Visual Studio 2013 or older, ensure that you have also installed Xamarin Studio and that you have activated the installation with your Xamarin account.

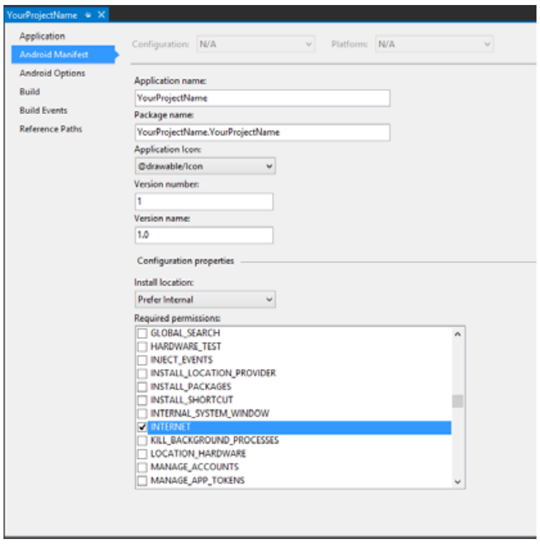
To build an Android application using the Sitecore Mobile SDK in Microsoft Visual Studio:

1. To create a project in Visual Studio, in the New Project wizard, click Visual C#, Android, Blank App (Android).


New Project
Recent
Installed
Templates
Visual C#
Store Apps
Windows Desktop
Web
Office/SharePoint
Android
Cloud
iOS
LightSwitch
Mobile Apps
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Android) Visual C#
Unit Test App (Android) Visual C#
Wear App (Android) Visual C#
WebView App (Android) Visual C#
OpenGL App (Android) Visual C#
Bindings Library (Android) Visual C#
Class Library (Android) Visual C#
Type: Visual C#
A project for creating a Xamarin.Android application.
Click here to go online and find templates.
Browse...
OK
Cancel

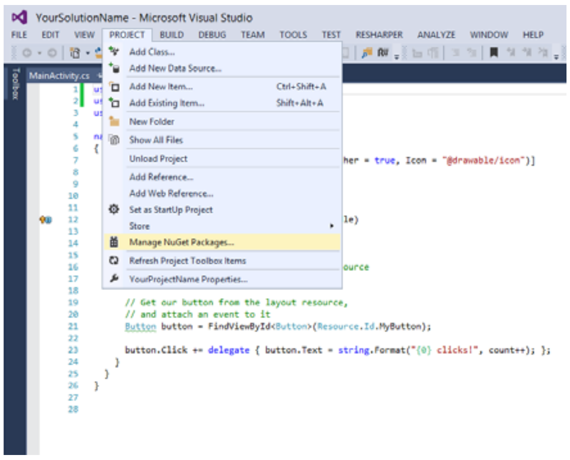
Name: YourProjectName
Location: C:\YourProjectLocation
Solution name: YourSolutionName
☑ Create directory for solution
☐ Add to source control

2. To add the permission for accessing network resources to the Android manifest file, in the Solution Explorer, right-click the project and select Properties.
3. In the Project Designer window, click Android Manifest, and in the Required permissions list box, select Internet.



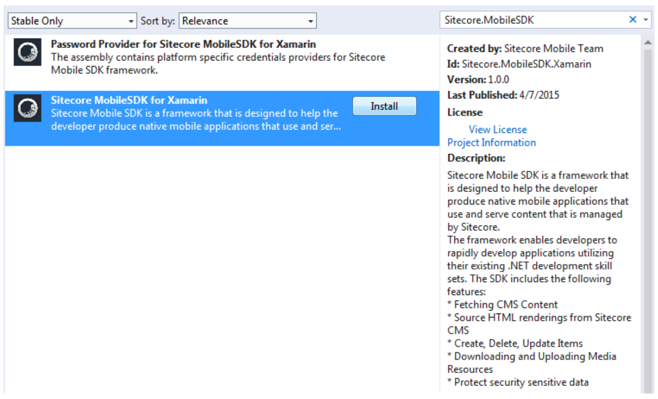
Application name: YourProjectName
Package name: YourProjectName.YourProjectName
Application Icon: @drawable/Icon
Version number: 1
Version name: 1.0
Configuration properties
Install location: Prefer Internal
Required permissions:
☐ GLOBAL_SEARCH
☐ HARDWARE_TEST
☐ INJECT_EVENTS
☐ INSTALL_LOCATION_PROVIDER
☐ INSTALL_PACKAGES
☐ INSTALL_SHORTCUT
☐ INTERNAL_SYSTEM_WINDOW
☑ INTERNET
☐ KILL_BACKGROUND_PROCESSES
☐ LOCATION_HARDWARE
☐ MANAGE_ACCOUNTS
☐ MANAGE_APP_TOKENS

4. To add the Sitecore Mobile SDK packages to your solution, on the toolbar, click Project and then click Manage NuGet Packages.


YourSolutionName - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST RESHARPER ANALYZE WINDOW HELP
MainActivity.cs
Toolbox
Add Class...
Add New Data Source...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
New Folder
Show All Files
Unload Project
Add Reference...
Add Web Reference...
Set as StartUp Project
Store
Manage NuGet Packages...
Refresh Project Toolbox Items
YourProjectName Properties...

```
her = true, Icon = "@drawable/icon")]
le)
ource
            // Get our button from the layout resource,
            // and attach an event to it
            Button button = FindViewById<Button>(Resource.Id.MyButton);

            button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };
        }
    }
}
```

5. In the Manage NuGet Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
6. In the search results, select *Sitecore MobileSDK for Xamarin* and click Install. Then select *Password Provider for Sitecore MobileSDK for Xamarin* and click Install.

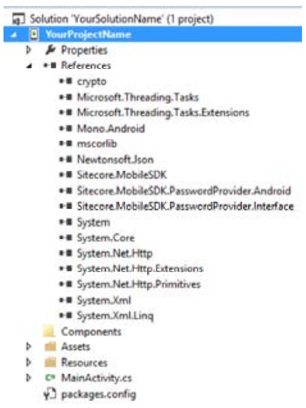

Stable Only | Sort by: Relevance | Sitecore.MobileSDK

Password Provider for Sitecore MobileSDK for Xamarin
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

Sitecore MobileSDK for Xamarin Install
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and ser...

Created by: Sitecore Mobile Team
Id: Sitecore.MobileSDK.Xamarin
Version: 1.0.0
Last Published: 4/7/2015
License
View License
Project Information
Description:
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and serve content that is managed by Sitecore.
The framework enables developers to rapidly develop applications utilizing their existing .NET development skill sets. The SDK includes the following features:
* Fetching CMS Content
* Source HTML renderings from Sitecore CMS
* Create, Delete, Update Items
* Downloading and Uploading Media Resources
* Protect security sensitive data

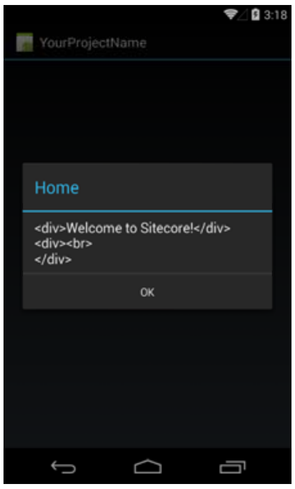
When the packages are added, you can see them in the *References* folder of the project in the Solution Explorer.


```
Solution 'YourSolutionName' (1 project)
  YourProjectName
    Properties
    References
      crypto
      Microsoft.Threading.Tasks
      Microsoft.Threading.Tasks.Extensions
      Mono.Android
      mscorlib
      Newtonsoft.Json
      Sitecore.MobileSDK
      Sitecore.MobileSDK.PasswordProvider.Android
      Sitecore.MobileSDK.PasswordProvider.Interface
      System
      System.Core
      System.Net.Http
      System.Net.Http.Extensions
      System.Net.Http.Primitives
      System.Xml
      System.Xml.Linq
    Components
    Assets
    Resources
    MainActivity.cs
    packages.config
```

Now you can use the Mobile SDK. To build a sample application, add the following lines to the `MainActivity.cs` file:

```
namespace YourProjectName
{
  using Android.App;
  using Android.Content;
  using Android.OS;
  using Sitecore.MobileSDK.API;
  using Sitecore.MobileSDK.API.Items;
  using Sitecore.MobileSDK.PasswordProvider.Android;
  using Sitecore.MobileSDK.API.Request.Parameters;
  [Activity(Label = "YourProjectName", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
    protected async override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);
      string instanceUrl = "http://my.site.com";
      using (var credentials = new SecureStringPasswordProvider("login", "password"))
      using (
           var session =
             SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
             .Credentials(credentials)
             .Site("/sitecore/shell")
             .DefaultDatabase("web")
             .DefaultLanguage("en")
             .MediaLibraryRoot("/sitecore/media library")
             .MediaPrefix("~/media/")
             .DefaultMediaResourceExtension("ashx")
             .BuildReadonlySession())
      {
        var request =
        ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                                .AddScope(ScopeType.Self)
                                .Payload(PayloadType.Content)
                                .Build();
        ScItemsResponse items = await session.ReadItemAsync(request);
        string fieldContent = items[0]["Text"].RawValue;
        var dialogBuilder = new AlertDialog.Builder(this);
        dialogBuilder.SetTitle(items[0].DisplayName);
        dialogBuilder.SetMessage(fieldContent);
        dialogBuilder.SetPositiveButton("OK", (object sender, DialogClickEventArgs e) => { });
        dialogBuilder.Create().Show();
      }
    }
  }
}
```

When it launches, the application displays an alert with the corresponding item name and field value.



```
<div>Welcome to Sitecore!</div>
<div><br>
</div>
```

Note

If you get this error:

Deployment failed because the device does not support the package's minimum Android version. You can change the minimum Android version in the Android Application section of the Project Options.

Open the Project Designer using the Properties command on the Project menu, and on the Application tab, change the *Minimum Android to target* setting to *Android 4.0*.

Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build an Android application in Xamarin Studio

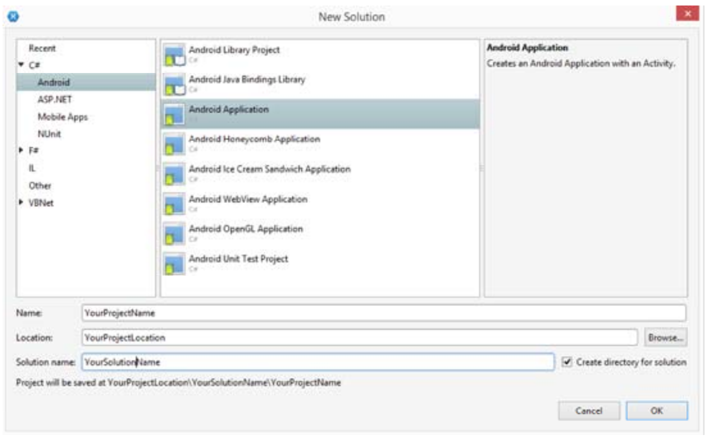
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Sitecore Mobile SDK using NuGet and create an Android application in Xamarin Studio.

Note

Ensure that you have installed Xamarin Studio and activated the installation with your Xamarin account.

To build an Android application using the Sitecore Mobile SDK in Xamarin Studio:

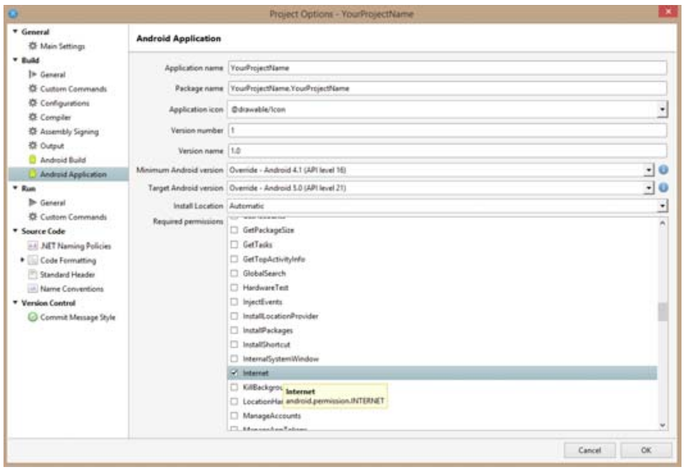
1. To create a project in Xamarin Studio, in the New Solution wizard, click C#, Android, Android Application.


New Solution
Recent
C#
Android
ASP.NET
Mobile Apps
NUnit
F#
IL
Other
VBNet
Android Library Project
Android Java Bindings Library
Android Application
Android Honeycomb Application
Android Ice Cream Sandwich Application
Android WebView Application
Android OpenGL Application
Android Unit Test Project
Browse...
Create directory for solution
Cancel
OK

Android Application
Creates an Android Application with an Activity.

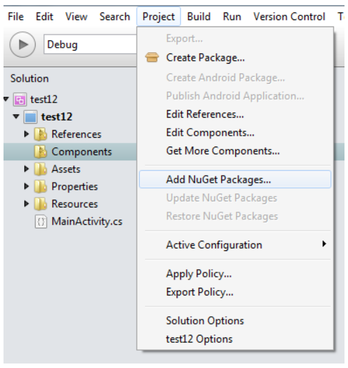
Name: YourProjectName
Location: YourProjectLocation
Solution name: YourSolutionName
Project will be saved at YourProjectLocation\YourSolutionName\YourProjectName

2. To add the permission for accessing network resources to the Android manifest file, in the Solution Pad, right-click the project and select Options.
3. In the Project Options window, click Android Application, and in the Required permissions list box, select Internet.

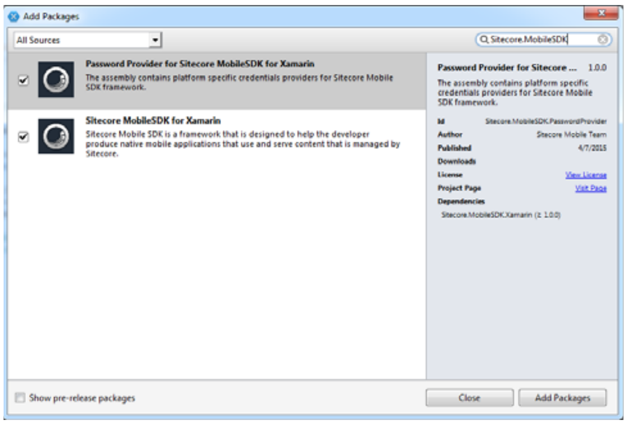


Android Application
Application name: YourProjectName
Package name: YourProjectName.YourProjectName
Application icon: @drawable/icon
Version number: 1
Version name: 1.0
Minimum Android version: Override - Android 4.1 (API level 16)
Target Android version: Override - Android 5.0 (API level 21)
Install Location: Automatic
Required permissions: ✓ Internet
Internet
android.permission.INTERNET

4. To add the Sitecore Mobile SDK packages to your solution, on the toolbar, click Project and then click Add NuGet Packages.



5. In the Add Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
6. In the search results, select *Sitecore MobileSDK for Xamarin* and *Password Provider for Sitecore Mobile SDK for Xamarin,* and then click Add Packages.



✓ Password Provider for Sitecore MobileSDK for Xamarin
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

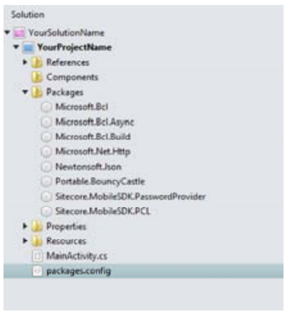
✓ Sitecore MobileSDK for Xamarin
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and serve content that is managed by Sitecore.

Password Provider for Sitecore ... 1.0.0
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

Id	Sitecore.MobileSDK.PasswordProvider
Author	Sitecore Mobile Team
Published	4/7/2015
Downloads	
License	View License
Project Page	Visit Page
Dependencies	

Sitecore.MobileSDK.Xamarin (≥ 1.0.0)

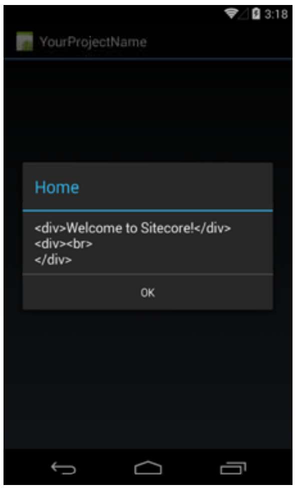
When the packages are added, you can see them in the *Packages* folder of the project in the Solution Pad.



Now you can use the Mobile SDK. To build a sample application, use the following code example:

```
namespace YourProjectName
{
  using Android.App;
  using Android.Content;
  using Android.OS;
  using Sitecore.MobileSDK.API;
  using Sitecore.MobileSDK.API.Items;
  using Sitecore.MobileSDK.PasswordProvider.Android;
  using Sitecore.MobileSDK.API.Request.Parameters;
  [Activity(Label = "YourProjectName", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
    protected async override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);
      string instanceUrl = "http://my.site.com";
      using (var credentials = new SecureStringPasswordProvider("login", "password"))
      using (
           var session =
             SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
             .Credentials(credentials)
             .Site("/sitecore/shell")
             .DefaultDatabase("web")
             .DefaultLanguage("en")
             .MediaLibraryRoot("/sitecore/media library")
             .MediaPrefix("~/media/")
             .DefaultMediaResourceExtension("ashx")
             .BuildReadonlySession())
      {
        var request =
        ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                                .AddScope(ScopeType.Self)
                                .Payload(PayloadType.Content)
                                .Build();
        ScItemsResponse items = await session.ReadItemAsync(request);
        string fieldContent = items[0]["Text"].RawValue;
        var dialogBuilder = new AlertDialog.Builder(this);
        dialogBuilder.SetTitle(items[0].DisplayName);
        dialogBuilder.SetMessage(fieldContent);
        dialogBuilder.SetPositiveButton("OK", (object sender, DialogClickEventArgs e) => { });
        dialogBuilder.Create().Show();
      }
    }
  }
}
```

When it launches, the application displays an alert with the corresponding item name and field value.



Home

<div>Welcome to Sitecore!</div>
<div>
</div>

Note

If you get this error:

Deployment failed because the device does not support the package's minimum Android version. You can change the minimum Android version in the Android Application section of the Project Options.

In the `Android.Manifest.xml` file, change the *Minimum Android version* setting to *Android 4.0*.

Send feedback about the documentation to docsite@sitecore.net.

Use the Mobile SDK to build an iOS application in Xamarin Studio on Mac

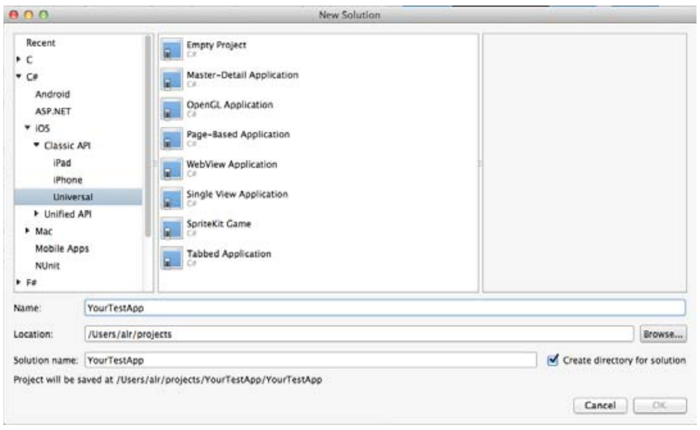
Before you can start using the Sitecore Mobile SDK, you must add its assembly reference to a dependent project. This topic describes how to install the Sitecore Mobile SDK using NuGet and create an iOS application in Xamarin Studio on Mac.

Note

To become familiar with the fundamentals of iOS application development with Xamarin, visit http://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/.

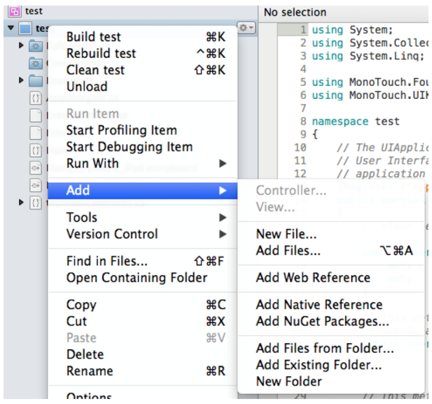
To build an iOS application using the Sitecore Mobile SDK in Xamarin Studio on Mac:

1. To create a project in Xamarin Studio, in the New Solution wizard, click iOS, Classic API or Unified API for a 64 bit application, Universal, Single View Application.



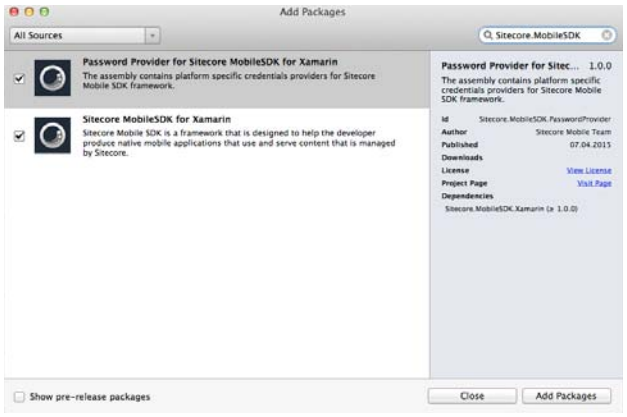
Name: YourTestApp
Location: /Users/alr/projects
Solution name: YourTestApp
✓ Create directory for solution
Project will be saved at /Users/alr/projects/YourTestApp/YourTestApp

2. To add the Sitecore Mobile SDK packages to your solution, in the Solution Pad, right-click the project node, click Add, Add NuGet Packages.



```
using System;
using System.Collec
using System.Linq;

using MonoTouch.Fou
using MonoTouch.UIK

namespace test
{
    // The UIApplic
    // User Interfa
    // application
```

3. In the Add Packages dialog box, in the search field, enter the `Sitecore.MobileSDK` ID. The ID is not case-sensitive.
4. In the search results, select *Sitecore MobileSDK for Xamarin* and *Password Provider for Sitecore MobileSDK for Xamarin* and click Add Packages.



Password Provider for Sitecore MobileSDK for Xamarin
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

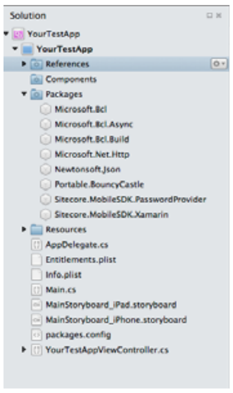
Sitecore MobileSDK for Xamarin
Sitecore Mobile SDK is a framework that is designed to help the developer produce native mobile applications that use and serve content that is managed by Sitecore.

Password Provider for Sitec... 1.0.0
The assembly contains platform specific credentials providers for Sitecore Mobile SDK framework.

Id	Sitecore.MobileSDK.PasswordProvider
Author	Sitecore Mobile Team
Published	07.04.2015
Downloads	
License	View License
Project Page	Visit Page
Dependencies	

Sitecore.MobileSDK.Xamarin (≥ 1.0.0)

When the NuGet packages are added, you can see the updated project references.



5. In the editor, add the following code to the `ViewDidAppear()` function of your application:

```
public override async void ViewDidAppear(bool animated)
{
  base.ViewDidAppear(animated);
  string instanceUrl = "http://my.site.com";
using (var demoCredentials = new SecureStringPasswordProvider("username", "password"))
using
(
  var session =
    SitecoreWebApiSessionBuilder.AuthenticatedSessionWithHost(instanceUrl)
                                .Credentials(demoCredentials)
                                .Site("/sitecore/shell")
                                .DefaultDatabase("web")
                                .DefaultLanguage("en")
                                .MediaLibraryRoot("/sitecore/media library")
                                .MediaPrefix("~/media/")
                                .DefaultMediaResourceExtension("ashx")
                                .BuildReadonlySession())
{
  var request =
    ItemWebApiRequestBuilder.ReadItemsRequestWithPath("/sitecore/content/home")
                            .AddScope(ScopeType.Self)
                            .Payload(PayloadType.Content)
                            .Build();
  ScItemsResponse items = await session.ReadItemAsync(request);
  string fieldContent = items[0]["Text"].RawValue;
    UIAlertView alert = new UIAlertView(
      "Home Item Demo",
      fieldContent,
      null,
      "Ok",
      null);
    alert.Show();
  }
}
```

6. Add the following namespaces to your project to make the code compile correctly:

```
using System;
using System.Drawing;
using System.Threading.Tasks;
using MonoTouch.Foundation;
using MonoTouch.UIKit;
using Sitecore.MobileSDK.API;
using Sitecore.MobileSDK.API.Items;
using Sitecore.MobileSDK.API.Request.Parameters;
using Sitecore.MobileSDK.PasswordProvider.Interface;
using Sitecore.MobileSDK.PasswordProvider.iOS;
```

   Note

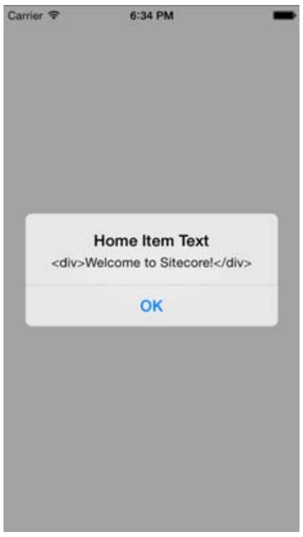
   If you have chosen the project template from iOS > Unified API section for your application, you should replace the following namespaces:

```
using MonoTouch.Foundation;

using MonoTouch.UIKit;
```

   With these ones:

```
using Foundation;

using UIKit;
```

When it launches, the application displays an alert with the contents of the Text field of the *Home* item.



Send feedback about the documentation to docsite@sitecore.net.