

The system allows you to transfer custom modules to other instances of Kentico using [NuGet packages](#).

When creating modules that you wish to deploy as installation packages, you need to take additional steps during the development process. The module must be represented by a separate web application project inside the Kentico solution. Based on [conventions](#), the installation package automatically includes files from specific folders and database objects related to the module.

The module project only exists on the Kentico instance that you use to develop the module. When you [install the module](#) on a different instance, all code (including code behinds of web forms and controls) is compiled inside the module DLL, and other files are integrated into the main Kentico project. Installation of modules is supported for both *web application* and *web site* type projects.

Module packages do not include any data stored in module classes. If you wish to provide default data along with your module, you can prepare additional [export packages](#) containing the data or an [SQL script](#) that creates the required data after the installation of the module.

### Limitations

Modules transferred via installation packages *currently* have the following limitations:

- To install or uninstall module packages, you need to open the target project in Visual Studio – the operations cannot be performed on certain types of deployed and running websites (for example on Microsoft Azure).
- You cannot use installation packages to transfer modules for development on other instances – the module always becomes sealed and uncustomizable after installation, and all code is compiled into a DLL. If you wish to develop the module on multiple instances, use [export](#) and [import](#) features instead.
- You cannot define dependencies between multiple module packages.
- Modules installed from packages cannot be directly downgraded to older versions. You can however uninstall a module and then install a package with an older version.
- You may encounter problems when using components (user controls etc.) from the main Kentico project inside the module's project. This may make it difficult to develop custom web forms, user controls or web parts within packageable modules.
- If you transfer a web part with a [custom layout](#) as part of the module, the layout does not work correctly when [Deployment mode](#) is enabled.
- You cannot create installation packages for:
  - The default Kentico modules.
  - The default *Custom* module, which is intended for non-transferable customizations.
  - Modules installed from other installation packages.

**To create an installation package for a custom module, you need to:**

1. [Create a separate project for the module in the Kentico solution](#)
2. Develop the module (see [Example - Creating a packageable module](#) for an example)
3. [Make sure the installation package includes database objects related to the module](#)
4. Optional steps:
  - [Add additional libraries to the module installation package](#)
  - [Configure the module's custom settings](#)
  - [Prepare additional SQL scripts for the module installation](#)
5. [Create the module installation package](#)

## Creating the module project



We strongly recommend using *web application* installations of Kentico when developing custom modules that you wish to deploy as installation packages.

Module projects must always use the [web application format](#) – having the same project type for the main Kentico project makes it easier to reuse files. With *web site* projects, you need to manually ensure that all module files are converted to the web application format.

Before you can develop a module that supports the creation of installation packages, you need to prepare a web application project and include this project in the Kentico solution.

The following steps are necessary to maintain the proper folder structure within the Kentico solution:

1. [Create the project](#)
2. [Copy the project files to the Kentico solution](#)
3. [Include the project in the Kentico solution](#)

### Creating the project

1. Open Visual Studio and create a new web application project.
  - **Important:** The project name must match the module's code name.
  - Select the **ASP.NET Web Application** project template (Empty).
2. Delete the project's *Web.config* file.
3. Rename the *packages.config* file to *packages.<project name>.config*.
4. In the project's **Properties** folder, rename the *AssemblyInfo.cs* file to *<project name>AssemblyInfo.cs*.
5. Save the project.

### Copying the project files to the Kentico solution

1. Open the project's folder on your file system.
2. Edit the *<project name>.csproj* file and make sure the content reflects the changes in the names of the *AssemblyInfo.cs* and *packages.config* files.

```
<ItemGroup>
    <Content Include="packages.{project name}.config" />
</ItemGroup>
<ItemGroup>
    <Compile Include="Properties\{project name}AssemblyInfo.cs" />
</ItemGroup>
```

3. Copy the *<project name>.csproj* file and *packages.<project name>.config* file (if present) to the **CMS** folder of your Kentico web project.
4. Copy *<project name>AssemblyInfo.cs* from the project's **Properties** directory to the **CMS\Properties** folder of your Kentico web project.

### Including the project in the Kentico solution

1. Open your Kentico solution in Visual Studio (using the *WebApp.sln* file).
2. Add your custom module project to the Kentico solution:
  - a. In the Solution Explorer, right-click the solution.
  - b. Click **Add -> Existing Project...**
  - c. Select the *<project name>.csproj* file in the project's **CMS** folder.
3. Add the required references to your custom module project:
  - a. Right-click the project and select **Add -> Reference**.
  - b. Open the **Browse** tab of the **Reference manager** dialog, click **Browse** and navigate to the **Lib** folder of your Kentico web project.
  - c. Add a reference to the **CMS.Core.dll** library (and any others that you require for custom code).

4. Expand the **Properties** folder of the module project and edit `<project name>AssemblyInfo.cs`.
5. Add the **AssemblyDiscoverable** assembly attribute:

```
using CMS;  
  
[assembly:AssemblyDiscoverable]
```

6. Save the solution and all files.

The Kentico solution now contains a web application project representing your custom module. You can start developing the module. After you compile the module code into an assembly (i.e. build the project), the system automatically includes the resulting DLL when you [create an installation package for the module](#).

The additional project only exists on the Kentico instance that you use to develop the module. When you [install the module](#) on a different instance, all code is already compiled inside the module's DLL and other files are integrated into the main Kentico project.



**Note:** If you ever need to rename a module during development, **Clean** your solution in Visual Studio **before** you rename the module project or manually delete the original DLL. Otherwise the module's old DLL may cause errors on your site.

## Including database objects in the package

When [creating](#) module installation packages, the system automatically includes the following database objects that are directly related to the given module:

- Classes
- Permissions
- UI elements
- Setting categories and groups, including all contained setting keys

You can configure [Page types](#) to be included in the installation packages of a specific custom module. Edit the page type in the **Page types** application and choose a module via the **Include in module package** selector.

By following [naming conventions](#), you can also create objects of the following types that the system includes in the module's installation packages:


- [Web parts](#)
- Web part categories
- [Form controls](#)



**Note:** Module installation packages cannot transfer other types of database objects, such as for example [scheduled tasks](#). You can create separate [export packages](#) for additional objects and deliver them together with your module's installation package.

## Adding additional libraries to the package

The system cannot automatically identify all dynamic link libraries (DLLs) required by your custom module. If your module uses third-party or custom libraries, you need to assign these libraries to the module before creating the installation package.

1. Open the **Modules** application and **Edit** () your custom module.
2. Switch to the **Additional libraries** tab.
3. Assign any number of libraries from the project's **bin** folder:
  - a. Click **Add library**.
  - b. Select the library that you wish to include in the module's installation package.
  - c. Click **Select**.

When you create the module installation package, the package also contains the selected libraries.

**Note:** Do NOT assign DLLs that are part of the default Kentico project as additional libraries of module packages. This includes any DLL files in the *CMS\bin* or *CMS\CMSDependencies* folders of unmodified Kentico projects. Adding these libraries may cause the system to work incorrectly after installing or uninstalling the custom module package.

## Configuring custom module settings

If you have defined [custom settings](#) for the module, we recommend configuring the default values of the settings before you create the installation package. Open the **Settings** application and set the desired (*global*) values for the module's settings.

The setting values configured on the development instance are used by default when the module is installed on other instances of Kentico.

## Preparing additional SQL scripts for the installation

Installation packages cannot automatically transfer certain types of database components, such as **indexes** and **views**. If you wish to deliver such components with the module, you need to manually prepare SQL scripts that create (and remove) the required database objects.

Add the script files into the following folders in the development project (create the folders if necessary):

- **~/App\_Data/CMSModules/<module code name>/**
  - **Install** – scripts run before or after the import of the module's database objects (do not run when updating the module from an older version)
  - **Update** – scripts run before or after the import of database objects when updating the module from an older version
  - **Uninstall** – scripts run before or after the removal of the module's database objects

In all cases, the script files must be named either **before.sql** or **after.sql**. In the uninstallation scripts, you typically need to remove objects that you created in the opposing installation scripts (for example **Uninstall/before.sql** to reverse **Install/after.sql**).

The system automatically includes the script files into the module's installation packages. When [installing](#) or [uninstalling](#) the module, the appropriate *before.sql* and *after.sql* scripts are executed against the target database.

## Creating the module installation package

Once your custom module is fully developed, you can create the installation package in the Kentico administration interface:

1. Open the **Modules** application and **Edit** (🔧) your custom module.
2. On the **General** tab, fill in the module's metadata:

- **Module description** (Default value: *No description provided*)
- **Module version** (Default value: *1.0.0*)
- **Module author** (Default value: *Unknown*)



**Tip:** Developers can perform more advanced modifications of the module package metadata (i.e. the [nuspec XML manifest](#)). Assign a custom handler to the system's **ModulePackagingEvents.BuildNuSpecManifest** [global event](#).

3. Click **Save**.
4. Click **Create installation package**.
  - The system opens a dialog showing the package's metadata, and the files and objects included in the package.
5. Click **Create**.

The system creates the module installation package in the project's export folder (**CMS\CMSSiteUtils\Export** by default). You can use the package to [install](#) the module on other instances of Kentico.



### Kentico version requirements of module packages

Module packages can be successfully installed only on Kentico instances that have the same version as the source instance, including minor hotfix versions. For example, if you create a module installation package on a 9.0.16 instance, users cannot install the package on Kentico 9.0.15 or older. We strongly recommend adding information about the minimum required Kentico version into the **Module description**.

Module packages from older versions can be installed on newer versions, but may cause problems or errors depending on the differences in the data structure and API of the two versions.

## Preparing update packages for modules

If you make further changes to a module after creating an installation package, you can prepare a new version of the package at any time. Use the following procedure:

1. Add or update the files and database objects for the module.
2. Set a higher **Module version** number when editing the module on the *General* tab in the *Modules* application.
3. Create the installation package.

When **installing** the newer version of the package on an instance that already contains an older version of the module, the system adds any new files or database objects and updates existing ones. If the instance does not contain the module at all, the standard installation occurs.



**Limitation:** The module update functionality does not automatically support **removing** of database objects related to the module (classes, web parts, form controls etc.). If the newer version of the module installation package does not contain an object that exists in an older version, updating the module will NOT automatically remove the given object on the target instance.

## Adding update SQL scripts

If you wish to perform additional database operations during the module update (for example add, update or remove database **indexes** and **views**), you need to manually prepare SQL scripts. You can use the update SQL scripts as a workaround for removing module-related objects from the database.

Add the script files into the `~/App_Data/CMSModules/<module code name>/Update` folder in the development project (create the folder if necessary). The script files must be named either **before.sql** or **after.sql**. The system runs the scripts before or after the import of the module's database objects (only when updating from an older module version to a newer one).



### Adjusting the update scripts based on the original module version

When running the scripts during the update of a module, the system supplies the original version number of the module via the **@FromVersion** SQL parameter. You can use the parameter to branch your module's update scripts and perform different operations according to the module version from which you are updating.

## Reference - Custom module conventions

To ensure that the installation packages created for your custom modules contain the required files and database objects, you need to follow naming and location conventions, or directly assign objects to modules in the administration interface.

- [File system conventions](#)
- [Conventions for database objects](#)

### Module code name

The naming conventions for folders, files and database objects are based on the **Module code name**, including any prefixes or namespaces. For example: *Acme.Forums*

Carefully consider the code name when creating custom modules. Choose a **sufficiently unique** module code name to avoid collisions with the default Kentico modules or other custom modules. Do NOT start the code names of custom modules with the **cms.** prefix, which is reserved for Kentico modules.

## File system conventions

### Important

Include all custom files related to your module into the module's web application project in Visual Studio, NOT the default Kentico web project (*CMSApp*).

Both projects share the same folder on the file system, so you can include files located in the default Kentico folders (such as *~/App\_Data/CMSModules* or *~/CMSWebParts*) without problems. When the module package is installed on a different instance, files from both projects are merged into the standard Kentico project.

Including files in the module project ensures that:

- Custom module files are clearly separated in your development solution from the default files.
- All required code is compiled into the module's DLL. Module installation packages transfer all code within the module DLL, without the original code files. Files with separate code behinds (web forms, user controls, web parts, form controls, handlers, etc.) are installed without the code behind and instead reference the appropriate class in the module DLL.

Module component	Convention
Module library	<p>The library containing the module's compiled code must be located in the <i>~/bin</i> folder of the Kentico project and have the same name as the module code name. For example, the <i>Acme.Forums</i> module has the <i>Acme.Forums.dll</i> library.</p> <p>As long as the module's web application project in the Kentico solution matches the module name, the library is created automatically when you compile the project.</p>
3rd party libraries	<p>You can <a href="#">manually include 3rd party libraries</a> into the module's installation package using the administration interface. The libraries must be located in the <i>~/bin</i> folder of the Kentico project.</p>
Code files	<p>All code files must be included in the module's web application project in Visual Studio. Examples of module code files are:</p> <ul style="list-style-type: none"> <li>• <i>Info</i> and <i>InfoProvider</i> files of the module's classes</li> <li>• The file containing the module's <a href="#">initialization code</a></li> <li>• <a href="#">User interface extenders</a></li> </ul> <p>We recommend organizing the code files inside a folder that matches the code name of the module, for example: <i>~/Acme.Forums/</i></p>
Web forms and controls	<p>Place web forms or user controls used by your module into the <i>~/CMSModules/&lt;module code name&gt;/</i> folder.</p>



Web part files	<p>Place the source files (user controls) of <a href="#">web parts</a> that are part of your module into the <code>~/CMSWebParts/&lt;module code name&gt;/</code> folder.</p> <p>For example: <code>~/CMSWebParts/Acme.Forums/</code></p>
Form control files	<p>Place the source files (user controls) of <a href="#">form controls</a> that are part of your module into the <code>~/CMSFormControls/&lt;module code name&gt;/</code> folder.</p> <p>For example: <code>~/CMSFormControls/Acme.Forums/</code></p>
UniGrid definitions	<p>Place the XML definitions for the module's <a href="#">UniGrid</a> components (object listings) into the <code>~/App_Data/CMSModules/&lt;module code name&gt;/UI/Grids/</code> folder.</p> <p>The default recommended location includes further subfolders named after specific object types. For example: <code>~/App_Data/CMSModules/Acme.Forums/UI/Grids/Acme_Forums_Forum/default.xml</code></p>
Resource strings	<p>Place the module's <a href="#">resource strings</a> into <i>resource files</i> (.resx) inside the <code>~/CMSResources/&lt;module code name&gt;/</code> folder. To add resource files for specific languages, create subfolders with names that match the required <i>culture code</i>.</p> <p>For example, <code>~/CMSResources/Acme.Forums/Common.resx</code> for strings in the default culture, and <code>~/CMSResources/Acme.Forums/&lt;language code&gt;/Common.resx</code> for the <code>&lt;culture code&gt;</code> culture.</p> <p>Note: If you plan to <a href="#">publish</a> your development project, set the <a href="#">Build Action</a> property of resource files to <b>Content</b> to ensure that they are included. The Build Action of resx files is automatically set to <i>Content</i> when installing module packages on other instances.</p>
JavaScript files	<p>If your module uses JavaScript files, place them into the <code>~/CMSScripts/CMSModules/&lt;module code name&gt;/</code> folder.</p> <p>For example: <code>~/CMSScripts/CMSModules/Acme.Forums/TextBox.js</code></p>
CSS	<p>If your module uses physical CSS files, place them into the <code>~/CMSModules/&lt;module code name&gt;/Content/</code> folder.</p> <p>For example: <code>~/CMSModules/&lt;module code name&gt;/Content/Forums.css</code></p> <p>To add links to module stylesheets on pages, call the following methods (in the code behind of pages or components such as form controls):</p> <ul style="list-style-type: none"> <li><code>ModuleCssRegistration.RegisterModuleStylesheets</code> - adds links for all stylesheets of the specified module.</li> <li><code>ModuleCssRegistration.RegisterModuleStylesheet</code> - adds a link for one particular module stylesheet.</li> </ul> <p>The <code>ModuleCssRegistration</code> class is located in the <code>CMS.Base.Web.UI</code> namespace.</p>
HTTP handlers	<p>If your module uses HTTP handlers or other types of system pages, place them into the <code>~/CMSModules/&lt;module code name&gt;/CMSPages/</code> folder.</p> <p>For example: <code>~/CMSModules/Acme.Forums/CMSPages/GetAttachmentHandler.ashx</code></p>

## Conventions for database objects

Object type	Convention
-------------	------------

<a href="#">Web part</a>	<p>The code names of web parts must start with the <b>&lt;module code name&gt;.</b> prefix (including the dot character).</p> <p>For example: <i>Acme.Forums.ForumList</i></p> <p>Module installation packages automatically include the parent categories of web parts that match the naming convention.</p>
Web part category	<p>The code names of web part categories (<b>Category name</b> field) must start with the <b>&lt;module code name&gt;.</b> prefix (including the dot character).</p> <p>For example: <i>Acme.Forums.WebpartCategory</i></p> <p>Including a web part category into the module installation package does NOT automatically include all web parts inside the given category. The web parts are only included if their code name matches the module naming convention.</p> <p>You only need to use the module name convention for web part categories to ensure that the system deletes the categories and their entire content when the <a href="#">module package is uninstalled</a>.</p>
<a href="#">Form control</a>	<p>The code names of form controls must start with the <b>&lt;module code name&gt;.</b> prefix (including the dot character).</p> <p>For example: <i>Acme.Forums.TextBox</i></p>
<a href="#">Page type</a>	<p>Pages types do not use module naming conventions. To include a page type into the installation packages of a custom module:</p> <ol style="list-style-type: none"> <li>1. Open the <b>Page types</b> application.</li> <li>2. Edit the page type.</li> <li>3. On the <b>General</b> tab, select the module in the <b>Include in module package</b> property (you can only choose custom modules that are still in development mode).</li> <li>4. Click <b>Save</b>.</li> </ol> <p><b>Note:</b> After you include a page type into the installation packages of a module, you can no longer specify on which <i>Sites</i> the page type is available. The page type automatically uses the site bindings of the given module, which you can edit in the <i>Modules</i> application.</p>