

---

# **Omnia Foundation Documentation**

**Precio Fishbone**

**Apr 06, 2018**



---

## Contents

---

<b>1</b>	<b>Topics</b>	<b>3</b>
1.1	Getting Started . . . . .	3
1.2	Omnia Foundation Fundamentals . . . . .	13
1.3	SharePoint Provisioning . . . . .	42
1.4	Omnia API . . . . .	54
1.5	Custom Web API for Omnia extensions . . . . .	70
1.6	Omnia Jobs . . . . .	73
1.7	Client-Side Development . . . . .	78
1.8	Performance . . . . .	97
1.9	Release Notes . . . . .	98
1.10	Contribute to this Documentation . . . . .	102



.. include:: ../common/authors.txt

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---



```
.. include:: ../common/authors.txt
```

## 1.1 Getting Started

### 1.1.1 1. Install Visual Studio 2017

Download and install Visual Studio 2017 (not Community edition)

### 1.1.2 2. Install Node JS

Download and install Node JS from <https://nodejs.org/> . Choose the latest **Current** version. (Tested with version 8.5.0)

Or, if you have the need for running different versions of nodejs, for various projects. NVM (Node Version Manager) for windows is recommended, it allows for easy switching between nodejs versions. It can be found at <https://github.com/coreybutler/nvm-windows> You will need to uninstall nodejs if you already have it, instructions are on the repo.

### 1.1.3 3. Install Office Developer Tools

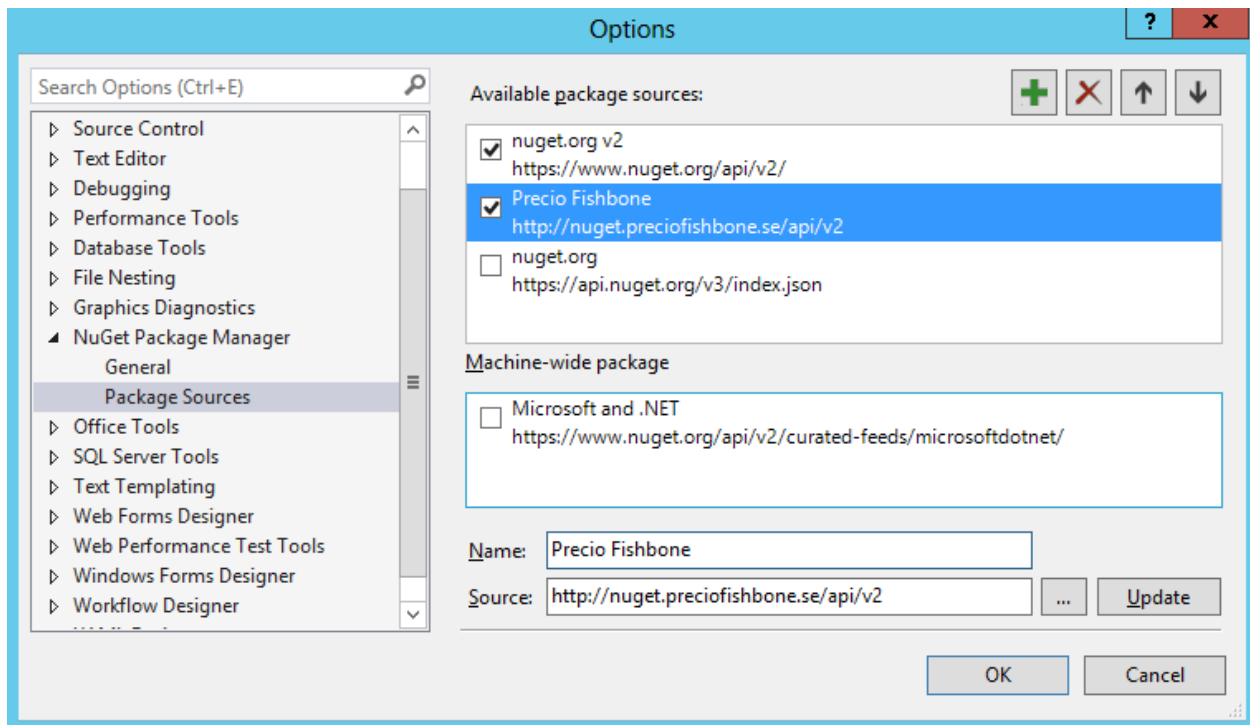
Download and install Office Developer Tools from <https://www.visualstudio.com/vs/office-tools/>

### 1.1.4 4. Configure Precio Fishbone NuGet package source in Visual Studio

Precio Fishbone NuGet package source is needed for download Omnia NuGet packages. To configure Precio Fishbone NuGet package source, follow these steps:

- In Visual Studio, navigate to **Tools > NuGet Packages Manager > Packages Manager Settings**
- In the Options dialog, select **Package Sources** in the left pane

- Add a new package source named Precio Fishbone with this URL <http://nuget.preciofishbone.se/api/v2>

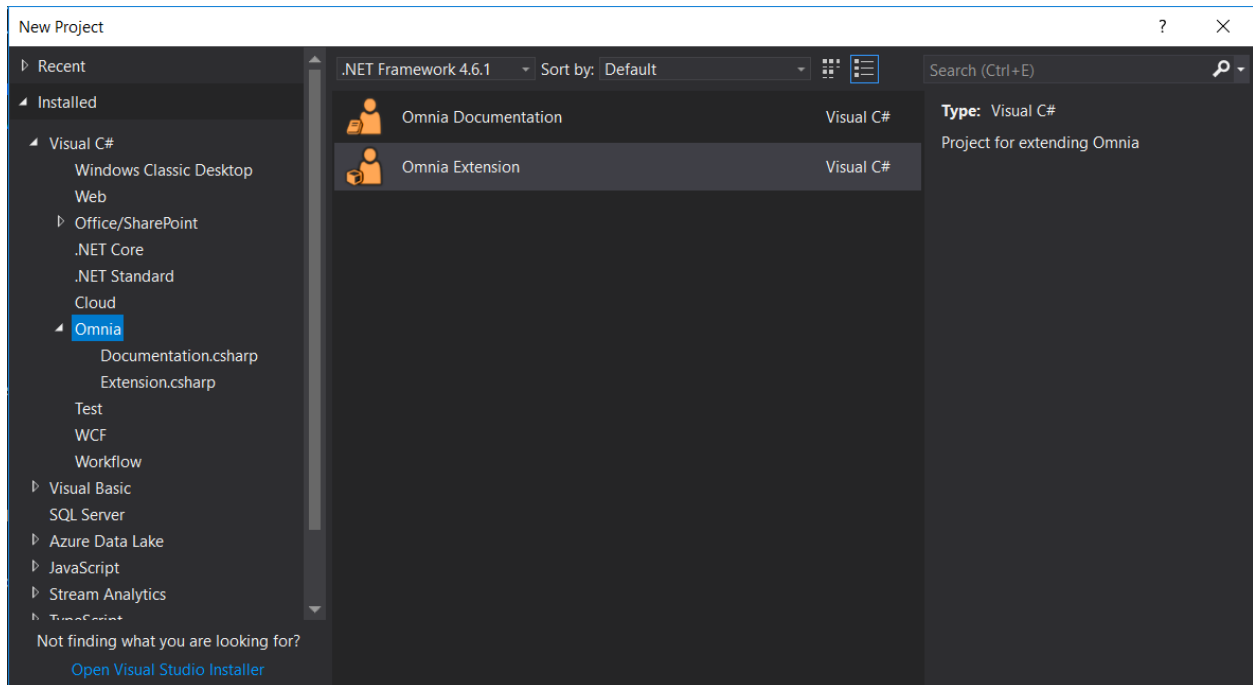


### 1.1.5 5. Install Omnia Tooling for Visual Studio

Download and install *Omnia Tooling for Visual Studio*



### 1.1.6 6. Create new project with Omnia Extension project template

**Note:**

- **Omnia Documentation** - Project template for creating an Omnia documentation package
- **Omnia Extension** - Project template for creating an Omnia extension package

Select the **Omnia Extension** template and name your project. Click OK.

The following screen opens

New Omnia Extension Project

**omnia**

**Extension**

**Id**  
d3282397-b325-469d-8b81-a7b521380256

**Title**  
DevelopDocumentation

**Developer Environment**

**Tenant Id**

**Foundation Url**

**Api Secret**

**Foundation Api Version**  
☐ Use Prerelease  
1.0.6917

**Sharepoint Api Version**  
SharePoint 2013 OnPrem / Online (15)

**Project Options**

☐ Extension Package

- ☐ Client Side
  - ☐ Angular Tooling
  - ☐ Less Tooling
- ☐ Server Side
  - ☒ Examples
  - ☐ Setup basic folder structure
- ☐ Web API

The first section is where you enter metadata about your extension. Normally there is no need to change these two values

**Extension**

**Id**  
d3282397-b325-469d-8b81-a7b521380256

**Title**  
DevelopDocumentation

The second section exists to help you configure the communication between your development environment and your Omnia tenant

### Developer Environment

Tenant Id

Foundation Url

Api Secret

Foundation Api Version

☐ Use Prerelease

1.0.6917 ▼ Lookup

- **Tenant Id** is found in Omnia Admin in your tenant, in the Settings section (see image below)
- **Foundation Url** is also found in Omnia Admin, in the Settings section (see image below)

**omnia** Start

Home System Features

**Tenant**

- Targeting Definitions
- Site Templates
- Navigation
- PowerPack
- Common Links
- Important Announcements
- Document Management
- Page Footer

**Site Collection**

- Quick Polls
- Reusable Banners

**System**

Settings Permissions Queues Logs Extensions

**Tenant Information**

**Tenant Id**

**Foundation Url**

**Developer Information**

**Foundation Version**  
1.0.6912.0

**NuGet Version**  
1.0.6335

- **Api Secret.** This secret is recieved from Omnia by navigating to Omnia Admin > System > Extensions > Register Extension and paste in your Extension Id (from the Visual Studio dialog)

System » Extensions » Register Extension

ExtensionId	Created At	
aaf868ac-469f-47f9-a5e1-6ec181a6de38	2016-06-10 10:24:18	
8c03468f-ef20-49ef-9349-88418305433c	2016-06-10 13:34:03	
0d54230e-d0ec-4b57-835f-92325218fc52	2016-06-29 09:26:40	
df041edc-14c1-4a07-a8be-f0775d48f07c	2016-07-06 11:51:13	
72e65df6-aa60-4f4c-bb27-1bdaa9df3107	2016-08-08 06:12:55	
b708fc36-57af-4ede-8334-5705a2e696a1	2016-10-13 12:51:14	
3e244375-6a6f-458c-bfe8-2993d4fd6f88	2017-07-26 11:36:30	

System » Extensions » Register Extension » New Extension

The Extension ID has been created successfully. This is the only time the API secret will be displayed in clear text, please save it somewhere else for later use.

**ExtensionId**  
d3282397-b325-469d-8b81-a7b521380256

**API Secret**  
sZxfAA5wn0f3Dyu5haOMWIEDSMabp6NI

The third section's purpose is there to help you target the correct API version, both in Omnia and SharePoint

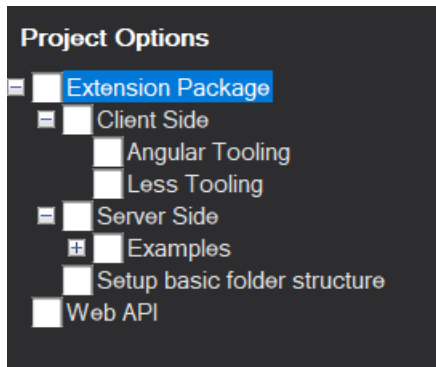
**Foundation Api Version**

☐ Use Prerelease

**Sharepoint Api Version**

The Foundation API version used in your tenant can be found in Omnia Admin > System, in the *Developer Information* section

The right hand side of the dialog determines the structure and files that will be created in the Extension projects

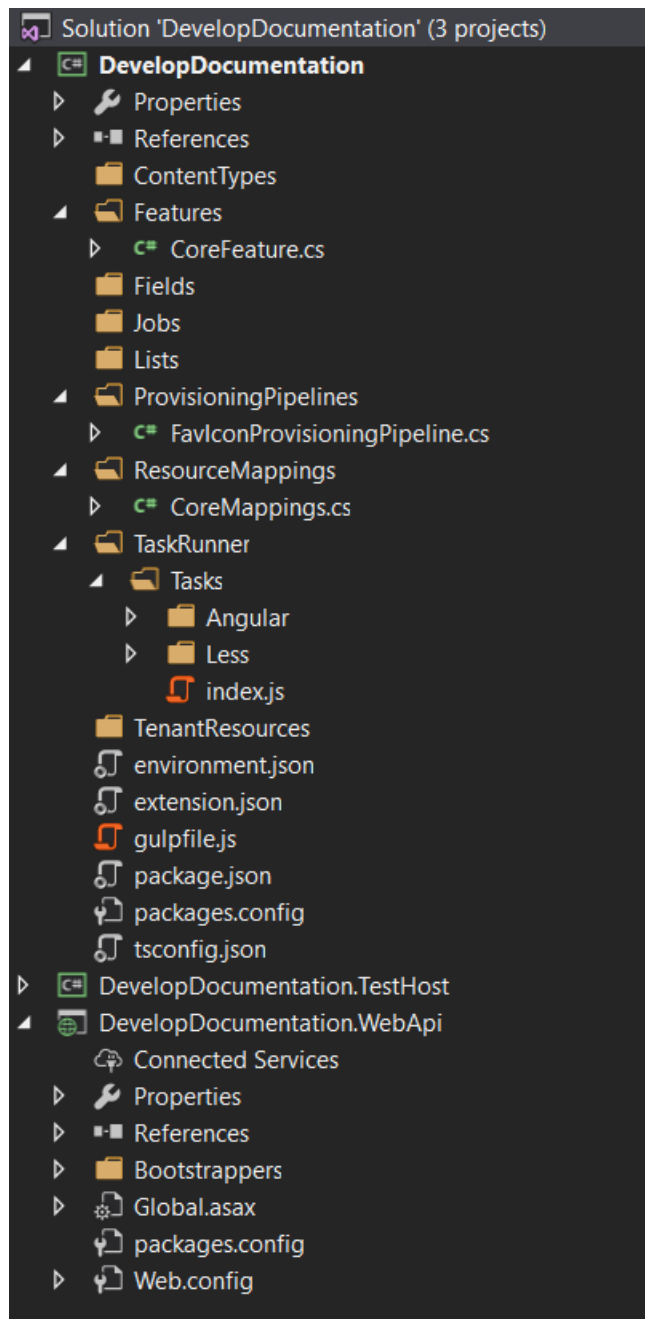


- The **Extension Package** checkbox controls if an Omnia Extension project is created in the solution
- The **Web API** checkbox controls if a Web API project, prepared to communicate with Omnia is created in the solution

The **Extension Package** checkbox has some child items that can be selected

- The **Client Side** checkbox determines if the project structure and files for doing client side development should be added to the solution
- The **Angular Tooling** checkbox makes sure the relevant files and tooling support for Angular development is added
- The **Less Tooling** checkbox makes sure the relevant files and tooling support for compiling Less files is added
- The **Server Side** checkbox determined if the project structure for doing server side development (like features, jobs etc.) is added to the project
- Below the **Examples** checkbox you can select different code examples to be provisioned to your project, to set you off to a quick start
- The **Setup basic folder structure** checkbox will add a best practice folder structure to your project

After filling in all the fields in the form, and selecting all the checkboxes, the following solution structure will be created



### 1.1.7 7. Alter the environment information in your project

(when changing tenant, or when passing the extension over to a fellow developer)

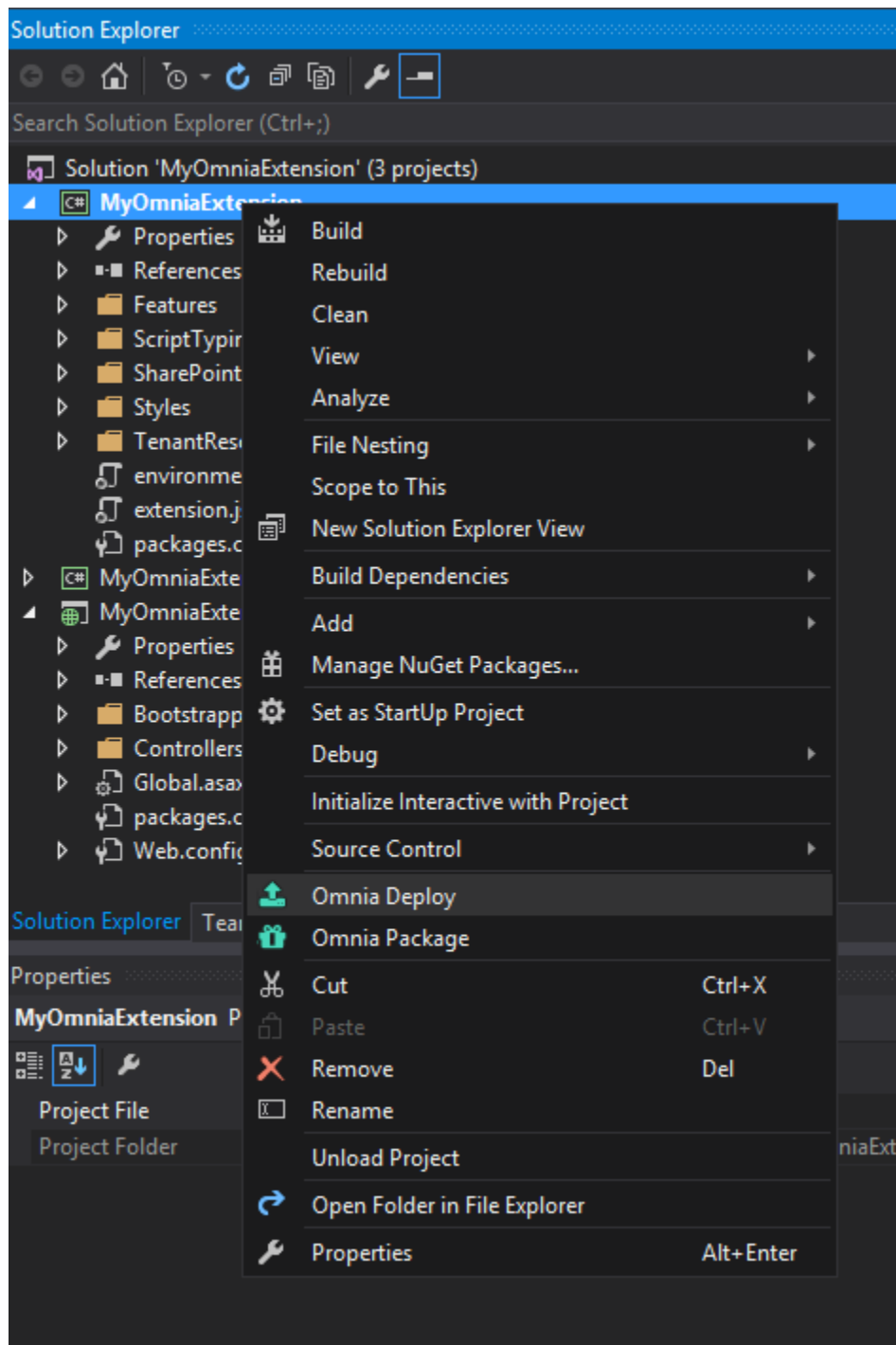
Open the file **environment.json** in MyOmniaExtension and fill in:

- TenantId: you get this from the System page in Omnia admin
- ApiSecret: the secret you got when you registered your extension in step 6
- FoundationUrl: you get this from the System page in Omnia admin

```
{
  "TenantId": "d4fb9588-8b1a-4007-b582-8d8b5e0d5f3e",
  "ApiSecret": "eo7PRFjXZ9cgLKCyXK3NIIfILLvdSIb5P",
  "FoundationUrl": "https://omniacil.azurewebsites.net",
  "Hosting": {
    "Enabled": false,
    "Port": 9900
  },
  "FeatureActivations": [ ]
}
```

### 1.1.8 9. Deploy your extension

Right click on Extension project (DevelopDocumentation) and click Omnia Deploy



You can see the deployment progress in the Output window in Visual Studio



```

Output
Show output from: Omnia Tooling

> Compiling extension package
> Extension package is saved at: C:\MyOmniaExtension\MyOmniaExtension\MyOmniaExtension\bin\Debug\Omnia.Publish\1.0.0
> Uploading extension package
> Working on it....
> Initializing Package....
> Extension Package was added

> Hurray!! Deployed the extension package successfully.
> Lets hope you dont have any bugs!!

```

Package Manager Console Task Runner Explorer Comment Preview [GhostDoc] Documentation Maintenance [GhostDoc] Error List Output Find Results 1 Find Symbol Results Call Hierarchy

### 1.1.9 10. Verify

After the extension has been deployed successfully to Omnia, you can verify it by navigating to **System > Extension** in Omnia admin

Name	Status	Description
MyOmniaExtension	Available	

And in the **Features** page you should see the features from your extension.

Congratulations, you are now ready to build your magical Omnia extension!

```
.. include:: ../common/authors.txt
```

## 1.2 Omnia Foundation Fundamentals

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

### 1.2.1 Topics

```
.. include:: ../common/authors.txt
```

#### Omnia Extension Package

Omnia Foundation is built with extensibility in mind since day one and an Omnia Extension Package is the main way to extend Omnia. This article will give an overview of what you can achieve with extension packages and the basic structure of an extension package. For a more practical guide on how to start developing Omnia extensions see the Getting Started guide.

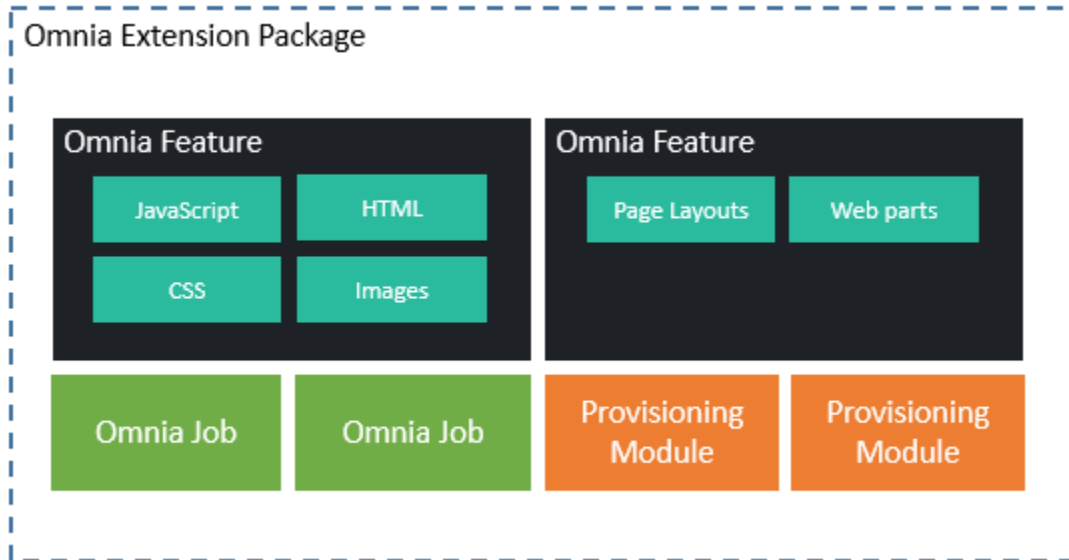
#### Sections:

- *What is an Omnia Extension Package?*

- *Manage extension packages*

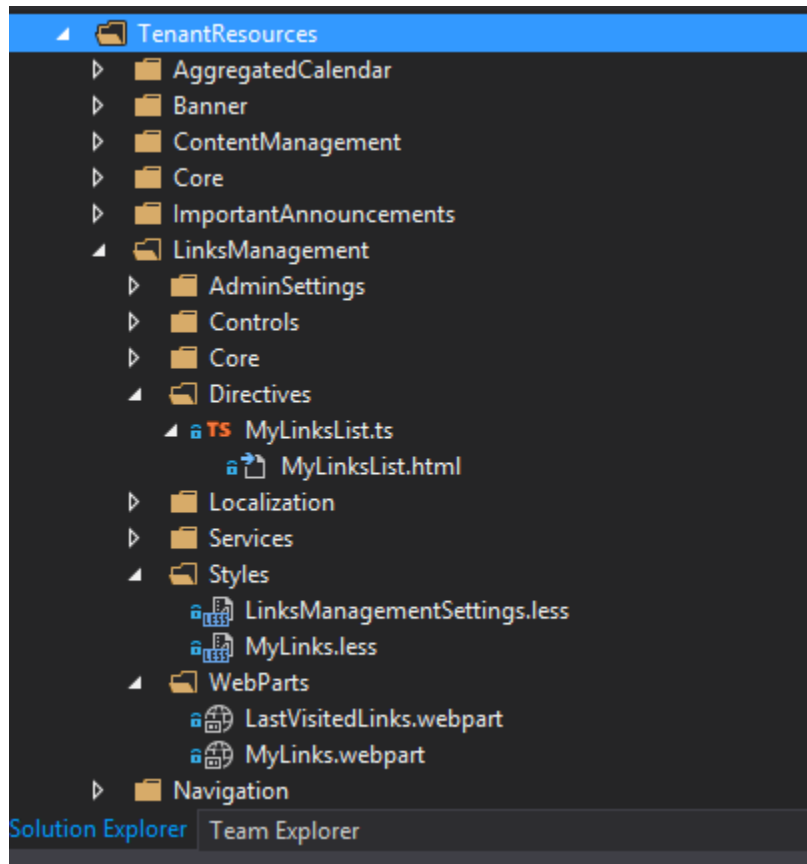
## What is an Omnia Extension Package?

An Omnia extension package is a pack of features or extensions for Omnia. You can think of it like SharePoint solution packages but for the Omnia platform. An extension can be used to deploy all kinds of new features, from small UI changes to non-trivial applications with its own back-end and database. Examples are *Omnia Intranet* and *Omnia Documentation Management*.



At the core of an extension package are resource files. Resource files can be anything from client-side code like JavaScript, CSS, HTML templates to SharePoint artifacts like page layouts and web parts. These resources are *mapped and deployed* by logical containers called *Omnia feature*.

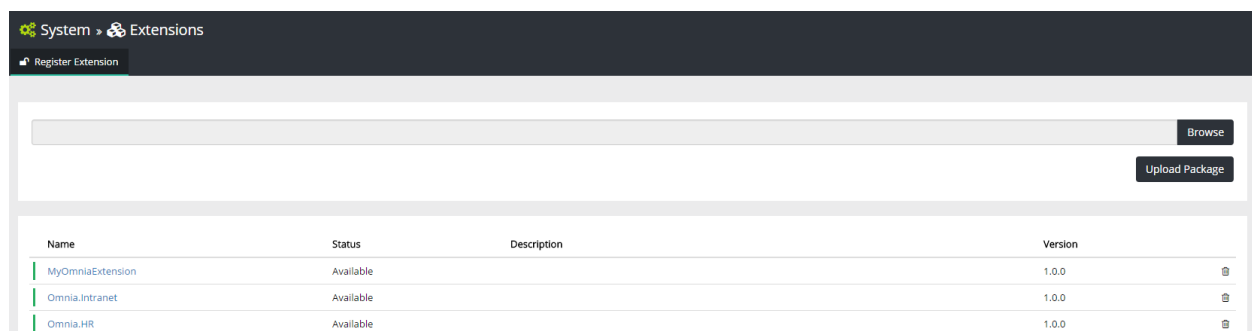
For example, the resources in a large extension might look like this



An extension package can also contain Omnia jobs, pieces of code that will run as scheduled tasks in Omnia, and *provisioning pipelines*, pieces of code that will be hooked into the site provisioning pipeline of Omnia.

## Manage extension packages

All extension packages in a tenant can be managed in Omnia admin, **System > Extensions**



New extensions can be uploaded directly from this UI by drag and drop or from Visual Studio using Omnia Tooling. An extension need to be registered before it can be uploaded, see the [Getting Started](#) guide for more details.

In this UI you can also manage configurations for each extension by clicking on the extension name

System » Extensions » Omnia.Intranet

Name	Region	Value	Included in Client
omnia.intranet.apiurl	omnia.intranet	<input type="text" value="https://omniacl1-intranet.azurewebsites.net/api/"/>	<input checked="" type="checkbox"/>
omnia.intranet.storageid	omnia.intranet	<input type="text" value="BF112171-AC40-4288-AEC0-DF794F146683"/>	<input checked="" type="checkbox"/>

Save

When a new version of an extension package has been deployed, all the new changes will not be available for users until the features of that extension are upgraded to the latest version.

```
.. include:: ../common/authors.txt
```

### Omnia Feature

Omnia features are modules containing customizations for Omnia. They are very similar to SharePoint features, except that they can provisioning resources to both SharePoint and to Omnia database. An Omnia feature can also run your custom code when they are activated, upgraded or removed.

#### Sections:

- *Feature scopes*
- *Built-in methods and properties*
- *Create new feature*
- *Feature activation, deactivation and upgrade*
- *Provisioning tenant resources*
- *Provisioning SharePoint artifacts*

### Feature scopes

A feature will have one of the three scopes **Site**, **SiteCollection** and **Tenant**, each scope can deploy different types of artifacts.

Scope	Artifacts	SharePoint ClientContext
Site	<ul style="list-style-type: none"><li>• Page</li><li>• List instance</li><li>• <a href="#">Site.js bundle</a></li><li>• <a href="#">Site.css bundle</a></li></ul>	App-only context of the target site
SiteCollection	<ul style="list-style-type: none"><li>• Page</li><li>• List instance</li><li>• Content Type</li><li>• Field</li><li>• Masterpage and pagelayout</li><li>• Webpart</li><li>• <a href="#">SiteCollection.js bundle</a></li><li>• <a href="#">SiteCollection.css bundle</a></li></ul>	App-only context of the root site of the target site collection
Tenant	<ul style="list-style-type: none"><li>• Tenant resource</li><li>• <a href="#">Tenant.js bundle</a></li><li>• <a href="#">Tenant.css bundle</a></li></ul>	Not available

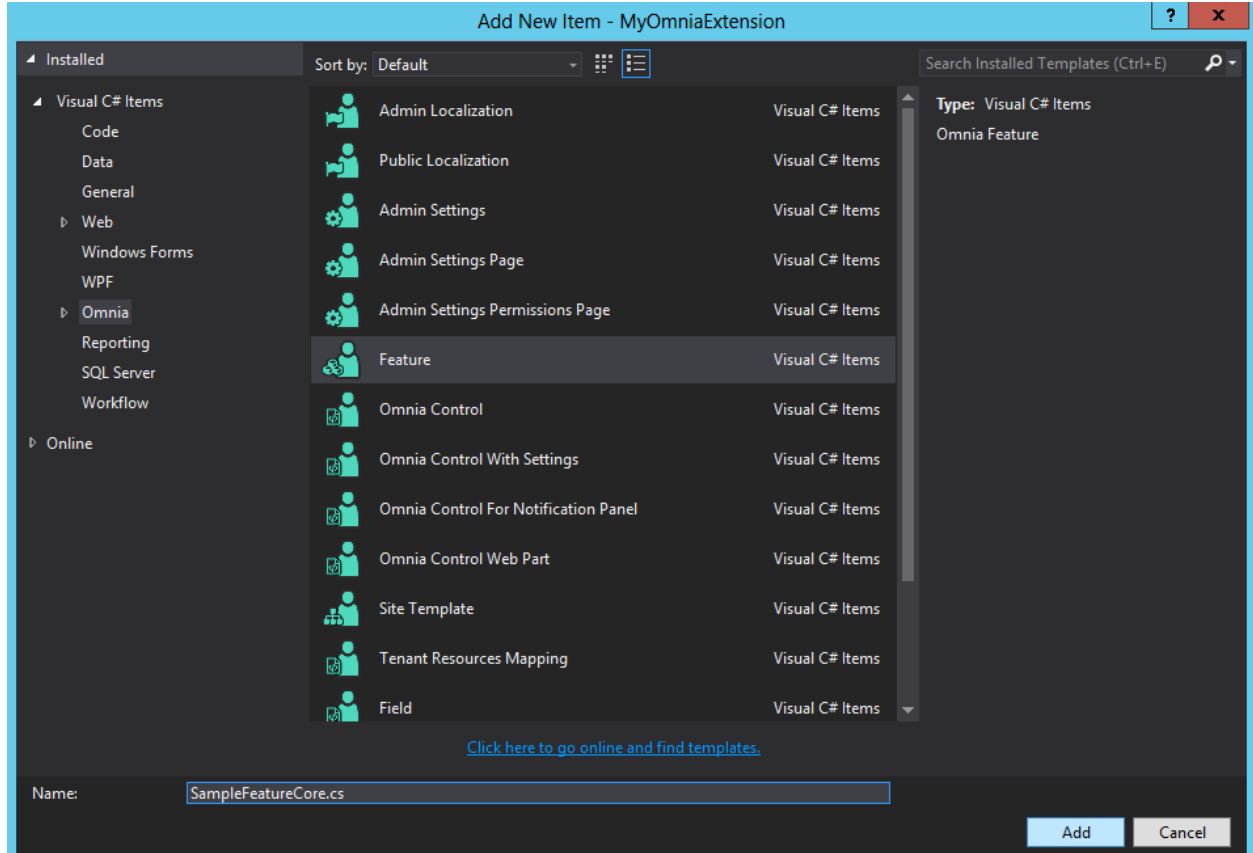
### Built-in methods and properties

Properties	Type	Description
Ctx	ClientContext	App-only client context for the targeting site. This property is not available in tenant scope features.

Methods	Type	Description
CreateContextFor(string spUrl)	ClientContext	Create a new app-only context for another site
Log(string log)	void	Add a new entry to this feature's logs
WorkWith()	ApiFactory	Return the ApiFactory that can call Omnia API Example: Work- With().Logging().AddLog(log)
Localize(string localizeKey)	string	Return the localized string

## Create new feature

As usual, you can create new Omnia feature using the template from Omnia tooling



At the top of the feature is the **FeatureDefinition** attribute which contains the feature's metadata

```
[FeatureDefinition(
    id: "85544C6C-9EB9-4F99-9410-95F1EA3D07B5",
    name: "MyOmniaExtension Sample Feature Core",
    version: "0.1.0",
    scope: FeatureScopes.Tenant
)]
public class SampleFeatureCore : Omnia.Foundation.Extensibility.Features.OmniaFeature
```

## Feature activation, deactivation and upgrade

In a feature you can override the activation, deactivation or upgrade events to run custom code. In these events, you can:

- Write CSOM code to work with data in SharePoint (except for Tenant scope features where the ClientContext is not available)
- Call the API of Omnia Foundation like [logging](#) and [configurations](#) using the built-in method **WorkWith**
- Trigger queue jobs using Omnia Queues API

**Example:** Provisioning new start page for the target site when activating the feature

```
/// <summary>
/// Activates the feature
/// </summary>
public override void Activate()
{
    try
    {
        string pageTitle = this.Localize("$Localize:MyOmniaExtension.StartPageTitle;
↪");
        var publishingClient = this.WorkWith().Publishing(this.Ctx);
        publishingClient.CreateStartPage(new Page
        {
            Name = "StartPage.aspx",
            PageLayout = MyOmniaExtension.PageLayouts.StartPage,
            Title = pageTitle
        });
    }
    catch (Exception ex)
    {
        this.Log("Activate feature", ex.Message, FeatureInstanceLogTypes.Error);
        throw;
    }
}

/// <summary>
/// Deactivates the feature.
/// </summary>
/// <param name="fromVersion">From version.</param>
public override void Deactivate(string fromVersion)
{
    // Your code to handle feature deactivation here
}
```

(continues on next page)

(continued from previous page)

```

/// <summary>
/// Upgrades the feature
/// </summary>
/// <param name="fromVersion">From version.</param>
public override void Upgrade(string fromVersion)
{
    // Your code to handle feature upgrade here
}

```

## Provisioning tenant resources

By overriding the method **OnTenantResourceMappings**, a feature can provision tenant resources to Omnia database.

For code resources like JavaScript and CSS, you will also need to add them to a bundle for them to be loaded and executed on SharePoint or Omnia admin application. Read more on [bundling in Omnia](#).

---

**Note:** Only tenant-scope features can provision tenant resources, though any features can add resources to bundles.

---

## Example

```

/// <summary>
/// Called when [OmniaFeature resource mappings is being performed].
/// </summary>
/// <param name="resourceMapper">The resource mapper.</param>
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    // Provisioning tenant resources to Omnia database
    // NOTE: Only tenant-scope features can provision tenant resources
    resourceMapper
        .AddOrUpdateTenantResourcesFrom<ResourcesMapping>();

    // Adding resources to the scope's bundles, in this case tenant.js and tenant.css
    resourceMapper
        .CreateBundleFor(BundleTargets.SharePoint)
        .Include<ResourcesMapping.Scripts.Core>()
        .Include<ResourcesMapping.Scripts.Services>()
        .Include<ResourcesMapping.Scripts.Directives>()
        .Include<ResourcesMapping.Scripts.Controls>()
        .Include<ResourcesMapping.Styles>();

    resourceMapper
        .CreateBundleFor(BundleTargets.OmniaAdmin)
        .Include<ResourcesMapping.Scripts.Core>()
        .Include<ResourcesMapping.Scripts.Services>()
        .Include<ResourcesMapping.Scripts.Directives>()
        .Include<ResourcesMapping.Scripts.AdminSettings>(q => q.
        ↪ SampleAdminSettingsFormJs)
        .Include<ResourcesMapping.Scripts.AdminSettings>(q => q.SampleAdminSettingsJs)
        .Include<ResourcesMapping.Scripts.AdminSettings>();
}

```



## Provisioning SharePoint artifacts

In the method **OnTenantResourceMappings** you can also provision files like masterpage and pagelayout to SharePoint. To provision other SharePoint artifacts like fields, content types and list instances you need to override the method **OnSharePointArtifactMappings**.

---

**Note:** Only site-scope and sitecollection-scope features can provision SharePoint artifacts

---

### Example

```

/// <summary>
/// Called when [OmniaFeature resource mappings is being performed].
/// </summary>
/// <param name="resourceMapper">The resource mapper.</param>
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper
        .MapTenantResource<ResourcesMapping.PageLayouts>(q => q.SamplePageLayout)
        .WithSettingsForPageLayout()
        .DeploysTo(SharePointFileDeploymentTargets.MasterPageGallery);
}

/// <summary>
/// Called when [OmniaFeature sharepoint artifacts mappings is being performed].
/// </summary>
/// <param name="artifactMapper">The artifacts mapper.</param>
public override void OnSharePointArtifactMappings(SharePointArtifactMapper_
↪artifactMapper)
{
    artifactMapper.MapToField<SampleField>()
        .DeployTo(Ctx.Site.RootWeb);

    artifactMapper
        .MapToContentType<SampleContentType>()
        .DeployTo(Ctx.Site.RootWeb)
        .UpdateChildren();

    artifactMapper
        .MapToList<SampleList>()
        .DeployTo(Ctx.Site.RootWeb);
}

```

.. include:: ../../common/authors.txt

## Resource Mappings

In an Omnia extension, every resource files need to be mapped with a unique ID. The resource mappings are defined with an attribute-based pattern like other entities in Omnia like feature, jobs and SharePoint artifacts.

```

namespace MyOmniaExtension.TenantResources
{
    [TenantResourceFolderMapping(id: "DDA93FDC-883E-4EF3-9012-54E7FD55C4F1", name: "TenantResources")]
    public class ResourcesMapping
    {
        [TenantResourceFolderMapping(id: "8E6FF640-8C3D-41F8-BE58-339C18699351", name: "Images")]
        public class Images
        {
            [TenantResourceFileMapping("79C53D94-AFD9-4227-8A5A-103726FC526E", "/TenantResources/Images/sample.gif")]
            public string SampleImage { get; set; }
        }

        [TenantResourceFolderMapping(id: "6F70EF97-FF6A-4011-B902-AFC615B378E6", name: "Localization")]
        public class Localization
        {
            [TenantResourceFileMapping("766F18D7-1E6B-486D-912B-C12D6683B28A", "/TenantResources/Localization/sample.loc.json")]
            public string SampleLocalization { get; set; }

            [TenantResourceFileMapping("0CD92A43-9065-49DC-83EB-054123445A9E", "/TenantResources/Localization/sample.loc.sv-se.json")]
            public string SampleLocalizationSvSe { get; set; }
        }
    }
}

```

The resource mappings also help create an logical hierarchy or grouping of resources, which can be used to distribute the resources to different Omnia features. If your extension is small (less than 50 resource files) you should put all resource mappings into one file, otherwise you should split it into multiple files by feature area.

In the resource mappings you can also specify different properties and metadata.

#### Sections:

- *Folders mapping*
- *Generic files mapping*
- *Site templates mapping*
- *Localization files mapping*
- *SharePoint masterpages mapping*
- *SharePoint webparts mapping*
- *SharePoint pagelayouts mapping*

### Folders mapping

```

[TenantResourceFolderMapping(id: "EE2FFD07-D5B6-4964-BD46-09472229F49C", name:
↪"Scripts")]
public class Scripts
{
    // Other folders or files mapping code in here
}

```

### Generic files mapping

```

[TenantResourceFileMapping(id: "DC37115E-9ED7-4017-BCA1-449C67D8EBC0",
    sourceRelativePath: "/TenantResources/Scripts/Core/sample.core.js")]
public string SampleCoreJs { get; set; }

```

## Site templates mapping

```
[TenantResourceFileMapping(id: "AA5A09DA-CA8D-4289-9EF2-D05AE99E4AB2",
    sourceRelativePath: "/TenantResources/SiteTemplates/sampletemplate.json",
    Category = BuiltInCategories.SiteProvisioning.SiteTemplate)]
public string SampleTemplate { get; set; }
```

## Localization files mapping

```
[TenantResourceFileMapping(id: "766F18D7-1E6B-486D-912B-C12D6683B28A",
    sourceRelativePath: "/TenantResources/Localization/sample.loc.json")]
public string SampleLocalization { get; set; }

[TenantResourceFileMapping(id: "0CD92A43-9065-49DC-83EB-054123445A9E",
    sourceRelativePath: "/TenantResources/Localization/sample.loc.sv-se.json")]
public string SampleLocalizationSvSe { get; set; }
```

## SharePoint masterpages mapping

```
[TenantResourceFileMapping(id: "69B74E73-CDA2-4BB9-A917-C16F2FACB5D1",
    sourceRelativePath: "/TenantResources/MasterPages/sample.master")]
[ContentTypeId(typeof(Omnia.Foundation.Extensibility.ContentTypes.BuiltIn.
    ↳MasterPage))]
[SharePointFileProperty("MasterPageDescription", "")]
[SharePointFileProperty("UIVersion", "15")]
public string PortalMaster { get; set; }
```

## SharePoint webparts mapping

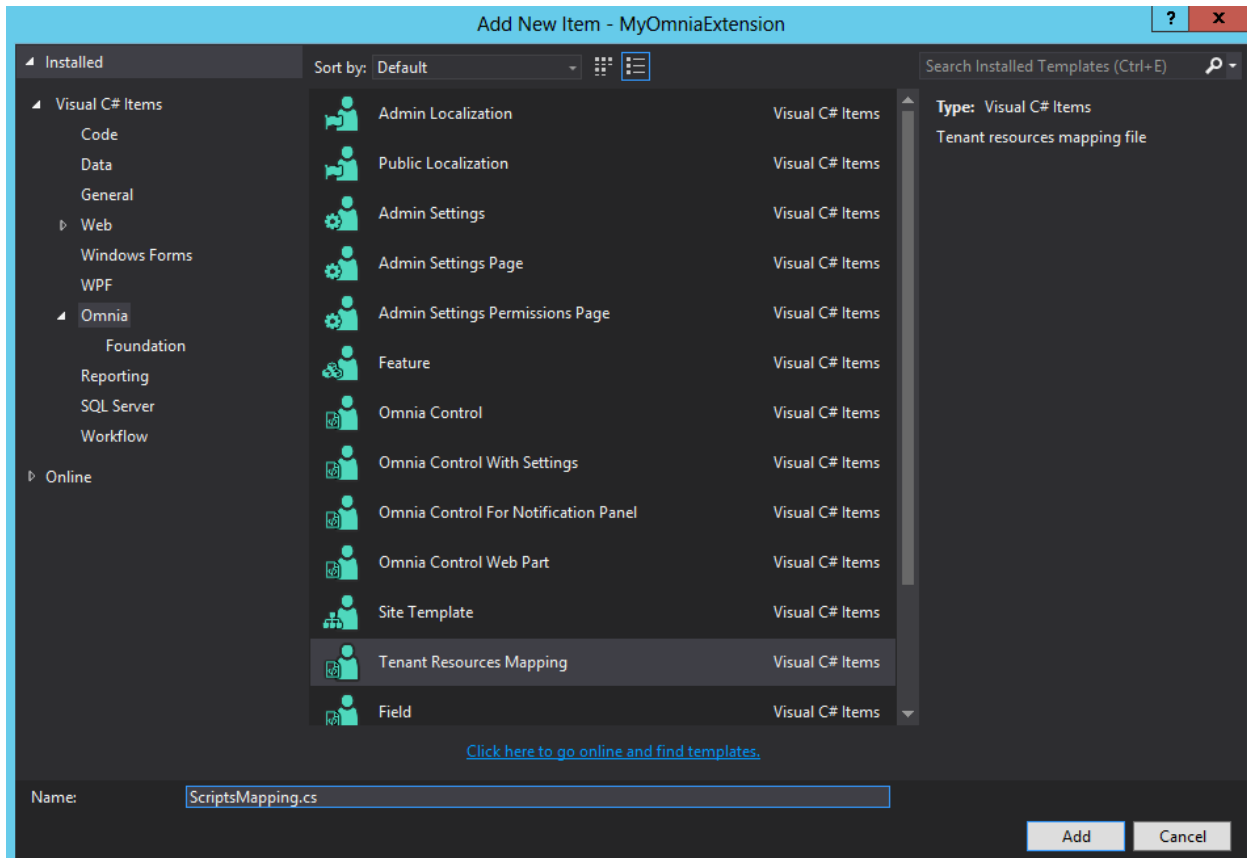
```
[TenantResourceFileMapping(id: "5486D161-E899-4AC5-BBCE-2F4B093B788C",
    sourceRelativePath: "/TenantResources/WebParts/sample.webpart")]
[SharePointFileProperty("Group", Omnia.Foundation.Core.Constants.WebPartGroups.Omnia)]
public string SampleWebPart { get; set; }
```

## SharePoint pagelayouts mapping

```
[TenantResourceFileMapping(id: "94D169CF-B8F0-4A55-9767-5F410DBAC9F5",
    sourceRelativePath: "/TenantResources/PageLayouts/SamplePageLayout.aspx")]
[ContentTypeId(typeof(Omnia.Foundation.Extensibility.ContentTypes.BuiltIn.
    ↳PageLayout))]
[PublishingAssociatedContentType(typeof(ArticlePage))]
[SharePointFileProperty("Title",
    "$Localize:MyOmniaExtension.Sample.PageLayouts.SamplePageLayout.Title;")]
public string SamplePageLayout { get; set; }
```

## Working with resource mappings

When developing new extension, after you have all the resources ready, you can create a new resource mapping class using Omnia tooling

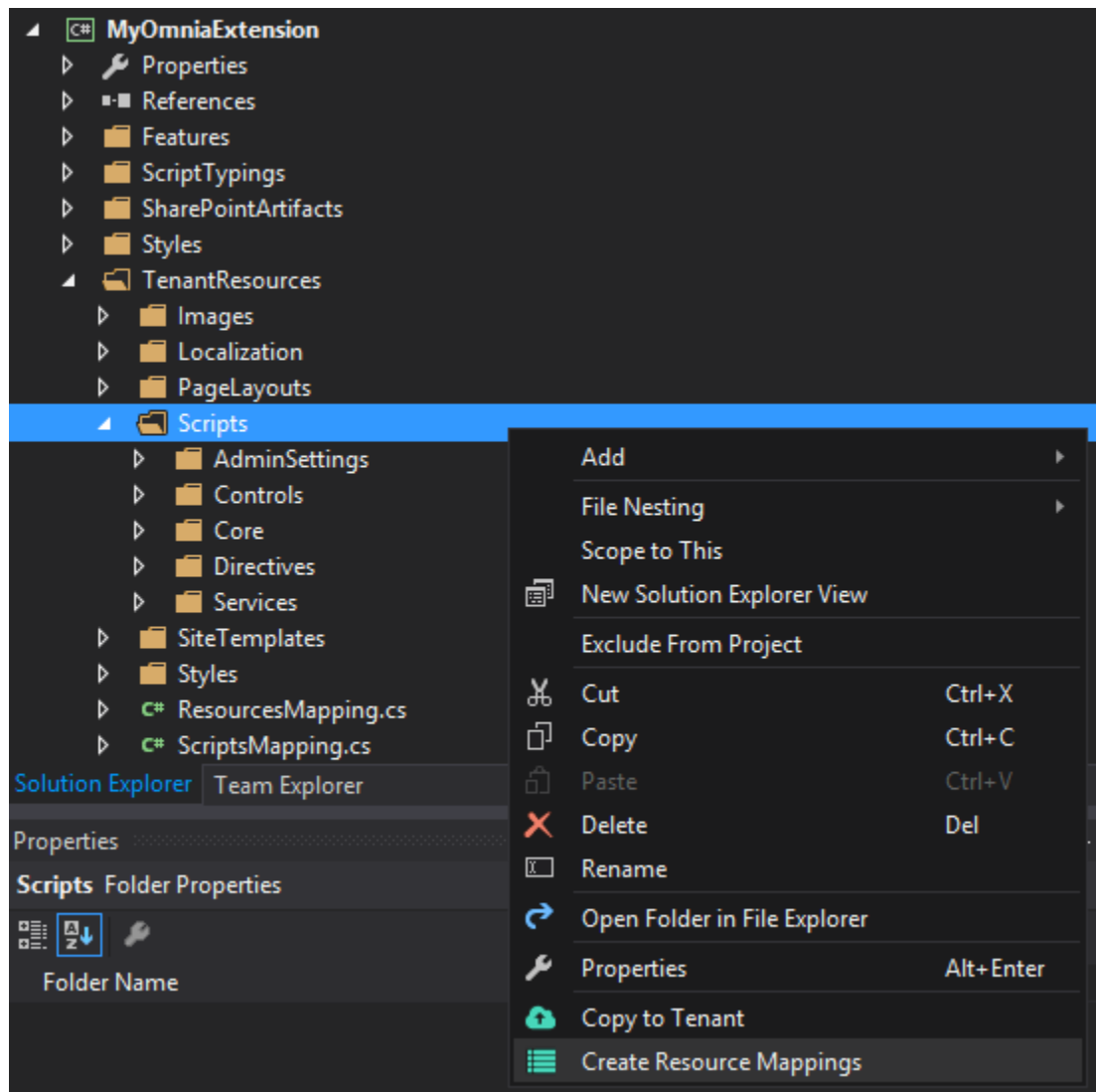


```

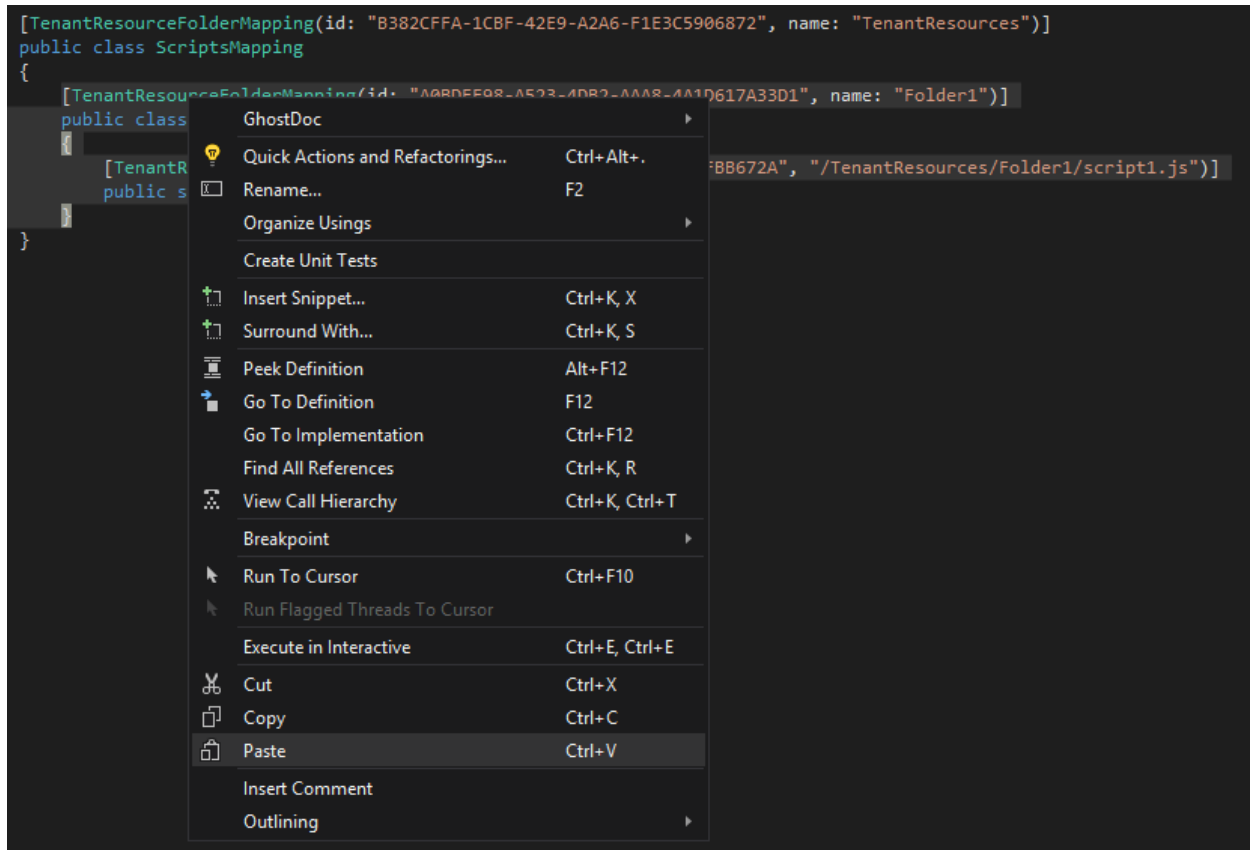
namespace MyOmniaExtension.TenantResources
{
    [TenantResourceFolderMapping(id: "B382CFFA-1CBF-42E9-A2A6-F1E3C5906872", name: "TenantResources")]
    public class ScriptsMapping
    {
        [TenantResourceFolderMapping(id: "A08DEE98-A523-40B2-AAA8-4A1D617A33D1", name: "Folder1")]
        public class Folder1
        {
            [TenantResourceFileMapping("006905C5-1D66-4561-B04E-669CDFBB672A", "/TenantResources/Folder1/script1.js")]
            public string Script1 { get; set; }
        }
    }
}

```

After the resource mapping class has been created, you can manually write code to map your resource files following that sample pattern here, or you can again use Omnia toolings to generate the mapping code for you. Right-click on the folder contains your new resources and select **Create Resource Mappings**



The generated mapping code has been copy to your clipboard, now go to the new **ScriptMappings** class you have created ealier and replace the sample code with the correct mapping



```
[TenantResourceFolderMapping(id: "B382CFFA-1CBF-42E9-A2A6-F1E3C5906872", name: "TenantResources")]
public class ScriptsMapping
{
    [TenantResourceFolderMapping(id: "EE2FFD07-05B6-4964-BD46-09472229F49C", name: "Scripts")]
    public class Scripts
    {
        [TenantResourceFolderMapping(id: "CC316600-2332-421B-8349-2F1F55786F0D", name: "AdminSettings")]
        public class AdminSettings
        {
            [TenantResourceFileMapping("534C8F21-B597-4016-B6E3-8F2F81202AE5", "/TenantResources/Scripts/AdminSettings/sample.adminsettings.form.js")]
            public string SampleAdminSettingsFormJs { get; set; }

            [TenantResourceFileMapping("308015D3-CCAC-4A7C-B011-C572B6086D97", "/TenantResources/Scripts/AdminSettings/sample.adminsettings.form.html")]
            public string SampleAdminSettingsFormView { get; set; }

            [TenantResourceFileMapping("D4666EF8-8E49-43B2-A4AA-91845ED15B3F", "/TenantResources/Scripts/AdminSettings/sample.adminsettings.js")]
            public string SampleAdminSettingsJs { get; set; }
        }

        [TenantResourceFolderMapping(id: "AB67D43E-2426-4BF4-BAB4-D3FC80067DE3", name: "Controls")]
        public class Controls
        {
            [TenantResourceFileMapping("9D916C28-4E24-4F32-9413-4B3BFFCDB764", "/TenantResources/Scripts/Controls/sample.controls.sample.settings.js")]
            public string SampleControlsSampleSettingsJs { get; set; }

            [TenantResourceFileMapping("5DCE4726-4394-4F6A-8981-BCED38AB3C25", "/TenantResources/Scripts/Controls/sample.controls.sample.settings.html")]
            public string SampleControlsSampleSettingsView { get; set; }

            [TenantResourceFileMapping("04D7EE5D-0DB9-47C5-B38D-DF41418D665E", "/TenantResources/Scripts/Controls/sample.controls.sample.js")]
            public string SampleControlsSampleJs { get; set; }

            [TenantResourceFileMapping("C20BCA87-2FFC-4F42-8737-E4CE0432054E", "/TenantResources/Scripts/Controls/sample.controls.sample.html")]
            public string SampleControlsSampleView { get; set; }
        }
    }
}
```

```
.. include:: ../common/authors.txt
```

## Localization

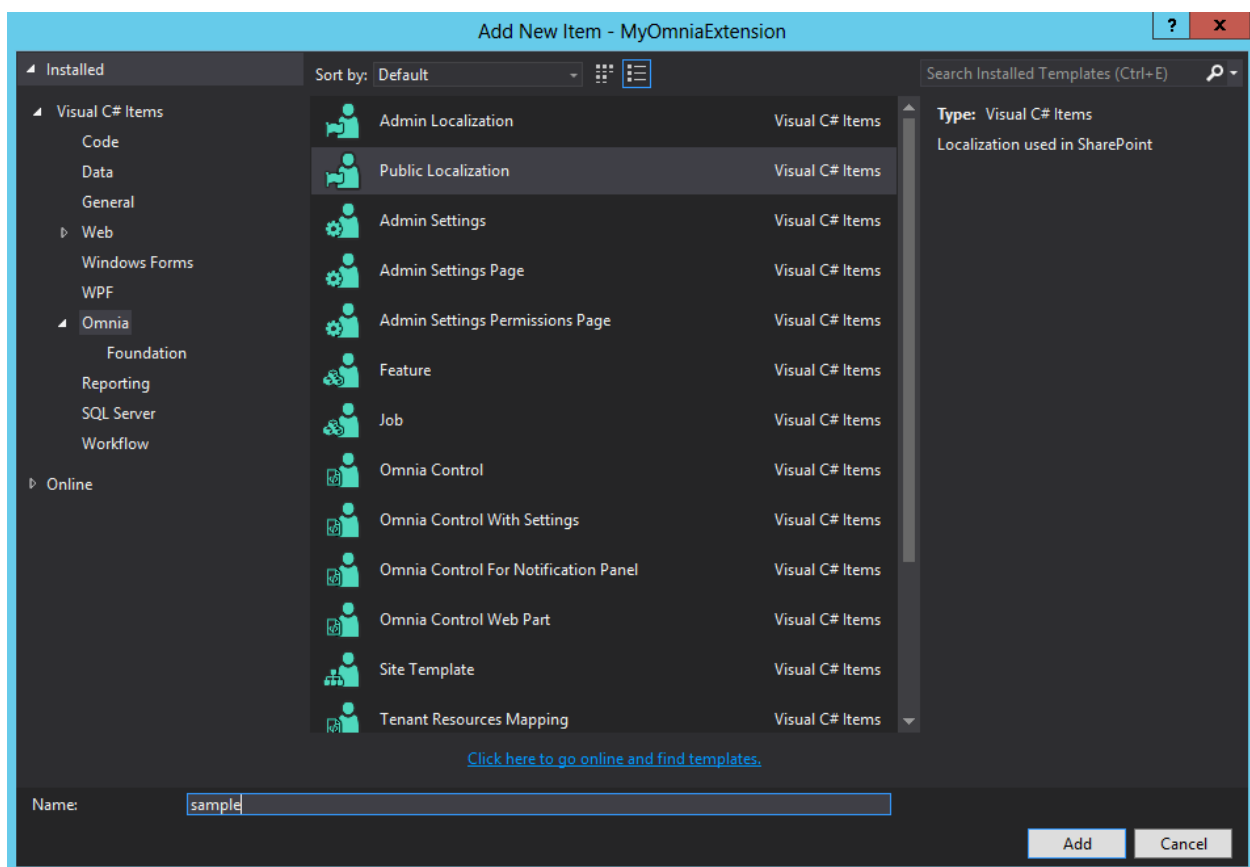
Omnia Foundation has a powerful localization engine that is using localization strings stored in JSON. By using json which is a language neutral format we can use it both server-side and client-side.

**Sections:**

- *Creating localization files*
- *Localization in client-side code*
- *Localization in server-side code*
- *Override localization in the Omnia admin app*

**Creating localization files**

You can create new localization file using the template from Omnia toolings

**Note:**

- **Public Localization** - Used to store localization strings for UI in SharePoint
- **Admin Localization** - Used to store localization strings for UI in Omnia admin app

Each localization file contains the localization strings for only one language, and it need to follow this naming convention:

- **\*.loc.json** for default language (English) localization. Example: sample.loc.json
- **\*.loc.[culture code].json** for other language localization. Example: sample.loc.sv-se.json

Most of the cases, localization files with the same name but different culture code should contain the same JSON structure with different string values.

---

**Note:** The first level of the JSON will always be “Public” or “Admin” depend on the localization target. You do not need to include “Public” or “Admin” part when getting localized strings.

---

### sample.loc.json

```
{
  "Public": {
    "MyOmniaExtension": {
      "Sample": {
        "SiteCollection": "Site Collection",
        "Site": "Site"
      }
    }
  }
}
```

### sample.loc.sv-se.json

```
{
  "Public": {
    "MyOmniaExtension": {
      "Sample": {
        "SiteCollection": "Webbplatssamling",
        "Site": "Webbplats"
      }
    }
  }
}
```

## Localization in client-side code

Once the localization resources have been deployed, your client-side code can get the localized strings in three ways:

- Using the global variable **\$localize**. This should only be used if you are writing non-Angular code. For Angular code you should use the other ways for better support.
- Using the filter **omfLocalization** for Angular

```
<span ng-bind="'MyOmniaExtension.Sample.SiteCollection' | omfLocalization">
```

- Using the service **localizationService** for Angular

```
constructor(private $scope: ISystemLogScope,
  private localizationService: Omnia.Foundation.Services.LocalizationService) {
  this.init();
}

private init = () => {
  this.$scope.logTypes = new Array<Log>();
  this.$scope.logTypes.push({
    source: this.localizationService.getText("System.SystemLogs.LogTypes.Info"),
    logType: LogTypes.Info
  });
}
```

(continues on next page)



(continued from previous page)

```

    });
    this.$scope.logTypes.push({
        source: this.localizationService.getText("System.SystemLogs.LogTypes.Warning
↪"),
        logType: LogTypes.Warning
    });
    this.$scope.logTypes.push({
        source: this.localizationService.getText("System.SystemLogs.LogTypes.Error"),
        logType: LogTypes.Error
    });
}

```

## Localization in server-side code

Server-side code can also use localized strings. Typical examples are localized email content in Omnia timer jobs and localized title of SharePoint fields and content types.

---

**Note:** Currently the title of Omnia features cannot be localized. This may become possible in future version.

---

### Example: Localized SharePoint content type

```

[ContentType(id: "78FBA358-10D6-459A-ABD9-6E1539EFF8C0",
    name: "$Localize:MyOmniaExtension.Sample.ContentTypes.SampleContentType.Name;",
    Group = "Sample Content Type Group",
    Description = "$Localize:MyOmniaExtension.Sample.ContentTypes.SampleContentType.
↪Description;")]
public class SampleContentType : Omnia.Foundation.Extensibility.ContentTypes.BuiltIn.
↪Item
{
    [FieldRef(typeof(SampleField))]
    public string SampleField { get; set; }
}

```

### Example: Get localized strings in timer jobs

```

public void SampleJobTimer([TimerTrigger("01:00:00")] TimerInfo timerInfo)
{
    try
    {
        string language = "en-US";

        string[] localizationKeys = new string[] {
            "$Localize:MyOmniaExtension.Sample.EmailSubject;",
            "$Localize:MyOmniaExtension.Sample.EmailContent;" };

        ILocalizationService localizationService = WorkWith().Localization();
        Dictionary<string, string> localizationsResult =
            localizationService.GetLocalization(localizationKeys, language);

        string localizedEmailSubject = "";
        localizationResult.TryGetValue(localizationKeys[0], out
↪localizedEmailSubject);
    }
}

```

(continues on next page)

(continued from previous page)

```

        string localizedEmailContent = "";
        localizationResult.TryGetValue(localizationKeys[1], out
↪ localizedEmailContent);
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("SampleJobTimer", ex.Message, DefaultLogTypes.
↪ Error, ex);
    }
}

```

## Override localization in the Omnia admin app

End users can change the localized strings using the Omnia admin app at **System > Localization**

**Note:** Once a localized string has been changed in the admin app it will not be updated when a newer version of extension package is deployed. To make get the latest version of the localization users need to undo the customization. On the otherhand, when an extension package is removed all customization will also be removed.

```
.. include:: ../../common/authors.txt
```

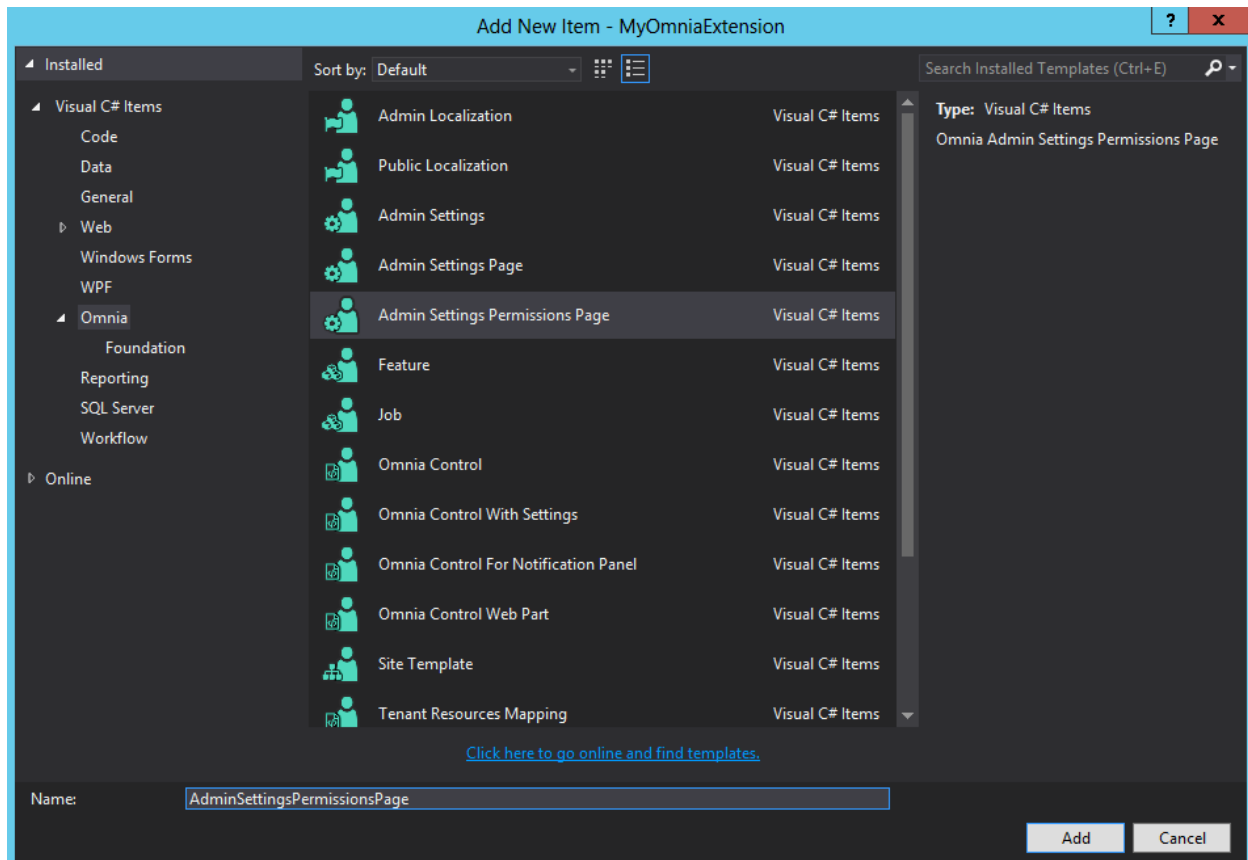
## Permissions

### Omnia permission roles

Omnia permission roles are security groups for managing access rights to Omnia resources. Permission roles can also be used to limit the access to the API of Omnia extensions. An Omnia permission role can be assign to a user or a security group in Office 365 and can be apply to site, site collection or tenant scope.

### Create form for managing permission roles in Omnia extension

Omnia toolings provides a template for managing permission roles in admin app for each feature



The permission settings page is the same as any other admin app settings page in an Omnia extensions. The first notable part is the **navigationNode**. In the navigationNode, along with other metadata, you need to specify the **authorizedRoles**: the permission roles that can access this page and manage other permission roles. Usually, the authorizedRoles for permission settings pages are set to the OmniaAdmin.

```
static navigationNode: NavigationNode = {
    title: "",
    state: "AdminSettingsPermissionsPagePermissions",
    url: "permission",
    controller: AdminSettingsPermissionsPagePermissionsController.ngName,
    viewId: "",
    iconClass: "fa-key",
    authorizedRoles: [
        {
            name: Omnia.Foundation.Security.PermissionRoles.OmniaAdmin,
            scope: Omnia.Foundation.Security.PermissionScopes.Tenant
        }
    ]
};
```

You will need to register this navigationNode to the navigation system of admin app. For more information about this read the article *admin settings pages*.

```
var node: NavigationNode = SampleSettings.SampleSettingsController.navigationNode;
if (node.children == null)
    node.children = [];
```

(continues on next page)

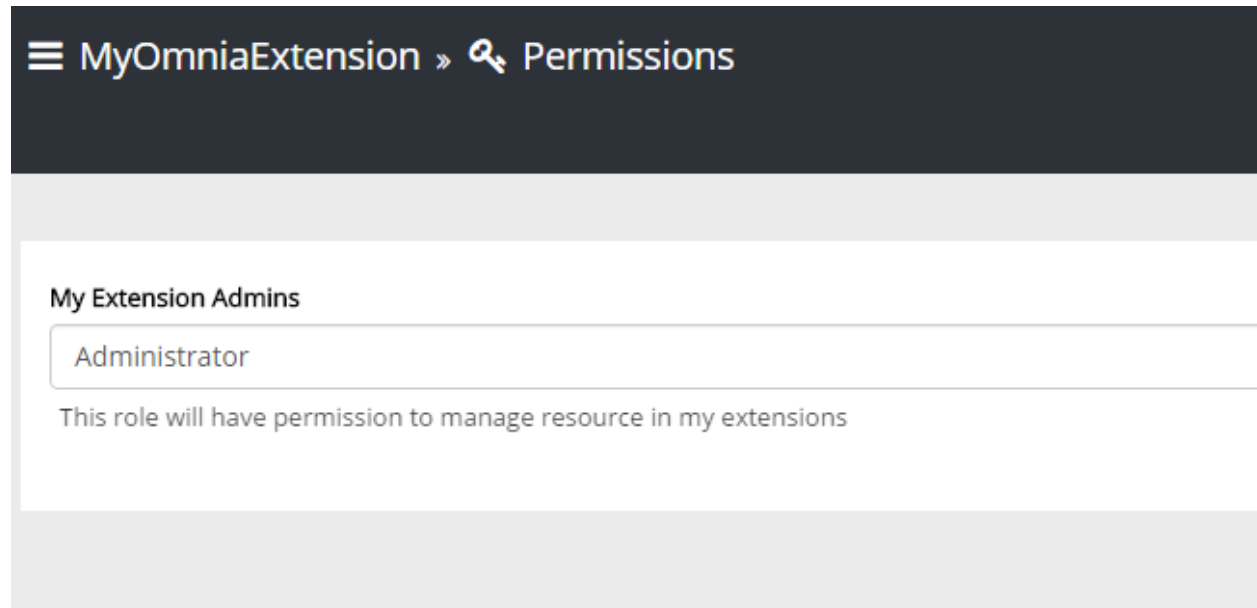
(continued from previous page)

```
node.children.push(SampleSettings.AdminSettingsPermissionsPagePermissionsController.
    ↪navigationNode);
$admin.Navigation.addNode(NavigationScope.tenant, node);
```

The last and most important part is the `roleDefinitionList` property on `$scope` where you define all permission roles that can be managed on this page.

```
private init = () => {
    this.$scope.roleDefinitionList = [
        {
            name: "AdminSettingsPermissionsPage.Admin",
            scope: PermissionScopes.Tenant,
            label: "AdminSettingsPermissionsPage.Permissions.Admin.Label",
            description: "AdminSettingsPermissionsPage.Permissions.Admin.Description"
        }
    ];
}
```

Deploy the extension and the permissions settings should be available



## Secured extension API

The API of extensions can limit the access to certain endpoints by using the **RequiredPermissionRoles** attribute

```
[HttpGet]
[Route("api/settings")]
[RequiredPermissionRoles("AdminSettingsPermissionsPage.Admin",
    PermissionScopes.Tenant)]
public ApiOperationResult<Settings> GetSettings()
{ }
```

```
.. include:: ../common/authors.txt
```

## Configuration

Omnia configuration API provides a key-value storage that can be accessed from both server-side and client-side code. A configuration can have the following properties:

Property	Type	Description
Name	String	Name of the configuration (case-insensitive)
Region	String	Group or category of the configuration (case-insensitive)
ExtensionPackageId	Guid	ID of the extension package that the configuration belongs to
Value	String	Value of the configuration
UIEditable	Boolean	Specify whether the configuration is editable in Omnia admin app
IncludedInClient	Boolean	Specify whether the configuration should be included in the client-side context in <code>_omniaContextInfo.customConfigurations</code>

A configuration “key” is the unique combination of its name, region and extension ID. Configuration value is string-based so it can be anything from plain string to URL or JSON.

### Configuration API for server-side code

---

**Note:** To use Omnia configuration API in server-side code you need to install the NuGet package **Omnia.Foundation.Extensibility.Core** to you project.

---

In server-side code, the configuration API can be called by using **OmniaApi** factory

```
IConfigurationService configurationService = OmniaApi.WorkWith(tenantId).
    ↳Configurations();
```

When working with configuration API, you can specify the extension ID that the configuration belongs to, or you can omit it. When omitted, the extension package ID will be read from the app setting **Omnia.Foundation.Settings.ExtensionId** of the current app domain.

**Example:** Get a single configuration by name and region

```
// Get a built-in configuration from Omnia Foundation
OmniaApi.WorkWith(tenantId).Configurations().GetConfiguration(
    name: "configuration-name",
    region: "configuration-region",
    extensionPackageId: Extensibility.Core.Constants.Extensions.
    ↳BuiltInExtensionPackageId);

// Get a configuration from your extension. When omitted, the extension package ID
    ↳will be read
// from the app setting 'Omnia.Foundation.Settings.ExtensionId' of the current app
    ↳domain
OmniaApi.WorkWith(tenantId).Configurations().GetConfiguration(
    name: "configuration-name",
    region: "configuration-region");

// Get a configuration from another extension.
OmniaApi.WorkWith(tenantId).Configurations().GetConfiguration(
    name: "configuration-name",
    region: "configuration-region",
    extensionPackageId: "extension-id");
```

**Example:** Get all configurations in a region

```
OmniaApi.WorkWith(tenantId).Configurations().GetConfigurationsInRegion(
    name: "configuration-region",
    extensionPackageId: "extension-id");
```

**Example:** Add or update a configuration

```
MyModel myModel = new MyModel();

OmniaApi.WorkWith(tenantId).Configurations().AddOrUpdateConfiguration(
    name: "configuration-name",
    value: JsonConvert.SerializeObject(myModel),
    region: "configuration-region",
    includedInClient: false,
    uiEditable: false,
    extensionPackageId: "extension-id");
```

**Example:** Delete a configuration



```
OmniaApi.WorkWith(tenantId).Configurations().DeleteConfiguration(
    name: "configuration-name",
    region: "configuration-region",
    extensionPackageId: "extension-id");
```

## Configuration for extension

You can automatically set the configurations for your extension when it is deployed by specifying those configurations in the **extension.json** file

```
{
  "Id": "3847fb18-8cb7-4597-83d8-6bcb7136ce7a",
  "Title": "MyOmniaExtension",
  "Description": "",
  "Version": "1.0.0",
  "PackageName": null,
  "TenantResourceFolders": [
    "TenantResources"
  ],
  "Configurations": [
    {
      "Name": "MyWebApiUrl",
      "Region": "MyOmniaExtension",
      "IncludedInClient": true,
      "UIEditable": true,
      "Required": true,
      "DefaultValue": "https://localhost:44300/api/"
    },
    {
      "Name": "TermSetId",
      "Region": "MyOmniaExtension",
      "IncludedInClient": true,
      "UIEditable": true,
      "Required": true,
      "DefaultValue": "888F3B90-9A90-4F77-B64D-305EFF1EB3D5"
    }
  ]
}
```

Configuration default values will be used when the extension package is deployed the first time from Visual Studio. When the package is uploaded from Omnia admin app user will need to fill in the configuration values.

 System > 
  Extensions > Upload Package

MyOmniaExtension

Name	Region	Value	Included in Client
mywebapiurl	myomniaextension	<input type="text"/>	<input checked="" type="checkbox"/>
termsetid	myomniaextension	<input type="text"/>	<input checked="" type="checkbox"/>

Upload Package

.. include:: ../common/authors.txt

### Bundling

Bundling is the process of concatenate multiple files into a single file. In web development, bundling can increase performance significantly, especially in high latency networks, by reducing the number of requests and round trips to the server.

In Omnia feature you can specify which JavaScript and CSS resources to be bundled and which application the bundle is targeted for (more on this in bundle targets section). When that feature is activated, those resources will be add to the bundle corresponding to the feature's scope and the target application.

#### Sections:

- *Creating bundles*
- *Bundle scopes*
- *Bundle targets*
- *Bundle sequence number*
- *Bundle minification*

### Creating bundles

To bundle resources, override the method `OnTenantResourceMappings` in your Omnia feature and use the method `CreateBundleFor` of the `resourceMapper`.

```
public override void OnTenantResourceMappings(TenantResourceMapper resourceMapper)
{
    resourceMapper
        .CreateBundleFor(Models.Features.BundleTargets.SharePoint)
        .Include<ResourcesMapping.Core>()
        .Include<ResourcesMapping.Services>(q => q.AjaxService)
        .Include<ResourcesMapping.Services>();
}
```

You can add specific file from the resources mapping

```
.Include<ResourcesMapping.Services>(q => q.AjaxService)
```

Or you can add a whole folder and the `resourceMapper` will recursively include every files in that folder. The resource mapper will be smart enough to not include a file again if it is already in the bundle

```
.Include<ResourcesMapping.Services>(q => q.AjaxService)
.Include<ResourcesMapping.Services>()
```

The resources will be bundled in the same order as they were included here. In this example, the resources will be bundled in the following order:

- All the files (recursively including child folders) in Core folder
- The `ajaxService` file in Services folder
- All the files in Services folder (recursively including child folders) except for `ajaxService`



## Bundle scopes

There are 3 bundle scopes: Tenant, Site Collection and Site. These scopes map with Omnia feature scopes, meaning an tenant-scope feature can only add resources to the tenant-scope bundle, etc.

On each Omnia page load (in SharePoint or in admin web), these 6 bundle files that will be loaded:

Scope	Bundle	Content
Site	<ul style="list-style-type: none"> <li>• site.js</li> <li>• site.css</li> </ul>	All resources bundled by site-scope features activated on the current site
SiteCollection	<ul style="list-style-type: none"> <li>• sitecollection.js</li> <li>• sitecollection.css</li> </ul>	All resources bundled by sitecollection-scope features activated on the current site collection
Tenant	<ul style="list-style-type: none"> <li>• tenant.js</li> <li>• tenant.css</li> </ul>	All resources bundled by activated tenant-scope features

## Bundle targets

Currently there are only two bundle targets, SharePoint and OmniaAdmin. As the names imply, resources bundles targeting SharePoint will only be loaded in SharePoint and the resources bundle targeting OmniaAdmin will only be loaded in Omnia admin web.

A general guideline on bundle targets would be:

- JavaScript services and directives usually can target both SharePoint and Omnia Admin.
- Omnia controls should only target SharePoint.
- Admin settings should only target OmniaAdmin.

```
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper
        .AddOrUpdateTenantResourcesFrom<TenantResources>();

    resourceMapper
        .CreateBundleFor(Models.Features.BundleTargets.SharePoint)
        .Include<ResourcesMapping.Enums>()
        .Include<ResourcesMapping.Core>()
        .Include<ResourcesMapping.Services>()
        .Include<ResourcesMapping.Directives>()
        .Include<ResourcesMapping.Styles>();

    resourceMapper
```

(continues on next page)

(continued from previous page)

```

        .CreateBundleFor (Models.Features.BundleTargets.OmniaAdmin)
        .Include<ResourcesMapping.Enums> ()
        .Include<ResourcesMapping.Core> ()
        .Include<ResourcesMapping.Services> ()
        .Include<ResourcesMapping.Directives> ()
        .Include<ResourcesMapping.Styles> ()
        .Include<ResourcesMapping.AdminSettings.Controllers> ()
        .Include<ResourcesMapping.AdminSettings> ();
    }

```

## Bundle sequence number

While you can specify the order of resources in your feature just by order they were included, sometimes you will also need to ensure the resources of one feature is loaded before the resources of other features. For that purpose you can set the sequence number for your feature bundle:

```

public override void OnTenantResourceMappings (TenantResourcesMapper resourceMapper)
{
    resourceMapper
        .AddOrUpdateTenantResourcesFrom<TenantResources> ();

    resourceMapper
        .CreateBundleFor (Models.Features.BundleTargets.SharePoint)
        .Include<ResourcesMapping.Enums> ()
        .Include<ResourcesMapping.Core> ()
        .Include<ResourcesMapping.Services> ()
        .Include<ResourcesMapping.Directives> ()
        .Include<ResourcesMapping.Styles> ();

    resourceMapper
        .CreateBundleFor (Models.Features.BundleTargets.OmniaAdmin)
        .Include<ResourcesMapping.Enums> ()
        .Include<ResourcesMapping.Core> ()
        .Include<ResourcesMapping.Services> ()
        .Include<ResourcesMapping.Directives> ()
        .Include<ResourcesMapping.Styles> ()
        .Include<ResourcesMapping.AdminSettings.Controllers> ()
        .Include<ResourcesMapping.AdminSettings> ();

    resourceMapper
        .SetBundlesSequence (90000, Models.Features.BundleTargets.SharePoint)
        .SetBundlesSequence (80000, Models.Features.BundleTargets.OmniaAdmin);
}

```

The bundle with lower sequence number will be included first in the bundle. The default sequence number is 100000. You should not set the sequence number to lower than 100 because the sequence numbers from 0 to 100 are reserved for core features of Omnia Foundation.

Also, from the example you can see that the sequence number can be different for bundling targets.

## Bundle minification

In non-development environments, all JavaScript bundles will be **minified** to reduce the size of the bundles and further improve performance. However, this sometimes can cause issues if the code was not written in a way that is compatible

with minification. If you have errors happened only in non-development environments and you suspect it could be from the minification, use the `querystring` parameter “**debug=true**” to un-minify your code.

One common issue with minification is Angular dependencies injection. For example, this code will not work when minified

```
var app = angular.module('bigApp', []);

app.controller('mainController', function($scope) {
    $scope.message = 'OH NO!';
});
```

But this will code will

```
var app = angular.module('bigApp', []);

app.controller('mainController', ['$scope', function($scope) {
    $scope.message = 'HOORAY!';
}]);
```

To understand why the second code block works with minification while the first does not, read [this article](#).

.. include:: ../common/authors.txt

## Logging

Omnia provides a logging API for extensions to write logs to Omnia Foundation’s logs database. There are three different types of logs in Omnia: System Logs, Queue Logs and Feature Logs.

System Logs is a general-purpose place for information or error logs from both Omnia Foundation and extensions. System Logs can be viewed in Omnia admin app at **System > Logs**

System > Q Logs

System Logs

Unable to connect to the remote server

Filter

Source	Message	Type
2016-08-05 04:51:11 Omnia.Foundation.Core.Security.SecurityServiceExtended	Unable to connect to the remote server	Error

**Exception**

System.Web.HttpException: Unable to connect to the remote server --> System.Net.Sockets.SocketException: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond 172.27.80.71:7777 at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress) at System.Net.ServicePoint.ConnectSocketInternal(Boolean connectFailure, Socket s4, Socket s6, Socket& socket, IPAddress& address, ConnectSocketState state, IAsyncResult asyncResult, Exception& exception) --- End of inner exception stack trace --- at System.Net.HttpWebRequest.GetRequestStream(TransportContext& context) at System.Net.HttpWebRequest.GetRequestStream() at Microsoft.SharePoint.Client.SPWebRequestExecutor.GetRequestStream() at Microsoft.SharePoint.Client.ClientRequest.ExecuteQueryToServer(ChunkStringBuilder sb) at Microsoft.SharePoint.Client.ClientRequest.ExecuteQuery() at Microsoft.SharePoint.Client.ClientRuntimeContext.ExecuteQuery() at Microsoft.SharePoint.Client.ClientContext.ExecuteQuery() at Omnia.Foundation.Core.SharePoint.ExtendedClientContext.ExecuteQueryRetry(Int32 retryCount, Int32 delay) in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\SharePoint\ExtendedClientContext.cs:line 84 at Omnia.Foundation.Core.SharePoint.ExtendedClientContext.ExecuteQuery() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\SharePoint\ExtendedClientContext.cs:line 36 at Omnia.Foundation.Core.Security.SecurityServiceExtended.GetUserPermissionRoles() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\Security\SecurityServiceExtended.cs:line 1040 at Omnia.Foundation.Core.Security.SecurityServiceExtended.GetCurrentUserPermissionRolesDictionary() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\Security\SecurityServiceExtended.cs:line 421

2016-07-29 12:02:05 Omnia.Foundation.Core.Security.SecurityServiceExtended Unable to connect to the remote server Error View Exception

Queue Logs contains logs from queue messages jobs. For errors related to long-running operations like uploading extension packages or creating site collection, check the Queue Logs. Queue Logs can be viewed in Omnia admin app at **System > Queues**

System > Queues

Queue Messages

Filter by Transaction Id...

Filter

Start Time	End Time	Name	Transaction	Status	
2016-08-04 13:25:56	2016-08-04 13:26:00	processfeature	d7a463c3-a577-458d-8f17-f098918968e0	Success	<a href="#">View Details</a>
2016-08-04 13:25:16	2016-08-04 13:25:28	processfeature	c2f41878-ec95-40d4-93d8-6501d5b0b7d3	Success	<a href="#">View Details</a>
2016-08-04 12:15:31	2016-08-04 12:15:59	processfeature	62f3e62a-7723-4047-8224-d83fcbcbffdf	Success	<a href="#">View Details</a>
2016-08-03 11:25:59	2016-08-03 11:26:04	newextensionpackage	85366cf5-b32e-4d93-bb56-6fdb419c6fd	Success	<a href="#">View Details</a>

The last type of logs is Feature Logs, which contains logs from custom code in feature activation, upgrade or deactivation. Feature Logs can be viewed in the feature detail page.

Features > Omnia Core MasterPage

Omnia Core MasterPage: 1.0.6

Current Site Collection

Url	Version	
/	1.0.6	<a href="#">Hide log</a> <a href="#">Remove</a> <a href="#">Upgrade</a>

Logs - Refresh

2016-06-30 06:12:45	Upgrade feature from version 1.0.6
2016-06-29 11:01:33	Upgrade feature from version 1.0.5

## Logging in extension Web API

In extension API that inherit from **SharePointContextProvidedController** you can use the built-in **Logging** service to write to the System Logs.

```
[HttpGet]
[Route("api/documents")]
public ApiOperationResult<IEnumerable<Document>> GetDocuments()
{
    try
    {
        // Web API code here
    }
    catch (Exception ex)
    {
        // The built-in Logging service can be used to write to System Logs
        this
            .Logging
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error, ex);

        return ApiUtils.CreateErrorResult<IEnumerable<Document>>(ex);
    }
}
```

For API that does not inherit from **SharePointContextProvidedController**, you can use **OmniaApi** factory to create the

**Logging** service.

```
[HttpGet]
[Route("api/documents")]
public ApiOperationResult<IEnumerable<Document>> GetDocuments (
    string tokenKey, string spUrl, string language)
{
    try
    {
        // Web API code here
    }
    catch (Exception ex)
    {
        // For controller that does not inherit from
        ↳ SharePointContextProvidedController
        // we need to create the ClientContext first.
        ClientContext ctx = SharePointContextProvider
            .CreateUserClientContext(tokenKey, spUrl, language);

        OmniaApi
            .WorkWith(Ctx.Omnia())
            .Logging()
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error, ex);

        return ApiUtils.CreateErrorResult<IEnumerable<Document>>(ex);
    }
}
```

## Logging in extension features

In Omnia feature you can write to System Logs using **OmniaApi** factory or write to Feature Logs using the built-in **Log** method

```
[FeatureDefinition(
    id: "85544C6C-9EB9-4F99-9410-95F1EA3D07B5",
    name: "MyOmniaExtension Sample Feature Core",
    version: "0.1.0",
    scope: FeatureScopes.Tenant
)]
public class SampleFeatureCore : Omnia.Foundation.Extensibility.Features.OmniaFeature
{
    /// <summary>
    /// Activates the OmniaFeature
    /// </summary>
    public override void Activate()
    {
        // This will write to System Logs
        this.WorkWith().Logging()
            .AddLog("SampleFeatureCore", "Feature activation", DefaultLogTypes.Info);

        // This will write to Feature Logs
        this.Log("Feature activation", "Success", FeatureInstanceLogTypes.
        ↳ Information);
    }
}
```

## Logging in extension jobs

Similar to features, in jobs you can use the **OmniaApi** factory to write to System Logs. One important thing to note is that currently any uncached error in queue job will be write to the Queue Logs, but for timer jobs you need to handle the error and explicitly write to the System Logs in your code.

```
public void SampleJobTimer ([TimerTrigger("01:00:00")] TimerInfo timerInfo)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("SampleJobTimer", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}

public void SampleJobQueue ([QueueTrigger("SampleJob")] object queueMessage)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("SampleJobQueue", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}
```

.. include:: ../common/authors.txt

## 1.3 SharePoint Provisioning

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### 1.3.1 Topics

.. include:: ../common/authors.txt

#### Field

Fields are used in Lists and Content Types in SharePoint and Omnia Foundation makes it easy to create field definitions by using attributes that can be provisioned using a, *Omnia feature* or referenced in a *List definition*.

All field attributes are located in the **Omnia.Foundation.Extensibility.Fields** namespace and the following list contains the different attributes that can be used to create field definitions.

Attribute	Description
FieldAttribute	The base attribute that all field attributes inherits from containing generic properties like title, description etc
BooleanFieldAttribute	Boolean field definition
CalculatedFieldAttribute	Calculated field definition
DateTimeFieldAttribute	DateTime field definition
HTMLFieldAttribute	HTML field definition
LookupFieldAttribute	Lookup field definition
ManagedMetadataFieldAttribute	Managed metadata field definition
NoteFieldAttribute	Note field definition
NumberFieldAttribute	Number field definition
TextFieldAttribute	Text field definition
UrlFieldAttribute	Url field definition
UserFieldAttribute	User field definition

Field definitions can be referenced both in list definitions and content type definitions, so they are defined using class level attribute decoration. Below is an example of how to define a basic NoteField using the NoteFieldAttribute

```
[NoteField(id: "37643EF6-2BB9-429B-BD19-4684FC7879DD", internalName: "MyNoteField",
    Title = "My Title", Group = "Custom Group")]
public class MyNoteField : FieldBase
{
}
```

**Note:** Even if the attribute is called **NoteFieldAttribute** its not necessary to include Attribute in the name, **NoteField** is enough and that will make the code look more clean.

All field definitions requires the minimal **id** and **internalName** parameters since this is required to be able to create the field. For the different field attributes there are always minimal required parameters that need to be provided but there is also optional properties that can be specified. Below is an example of all the optional properties that can be specified on the NoteFieldAttribute

```
namespace MyOmniaExtension.Fields
{
    [NoteField(id: "37643EF6-2BB9-429B-BD19-4684FC7879DD", internalName: "MyNoteField",
        Title = "My Title", Group = "Custom Group")]
    public class NoteFieldAttribute : FieldAttribute
    {
        public NoteFieldAttribute(string id, string internalName, Properties: [AllowDeletion = bool], [AllowHyperlink = bool],
            [AppendOnly = bool], [DefaultValue = string], [Description = string], [Direction = string], [DisplaySize = string],
            [EnforceUniqueValues = bool], [FieldTypeKind = Microsoft.SharePoint.Client.FieldType], [Group = string], [Hidden = bool],
            [Id = Guid], [Indexed = bool], [InternalName = string], [IsolateStyles = bool], [JSLink = string],
            [LinkToItemAllowed = ListItemMenuState], [ListItemMenuAllowed = ListItemMenuState], [NumberOfLines = int], [PIAttribute = string],
            [PITarget = string], [PrimaryPIAttribute = string], [PrimaryPITarget = string], [ReadOnlyField = bool],
            [Required = bool], [RestrictedMode = bool], [RichText = bool], [RichTextMode = RichTextMode], [SchemaXml = string],
            [Sealed = bool], [ShowInDisplayForm = bool], [ShowInEditForm = bool], [ShowInListSettings = bool],
            [ShowInNewForm = bool], [ShowInVersionHistory = bool], [StaticName = string], [Title = string], [TypeAsString = string],
            [UnlimitedLengthInDocumentLibrary = bool], [ValidationFormula = string], [ValidationMessage = string])
        {
        }
    }
}
```

Now that we have a field definition we can either use a site-scoped or sitecollection-scoped *Omnia feature* to deploy it to a site or use it in a *List definition*.

```
.. include:: ../common/authors.txt
```

## Content Type

Content Types are used in SharePoint to classify information. Omnia Foundation makes it easy to create content type definitions by using attributes that can be provisioned using an *Omnia feature* or referenced in a *List definition*.

Content Types are hierarchical, meaning that new content types needs to inherit from an existing one. The base content type in SharePoint is **Item**.

## Create a Content Type

1. In your Visual Studio project, right click and choose **Add -> New item**
2. From the Omnia group, choose the **Content Type** template and give your content type class a name. Click **Add**

When doing this a new class is created in your project, inheriting from **Omnia.Foundation.Extensibility.ContentTypes.BuiltIn.Item**. This class represents the **Item** content type in SharePoint.

## Inheritance

To make your content type inherit from another content type than **Item**, you can either choose to inherit from another class in the **Omnia.Foundation.Extensibility.ContentTypes.BuiltIn** or from any other content type class in your project.

## Properties

The content type class is also decorated with a **ContentType** attribute. This attribute tells Omnia that your class should be treated as a content type and contains a number of required properties:

Attribute	Description
id	A unique GUID for the content type, adds as part of the ContentTypeId
name	The name of the content type. Could be a <i>Localization</i> string or a plain text string
Group	The group of the content type
Description	The description of the content type. Could be a <i>Localization</i> string or a plain text string

There is also a large number of optional properties that can help you configure the content type the way you want.

## Adding fields

To add fields to your content type you add properties to your class:

```
[FieldRef(typeof(Preamble))]  
public string Preamble { get; set; }
```

In the **FieldRef** attribute you supply the class of an existing *Field*

In the **FieldRef** attribute you can also specify a number of optional properties, to for example make the field required or hidden:

```
[FieldRef(typeof(Preamble), Required = true)]  
public string Preamble { get; set; }
```



## Provisioning

As mentioned, a content type can be provisioned directly via an *Omnia feature*. This way the content type will be created as a Site Content Type.

It can also be provisioned via a List definition class. This way the content type will be a List Content Type.

### Provision Site Content Type

In a site scoped feature, in the **OnSharePointArtifactMappings(SharePointArtifactMapper artifactMapper)** method, add the following code

```
public override void OnSharePointArtifactMappings(SharePointArtifactMapper
↪artifactMapper)
{
    artifactMapper.MapToContentType<MyContentTypeClass>().
    ApplyChangeOn(Ctx.Web);
}
```

You can also tell Omnia to add the content type to a list:

```
public override void OnSharePointArtifactMappings(SharePointArtifactMapper
↪artifactMapper)
{
    artifactMapper.MapToContentType<MyContentTypeClass>().
    ApplyChangeOn(Ctx.Web).
    AddToList<Omnia.Foundation.Extensibility.Lists.Builtin.Pages>();
}
```

### Provision List Content Type

In a List class find the **public IEnumerable<ContentTypeBase> ContentTypes** property.

Decorate this property with a **ContentTypeRef** attribute

```
[ContentTypeRef(typeof(MyContentTypeClass))]
public IEnumerable<ContentTypeBase> ContentTypes
{
    get { return GetContentTypes(); }
}
```

In the **ContentTypeRef** attribute, supply the class of your content type, or the ContentTypeId of an existing content type.

The **ContentTypeRef** attribute also contains a number of optional parameters to allow you to for example make the content type the default one for the list.

.. include:: ../common/authors.txt

## List

A List is one of the core SharePoint artifacts and Omnia Foundation makes it easy to create list definitions by using attributes that can be provisioned using *Omnia features*.

### Create a List

1. In your Visual Studio project, right click and choose **Add -> New item**

2. From the Omnia group, choose the **List** template and give your list class a name. Click **Add**

When doing this a new class is created in your project, inheriting from **Omnia.Foundation.Extensibility.Lists.ListBase** and implementing the **Omnia.Foundation.Extensibility.Lists.IListBase** interface.

The class is also decorated with a **List** attribute with a number of required properties

## Properties

The List attribute has the following required properties

Attribute	Description
url	The site relative path of the list
title	The title of the list. Could be a <i>Localization</i> string or a plain text string
listTemplate	The list template type to inherit from, defaults to <b>ListTemplateType.GenericList</b>

There is also a large number of optional properties that can help you configure the list the way you want.

## Adding Fields

You can add *Fields* to your list by following the steps below.

In the List class find the **public IEnumerable<FieldBase> Fields** property.

Decorate this property with a **FieldRef** attribute

```
[FieldRef(typeof(LinkTitle))]  
public IEnumerable<FieldBase> Fields  
{  
    get { return GetFields(); }  
}
```

In the **FieldRef** attribute, supply the class of your field or a class from **Omnia.Foundation.Extensibility.Fields.BuiltIn**.

The **FieldRef** attribute also contains a number of optional parameters to allow you to for example make the field required in the list.

## Adding Content Types

You can add *Content Types* to your list by following the steps below.

In the List class find the **public IEnumerable<ContentTypeBase> ContentTypes** property.

Decorate this property with a **ContentTypeRef** attribute

```
[ContentTypeRef(typeof(MyContentTypeClass))]  
public IEnumerable<ContentTypeBase> ContentTypes  
{  
    get { return GetContentTypes(); }  
}
```

In the **ContentTypeRef** attribute, supply the class of your content type, or the ContentTypeId of an existing content type.

The **ContentTypeRef** attribute also contains a number of optional parameters to allow you to for example make the content type the default one for the list.

### Setup the default view

Omnia also provides logic to configure the default view of the list.

In the Lists class find the **public IEnumerable<FieldBase> DefaultView** property. Decorate it with **FieldRef** attributes where you supply the type for the field and the index you want the field to have in the view.

```
[FieldRef(typeof(DocIcon), 1)]
[FieldRef(typeof(LinkTitle), 2)]
[FieldRef(typeof(Modified), 3)]
[FieldRef(typeof(Author), 4)]
public IEnumerable<FieldBase> DefaultView
{
    get { return GetDefaultViewFields(); }
}
```

### Provisioning

A list is provisioned directly via a **site scoped Omnia feature**


In the **OnSharePointArtifactMappings(SharePointArtifactMapper artifactMapper)** method, add the following code

```
public override void OnSharePointArtifactMappings(SharePointArtifactMapper
    artifactMapper)
{
    artifactMapper
        .MapToList<MyListClass>()
        .DeployTo(Ctx.Web);
}
```

```
.. include:: ../common/authors.txt
```

### Image Rendition

By using attributes it is easy to specify and provision Image Renditions to SharePoint

 Start ▾ EDIT LINKS					
Image Renditions ⓘ					
Start	ID	Name	Width	Height	Edit Delete
About us	<a href="#">Add new item</a>				
How we work	1	Display Template Picture 3 Lines	100px	100px	
My employment	2	Display Template Picture On Top	304px	100px	
	3	Display Template Large Picture	468px	220px	
EDIT LINKS	4	Display Template Video	120px	68px	
Site Contents	1001	Landscape	600px	300px	
	1002	Square	300px	300px	
	1003	Portrait	300px	450px	
	1004	Landscape (small size)	300px	150px	

To create Image Renditions start by creating a new class that inherits from **ImageRenditionBase** then apply the **ImageRendition** attribute on your class. The **id** of your Image Rendition should start on a higher number so that it won't conflict with the built in SharePoint and Omnia Image Renditions. SharePoint starts with Id 1 and ends on 4 and Omnia on 1001 and ends on 1004 but your customization could start on 2000 to be sure not to write over existing renditions. If you want the height to be dynamic then you can set it to **0**

```
[ImageRendition(id: 1001, name: "$Localize:OMF.Core.ImageRenditions.Landscape;",
↪width: 600, height: 300)]
[ImageRendition(id: 1002, name: "$Localize:OMF.Core.ImageRenditions.Square;", width:
↪300, height: 300)]
[ImageRendition(id: 1003, name: "$Localize:OMF.Core.ImageRenditions.Portrait;",
↪width: 300, height: 450)]
[ImageRendition(id: 1004, name: "$Localize:OMF.Core.ImageRenditions.
↪LandscapeSmallSize;", width: 300, height: 150)]
public class PortalCoreImageRendition : ImageRenditionBase
{
}
```

Now that we have the specification of the Image Renditions all we need to do is to add the mapping code in the **OnSharePointArtifacts** method of an *Omnia feature* that will perform the deployment.

**Note:** The provisioning must be done from a Site Collection scoped feature

```
public override void OnSharePointArtifactMappings(SharePointArtifactMapper
↪artifactMapper)
{
    artifactMapper.MapToImageRendition<PortalCoreImageRendition>();
}
```

.. include:: ../common/authors.txt

## File

Using Omnia it is easy to provision different kinds of files to SharePoint. You can for example provision Page Layouts and Webpart Definitions, but also any other file you need.

The first thing you need to do is to add the file to your Visual Studio Project, inside a Tenant Resources folder. Then add the file to a Resource Mappings class as *described here*.

**Note:** You need to decorate your mappings differently based on the type of file you are adding (ex. for Page Layouts).

After creating the needed Resource Mappings, follow the steps in the section below matching your file type.

```
.. include:: ../common/authors.txt
```

## Page Layout

Page layouts in SharePoint are stored in the Master Page gallery on the root web of a site collection. Therefore it is recommended to provision page layout files in a Site Collection scoped feature.

In a Site Collection scope feature, locate the **public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)** method.

Add the following code, replacing **TenantResourcesMapping.PageLayouts** with the class name of your Page layout in the Resource Mappings class, and **StartPage** with the property representing your page layout.

```
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.PageLayouts>(q => q.
    ↪StartPage)
        .WithSettingsForPageLayout()
        .DeploysTo(SharePointFileDeploymentTargets.MasterPageGallery);
}
```

Note that this is a fluent API where you after the **.WithSettingsForPageLayout()** can modify a number of additional things about how the file will be provisioned, for example the file name

```
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.PageLayouts>(q => q.
    ↪StartPage)
        .WithSettingsForPageLayout()
        .SetFileName("StartPageLayout.aspx")
        .DeploysTo(SharePointFileDeploymentTargets.MasterPageGallery);
}
```

```
.. include:: ../common/authors.txt
```

## Glue Layout

Glue layouts in Omnia are used as layouts for Quick Pages.

They are stored in the Master Page gallery on the root web of a site collection. Therefore it is recommended to provision page layout files in a Site Collection scoped feature.

In a Site Collection scope feature, locate the **public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)** method.

Add the following code, replacing **TenantResourcesMapping.GlueLayouts** with the class name of your Glue layout in the Resource Mappings class, and **StartPage** with the property representing the specific glue layout.

```
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.GlueLayouts>(q => q.
    ↪StartPage)
```

(continues on next page)

(continued from previous page)

```

        .WithSettingsForGlueLayout ()
        .DeploysTo (SharePointFileDeploymentTargets.MasterPageGallery);
    }

```

Note that this is a fluent API where you after the **.WithSettingsForGlueLayout()** can modify a number of additional things about how the file will be provisioned, for example the file name

```

public override void OnTenantResourceMappings (TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.GlueLayouts>(q => q.
    ↪StartPage)
        .WithSettingsForGlueLayout ()
        .SetFileName ("StartPageLayout.aspx")
        .DeploysTo (SharePointFileDeploymentTargets.MasterPageGallery);
}

```

or make it the default layout for new quick pages

```

public override void OnTenantResourceMappings (TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.GlueLayouts>(q => q.
    ↪StartPage)
        .WithSettingsForGlueLayout ()
        .SetAsDefault ()
        .DeploysTo (SharePointFileDeploymentTargets.MasterPageGallery);
}

```

You can also add tokens in your layouts that can be replaced when the file is provisioned to SharePoint

```

public override void OnTenantResourceMappings (TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.GlueLayouts>(q => q.
    ↪StartPage)
        .WithSettingsForGlueLayout ()
        .TokenReplace (" [WeWantTheWebsTitleHere] ", Ctx.Web.Title)
        .DeploysTo (SharePointFileDeploymentTargets.MasterPageGallery);
}

```

```
.. include:: ../common/authors.txt
```

## Webpart Definition

Deploying a webpart definition is very similar to provisioning a page layout. Web part definitions are also stored in the root web of a site collection, but in the Webpart gallery library. Therefore it is recommended to provision page layout files in a Site Collection scoped feature.

In a Site Collection scope feature, locate the **public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)** method.

Add the following code, replacing **TenantResourcesMapping.Webparts** with the class name of your Webpart in the Resource Mappings class, and **MyWebPart** with the property representing your webpart definition.

```

public override void OnTenantResourceMappings (TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.Webparts>(q => q.
    ↪MyWebPart)
}

```

(continues on next page)

(continued from previous page)

```

        .WithSettingsForWebPart()
        .DeploysTo(SharePointFileDeploymentTargets.WebPartGallery);
    }

```

**Note:** This is a fluent API, see the [Page layout section](#) for more information about how to for example change the file name of the webpart definition file when it is provisioned.

```
.. include:: ../common/authors.txt
```

## Generic File

There could be a need in your project to provision other kinds of files than the ones outlined in the previous sections. This can be done in either a **Site Collection** scoped feature or a **Site** scoped feature.

Locate the **public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)** method.

Add the following code, replacing **TenantResourcesMapping.Files** with the class name of your Files in the Resource Mappings class, and **MyFile** with the property representing your file.

```

public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.Files>(q => q.MyFile)
        .WithSettingsForGenericFile()
        .SetFileName("MyFile.html")
        .DeploysTo(SharePointFileDeploymentTargets.StyleLibrary);
}

```

Notice that in the **SetFileName** you can decide the filename of the uploaded file.

You can also use the **RenameFile("OldFileName.aspx")** to update an already provisioned file, where you have changed the file name in your resource mapping.

```

public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.Files>(q => q.MyFile)
        .WithSettingsForGenericFile()
        .RenameFile("OldFileName.html")
        .DeploysTo(SharePointFileDeploymentTargets.StyleLibrary);
}

```

The **DeploysTo()** method can either be called, like above with a predefined library from the **SharePointFileDeploymentTargets** class, or with a site (web) relative path to a library

```

public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.Files>(q => q.MyFile)
        .WithSettingsForGenericFile()
        .SetFileName("MyFile.html")
        .DeploysTo("/MyLibrary");
}

```

Or even to a specific folder in a library

```
public override void OnTenantResourceMappings(TenantResourcesMapper resourceMapper)
{
    resourceMapper.MapTenantResource<TenantResourcesMapping.Files>(q => q.MyFile)
        .WithSettingsForGenericFile()
        .SetFileName("MyFile.html")
        .DeploysTo("/MyLibrary", "AFolder");
}
```

.. include:: ../common/authors.txt

## Provisioning Pipeline

Omnia provides a way of doing modifications to resources provisioned as part of core Omnia features.

A Provisioning Pipeline makes it possible to modify the content of a resource file at the time it is provisioned to a SharePoint site. This enables you to adapt those resources to specific needs.

### Add a Provisioning Pipeline

To add a new Provisioning Pipeline to your project, add a new class inheriting from **SharePointFileProvisioningPipeline** (found in `Omnia.Foundation.Extensibility.TenantResources`).

Decorate it with a **SharePointFileProvisioningPipelineDefinition** attribute and a unique GUID:

```
using Omnia.Foundation.Extensibility.TenantResources;

[SharePointFileProvisioningPipelineDefinition(id: "2F5450A8-8C6E-4C8F-A907-
↪4F285691BEEB")]
public class MyMasterPageProvisioningPipeline : SharePointFileProvisioningPipeline
{
}
```

## Methods

The **SharePointFileProvisioningPipeline** class has two overridable methods:

```
public override string BeforeContentReplacements(string content, TenantResource_
↪tenantResource)
```

```
public override string AfterContentReplacements(string content, TenantResource_
↪tenantResource)
```

The **content** parameter of the methods contains the full file content in a string format. This allows you to do string operations like **Contains** and **Replace**. Omnia also provides a number of extensions to the String object in C#, like **InsertBefore** and **InsertAfter**.

The **tenantResource** parameter contains information about the resource that is currently being processed. You can do comparisons with properties in this object to make sure your modifications only affects a specific Omnia resource:

```
public override string BeforeContentReplacements(string content, TenantResource_
↪tenantResource)
{
```

(continues on next page)



(continued from previous page)

```

var portalMasterPageId = new Guid(BuiltInResources.MasterPages.Omnia);

if (tenantResource.Id == portalMasterPageId)
{
    // Do code modifications to content here
}
}

```

The main difference between the **BeforeContentReplacements** and **AfterContentReplacements** is that in the **BeforeContentReplacements** the internal tokens Omnia uses to insert functionality to the files are still present in the content string.

The content in the **AfterContentReplacements** resembles exactly the content of the file that is provisioned to SharePoint.

### Example of a BeforeContentReplacements method

```

public override string BeforeContentReplacements(string content, TenantResource_
↵tenantResource)
{
    var portalMasterPageId = new Guid(BuiltInResources.MasterPages.Omnia);

    if (tenantResource.Id == portalMasterPageId)
    {
        try
        {
            content = content.InsertBefore(ProvisioningPipelineTokens.
↵BodyContainerTop, "<div class=\"myClass\">Hello World</div>");
        }
        catch (Exception ex)
        {
            return content.InsertBefore(ProvisioningPipelineTokens.GlobalNavLeft, ex.
↵Message);
        }
    }

    return content;
}

```

**Note:** The **ProvisioningPipelineTokens** class contains the internal tokens still present in the resource files in the **BeforeContentReplacements** method

### Example of an AfterContentReplacements method

```

public override string AfterContentReplacements(string content, TenantResource_
↵tenantResource)
{
    var portalMasterPageId = new Guid(BuiltInResources.MasterPages.Omnia);

    if (tenantResource.Id == portalMasterPageId)
    {

```

(continues on next page)

(continued from previous page)

```

        try
        {
            content = content.InsertBefore("<div class=\"myClass\">", "<div>This is_
↪added in front of the Hello World tag added in the BeforeContentReplacements method
↪</div>");
        }
        catch (Exception ex)
        {
            content.InsertBefore("</head>", "<div>" + ex.Message + "</div>");
        }
    }

    return content;
}

```

```
.. include:: ../common/authors.txt
```

## 1.4 Omnia API

Omnia contains various API's designed to simplify various tasks.

The API's are located in the **OmniaApi** class in the **Omnia.Foundation.Extensibility.Core** namespace.

The basic C# code needed to begin working with the API's from an *Omnia Feature* or a *Custom Web API for Omnia extensions* looks like below.

```
OmniaApi.WorkWith(Ctx.Omnia());
```

By adding a **.** after **WorkWith(Ctx.Omnia())** you will reach the available API services listed below. E.g.

```
OmniaApi.WorkWith(Ctx.Omnia()).Caching();
```

### 1.4.1 Services

```
.. include:: ../common/authors.txt
```

#### Caching

The Caching API makes it easy to work with object caching, either locally on one server or distributed to all servers in an Omnia server cluster (similar to Distributed Cache used in SharePoint).

You reach the Caching API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Caching();
```

**The API contains the following methods:**

- *AddOrUpdateMemoryCache*
- *GetFromMemoryCache*
- *GetFromMemoryCache<T>*

- *MemoryCacheContains*
- *RemoveFromMemoryCache*
- *AddOrUpdateDistributedCache*
- *GetFromDistributedCache*
- *GetFromDistributedCache<T>*
- *RemoveFromDistributedCache*

### AddOrUpdateMemoryCache

Adds or updates an already existing object in the memory cache

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().AddOrUpdateMemoryCache(string key,
↪object value);
```

Optionally you can supply an expiration time as a **DateTimeOffset**

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().AddOrUpdateMemoryCache(string key,
↪object value, DateTimeOffset expires);
```

### GetFromMemoryCache

Gets an object from the memory cached by supplying the key the object was stored with

```
object OmniaApi.WorkWith(Ctx.Omnia()).Caching().GetFromMemoryCache(string key);
```

### GetFromMemoryCache<T>

Gets an object from the memory cached by supplying the key the object was stored with. Casts the object to the supplied type

```
T OmniaApi.WorkWith(Ctx.Omnia()).Caching().GetFromMemoryCache<T>(string key);
```

### MemoryCacheContains

Checks if an object with the given key is present in the memory cache

```
bool OmniaApi.WorkWith(Ctx.Omnia()).Caching().MemoryCacheContains(string key);
```

### RemoveFromMemoryCache

Deletes the object with the given key from the memory cache

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().RemoveFromMemoryCache(string key)
```

### AddOrUpdateDistributedCache

Adds or updates an already existing object in the distributed cache

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().AddOrUpdateDistributedCache(string key,
↪object value, DateTimeOffset expires);
```

Optionally you can cache the data encrypted

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().AddOrUpdateMemoryCache(string key,
↪object value, bool encrypted, DateTimeOffset expires);
```

You can also cache multiple objects at once by creating a **List<CachedItem>** (**CachedItem** is found in the **Omnia.Foundation.Extensibility.Core.Caching** namespace)

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().AddOrUpdateDistributedCache(List
↪<CachedItem> objectsToCache);
```

### GetFromDistributedCache

Gets objects from the distributed cache by supplying a list of the keys the items are stored with

```
List<CachedItem> OmniaApi.WorkWith(Ctx.Omnia()).Caching().GetFromDistributedCache(List
↪<string> keys);
```

### GetFromDistributedCache<T>

Gets an object from the distributed cache, cast to the specified type

```
T OmniaApi.WorkWith(Ctx.Omnia()).Caching().GetFromDistributedCache<T>(string key);
```

### RemoveFromDistributedCache

Deletes objects from the distributed cache by supplying a list of keys

```
void OmniaApi.WorkWith(Ctx.Omnia()).Caching().RemoveFromDistributedCache(List<string>
↪keys);
```

```
.. include:: ../common/authors.txt
```

### Configurations

The Configurations API allows you to read, store, update and delete configurations for your solution.

Use Configurations as a means for allowing certain values in your solution to be configurable, via code or from the Omnia Admin app, to make it more adaptable.

You reach the Configurations API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Configurations();
```

The API contains the following methods:

- *AddOrUpdateConfiguration*
- *AddOrUpdateConfigurations*
- *DeleteConfiguration*
- *GetConfiguration*
- *GetConfigurations*
- *GetConfigurationsInRegion*
- *GetOmniaInstanceMode*
- *GetParentSiteConfigurations*

## AddOrUpdateConfiguration

Adds a new configuration, or updates an existing one.

```
void OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .AddOrUpdateConfiguration(Configuration configuration);
```

You find the **Configuration** class in the **Omnia.Foundation.Models.Configurations** namespace

You can also add or update a configuration with the following overload

```
void OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .AddOrUpdateConfiguration(string key, dynamic value, [string region = ""],
    ↪ [bool includedInClient = false], [bool uiEditable = false], [string permissionRoles
    ↪ = ""]);
```

## AddOrUpdateConfigurations

To add / update multiple configurations at once, use the following method

```
void OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .AddOrUpdateConfigurations(IEnumerable<Configuration> configurations);
```

You find the **Configuration** class in the **Omnia.Foundation.Models.Configurations** namespace

## DeleteConfiguration

Deletes an existing configuration by passing the name and region of it.

```
void OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .DeleteConfiguration(string name, string region);
```

Optionally you can also pass in an extension id as the last parameter, to target configurations for a specific extension

```
void OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .DeleteConfiguration(string name, string region, [Guid? extensionPackageId =
    ↪ null]);
```

(continues on next page)

## GetConfiguration

Gets a specific configuration by name and region.

```
Configuration OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfiguration(string name, string region);
```

Optionally you can also pass in an extension id as the last parameter, to target configurations for a specific extension

```
Configuration OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfiguration(string name, string region, Guid?   
↳extensionPackageId = null]);
```

## GetConfigurations

To get all existing configurations, use the following method

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfigurations();
```

You can also scope this to only get configurations for a specific extension by supplying the Id of the solution

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfigurations([Guid? extensionPackageId = null]);
```

To specify which configurations to get, use the following method

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfigurations(List<string> names, string region);
```

Where you pass in the names and region of the configurations to retrieve.

## GetConfigurationsInRegion

To get all configurations in a given region, use the following method

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfigurationsInRegion(string region);
```

You can also scope this to only get configurations for a specific extension by supplying the Id of the solution

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()  
    .GetConfigurationsInRegion(string region, [Guid? extensionPackageId = null]);
```

## GetOmniaInstanceMode

In some scenarios you might need know if Omnia is running in **Site collection** or **Tenant** mode. To get this information, call the following method

```
OmniaInstanceModes OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .GetOmniaInstanceMode();
```

This will return a value from the enum **Omnia.Foundation.Models.Shared.OmniaInstanceModes**, either **SiteCollection** or **Tenant**

## GetParentSiteConfigurations

Gets configurations from a specified parent site, based on the site URL and region of the configuration.

```
IEnumerable<Configuration> OmniaApi.WorkWith(Ctx.Omnia()).Configurations()
    .GetParentSiteConfigurations(string fromSiteUrl, string region);
```

.. include:: ../common/authors.txt

## Controls

The Controls API contains methods that makes it easier to work with Omnia Controls.

You reach the Controls API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Controls();
```

### The API contains the following methods:

- *GetControlSettings*

## GetControlSettings

Use this method to get the stored settings for a given instance of an Omnia Control

```
string OmniaApi.WorkWith(Ctx.Omnia()).Controls()
    .GetControlSettings(string scope, Guid controlId, string siteCollectionUrl,
    ↪ string siteUrl, int? pageItemId, Guid? featureResourceId);
```

Pass in the scope of the control (“masterpage”, “site”, “page” or “webpart”), the Guid of the control, the url of the site collection and site.

To target a control on a certain page pass the a item ID of the page.

You can also pass in the feature resource Id of your control.

.. include:: ../common/authors.txt

## Email

The Email API allows you to send email from your solution.

You reach the Caching API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Email(Ctx);
```

The API contains the following methods:

- *SendEmail*

### SendEmail

Sends an email message

```
void OmniaApi.WorkWith(Ctx.Omnia()).Email(Ctx).SendEmail(string subject, string body, List<string> emailTo);
```

You can add multiple email addresses in the **emailTo** list

There is also two optional input parameters: **emailCC** and **emailBcc**, both of type **List<string>**.

- Add email addresses to **emailCC** to add them to the the carbone copies list of the email .
- Add email addresses to **emailBCC** to add them to the the blind carbone copies list of the email.

.. include:: ../common/authors.txt

### Features

The Features API allows you to Activate, Deactivate and in other ways work with *Omnia Features*.

---

**Note:** This API works with Omnia Features, not SharePoint Features.

---

You reach the Features API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Features();
```

The API contains the following methods:

- *ActivateFeature*
- *DeactivateFeature*
- *GetFeature*
- *GetFeatures*
- *GetFeatureActivationStatus*
- *GetFeatureInstanceLogs*
- *GetFeatureInstances*
- *UpgradeFeature*

### ActivateFeature

To activate an Omnia Feature, use the following method



```
FeatureInstance OmniaApi.WorkWith(Ctx.Omnia()).Features()
    .ActivateFeature(Guid id, string spUrl, bool force);
```

Supply the unique Guid of the feature, the URL of the tenant authority / site collection / web where the feature should be activated. Set the **force** parameter to **true** to ignore errors on activation.

## DeactivateFeature

To deactivate an Omnia Feature, use the following method

```
FeatureInstance OmniaApi.WorkWith(Ctx.Omnia()).Features()
    .DeactivateFeature(Guid id, string spUrl);
```

Supply the unique Guid of the feature and the URL of the tenant authority / site collection / web where the feature should be deactivated.

## GetFeature

To get more information about a specific feature, use the following method

```
FeatureModel OmniaApi.WorkWith(Ctx.Omnia()).Features()
    .GetFeature(Guid id);
```

Supply the unique Guid of the feature. In return you get a **Omnia.Foundation.Models.Features.FeatureInstance** object containing feature details like Name, Description and Scope

## GetFeatures

To get more information about a all existing features, use the following method

```
IEnumerable<FeatureModel> OmniaApi.WorkWith(Ctx.Omnia()).Features()
    .GetFeatures();
```

## GetFeatureActivationStatus

To check if a given feature is activated or not on a site / site collection / tenant, use the following method

```
FeatureInstanceStatus OmniaApi.WorkWith(Ctx.Omnia()).Features()
    .GetFeatureActivationStatus(Guid id, string spUrl);
```

This will return a value from the **Omnia.Foundation.Models.Features.FeatureInstanceStatus** enumeration.

This can have any of the following values:

- NotActivated
- Activating
- Activated
- Upgrading
- Deactivating

- Error

### GetFeatureInstanceLogs

To get the log messages written for a specific feature, use the following method

```
IEnumerable<FeatureInstanceLog> OmniaApi.WorkWith(Ctx.Omnia()).Features()  
    .GetFeatureInstanceLogs(Guid id, DateTimeOffset? startingBefore =       
↪ default(DateTimeOffset?), int take = -1);
```

### GetFeatureInstances

To get all instances of a feature (e.g. all places where a feature is activated), use the following method

```
IEnumerable<FeatureInstance> OmniaApi.WorkWith(Ctx.Omnia()).Features()  
    .GetFeatureInstances(Guid id);
```

This will return a collection of **Omnia.Foundation.Models.Features.FeatureInstance**, containing for example Status and Target of the feature

### UpgradeFeature

To upgrade a feature on one target (Tenant / Site collection / Site), use the following method

```
FeatureInstance OmniaApi.WorkWith(Ctx.Omnia()).Features()  
    .UpgradeFeature(Guid id, string spUrl);
```

To upgrade features in multiple places at once, you can use the following method

```
Dictionary<string, ApiOperationResult> OmniaApi.WorkWith(Ctx.Omnia()).Features()  
    .UpgradeFeature(Guid id, List<string> spUrls);
```

This will update the feature in all of the instances specified by the list of URLs supplied in the **spUrls** parameter.

.. include:: ../common/authors.txt

### Lists

The Lists API contains methods that makes it easier to work with lists and libraries in SharePoint.

You reach the Lists API through the following service

```
OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx);
```

**The API contains the following methods:**

- *AddFileToList*
- *GetAllDocumentLibraries*
- *GetAllImageLibraries*

- *GetDocuments*
- *GetDocumentsByFolder*
- *GetListItems*
- *GetPageList*
- *GetPageListId*
- *GetPageListUrl*

## AddFileToList

Use this method to create a file in a library based on a supplied byte array containing the file data.

```
void OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .AddFileToList(Guid listId, string folderServerRelativeUrl, string fileName,
↳byte[] data);
```

Pass in the GUID of the library, a server relative Url to a folder (or empty string to add in root folder) and a file name. Optionally, you can also supply a boolean indicating if the file should be published or not

```
void OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .AddFileToList(Guid listId, string folderServerRelativeUrl, string fileName,
↳byte[] data, bool publish);
```

## GetAllDocumentLibraries

To get information about all document libraries on the current web, use the following method

```
IEnumerable<ListIdentifier> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetAllDocumentLibraries();
```

The returned array contains **ListIdentifier** objects containing for example *Title*, *Id* and *ListUrl* for the document libraries.

## GetAllImageLibraries

To get information about all image libraries on the current web, use the following method

```
IEnumerable<ListIdentifier> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetAllImageLibraries();
```

The returned array contains **ListIdentifier** objects containing for example *Title*, *Id* and *ListUrl* for the image libraries.

## GetDocuments

To get documents from a library, you can use the following method

```
IEnumerable<DocumentIdentifier> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetDocuments(Guid listId, ListQuery listQuery, bool recursive = true, string
↳folderServerRelativeUrl = "");
```

(continues on next page)

---

Supply a **ListQuery** object to filter the results.

To get a paged subset of the documents, you can instead use the following method

```
IEnumerable<DocumentIdentifier> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    GetDocuments(Guid listId, string searchString = null, int skipId = 0, int
    ↳take = -1, string orderBy = null, bool ascending = true, bool isGetAbsoluteUrl =
    ↳false);
```

Passing in a **skipId** decides from which item to start fetching, and **take** sets the number of documents to return. You can also supply *orderBy* and *ascending* parameters to decide the sort order.

The returned **DocumentIdentifier** class contains basic information about the documents in the library, for example *Id*, *Title*, *FileName*, *DocumentUrl* and more.

### GetDocumentsByFolder

Much like the **GetDocuments** method, you can use this method to get a paged subset of documents, but in a specific folder

```
IEnumerable<DocumentIdentifier> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    GetDocumentsByFolder(Guid listId, string folderUrl, string searchString =
    ↳null, int skipId = 0, int take = -1, string orderBy = null, bool ascending = true);
```

Pass in site relative **folderUrl**

### GetListItems

Use this method to get list items from a list

```
IEnumerable<ListItem> OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetListItems(Guid listId, ListQuery listQuery);
```

Note that this returns the full **ListItem** objects. Use the **listQuery** parameter to filter what items are returned.

### GetPageList

---

**Note:** Publishing webs only

---

To get the **Pages** list of the current web use one of the following methods

```
List OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetPageList(Web web, string webUrl);
```

or (to target a specific list)

```
List OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    .GetPageList(Web web, string webUrl, string listId);
```

## GetPageListId

**Note:** Publishing webs only

To get the **Guid** of the **Pages** library on a publishing web, use the following method

```
Guid OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    GetPageListId(Web web, string webUrl);
```

## GetPageListUrl

**Note:** Publishing webs only

To get the **URL** of the **Pages** library on a publishing web, use the following method

```
string OmniaApi.WorkWith(Ctx.Omnia()).Lists(Ctx)
    GetPageListUrl(Web web, string webUrl);
```

```
.. include:: ../common/authors.txt
```

## Logging

Omnia provides a logging API for extensions to write logs to Omnia Foundation's logs database. There are three different types of logs in Omnia: System Logs, Queue Logs and Feature Logs.

System Logs is a general-purpose place for information or error logs from both Omnia Foundation and extensions. System Logs can be viewed in Omnia admin app at **System > Logs**

System > Logs

System Logs

Unable to connect to the remote server

Filter

Source	Message	Type
2016-08-05 04:51:11 Omnia.Foundation.Core.Security.SecurityServiceExtended	Unable to connect to the remote server	Error
<b>Exception</b> System.WebException: Unable to connect to the remote server --> System.Net.Sockets.SocketException: A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond 172.27.80.71:7777 at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress) at System.Net.ServicePoint.ConnectSocketInternal(Boolean connectFailure, Socket s4, Socket s6, Socket& socket, IPAddress& address, ConnectSocketState state, IAsyncResult asyncResult, Exception& exception) --- End of inner exception stack trace --- at System.Net.HttpWebRequest.GetRequestStream(TransportContext& context) at System.Net.HttpWebRequest.GetRequestStream() at Microsoft.SharePoint.Client.SPWebRequestExecutor.GetRequestStream() at Microsoft.SharePoint.Client.ClientRequest.ExecuteQueryToServer(ChunkStringBuilder sb) at Microsoft.SharePoint.Client.ClientRequest.ExecuteQuery() at Microsoft.SharePoint.Client.ClientRuntimeContext.ExecuteQuery() at Microsoft.SharePoint.Client.ClientContext.ExecuteQuery() at Omnia.Foundation.Core.SharePoint.ExtendedClientContext.ExecuteQueryRetry(Int32 retryCount, Int32 delay) in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\SharePoint\ExtendedClientContext.cs:line 84 at Omnia.Foundation.Core.SharePoint.ExtendedClientContext.ExecuteQuery() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\SharePoint\ExtendedClientContext.cs:line 36 at Omnia.Foundation.Core.Security.SecurityServiceExtended.GetUserPermissionRoles() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\Security\SecurityServiceExtended.cs:line 1040 at Omnia.Foundation.Core.Security.SecurityServiceExtended.GetCurrentUserPermissionRolesDictionary() in C:\Omnia\Foundation\Dev\Omnia.Foundation.Core\Security\SecurityServiceExtended.cs:line 421		
2016-07-29 12:02:05 Omnia.Foundation.Core.Security.SecurityServiceExtended	Unable to connect to the remote server	Error

Queue Logs contains logs from queue messages jobs. For errors related to long-running operations like uploading extension packages or creating site collection, check the Queue Logs. Queue Logs can be viewed in Omnia admin app at **System > Queues**

System > Queues

Queue Messages

Filter by Transaction Id...

Filter

Start Time	End Time	Name	Transaction	Status	
2016-08-04 13:25:56	2016-08-04 13:26:00	processfeature	d7a463c3-a577-458d-8f17-f098918968e0	Success	<a href="#">View Details</a>
2016-08-04 13:25:16	2016-08-04 13:25:28	processfeature	c2f41878-ec95-40d4-93d8-6501d5b0b7d3	Success	<a href="#">View Details</a>
2016-08-04 12:15:31	2016-08-04 12:15:59	processfeature	62f3e62a-7723-4047-8224-d83fcbcbffdf	Success	<a href="#">View Details</a>
2016-08-03 11:25:59	2016-08-03 11:26:04	newextensionpackage	85366cf5-b32e-4d93-bb56-6fdb419c6fd	Success	<a href="#">View Details</a>

The last type of logs is Feature Logs, which contains logs from custom code in feature activation, upgrade or deactivation. Feature Logs can be viewed in the feature detail page.

Features > Omnia Core MasterPage

Omnia Core MasterPage: 1.0.6

Current Site Collection

Url	Version	
/	1.0.6	<a href="#">Hide log</a> <a href="#">Remove</a> <a href="#">Upgrade</a>

Logs - Refresh

2016-06-30 06:12:45 Upgrade feature from version 1.0.6  
2016-06-29 11:01:33 Upgrade feature from version 1.0.5

## Logging in extension Web API

In extension API that inherit from **SharePointContextProvidedController** you can use the built-in **Logging** service to write to the System Logs.

```
[HttpGet]
[Route("api/documents")]
public ApiOperationResult<IEnumerable<Document>> GetDocuments()
{
    try
    {
        // Web API code here
    }
    catch (Exception ex)
    {
        // The built-in Logging service can be used to write to System Logs
        this
            .Logging
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error, ex);

        return ApiUtils.CreateErrorResult<IEnumerable<Document>>(ex);
    }
}
```

For API that does not inherit from **SharePointContextProvidedController**, you can use **OmniaApi** factory to create the

**Logging** service.

```
[HttpGet]
[Route("api/documents")]
public ApiOperationResult<IEnumerable<Document>> GetDocuments (
    string tokenKey, string spUrl, string language)
{
    try
    {
        // Web API code here
    }
    catch (Exception ex)
    {
        // For controller that does not inherit from
        ↳ SharePointContextProvidedController
        // we need to create the ClientContext first.
        ClientContext ctx = SharePointContextProvider
            .CreateUserClientContext(tokenKey, spUrl, language);

        OmniaApi
            .WorkWith(Ctx.Omnia())
            .Logging()
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error, ex);

        return ApiUtils.CreateErrorResult<IEnumerable<Document>>(ex);
    }
}
```

## Logging in extension features

In Omnia feature you can write to System Logs using **OmniaApi** factory or write to Feature Logs using the built-in **Log** method

```
[FeatureDefinition(
    id: "85544C6C-9EB9-4F99-9410-95F1EA3D07B5",
    name: "MyOmniaExtension Sample Feature Core",
    version: "0.1.0",
    scope: FeatureScopes.Tenant
)]
public class SampleFeatureCore : Omnia.Foundation.Extensibility.Features.OmniaFeature
{
    /// <summary>
    /// Activates the OmniaFeature
    /// </summary>
    public override void Activate()
    {
        // This will write to System Logs
        this.WorkWith().Logging()
            .AddLog("SampleFeatureCore", "Feature activation", DefaultLogTypes.Info);

        // This will write to Feature Logs
        this.Log("Feature activation", "Success", FeatureInstanceLogTypes.
        ↳ Information);
    }
}
```

### Logging in extension jobs

Similar to features, in jobs you can use the **OmniaApi** factory to write to System Logs. One important thing to note is that currently any uncached error in queue job will be write to the Queue Logs, but for timer jobs you need to handle the error and explicitly write to the System Logs in your code.

```
public void SampleJobTimer([TimerTrigger("01:00:00")] TimerInfo timerInfo)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("SampleJobTimer", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}

public void SampleJobQueue([QueueTrigger("SampleJob")] object queueMessage)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("SampleJobQueue", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}
```

.. include:: ../common/authors.txt

### Security

---

**Note:** This is a draft version of the documentation, it is not yet complete. Feel free to contribute to it via GitHub.

---

The Security API allows you to work with permissions, both Omnia permissions and SharePoint permissions.

You can for example check if the current user has a certain permission on a web, or a certain role in Omnia.

You reach the Security API through the following service

```
OmniaApi.WorkWith(Ctx.Omia()).Security(Ctx);
```

#### The API contains the following methods:

- *AddOrUpdateOmniaPermissionRoles*
- *DeleteOmniaPermissionRolesForExtensionPackage*
- *DoesUserHavePermissionOnWeb*
- *GetAllOmniaPermissionRoles*



- *GetCurrentUserADGroups*
- *GetPermissionRoles*
- *IsUserAuthorized*
- *SearchOmniaPermissionRoles*

## AddOrUpdateOmniaPermissionRoles

*Documentation in progress*

## DeleteOmniaPermissionRolesForExtensionPackage

*Documentation in progress*

## DoesUserHavePermissionOnWeb

Use this method to check if the user has the specified permission on the current web.

```
bool OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .DoesUserHavePermissionOnWeb(PermissionKind permissionKind);
```

Returns **true** if the user has the supplied **permissionKind**, otherwise **false**.

## GetAllOmniaPermissionRoles

Use this method to get all available Omnia Permission roles for a given URL.

```
IEnumerable<PermissionRole> OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .GetAllOmniaPermissionRoles(string targetUrl);
```

Returns an array of **PermissionRole** for the given **targetUrl**.

## GetCurrentUserADGroups

If you need to get all AD groups the current user is a member of, use the following end-point.

**Note:** Only AD groups used somewhere in Omnia Foundation will be returned, e.x. groups used for security in Omnia Administration or as part of targeting definitions.

```
IEnumerable<string> OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .GetCurrentUserADGroups(string[] groups);
```

If you supply and empty **groups** parameter, you will get all AD groups, used somewhere in Omnia, that the user is a member of.

```
var allGroups = OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .GetCurrentUserADGroups(new string[0]());
```

To only check specific AD groups, pass their names in the **groups** parameter

```
var someGroups = OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .GetCurrentUserADGroups(new string[] { "ADGroup1", "ADGroup2" });
```

### GetPermissionRoles

To get Permissions Roles for given Permission Role Definitions you can use the following method, passing in a **List<PermissionRoleDefinition>** containing the role definitions of interest.

```
IEnumerable<PermissionRole> OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .GetPermissionRoles(List<PermissionRoleDefinition> requestedRoles);
```

This will return an array containing all permissions roles existing for the supplied definitions.

### IsUserAuthorized

Use this method to check if the user is authorized, e.g. is a member of the required Omnia **PermissionRoleDefinition**

```
bool OmniaApi.WorkWith(Ctx.Omnia()).Security(Ctx)
    .IsUserAuthorized(PermissionRoleDefinition requiredRole);
```

Returns **true** if the user is a member of the required Omnia role, else **false**.

### SearchOmniaPermissionRoles

*Documentation in progress*

.. include:: ../common/authors.txt

## 1.5 Custom Web API for Omnia extensions

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### 1.5.1 Topics

.. include:: ../common/authors.txt

#### Omnia base controller

When building Web API for Omnia extension it's recommended to use the base class **SharePointContextProvided-Controller** from Omnia. This base class provides many useful services when working with SharePoint and Omnia.

```

using Omnia.Foundation.Extensibility.Core;
using Omnia.Foundation.Extensibility.Core.Utilities;
using Omnia.Foundation.Extensibility.WebApi;
using Omnia.Foundation.Models.Shared;
using Omnia.Foundation.Models.Logging;
using System;
using System.Web.Http;
using System.Collections.Generic;
using Microsoft.SharePoint.Client;

public class MyController: SharePointContextProvidedController
{
    [HttpGet, Route("api/items")]
    public ApiOperationResult<IEnumerable<Item>> GetItems()
    {
        try
        {
            // Your controller logic here
        }
        catch (Exception ex)
        {
            this
                .Logging
                .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error,
↳ex);

            return ApiUtils.CreateErrorResult<IEnumerable<Item>>(ex);
        }
    }
}

```

## SharePoint ClientContext

The base class **SharePointContextProvidedController** will enforce the parameters **SPUrl** and **TokenKey** on every endpoints in the controller, either in querystring or in the request headers. These paramters will be used to authenticate who is calling the API and create a SharePoint user ClientContext. This client context can be accessed using the base controller's property **Ctx**. This way the Web API developers will not need to think about authentication and how to communicate with Omnia Foundation services.

The base controller class also provides a helper method, named **CreateContextFor**, to create ClientContext for another SharePoint site than the site specified by SPUrl paramter, or to create ClientContext with elevated permission.

```

using Omnia.Foundation.Extensibility.Core;
using Omnia.Foundation.Extensibility.Core.Utilities;
using Omnia.Foundation.Extensibility.WebApi;
using Omnia.Foundation.Models.Shared;
using Omnia.Foundation.Models.Logging;
using System;
using System.Web.Http;
using System.Collections.Generic;
using Microsoft.SharePoint.Client;

public class MyController: SharePointContextProvidedController
{
    [HttpGet, Route("api/items")]

```

(continues on next page)

(continued from previous page)

```

public ApiOperationResult<IEnumerable<Item>> GetItems()
{
    try
    {
        // Use the current ClientContext
        this.Ctx.Load(this.Ctx.Web, w => w.Url);
        this.Ctx.ExecuteQuery();

        // Create ClientContext with elevated permission
        using (ClientContext appContext = CreateContextFor(this.Ctx.Web.Url,
↪elevated: true))
        {

        }

    }
    catch (Exception ex)
    {
        this
            .Logging
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error,
↪ex);

        return ApiUtils.CreateErrorResult<IEnumerable<Item>>(ex);
    }
}

```

To Create ClientContext with elevated permission you need to have a valid ClientId and ClientSecret in web.config (for Office 365) or have a high-trust certificate configured (for on-premise).

```

<appSettings>
  <add key="ClientSecret" value="DljsSY3lwlsZFytMRhjr4xE11NynROTf0K1p/9XDWnM=" />
  <add key="ClientId" value="62669ea8-15f0-4b24-83e4-1e99fce5ecd5" />
  ...
</appSettings>

```

## Omnia Foundation Services

SharePointContextProvidedController has a built-in Logging service which write to Omnia Foundation logs database. Other Omnia Foundation services can be accessed using the factory method **WorkWith()**

```

using Omnia.Foundation.Extensibility.Core;
using Omnia.Foundation.Extensibility.Core.Utilities;
using Omnia.Foundation.Extensibility.WebApi;
using Omnia.Foundation.Extensibility.Core.Configurations;
using Omnia.Foundation.Models.Shared;
using Omnia.Foundation.Models.Logging;
using System;
using System.Web.Http;
using System.Collections.Generic;
using Microsoft.SharePoint.Client;

public class MyController: SharePointContextProvidedController
{
    [HttpGet, Route("api/items")]

```

(continues on next page)

(continued from previous page)

```

public ApiOperationResult<IEnumerable<Item>> GetItems()
{
    try
    {
        // Use Omnia configuration service
        var configuration = WorkWith().Configurations().GetConfiguration(
            name: "configuration-name",
            region: "configuration-region");
    }
    catch (Exception ex)
    {
        // Built-in logging service
        this
            .Logging
            .AddLog(this.GetType().ToString(), ex.Message, DefaultLogTypes.Error, ex);
    }

    return ApiUtils.CreateErrorResult<IEnumerable<Item>>(ex);
}
}

```

Other contextual information are also provided:

- **TenantId**: ID of the Omnia tenant that the current SharePoint site belongs to.
- **LoginName**: SharePoint loginname of the current user.
- **OmniaInstanceMode**: The mode that Omnia Foundation is running in, either Tenant or SiteCollectionOnly

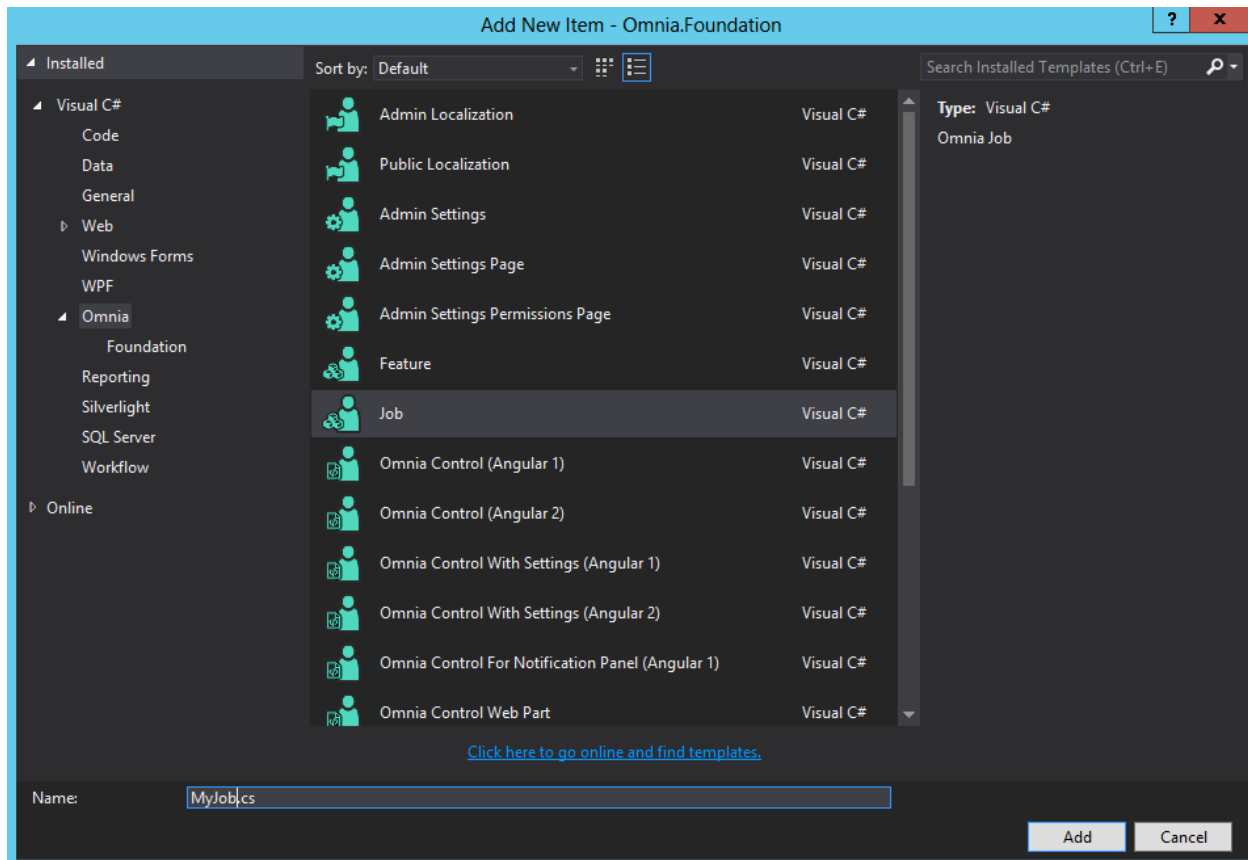
.. include:: ../common/authors.txt

## 1.6 Omnia Jobs

Omnia Jobs are pieces of code for handling long-running operations that can be run either as a message queue or as a scheduled timer job.

### 1.6.1 Create new Omnia Job

You can create Omnia Job using the template from Omnia Tooling



```
[JobDefinition(
    id: "368D7722-9F75-4789-A1BA-460DBB6595F8",
    name: "MyJob",
    description: ""
)]
public class MyJob : OmniaJob
{
    // Scheduled Job job function
    public void MyJobTimer([TimerTrigger("01:00:00")] TimerInfo timerInfo)
    {
        try
        {
            // Your job code here
        }
        catch (Exception ex)
        {
            WorkWith().Logging().AddLog("MyJobTimer", ex.Message, DefaultLogTypes.
↪Error, ex);
        }
    }

    // Message Queue job function
    public void MyJobQueue([QueueTrigger("MyJob")] object queueMessage)
    {
        try
        {
            // Your job code here
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("MyJobQueue", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}

```

The job metadata is defined by the attribute **JobDefinition**. Each Omnia Job can contains any number of **independent** job functions, each can be of one of the two types Scheduled Job or Message Queue.

## 1.6.2 Built-in methods and properties

Similar to Omnia Feature and Omnia Web API, Omnia common services are available through the built-in **Work-With** method. The code in Omnia Jobs is run at tenant scope so there is no user context, you can get an app-only ClientContext using the method **CreateContextFor**

Properties	Type	Description
Tenant	Tenant	The current tenant.
OmniaInstanceMode	Enum	The mode of the current tenant. could be Tenant or Sitecollection

Methods	Type	Description
CreateContextFor(string spUrl)	ClientContext	Create an app-only context
WorkWith()	ApiFactory	Return the ApiFactory that can call Omnia API Example: Work- With().Logging().AddLog(log)

## 1.6.3 Manage jobs in a tenant

Unlike tenant resources, jobs are available and running immediately after the extension is deployed, there is no need to activate any feature.

All jobs deployed by extensions can be viewed in admin app at **Systems > Jobs**. You can also change the interval of scheduled jobs or stop/force run them from this interface. Note that built-in jobs will not be displayed here.

System > Jobs

Name	Description	Extension	Trigger	Started At	Finished At	
CalendarSyncJob		3e244375-6a6f-458c-bfe8-2993d4fd6f88				
QMSSearchSyncJob	QMS Search sync Job	0d54230e-d0ec-4b57-835f-92325218fc52				
SearchSynchronize	Timer	0d54230e-d0ec-4b57-835f-92325218fc52	00:30:00	2016-11-21 11:18:28	2016-11-21 11:18:37	Edit Stop Run now
PowerPack Email Job		3e244375-6a6f-458c-bfe8-2993d4fd6f88				
AssetRequestReminderJob		3e244375-6a6f-458c-bfe8-2993d4fd6f88				
IntranetNavigationSyncJob	Sync the navigations from configuration to SharePoint	8c03468f-ef20-49ef-9349-88418305433c				
EmployeeSyncJob		3e244375-6a6f-458c-bfe8-2993d4fd6f88				
QMSRunEmailSenderJob	QMS Read and Understood Email Sender Job	0d54230e-d0ec-4b57-835f-92325218fc52				
ODMReviewWorkflow	Review Workflow Job	aaf868ac-469f-47f9-a5e1-6ec181a6de38				
ODMReviewReminder	Review Reminder Job	aaf868ac-469f-47f9-a5e1-6ec181a6de38				
ExtranetSLALevelCheckUp		e982ff0d-7cb6-4297-b858-4ce17a0bca96				
ExtranetNotificationEmailServiceJob		e982ff0d-7cb6-4297-b858-4ce17a0bca96				

## 1.6.4 Job Types

```
.. include:: ../../common/authors.txt
```

### Scheduled Job

```
// Scheduled Job job function
public void MyJobTimer([TimerTrigger("01:00:00")] TimerInfo timerInfo)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("MyJobTimer", ex.Message, DefaultLogTypes.Error,
↪ex);
    }
}
```

Scheduled jobs will run at the interval defined in the **TimerTrigger**. There are 2 different constructors you can use:

**Note:** The minimum interval supported by Omnia Jobs is 10 seconds

**Note:** **TimerInfo** is a place-holder for future features, currently it contains no information

```
public void MyJobTimer([TimerTrigger("01:00:00")] TimerInfo timerInfo)
```

```
public void MyJobTimer([TimerTrigger(1, 0, 0)] TimerInfo timerInfo)
```

```
.. include:: ../../common/authors.txt
```



## Message Queues

```
public void MyJobQueue([QueueTrigger("MyQueue")] object queueMessage)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("MyJobQueue", ex.Message, DefaultLogTypes.Error,
↪ex);
    }
}
```

Message queue jobs will be triggered everytime a message is added to the queue that the jobs listen to ("MyQueue" in this example).

Queue messages can be added when a feature is activated/deactivated, when an web api endpoint is called or from another Omnia Job. To add a queue message, use the Queues service:

```
public void MyJobTimer([TimerTrigger(1, 0, 0)] TimerInfo timerInfo)
{
    try
    {
        // Add a queue message to the queue "MyQueue" every hour, triggering the
↪queue message job.
        WorkWith().Queues().AddQueueMessage("MyQueue", new MyModel());
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("MyJobTimer", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}

// The content of the queue message will de deserialized to type MyModel
public void MyJobQueue([QueueTrigger("MyQueue")] MyModel queueMessage)
{
    try
    {
        // Your job code here
    }
    catch (Exception ex)
    {
        WorkWith().Logging().AddLog("MyJobQueue", ex.Message, DefaultLogTypes.
↪Error, ex);
    }
}
```

## Dequeue Mode

When adding a queue message you can set the **transaction ID** for the message, so that multiple messages can be grouped together in the same transaction. By default, the message queue job will be triggered immediately when a new message is added, however you can change this behavior by setting the dequeue mode to **SynchronousTransaction**.

When running synchronous dequeue mode, all messages in the same transaction will be processed sequentially by

the order that they was added. In other words, if a new message of the same transaction is added while the previous message was being processed, the new message will not be processed until the previous message has been finished.

```
public void MyJobQueue([QueueTrigger("MyJob", DequeueMode = Models.Queues.  
↔DequeueModes.SynchronousTransaction)] MyModel queueMessage)
```

```
.. include:: ../common/authors.txt
```

## 1.7 Client-Side Development

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### 1.7.1 Topics

```
.. include:: ../common/authors.txt
```

#### Live reload

When doing client side development, you can enable a mode called “Live reload”. In this mode, the page is automatically updated to reflect your new HTML, TypeScript and Less code when you save a file in Visual Studio.

##### Sections:

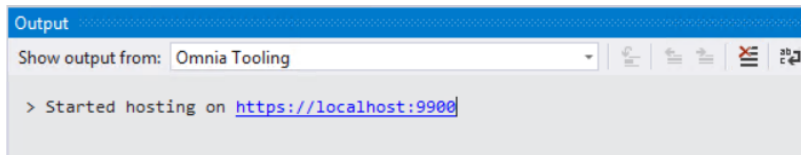
- *Enable Live reload*
- *Working in Live reload mode*
- *Disable Live reload*

#### Enable Live reload

1. Open Visual Studio as Administrator
2. In the **environment.json** file of your Omnia extension, do the following
  - In the *Hosting* section set *Enabled* to **true**, and *UseLiveReload* to **true**
  - Note the **Port** number (ex 9900)
  - Save the file

```
{
  "TenantId": "00000000-0000-0000-0000-000000000000",
  "ApiSecret": "00000000-0000-0000-0000-000000000000",
  "FoundationUrl": "https://localhost:9900",
  "Hosting": {
    "Enabled": true,
    "Port": 9900,
    "UseLiveReload": true
  },
  "Angular": {
    "AOT": {
      "RunOnBuild": true
    }
  },
  "FeatureActivations": []
}
```

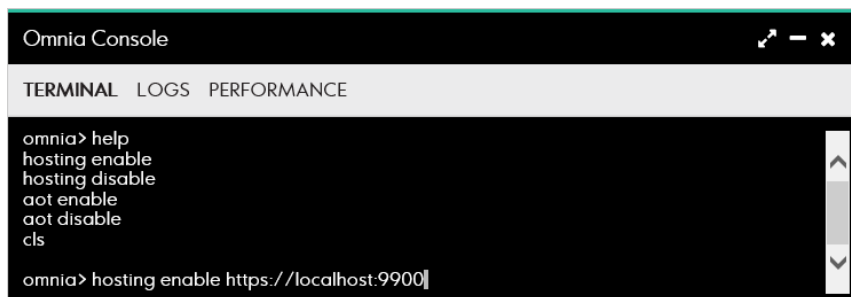
3. In the **Output** window of Visual Studio, select “Omnia Tooling” in the *Show output from* dropdown
4. Make sure the Hosting server has started correctly by reading the output.



**Note:** If you get an error message in the output, you need to make sure that the IIS Express SSL certificate is installed on your computer.

You can trigger the installation by creating an MVC project, Enable SSL and then start Debugging the project. You will now get a dialog asking if you want to trust the IIS Express SSL certificate. Click “Yes”

3. In your web browser
  - Navigate to a site in the Office 365 Tenant used for developing your Omnia Extension
  - Add **?console=on** to the URL of the page, ex `/somepage?console=on`. If you are on a quick page, instead add **!console=on**, ex `##/somepage!console=on`
  - This will display the Omnia Console on the current page. Type **help** in the console to get a list of all available commands
  - To enable Live reload, type **hosting enable https://localhost:9900** where 9900 is the port number you noted in Step 2, from the environment.json



- The page should now be reloaded and after some time your controls should be displayed, now served from **https://localhost:9900** instead of the Omnia server

### Working in Live reload mode

Edit your LESS, TypeScript or HTML files as you normally would. When saved, the browser will update to instantly display the changes.

### Disable Live reload

To disable Live reload, do the following:

1. In the Omnia console in the browser window, write **hosting disable**
2. In the Visual Studio project, in the environment.json file, set the *Enabled* parameter in the *Hosting* section to **false**

```
.. include:: ../common/authors.txt
```

### Angular 1

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

```
.. include:: ../common/authors.txt
```

### UI Components

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

```
.. include:: ../common/authors.txt
```

### Single Picker

---

**Note:** This component is an Angular 1 directive registered in module **Omnia.Foundation.Core.Module**

---

The single picker is a dropdown component that supports type-ahead search and customizable UI.

## Department Manager

## Sample

```
$scope.employees = [
  { id: 1, firstName: 'Mary', lastName: 'Brown', age: 27 },
  { id: 2, firstName: 'John', lastName: 'Smith', age: 36 }
];

$scope.department = {
  id: 1,
  name: "Marketing",
  managerId: 1
}
```

```
<omf-single-picker
  items="employees"
  title-expression="[firstName] [lastName]"
  id-prop="id"
  bind-selected-item-id="department.managerId"
  pre-selected-item-id="department.managerId">
</omf-single-picker>
```

## Properties

Name	Bind- ing	Description
items	=	The list of options in the dropdown
titleProp	@	The property on model to be used as the display text
titleExpression	@	The format string for display text. Example: <b>[firstName] [lastName] ([email])</b>
idProp	@	The property on model to be used as the value
onSelect	&	Callback when an option is selected. Parameters: (selectedItem, parentItem)
onDeselect	&	Callback when the dropdown is cleared.
onOpen	&	Callback when the dropdown is opened.
bindSelectedItemId	=	The selected value. This property is only one-way binding from the dropdown to the consumer scope.
preSelectedItemId	=	The initial selected value.
updateSelectedItemEvent	@	The event name used to update selected value of the dropdown from outside scope.
preSelectFirstItem	@	<b>true</b> or <b>false</b> - whether to preselect the first option if no initial selected value is provided.
parentItem	=	An object to be pass along the selected item on the <b>onSelect</b> callback.

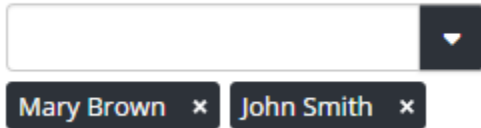
```
.. include:: ../common/authors.txt
```

## Multi Picker

**Note:** This component is an Angular 1 directive registered in module **Omnia.Foundation.Core.Module**

The multi picker is a dropdown component that supports multiple selected values, type-ahead search and customizable UI.

### Department Members



## Sample

```
$scope.employees = [
  { id: 1, firstName: 'Mary', lastName: 'Brown', age: 27 },
  { id: 2, firstName: 'John', lastName: 'Smith', age: 36 }
];

$scope.department = {
  id: 1,
  name: "Marketing",
  managerId: 1,
  members: [
    { id: 1, firstName: 'Mary', lastName: 'Brown', age: 27 },
    { id: 2, firstName: 'John', lastName: 'Smith', age: 36 }
  ]
}
```

```
<omf-multi-picker items="employees"
  title-expression="[firstName] [lastName]"
  selected-items="department.members">
</omf-multi-picker>
```

## Properties

Name	Bind- ing	Description
items	=	The list of options in the dropdown
titleProp	@	The property on model to be used as the display text
titleExpres- sion	@	The format string for display text. Example: <b>[firstName] [lastName] ([email])</b>
idProp	@	The property on model to be used as the value
onSelect	&	Callback when an option is selected. Parameters: (selectedItem)
onDeselect	&	Callback when the dropdown is cleared.
onOpen	&	Callback when the dropdown is opened.
selecte- dItems	=	The selected value. This property is only two-way binding between the dropdown and the consumer scope.

.. include:: ../common/authors.txt

## Client-Side Services

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

.. include:: ../common/authors.txt

## Angular 2

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

.. include:: ../common/authors.txt

## Components

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

.. include:: ../common/authors.txt

## DropDownList

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

DropDownList is a dropdown component that supports type-ahead search and customizable UI.

### How it looks like

Department Manager



### Selector

```
<omf-dropdown-list></omf-dropdown-list>
```

### Parameters

**- items: Array<any>**

The list of options for the dropdown. Could be an array of string or an array of object.

**allowMultipleValues: boolean**

Flag for allowing multiple values to be selected. Default value is **false**.

**textProperty: string**

The name of property to be used as the display text of the options. Only applicable if **items** is an array of object.

**textExpression: string**

The expression for formatting the display text of the options. **textProperty** will be ignored if **textExpression** is set. Only applicable if **items** is an array of object.

Example: text-expression="[firstName] [lastName] ([email])"

**valueProperty: string**

The name of the property to be used as the selected value. Only applicable if **items** is an array of object.

**preSelectFirstItem: boolean**

Flag for setting the first item in **items** as selected if no selected value is provided. Default value is **false**.

**onItemSelected: (selectedItem: any, parentItem?: any) => void**

Event handler for item selected event. If **items** is an array of object, **selectedItem** will be the object that was selected, not just the value property.



**onItemDeselected: (deselectedItem: any, parentItem?: any) => void**

Event handler for item deselected event. If **items** is an array of object, **deselectedItem** will be the object that was selected, not just the value property.

**onDropDownOpen: () => void**

Event handler for dropdown open event.

**onInputChange: (newValue: string) => void**

Event handler for

**showLoading: boolean**

Flag for showing the loading indicator. Should be set to true while **items** is being loaded. Default value is **false**.

**selectedItemValue: any**

Two-way bind selected value(s). If **items** is an array of object then the selected value(s) will be taken from the **valueProperty** of the selected item.

If **allowMultipleValues** is true then **selectedItemValue** will be an array, otherwise it will be a single value.

**parentItem: any**

The related object to be passed in **onItemSelected** and **onItemDeselected**.

## Examples

```
import { Component, Inject, ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-component'
})
export class MyComponent {
```

(continues on next page)

(continued from previous page)

```

employees = [
    { id: 1, firstName: 'Mary', lastName: 'Brown', age: 27 },
    { id: 2, firstName: 'John', lastName: 'Smith', age: 36 }
];

selectedEmployeeId = 1;

constructor(@Inject(ViewContainerRef) private viewContainer: ViewContainerRef) {
}

```

```

<omf-dropdown-list [items]="employees"
    textProperty="firstName"
    valueProperty="id"
    [(selectedItemValue)]="selectedEmployeeId">
</omf-dropdown-list>

```

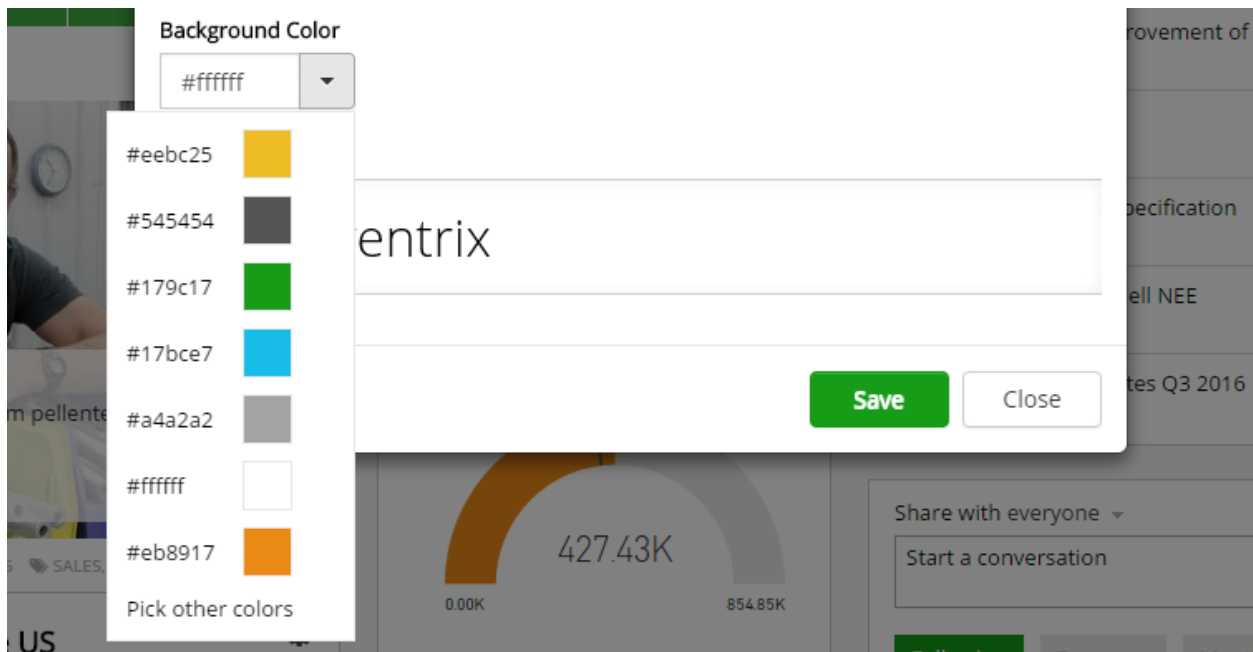
```
.. include:: ../common/authors.txt
```

## ColorPicker

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

ColorPicker is a component for selecting color from Omnia corporate colors feature or from custom colors.

### How it looks like



## Selector

```
<omf-color-picker></omf-color-picker>
```

## Parameters

### - color: string

Two-way bind selected color.

### onColorChange: (color: string) => void

Even handler for the color selected event.

### position: string

The position of the color picker popover. Must be one of these: 'top', 'bottom', 'left', 'right'

## Examples

```
import { Component, Inject, ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-component'
})
export class MyComponent {
  selectedColor = '#ffffff';

  constructor(@Inject(ViewContainerRef) private viewContainer: ViewContainerRef) {
  }
}
```

```
<omf-color-picker [color]="selectedColor" [position]="'bottom'"></omf-color-picker>
```

```
.. include:: ../../common/authors.txt
```

## Services

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

### Topics

.. include:: ../common/authors.txt

### AJAX Service

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

AJAX service is a utility for making AJAX requests to Omnia Foundation and Omnia extensions web API. The AJAX service is usually not used directly in UI components but in other services.

### Available Methods

Method	Description	Parameters
buildRequest	Fluent API for making an AJAX request	<ul style="list-style-type: none"><li>- <b>apiPath (string)</b>: The request's URL. If apiPath is a relative URL, it will be direct to Omnia Foundation. Other extensions can inherit this AJAX and override the internal method <b>getFullApiPath</b> to make calls to it's API instead</li><li>- <b>dataType (string) (optional)</b>: The request's Content-Type. Default value is 'application/json'</li></ul>

### Examples

---

**Note:** To use the AjaxService, you need to import The module OmniaExtensibilityModule into the NgModule of your component or add it directly to the list of providers of your compoment

---

### Normal Usage

```
import { AjaxService } from "Omnia/Foundation/Extensibility/Services";
import { Pipe, Injectable , Inject } from '@angular/core';

@Injectable()
export class ConfigurationService {
  constructor(@Inject(AjaxService) private ajaxService: AjaxService) {

  }

  public getConfiguration = (callback: (result: Configurations.IConfiguration) =>
  void, name: string, region: string, extensionPackageId: string = null) => {
```

(continues on next page)

(continued from previous page)

```

        var params = {
            name: name,
            region: region,
            extensionPackageId: extensionPackageId
        };

        this.ajaxService.buildRequest("configuration/configurations")
            .addQueryStrings(params)
            .doGet<Configurations.IConfiguration>()
            .subscribe((result) => { callback(result.json()); });
    }
}

```

### Inherit AJAX Service

```

import { AjaxService as FoundationAjaxService } from 'Omnia/Foundation/Extensibility/
↳Services'
import { Utils } from "Omnia/Foundation/Extensibility";

@Injectable()
export class AjaxService extends FoundationAjaxService{
    static apiBaseUrl: string = "";

    public getFullApiPath(apiPath: string): string {
        if (Utils.isNullOrEmpty(AjaxService.apiBaseUrl)) {
            AjaxService.apiBaseUrl = Utils.ensureTrailingSlash("<my-extension-api-url>
↳");
        }

        return AjaxService.apiBaseUrl + apiPath;
    }
}

```

```
.. include:: ../common/authors.txt
```

### Dialogs Service

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

Dialogs service is a utility for working with modal dialogs in Angular 2 and Omnia.



## Available Methods

Method	Description	Parameters
onConfirmationDialog	Show a confirmation dialog with yes/no outcome	<ul style="list-style-type: none"> <li>- <b>title (string)</b>: The dialog's title</li> <li>- <b>body (string)</b>: The dialog's body text</li> <li>- <b>viewControllerRef (ViewControllerRef)</b>: the viewControllerRef of the calling component</li> <li>- <b>okCallback () =&gt; void</b>: The callback for 'OK' outcome</li> <li>- <b>cancelCallback () =&gt; void</b>: The callback for 'Cancel' outcome</li> </ul>
openDialog	Show a custom dialog	<ul style="list-style-type: none"> <li>- <b>componentType (Type&lt;any&gt;)</b>: The component to show in the dialog</li> <li>- <b>params (any)</b>: the parameters passed to the dialog</li> <li>- <b>viewControllerRef (ViewControllerRef)</b>: the viewControllerRef of the calling component</li> <li>- <b>dialogSize (DialogSize)</b>: can be small, medium and large. Default value is medium</li> <li>- <b>okCallback () =&gt; void</b>: The callback for 'OK' outcome</li> <li>- <b>cancelCallback () =&gt; void</b>: The callback for 'Cancel' outcome</li> </ul>
blockUI	Show a loading indicator and block the whole page UI	
unblockUI	Remove the UI block from	
1.7. Client-Side Development	blockUI method	91

### Examples

**Note:** To use the DialogService, you need to import The module OmniaExtensibilityModule into the NgModule of your component or add it directly to the list of providers of your component

---

#### Injection

```
import { DialogService } from "Omnia/Foundation/Extensibility/Services";
import { Component, Inject, ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-component',
  providers: [ DialogService ]
})
export class MyComponent {
  constructor(@Inject(ViewContainerRef) private viewContainer: ViewContainerRef,
    @Inject(DialogService) private dialogService: DialogService) {
  }
}
```

#### Open confirmation dialog

```
private deleteItem(item) {
  let dialogTitle = "Delete Item";
  let dialogBody = "Are you sure you want to delete this item?";

  this.dialogService.onConfirmationDialog(dialogTitle, dialogBody, this.
    viewContainer, () => {
    // Confirmed, proceed to delete the item
  });
}
```

#### Open custom dialog

```
import { EditItemForm } from "MyComponent/EditItemForm";
import { DialogSize } from "Omnia/Foundation/Extensibility/Enums";

// ...

private editItem(item) {
  this.dialogService.openDialog(EditItemForm, { item: item },
    this.viewContainer, DialogSize.Large);
}
```

```
import { Component, Inject, ViewContainerRef, OnDestroy , OnInit } from '@angular/core';
import { DialogRef } from 'angular2-modal';
import { BaseDialogComponent, BaseDialogModel } from "Omnia/Foundation/Extensibility/Services";

@Component({
  selector: 'edit-item-form'
})
export class EditItemForm extends BaseDialogComponent<BaseDialogModel<any>> {
  implements OnInit {
```

(continues on next page)



(continued from previous page)

```
item: Item;

constructor(@Inject(DialogRef) public dialog: DialogRef<BaseDialogModel<any>>) {
    super(dialog);
}

ngOnInit() {
    this.item = this.context.params.item;
}
}
```

```
.. include:: ../../common/authors.txt
```

## Configuration Service

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

Configuration service is a utility for working with Omnia configurations. Read more about configuration in Omnia [here](#)



## Available Methods

Method	Description	Parameters
getConfiguration	Get configuration from Omnia Foundation API	<ul style="list-style-type: none"> <li>- <b>callback</b> ((<b>result: Configurations.IConfiguration</b>) =&gt; <b>void</b>): The callback with the requested configuration</li> <li>- <b>name</b> (<b>string</b>): The name of the configuration</li> <li>- <b>region</b> (<b>string</b>): The region of the configuration</li> <li>- <b>extensionPackageId</b> (<b>string</b>) (<b>optional</b>): The ID of the extension that created the configuration. Default value is built-in configurations</li> </ul>
getClientConfiguration	Get configuration included in client-side	<ul style="list-style-type: none"> <li>- <b>name</b> (<b>string</b>): The name of the configuration</li> <li>- <b>region</b> (<b>string</b>): The region of the configuration</li> <li>- <b>extensionPackageId</b> (<b>string</b>) (<b>optional</b>): The ID of the extension that created the configuration. Default value is built-in configurations</li> </ul>
addOrUpdateConfigurations	Add or update a list of configurations	<ul style="list-style-type: none"> <li>- <b>configurations</b> (<b>Array&lt;Configurations.IConfiguration&gt;</b>): The list of configurations to add or update</li> <li>- <b>callback</b> ((<b>isSuccess: boolean</b>) =&gt; <b>void</b>): The callback function</li> </ul>
getConfigurationsInRegion	Get all configurations by region	<ul style="list-style-type: none"> <li>- <b>region</b> (<b>string</b>): The region of the configurations</li> <li>- <b>callback</b> ((<b>result: Array&lt;Configurations.IConfiguration&gt;</b>) =&gt; <b>void</b>): The callback function</li> </ul>
<b>1.7. Client-Side Development</b>		with result <b>95</b>
updateConfiguration	Update a configuration	<ul style="list-style-type: none"> <li>- <b>configuration</b></li> </ul>

### Examples

---

**Note:** To use the ConfigurationService, you need to import The module OmniaExtensibilityModule into the NgModule of your component or add it directly to the list of providers of your component

---

#### Injection

```
import { ConfigurationService } from "Omnia/Foundation/Extensibility/Services";
import { Component, Inject, ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-component',
  providers: [ ConfigurationService ]
})
export class MyComponent {
  constructor(@Inject(ViewContainerRef) private viewContainer: ViewContainerRef,
    @Inject(ConfigurationService) private configurationService: ↳
    ConfigurationService) {
  }
}
```

#### Get configuration

```
private getDefaultColors() {
  this.configurationService.getConfiguration((configuration: Configurations.
  ↳ IConfiguration) => {
    let defaultColors = configuration.value;
  }, "defaultcolors", "");
}
```

```
.. include:: ../common/authors.txt
```

### Localization Service

---

**Note:** This documentation is a work in progress and contributions can be made on our Github repo

---

Localization service is a utility for getting localized text in Omnia. Read more about localization in Omnia [here](#)

#### Available Methods

Method	Description	Parameters
getText	Get the localized text for a label in Omnia	- <b>key (string)</b> : The key for the localization. If no localized text match, this key will be return

## Examples

**Note:** To use the `LocalizationService`, you need to import The module `OmniaExtensibilityModule` into the `NgModule` of your component or add it directly to the list of providers of your component

### Injection

```
import { LocalizationService } from "Omnia/Foundation/Extensibility/Services";
import { Component, Inject, ViewContainerRef } from '@angular/core';

@Component({
  selector: 'my-component',
  providers: [ ConfigurationService ]
})
export class MyComponent {
  constructor(@Inject(ViewContainerRef) private viewContainer: ViewContainerRef,
    @Inject(LocalizationService) private localizationService: LocalizationService) {
  }
}
```

### Get localized text

```
private getItemTypes() {
  return [
    { id: 0, title: this.localizationService.getText('MyExtension.ItemTypes.Small') },
    { id: 1, title: this.localizationService.getText('MyExtension.ItemTypes.Medium') },
    { id: 2, title: this.localizationService.getText('MyExtension.ItemTypes.Large') }
  ];
}
```

```
.. include:: ../../common/authors.txt
```

## 1.8 Performance

The following sections will guide you in using functionality in Omnia to improve the performance in your solution

### 1.8.1 ClientContext.LoadIfNeeded and ClientContext.ExecuteQueryIfNeeded

Often, a single SharePoint Client Context is passed between different services and you do not know if the data you need has already been loaded or if an **ExecuteQuery** is needed.

To make this process more effective and straight forward, Omnia has implemented two extension methods on **ClientContext**: **LoadIfNeeded** and **ExecuteQueryIfNeeded**.

By using the above methods, already loaded and fetched data will not trigger a new **ExecuteQuery** to the server.

Example:

```
public string GetServerRelativeUrl(ClientContext ctx)
{
    ctx.LoadIfNeeded(Ctx.Web, x => x.ServerRelativeUrl).ExecuteQueryIfNeeded();

    return ctx.Web.ServerRelativeUrl;
}
```

The **ExecuteQueryIfNeeded** method also takes an optional parameter of type **Microsoft.SharePoint.Client.ClientContextExtensions.ExecuteOption**. This is an enum with two possible values:

- LoadAllIfOneIsNeeded
- LoadOnlyNeeded

This is useful to decide if only missing property should be loaded from the server, or if all loaded properties should be reloaded if an execute needs to be done.

If this parameter is not supplied, **LoadOnlyNeeded** is used.

```
.. include:: ../common/authors.txt
```

## 1.9 Release Notes

### 1.9.1 Topics

```
.. include:: ../common/authors.txt
```

## Omnia Foundation

### Sections:

- *1.0.1.1681 (2016-09-20)*

### 1.0.1.1681 (2016-09-20)

#### What's new

- It's now possible in the API for an extension to change the configuration created by another extension.
- A new feature called "Omnia Statistics Provider" makes it possible to register a third-party analytics script in the Admin UI to be included on all pages within a specific site collection.
- Targeting api that makes it possible to specify targeting definitions in Omnia Admin using Taxonomy/Profile Properties or Security Groups and then use the api in extensions to filter content based on selected targeting.
- Omnia Jobs is a new feature that makes it possible to create Jobs in Extensions that can be based on a Timer or by listening on a Queue.
- OmniaApi support to work with queues.
- Updated font awesome to version 4.6.3.
- Omnia Console now has support to listen to commands. Currently we support two commands that enables support for the built in webserver in Omnia Tooling for example:

- hosting enable <https://localhost:9930>
- hosting disable
- A new section in Omnia Admin > System called Developers was added. It lists the version of Omnia Foundation and also displays the nuget version to be used for Extensions.
- OmniaApi now has a SitesService that has the following methods GetSiteRequestById, GetRecentSites, AddOrUpdateRecentSite.

### Bug fixes

- When provisioning a Field without a description an error was thrown.
- Left navigation in admin missing scroll support.
- Slow performance in Features view in Admin.
- Fixed an performance issue where the JobHost was loading all extension packages from database at startup.
- When adding new properties to a site template and performing a migration the migration would stop if a site was deleted.
- When using approval flow in site requests there was sometimes a timeout error when approving the request.
- When editing site templates deployed by extensions the save button was always disabled.
- The announcements control had a problem with overflow when long text without space was in the message.
- When uploading a extension package with configuration we now always sort it in alphabetic order.
- The customized localization made in the localization editor in Omnia Admin for an extension package was removed when a feature was upgraded.
- Selected features in site templates was not checked in UI when the template was loaded.
- When specifying a FieldRef using [FieldRef(typeof(Field1), Required = true)] it didn't listen on the Required property.

.. include:: ../common/authors.txt

## Omnia Tooling

Here you can find the releases of the Omnia Tooling for Visual Studio 2013/2015

### Sections:

- *Visual Studio 2017 1.0.0*
- *Visual Studio 2017 1.0.6924 Beta 4 (Vietnam Codebase Edition)*
- *Visual Studio 2017 1.0.6828 Beta 3 (France Codebase Edition)*
- *Visual Studio 2017 1.0.6250 Beta 2*
- *Visual Studio 2017 1.0.5935 Beta 1*
- *Visual Studio 2017 Preview*
- *Stable 1.0.1.3965*
- *Stable 1.0.1.2305-62*
- *Stable 1.0.1.1699-48*

- *Stable 1.0.1.1115-38*

### Visual Studio 2017 1.0.0

Please report issues with tooling on our [Github repo](#)

[Download vsix](#)

### Visual Studio 2017 1.0.6924 Beta 4 (Vietnam Codebase Edition)

---

#### Note:

- To be able to use this Tooling you need to target a Tenant with Omnia Foundation version 1.0.5862 or higher

#### What's new

- Improvement in Resource Mapper
- Its now possible to create new Omnia Controls without any additional setup
- Bugfixes

Please report issues with tooling on our [Github repo](#)

[Download vsix](#)

### Visual Studio 2017 1.0.6828 Beta 3 (France Codebase Edition)

---

#### Note:

- To be able to use this Tooling you need to target a Tenant with Omnia Foundation version 1.0.5862 or higher

#### What's new

- Visual Resource Mapper makes it easy adding resources to mapping files
- Support for Visual Resource Mapper when adding TenantResources from item templates

Please report issues with tooling on our [Github repo](#)

[Download vsix](#)

### Visual Studio 2017 1.0.6250 Beta 2

---

#### Note:

- To be able to use this Tooling you need to target a Tenant with Omnia Foundation version 1.0.5862 or higher

#### What's new

- Improved Copy to Tenant



- Remove extra folder from solution
- Upgrade TypeScript to version 2.4.1

Please report issues with tooling on our [Github repo](#)

[Download vsix](#)

## Visual Studio 2017 1.0.5935 Beta 1

---

### Note:

- To be able to use this Tooling you need to target a Tenant with Omnia Foundation version 1.0.5862 or higher
- 

### What's new

- Completely new Project Wizard and Item Template Wizard that uses Github as source
- Angular 4 support
- Extensible TaskRunner (angular aot, less etc)
- We now use npm for Foundation (<https://www.npmjs.com/package/@omnia/foundation>)

Please report issues with tooling on our [Github repo](#)

[Download vsix](#)

## Visual Studio 2017 Preview

### What's new

- Added support for Visual Studio 2017

[Download vsix](#)

## Stable 1.0.1.3965

### What's new

- New extension projects can now build without errors

[Download vsix](#)

## Stable 1.0.1.2305-62

### What's new

- Update Omnia Control Item Templates for Angular 2 to comply with Angular 2.0.0
- Update Omnia Extension Sample Project Template with new samples of Angular 2

[Download vsix](#)

### Stable 1.0.1.1699-48

#### What's new

- Omnia Control Item Templates for Angular 2
- Built in webserver for hosting Tenant bundles locally
- Live Reload support for Tenant bundles

---

**Note:** The Omnia Control Templates for Angular 2 is only for preview purposes since the Angular 2 RTM was just released we removed the bootstrapping for Angular 2 in Foundation until we have a working version running on Angular 2 RTM

---

#### Bug fixes

- The item template for Field contained a space in the internalname which could cause problems in provisioning
- Item Template for Omnia Control without settings should have enableSettings value set to false and the constructor should not have the ControlConfigService injected

[Download vsix](#)

### Stable 1.0.1.1115-38

[Download vsix](#)

```
.. include:: ../common/authors.txt
```

## 1.10 Contribute to this Documentation

This documentation is a constant work in progress, and contributions can be made on our Github repo. For more extensive contributions please follow the steps below to get started with using Sphinx.

```
.. include:: ../common/authors.txt
```

### 1.10.1 Get started

Please follow the steps below to install Sphinx and get the source code from our Git repository.

#### 1. Install Python

Download and install Python 3.6.0 from here <https://www.python.org/downloads/>

---

**Note:** Make sure to select **pip** and **Add Python to environment variables** during the installation

---

## 2. Install Sphinx

1. Open a command prompt
2. Run

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme
```

## 3. Install Git for Windows

Download and install Git from here <https://git-scm.com/download/win>

## 4. Get the Git repository

1. Create a folder in your file system where you would like to keep the documentation source code, for example

```
C:\Git\OMF-docs
```

2. In a command prompt, navigate to the folder you just created and execute the following command

```
git clone https://github.com/preciofishbone/Omnia-Foundation-Docs.git
```

The source code is now cloned to the current folder

```
.. include:: ../common/authors.txt
```

### 1.10.2 Work with the documentation

To work with the source code locally using Sphinx, follow the steps below

1. Open a command prompt and navigate to the folder containing the source code for the documentation
2. Execute the following command

```
make livehtml
```

After a while the console window will settle down, and you will find a link in the window, like this  
**Serving on http://127.0.0.1:8000**

3. In a web browser, enter the link from above. A locally rendered version of the documentation will be displayed.

Now, any time you add a file, or edit an existing file and then save it, Sphinx will rebuild the documentation locally and reload the browser window to display the latest changes.

You do edits, additions and deletes by just working with the files in the source code folder in your favorite text editor.

---

**Note:** The documentation is written with **reStructuredText**. To learn more about this markup language, you can use this link <http://sphinx-doc.org/rest.html#rst-primer>

---

To stop Sphinx, press **Ctrl + C** in the command prompt and then select **Y**

```
.. include:: ../common/authors.txt
```

### 1.10.3 Commit your changes

To commit your change to the Git-repo perform the following steps

1. Open a command prompt and navigate to the folder containing the source code for the documentation
2. Execute the following command

```
git pull
```

3. Execute the following command

```
git commit -a -m "A description of your changes"
```

4. Execute the following command

```
git push
```

You have now successfully contributed to this documentation. Thank you for your time and effort!