The CMS.IO library allows you to customize Kentico to support a file system of your choice. As described in the Overview page, you can achieve this by developing a custom provider based on the classes contained within CMS.IO.

Preparation

To implement a custom file system provider, you need to add a new assembly to the Kentico solution:

1. Open your Kentico solution in Visual Studio.
2. Add a new **Class Library** project to the Kentico solution, for example named *CustomFileSystemProvider*.
3. Add references to the required Kentico libraries (DLLs) for the new project:
    a. Right-click the project and select **Add -> Reference**.
    b. Select the **Browse** tab of the **Reference manager** dialog, click **Browse** and navigate to the ***Lib*** folder of your Kentico web project.
    c. Add references to the following libraries:
        - **CMS.Base.dll**
        - **CMS.Core.dll**
        - **CMS.DataEngine.dll**
        - **CMS.Helpers.dll**
        - **CMS.IO.dll**
4. Reference the *CustomFileSystemProvider* project from the Kentico web project *(CMSApp* or *CMS)*.

The Kentico installation directory contains sample definitions of all classes required to implement a generic file system provider. To help with the implementation, you can copy these classes into the custom file system provider's project:

1. Open your Kentico program files directory (by default *C:\Program Files\Kentico\<version>*).
2. Expand the *CodeSamples\CustomizationSamples\CustomFileSystemProvider* subfolder.
3. Copy all contained files into the *CustomFileSystemProvider* directory in your web project**.**
4. Include the new files into the *CustomFileSystemProvider* project in Visual Studio:
    a. Expand the *CustomFileSystemProvider* project in the Solution Explorer.
    b. Click **Show all files** at the top of the Solution Explorer.
    c. Select the new files while holding the **Ctrl** key.
    d. Right-click one of the files and select **Include in Project**.
5. Edit the copied files and rename the namespaces to **exactly** match the assembly name of your custom project (*CustomFileSystemProvider* in this example).

Implementation

If you choose to take advantage of the prepared classes, go through all files contained in the project and replace the **NotImplementedExceptions** inside methods with your implementation, and implement property and method overrides. You can consult the following steps for reference or if you wish to create a provider from scratch.

1. Create two separate classes that inherit from the following abstract classes:
    - CMS.IO.AbstractDirectory
    - CMS.IO.AbstractFile
2. Implement all methods defined in the abstract classes.
3. Create three other classes that inherit from the following classes:
    - CMS.IO.DirectoryInfo
    - CMS.IO.FileInfo
    - CMS.IO.FileStream
4. Override all methods and properties from those classes.
5. Create constructors for the classes listed in step 3 according to the following table:

Inherits from	Constructors
CMS.IO.DirectoryInfo	public DirectoryInfo(string path)

CMS.IO.FileInfo	public FileInfo(string filename)
CMS.IO. FileStream	public FileStream(string path, CMS.IO.FileMode mode) public FileStream(string path, CMS.IO.FileMode mode, CMS.IO.FileAccess access) public FileStream(string path, CMS.IO.FileMode mode, CMS.IO.FileAccess access, CMS.IO. FileShare share) public FileStream(string path, CMS.IO.FileMode mode, CMS.IO.FileAccess access, CMS.IO. FileShare share, int bSize)

Note: The custom file system provider classes must be placed into a namespace that **exactly** matches the name of the given assembly.

Configuration

Perform the following configuration steps to start using your custom file system provider:

1. Add the **CMSStorageProviderAssembly** key to the *appSettings* section of your project's **web.config** file. Set the key's value to the assembly name of your custom provider.
2. Configure the project to use the provider. Choose between the following options:

   - Add the **CMSExternalStorageName** key to the *appSettings* section of the web.config file and set the value to any identifier. This maps the project's ***entire*** file system to the custom provider.

     – OR –

   - Use the API to create an instance of the **CMS.IO.StorageProvider** class and map specific project folders to your custom provider.

Examples

- The following web.config keys configure the application to store its ***entire*** file system using a custom provider implemented in the "CustomFileSystemProvider" assembly with "custom" as the identifier:

```
<appSettings>
...
  <add key="CMSStorageProviderAssembly" value="CustomFileSystemProvider" />
  <add key="CMSExternalStorageName" value="custom" />
...
</appSettings>
```

- The following code registers a module that maps a site's media library folder to a custom storage provider:

```
using CMS;
using CMS.Base;
using CMS.DataEngine;
using CMS.IO;

// Registers the custom module into the system
[assembly: RegisterModule(typeof(CustomInitializationModule))]

public class CustomInitializationModule : Module
{
        // Module class constructor, the system registers the module under the
name "CustomInit"
        public CustomInitializationModule()
                : base("CustomInit")
        {
        }

        // Contains initialization code that is executed when the application
starts
        protected override void OnInit()
        {
                base.OnInit();

        // Creates a new StorageProvider instance using the custom
'CustomFileSystemProvider' assembly
        var customMediaProvider = new StorageProvider("custom",
"CustomFileSystemProvider");

                // Maps a directory to the provider
                StorageHelper.MapStoragePath("~/MySite/Media/",
customMediaProvider);
        }
}
```

To create an instance of the *StorageProvider* class for your custom file system provider, call the constructor with the following parameters:

1. The provider's external storage name. Can be an empty string for providers that use the local file system, or any string except *azure* or *amazon* for external providers. The value is assigned to the *ExternalStorageName* property of the StorageProvider class.
2. The name of the code assembly containing the provider's implementation.
3. (Optional) A *bool* parameter that specifies whether the storage is shared.

For more information about storage provider mapping options, see: Configuring file system providers