CMSImport PRO
User manual
Version 9.0

1 Introduction

CMSImport PRO helps you import content, members or dictionary items from any datasource into Umbraco. The following data sources are supported by default:

- BlogML
- CSV
- Excel File
- JSON File
- RSS Feed
- SQL Server
- WordPress
- XML

CMSImport PRO allows you to save wizard steps so you can run the import later or even schedule it for a certain date and time. When you re-run an import already existing records will be updated and only new records will be added. When media is imported references in content or member data will be updated automatically.

And best of all with CMSImport Pro it's possible to import complete content structures also! This allows you to import a complete product catalog (Categories and Products), blogposts + comments, or any structure you want to import.

This document describes PRO features only. The free edition is limited in functionality.
IMPORTANT:
This version of CMSImport is compatible with **Umbraco V9.0 and newer**(support for older versions of Umbraco can be found on our website). CMSImport is compatible with SQL Server (Express).

CMSImport uses HTML Agility pack which is licensed under MS-PL License.
https://html-agility-pack.net/
CMSImport uses Lumenworks Framework IO which is licensed under MIT license
http://www.codeproject.com/Articles/9258/A-Fast-CSV-Reader
CMSImport uses CLosedXML for reading Excel files which is licensed under MIT license
https://github.com/ClosedXML/ClosedXML

2 Installation

Install CMSImport by using Nuget.

dotnet add package CMSImport

Once the package is installed you have an section called Import. You might need a page refresh, or even logout and in again to see this folder.

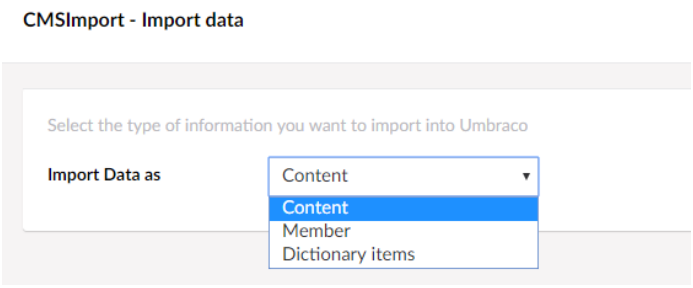
When you want to use the PRO functionality add the license file to the bin folder of your Umbraco install, or upload it via the Dashboard in the Import section.

3 Import

You can go to the import section, open CMSImport and select “import data”. The following wizard will open.

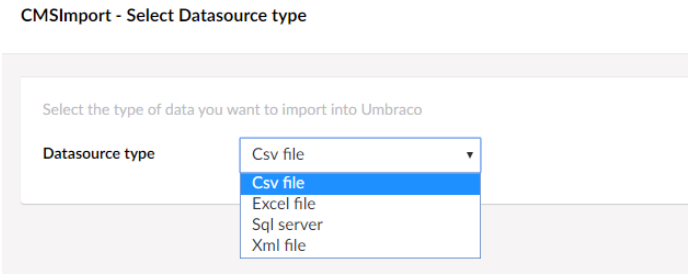
3.1 Select import type

In the first step you need to select what to import.



3.2 Select Datasource Type

In this step you are asked to specify the type of data you want to import.



The following datasource types are supported by default[1]:

- BlogML
- CSV
- JSON File
- Excel File
- RSS Feed
- SQL Server
- WordPress
- XML

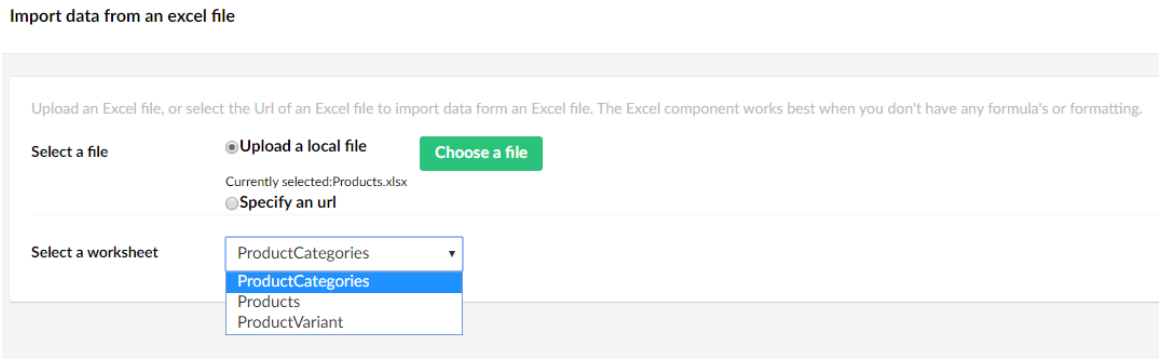
It is possible to create extra datasource types. See chapter extend CMSImport for more information.

3.3 Select datasource

In the select datasource step you need to provide the information for the selected datasource Type. Options can be different for each datasource. In general you specify a file, url, or conection string so CMSImport can retrieve the data. Excel

[1] BlogML,WordPress, RSS feed and JSOn File require that you install additional packages containing the Data providers.

In case of an Excel sheet you can upload a xls(x) file, or point to a location. When you **click next** you can select the worksheet from the pull-down menu.

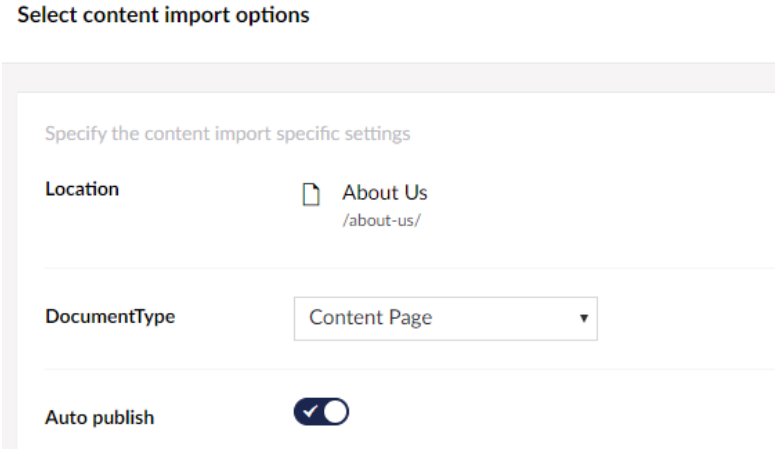


3.4 Set options for Content import

Specify all options specific for content imports

3.4.1 Default options

Specify the location where you want to store the imported documents. You can also specify the document type and you select the auto publish checkbox, when checked items are published automatically.

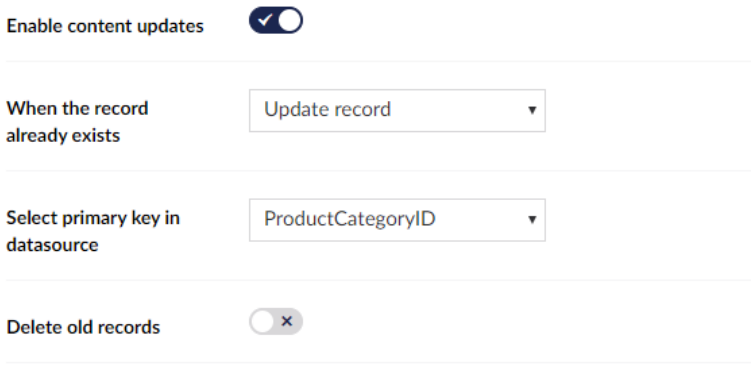


3.4.2 Update options

The “When the item already exists” and the primary key option are needed for content updates. With the “When the item already exists” option you specify what to do when an item is already imported. Possible options are skip and update record. With the primary key you specify the key in the datasource. This field will be used to determine if an item is already imported.

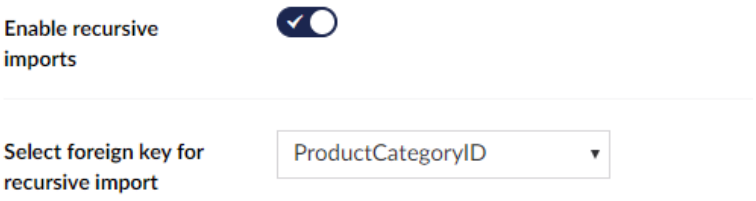
To disable content updates uncheck the “Enable content updates” option. **Only do this when you don’t have a primary key in your datasource**. No relation between imported data and Umbraco document is stored so even when you run the import for the second time data will be imported as new records.

Select Delete old records when you want to remove



3.4.3 Recursive options

CMSImport can maintain the structure for you when you import content. This is normally be done using parent/child import definitions but sometimes you import content with a recursive foreign key to itself, for example when you import Product categories as in the example below. Some categories have a relation to a parent category. By selecting the recursive import option you can specify the key to its parent, in this case ParentProductcategoryID.



Some of the options might be disabled, or you might have a few extra options when run this step as a child import. See section structured import for more information

3.4.4 Delete old records

When selected CMSImport will delete all records that are no longer in the current data source. This will be checked during import and allows you to remove old records automatically. The check will be based on Data source type, primary key name and primary key value. Make sure this is unique when you have multiple definitions otherwise records from a different data source might be deleted.

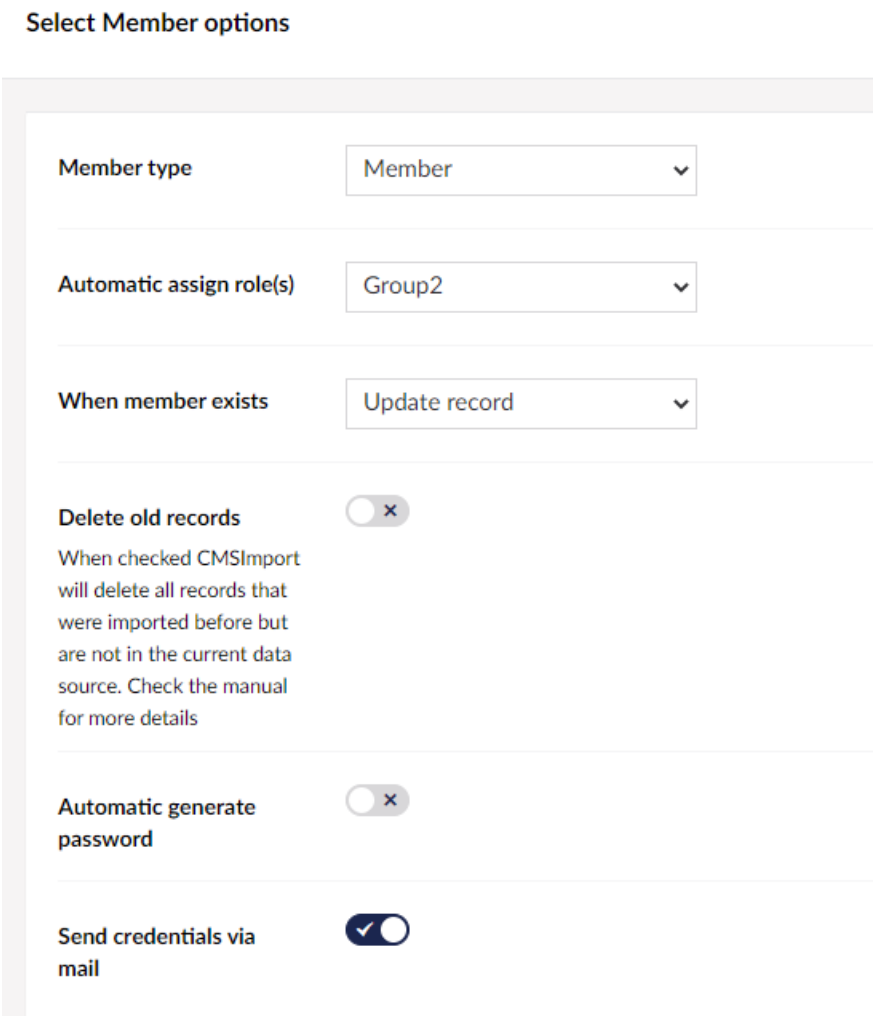
3.5 Set options for member import

For member import you can select the member type and assign a role. With the “When the item already exists” option you specify what to do when an item is already imported. Possible options are skip and update record.

When the “Automatic generate password” option is checked a password is automatically generated for the imported member. When the “Send credentials via mail” is checked an email with login credentials is send to the imported member. You can edit the email template , check chapter settings on how to do this.

When delete old records is selected CMSImport will delete all records that are no longer in the current data source. This will be checked during import and allows you to remove old records automatically. The check will be based on Data source type, primary key name and primary key

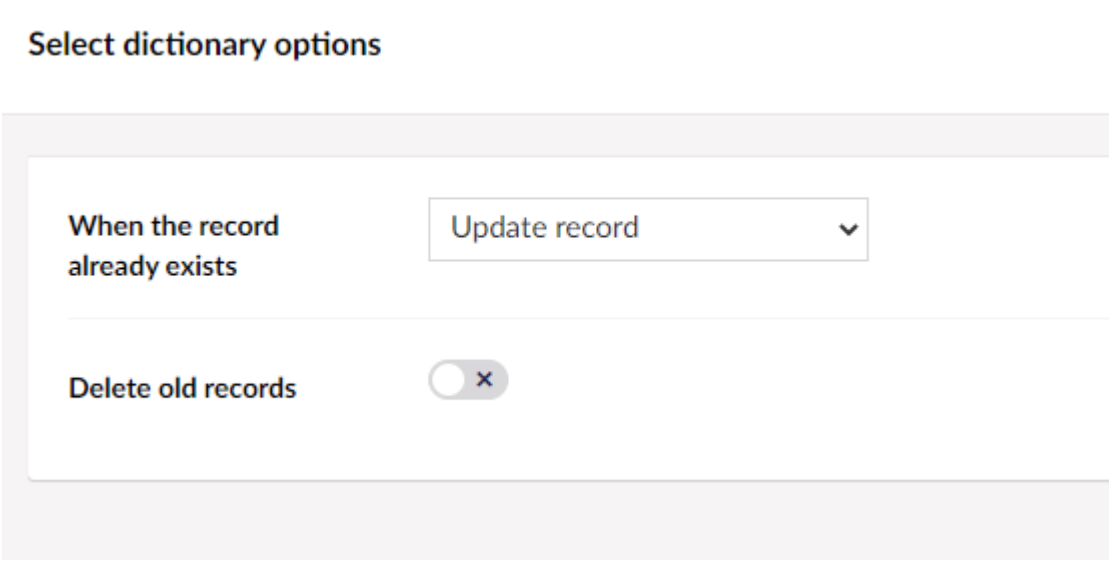
value. Make sure this is unique when you have multiple definitions otherwise records from a different data source might be deleted.



3.6 Select dictionary options

In this step you can select what to do when a record already exists, update or skip.

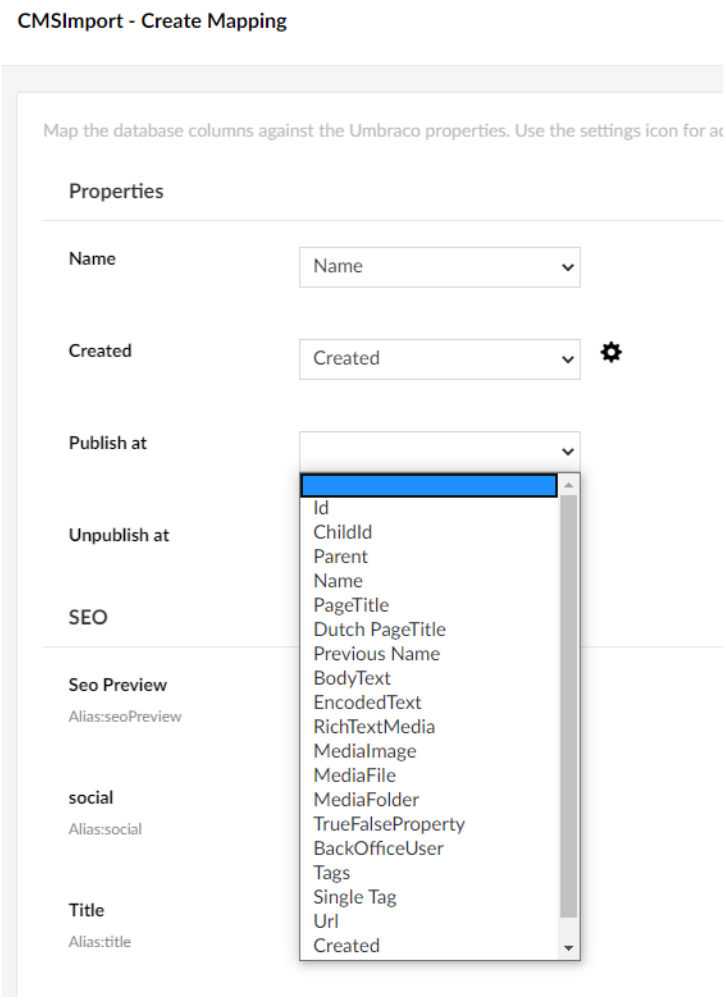
When delete old records is selected CMSImport will delete all records that are no longer in the current data source. This will be checked during import and allows you to remove old records automatically. The check will be based on Data source type, primary key name and primary key value. Make sure this is unique when you have multiple definitions otherwise records from a different data source might be deleted.



3.7 Create mapping

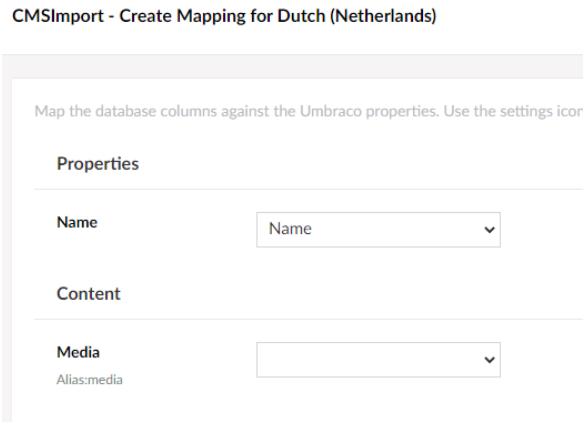
In this step you can specify the mapping between the fields from the data source and the properties of the Umbraco document type.

Quick tip:
When your fieldnames from the datasource are the same as the alias of the document property CMSImport will automatically map this field.



3.8 Multi lingual mapping

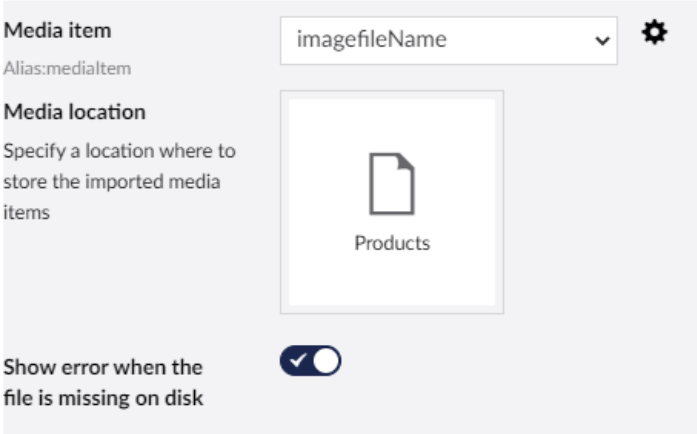
In case the Umbraco document type support Multi lingual Variants the mapping will be a multi-step process. Each step will mention the locale you are mapping against and will only show the allowed properties that can be used for Multi lingual variants.



3.9 Additional settings

Every datatype has its own options. Using CMSImport PRO it's also possible to import media related to content the only requirement is that media is stored in the root of the website. Whenever you map against a media picker or rich text editor screen it's possible for you to specify a media location and it's even possible to specify a default value;

See chapter "Related media import" for more information.



3.10 Confirm

In this step you can validate the selected options one more time. When you click next the import will start.

CMSImport - Confirm

Please validate your input and press Next to import the data.

Datasource options

Select a file	Products.xlsx
Select a worksheet	Products

Import options

Select parent relation column	ProductCategoryID
DocumentType	Content Page
Auto publish	Yes
Enable content updates	Yes
When the record already exists	Update record
Select primary key in datasource	productID
Delete old records	No
Enable recursive imports	No

Mapping

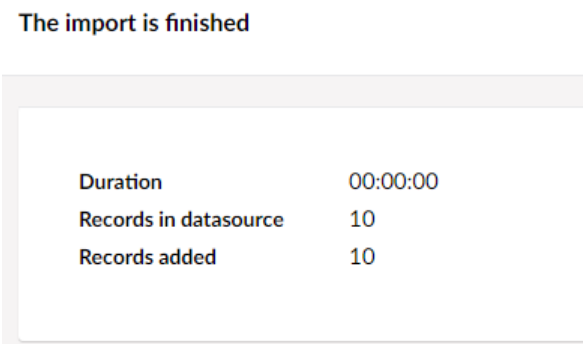
Name	ProductName
Page Title	ProductName
Media item	imagefileName

Additional settings

Media location	Products
Show error when the file is missing on disk	Yes

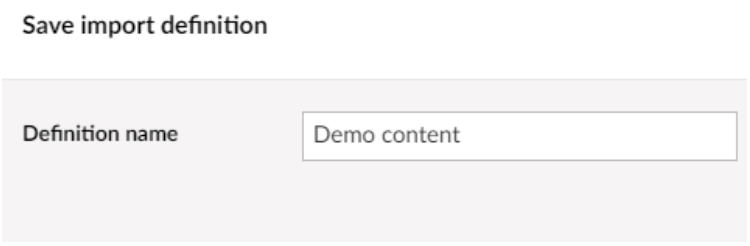
3.11 Import

When you click next in this step the import starts. When the import is finished it will report what it did. If there were any errors it will also report the errors.

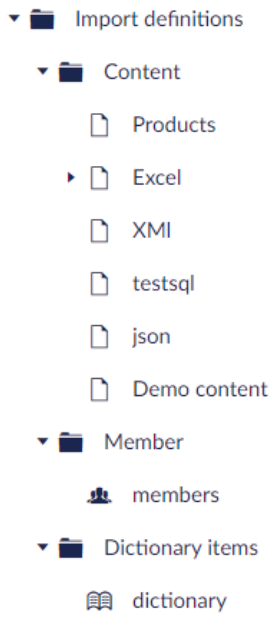


3.12 Save Import steps

When you click save, you can specify a name and when hitting the save button again the import steps are saved for later use. Create copy will create a copy of the current item.

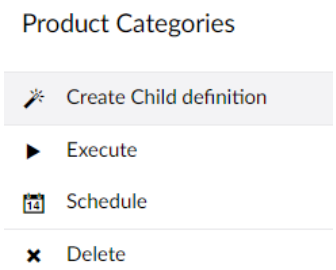


Saved imports are stored as Import definitions and can be found in the Import definitions tree.



4 Structured content import

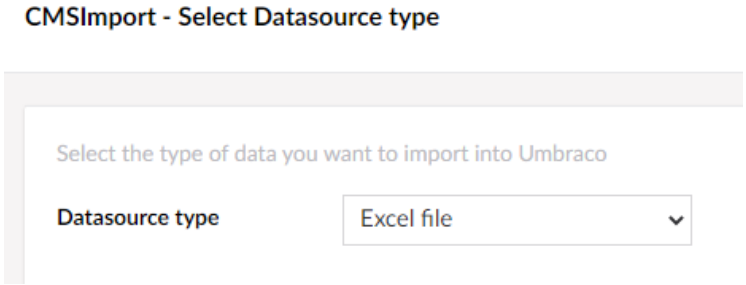
When you want to use structured imports it's possible to create a child definition. In the previous example we were importing Product categories. Now we can import products for those categories. Open the context menu on the parent definition (in this case ProductCategory) and select **Create child definition**



This will start the import wizard again, with a few small changes.

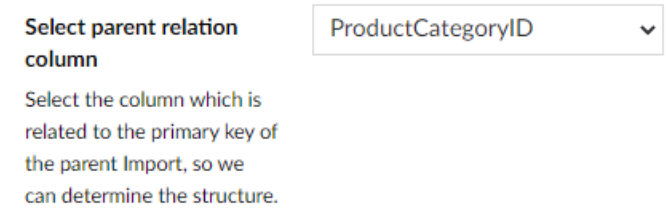
4.1.1 Select datasource

When selecting the same datasource type as the parent it will automatically select the same datasource as its parent.



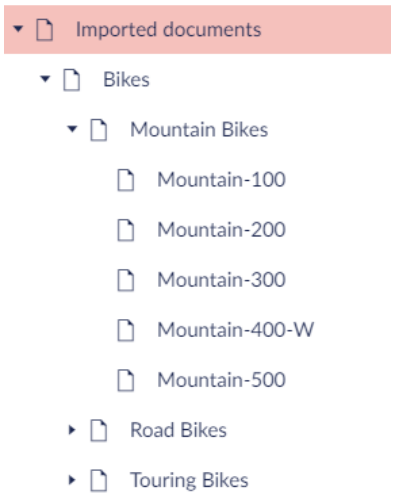
4.1.2 Specify content import options

When you set the content import options you don't need to specify the location since that will be determined based on the parent record. Instead you specify the **parent relation key.** In this case we want to add a relation based on category id so we specify ProductCategoryId as the relation. All other steps are the same.



4.2 End result

When importing these two definitions the content tree is filled with categories and products.



The import definition tree contains two items ProductCategory and Products as the child import definition

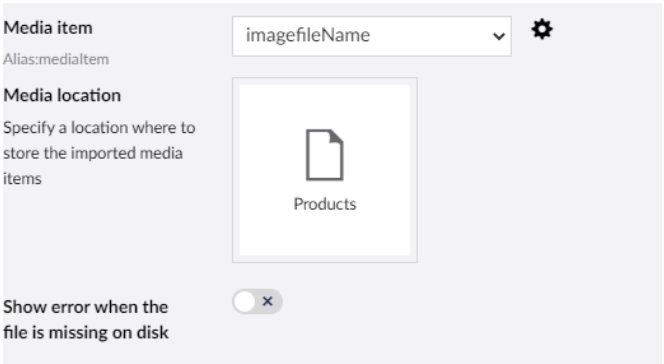
5 Related media import

CMSImport can import media also. This isn't a separate import process but integrated in content or media import. When CMSImport finds a reference to a relative path it will try to get the item and convert it to a media item, or store it in the media folder in case of an upload field**.**
The only required thing is that the original media folder is copied to the root, or configured media directory of your Umbraco folder.

When the relative path in the datasource is a folder CMSImport will import the complete folder and assign the folder id to the mediapicker when assigning a folder is possible with the datatype.

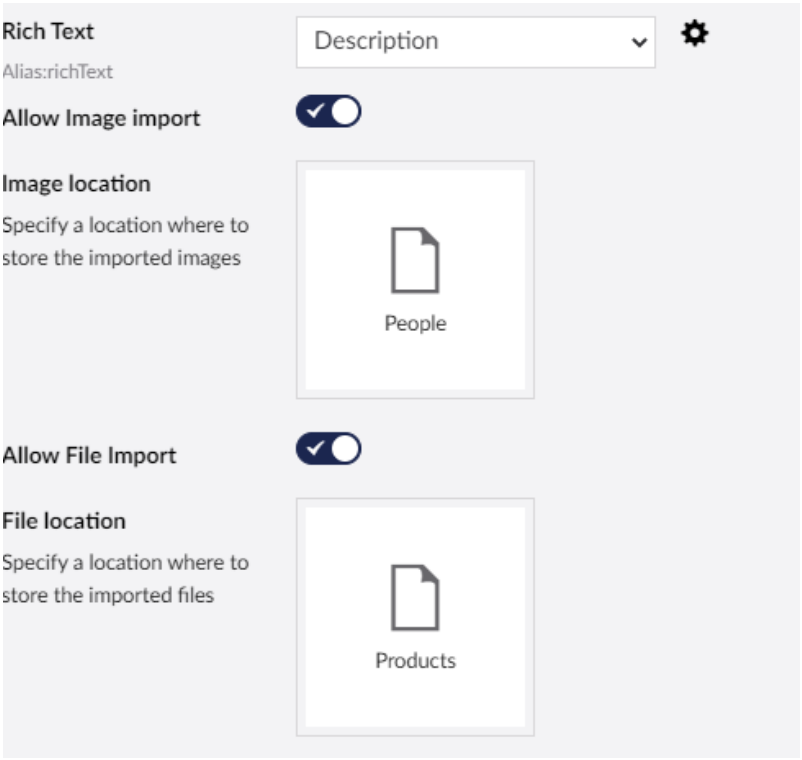
In the example below the img folder of the original site containing two images is stored in the Umbraco root.

Use the advance settings icon on the media picker to set the media location



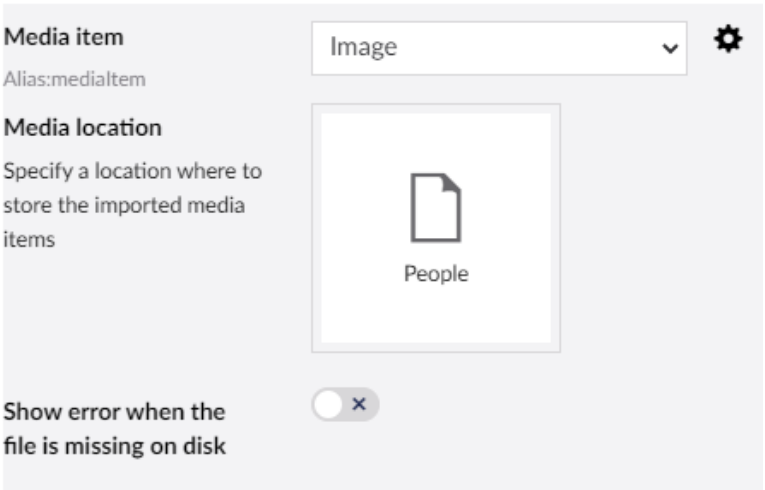
5.1 Rich text editor

When you map against a Richtext datatype you can specify the import options as shown below. When a reference to an image is found in the content, CMSImport will create a media Item and update the image source to the new Media item.



5.2 Media picker/ Multimedia picker

When you map against a mediapicker datatype you can specify the import options as shown below and CMSImport will create a media item and store the Id of the media item. When you set “Show error when file is missing on disk” an error will be displayed when the file is referenced in the data source but not on disk.



5.3 Upload, or cropper field

When an image (could also be a file) reference is mapped against an upload field. CMSImport will store the image in the Umbraco Media folder on disk automatically and update the reference in the Upload field

5.4 Supported Media types

Currently this media import process will work for the following datatypes:

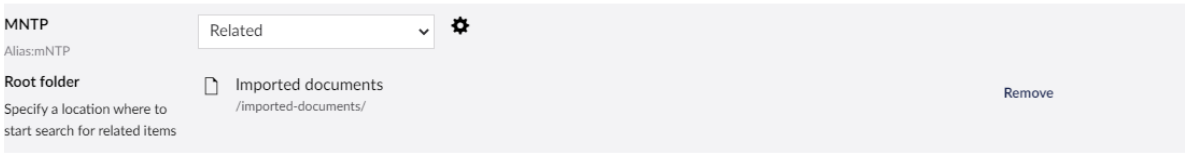
- Upload field
- (multiple) Media Pickers
- Multi Node Tree Picker (Media only)
- Cropper

6 Lookups

CMSImport support content lookups for Multi Node Tree picker values.

6.1 Configuration

CMSImport uses the datatype configuration to determine where to look for the nodes.



Based on the configuration it will look for nodes that it can map against the assigned value. In the image below we imported related record **Bikes**.

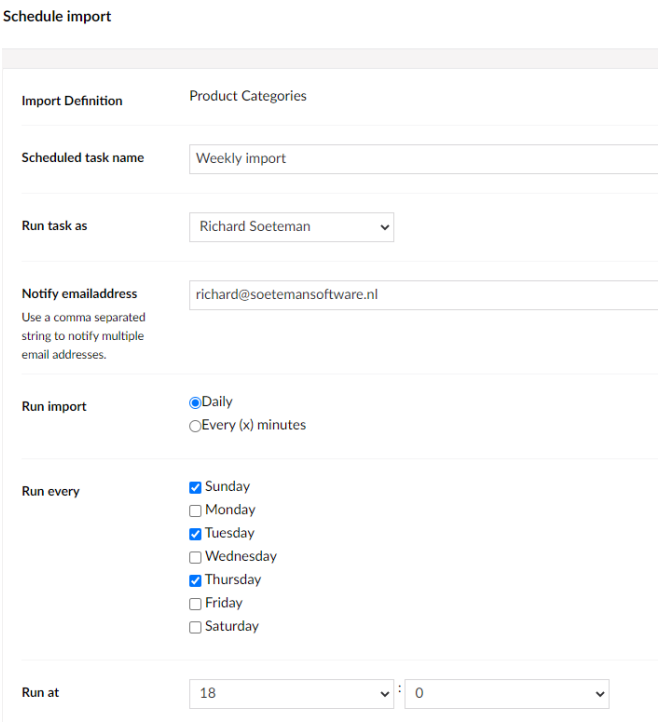
For this record the datasource value contains “Hitch Rack - 4-Bike”

CMSImport will try to map the values based on Id, or NodeName. In case the value cannot be found the value will be ignored.

When an item is pointing to a page that is not yet imported the relation will not be inserted the first time. An extra import to update the relations is required in that case.

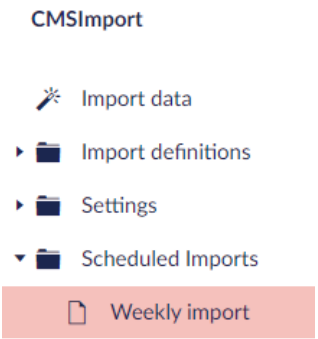
7 Schedule Imports

With CMSImport it's possible to run imports for on a certain day/time. When you right click on a saved import item (content/member) and click schedule the following screen will show up.



You can schedule imports to run every week on certain days/time, every day on a certain time or every hour.

You can specify an email address to receive a notification when the import is finished and you can specify a user that will be assigned as creator for imported items. When you click save the item will run on the selected day/time.



When the import is finished you will receive a notification email on the specified email address.

7.1 Scheduled task log

When you right click on a scheduled task you can view when the task did execute. As you already can see in the image below the scheduled time can be a little different from the real executed time. This can be caused by the website not being up at the given time. The scheduler will always execute the task when the site is back up and did not run the task.

8 Settings

CMSImport comes with a default config. In case you need to want to update the config you can add the following json to the appsettings.json section.

```
{
   "CmsImportConfig":{
      "MediaConfig":{
         "MediaImportLocation":"/",
         "AllowedFileExtensions":[
            ".doc",
            ".docx",
            ".pdf",
            ".ppt",
            ".pptx",
            ".rar",
            ".xls",
            ".xlsx",
            ".zip"
         ],
         "AllowedDomains":[
            "some domain to parse here"
         ],
         "MediaImportKeepFolderStructure":true,
         "MediaImportFileTypeAlias":"File",
         "MediaImportFolderTypeAlias":"Folder",
         "MediaImportImageTypeAlias":"Image"
      },
      "IgnoredPropertyAliasses":[
         "umbracoMemberFailedPasswordAttempts",
         "umbracoMemberLastLockoutDate",
         "umbracoMemberLastLogin",
         "umbracoMemberLastPasswordChangeDate",
         "umbracoMemberFailedPasswordAttempts"
      ],
      "LogDebugInfo":false
   },
   "LoginCredentialsMailConfig":{
      "FromAddress":"robot@cmsimport.com",
      "Subject":"Your account is ready",
      "ViewLocation":"~/App_Plugins/cmsimport/config/loginmail.cshtml"
   },
   "ScheduledTaskMailConfig":{
      "FromAddress":"robot@cmsimport.com",
      "Subject":"Scheduled task executed",
      "ViewLocation":"~/App_Plugins/cmsimport/config/scheduledtaskmail.cshtml"
   }
}
```

8.1 LoginCredentialsMailconfig

These settings will be used when an email with login credentials is send to an imported member.
You can specify:

- The from email address
- The email subject
- The email Razor template[2]

[2] Can be found on disk as view App_Plugins/cmsimport/config/loginmail.cshtml

In the email body you can use the CMSImport.Core.Models.Mail.LoginMail model that contains the following

Model	Description
LoginName	The loginname of the imported member
Password	The password (NOT ENCRYPTED) of the imported member
MemberName	The fullname of the imported member
Properties	All the properties for the member as a dictionary of <string,object>.

8.2 ScheduledTaskMailConfig

These settings will be used when an email is send to inform a user that a scheduled task is finished. You can specify:

- The from email address
- The email subject
- The email razor template [3]

In the email body you can use the CMSImport.Core.Models.Mail.ScheduledTaskMail model that contains the following:

Snippet	Description
Taskname	The name of the scheduled task
RecordCount	The amount of records in the datasource
RecordsAdded	The amount of records added during the import process
RecordsUpdated	The amount of records updated during the import process
RecorsdSkipped	The amount of records skipped during the import process
Errors	The amount of errors during the import process
ErrorMessages	The error descriptions that occurred during the import process

[3] Can be found on disk as view App_Plugins/cmsimport/config/ scheduledtaskmail.cshtml

8.3 Mediaconfig

8.3.1 MediaImportLocation

By default all media is expected in the www root of your Umbraco installation. If for some reason you want to store it in a subfolder you can specify this in the Import path.

8.3.2 MediaImportKeepFolderStructure

By default CMSImport respects the folder structure if you want to import all into a single folder just set MediaImportKeepFolderStructure to false.

8.3.3 AllowedFileExtensions

Specify which file extensions CMSImport needs to parse in case of media import mapped against a Rich Text editor. Only the selected file extensions will be picked up to import.

8.3.4 AlloweDomains

You can also specify some domains url's that needs to be picked up. Url's starting with that domain will also be picked up. You still need to make sure files are located in the root of your Umbraco install.

8.3.5 Media type settings

By default CMSImport is using the standard media types for Images, files and folders. You can change it by mapping the correct alias to the MediaImportFileTypeAlias, or MediaImportFolderTypeAlias, or MediaImportImageTypeAlias properties.

9 Extend CMSImport

Although you don't need to, there are several ways to extend CMSImport. This chapter describes how you can make use of these extension points in your code.

9.1 Setting up Visual Studio

When you want to create an extension for CMSImport you can create a new Class Library.

Install the following NuGet Packages:

- UmbracoCms.Web
- CMSImport.Core

9.2 Dependency Injection

You can use the same Dependency Injection Principles as you would use in other Umbraco V8 extensions. Just inject your dependencies via Constructor
https://our.umbraco.com/documentation/reference/using-ioc/

9.3 FieldProvider

A FieldProvider is used in CMSImport to transform a value from the datasource to the value Umbraco expects without any user interface. You can create your own FieldProvider by implementing the CMSImport.Providers.FieldProviders.IFieldProvider interface. And Decorate the class with one or more FieldProvider attributes. In the FieldProvider attribute you specify the property editor alias so CMSimport only parses the value when the assigned property alias matches the alias specified in the FieldProvider.

9.3.1 Sample

The sample below converts an EmailAddress value to the real member UDI for the Umbraco.MemmberPicker property Editor. First the MemberService gets injected via the constructor. Then In Parse method the value will be used to get the user object based on the value and from that the key will be returned as UDI.

```
    [FieldProvider(PropertyEditorAlias = "Umbraco.MemberPicker")]
    public class MemberPickerFieldProvider :IFieldProvider
    {
        private readonly IMemberService _memberService;
        public MemberPickerFieldProvider(IMemberService memberService)
        {
            _memberService = memberService;
        }

        public object Parse(object value, ImportPropertyInfo property,
FieldProviderOptions fieldProviderOptions)
        {
            var emailAddress = value.AsString();
            if (!string.IsNullOrWhiteSpace(emailAddress))
            {
                var user = _memberService.GetByEmail(emailAddress);
                if (user != null)
                {
                    value = UdiHelper.ParseUdi(user.Key, "member");
                }
            }
```

```
            return value;
        }
    }
```

9.4 AdvancedSettingProvider

The advanced Settings provider is the same as a FieldProvider, only difference is that you can add UI to specify some settings during mapping.

9.4.1 Sample

In the sample below we Convert a DateTime (string) field from the import file to the correct DateTime object Umbraco Expects.

9.4.1.1 Advanced Settings UI

To create UI for the Advanced Settings provider you need to implement a class that derives from the IProviderOptions interface. In the sample Below we created UI for the DateTime Advanced settings provider. In this sample we create UI that allows the user to specify a format that will be used to transform the value from datasource to a real DateTime object.

To make a connection between the Provider and UI you need to specify the ProviderAlias. Renderfield can be used to enable/disable certain options (fieldAlias is property name as lowercase). In the sample below we specify the DateTimeFormat and Decorate the string with the DateFormatField Attribute which will generate the UI for the property. See Section UI Fields for a complete list of all attributes.

```
public class AdvancedDateSettingsOptions :IProviderOptions
    {
        /// <summary>
        /// Gets/Sets the datetimeformat we can use to parse the data.
        /// </summary>
        [DateFormatField(Name = "Date time format", Description = "Specify the format
the import uses to parse the date/time")]
        public string DateTimeFormat { get; set; }

        public string Alias => “AdvancedDateSettingsProvider”;

        public bool RenderField(string fieldAlias)
        {
            return true;
        }
    }
```

9.4.1.2 Andvanced Settings provider

The Advanced Settings provider will convert the value to the correct Date Time object using the specified Format. The Provider needs to derive from AdvancedSettingProvider class. The class needs to be decorated With the AdvancedSettingsProvider attribute where you specify the Alias and the Supported Umbraco Property Editor(s)

Parse will be used in the importProcess to convert the data to the desired format Umbraco expects.

- Value contains the value from the Datasource
- Options contains the configuration for the Advanced Setting Provider (Cast to concrete type to read all values)
- ImportOptions contains the options from the Importprovider

Validate allows you to validate the selected import options and return a list of errormessages.

GetAdvancedSettingProviderOptions allows you to return a new UI Provider Object

```
[AdvancedSettingsProvider(Alias = "AdvancedDateSettingsProvider",
SupportedPropertyEditorAliasses = "Umbraco.DateTimePicker")]
    public class AdvancedDateSettingsProvider : AdvancedSettingProvider
    {
        public override object Parse(object value, IProviderOptions options,
ImportOptions importOptions)
        {
            var s = value.AsString();
            var dateParseOptions = options as AdvancedDateSettingsOptions;
            if (!string.IsNullOrWhiteSpace(dateParseOptions.DateTimeFormat))
            {
                if (DateTime.TryParseExact(string.Format("{0}", value),
dateParseOptions.DateTimeFormat,
                    CultureInfo.CurrentUICulture, DateTimeStyles.None, out var dt))
                {
                    value = dt;
                }
            }

            return value;
        }

        public override IEnumerable<string> Validate(IProviderOptions options)
        {
            var result = new List<string>();
            var dateOptions = options as AdvancedDateSettingsOptions;

            if (string.IsNullOrWhiteSpace(dateOptions.DateTimeFormat))
            {

result.Add(TranslationHelper.Localize("mapping_dateTimeFormatRequired"));
            }
            return result;
        }

        public override IProviderOptions
GetAdvancedSettingProviderOptions(ImportPropertyInfo importPropertyInfo)
        {
            return new AdvancedDateSettingsOptions();
        }
    }
```

9.5 DataProvider

The Dataprovider allows you to upload/ select a datasource and return the result in a datareader so CMSImport can use the data from the datareader to import the data. The Dataprovider needs two classes, the actual provider and the Provider UI. In the sample below we use a Dataprovider to load RSS feed data.

9.5.1 DataProvider class

GetData Allows you to return a Datareader based on the given Provideroptions that come from the UI. OnlyFirstRow parameter is used when CMSImport needs column info.

Validate allows you to validate the selected Data options and return a list of errormessages.

GetDataProviderOptions allows you to return a new Dataprovider UI object.

```
[DataProvider(Alias = “Rss File”, Name = “RSS File”,ContentType = “Content”, Title =
"RSS Demo", Intro = "Demo provider for RSS feed")]
    public  class RssFeedDataProvider : DataProvider
    {
        public override IDataReader GetData(IProviderOptions dataProviderOptions, bool
onlyFirstRow = false)
        {
            var provider = dataProviderOptions as RSSFeedDataProviderOptions;
            return
XmlDataHelper.XmlToDataReader(provider.Datasource.GetDataFromFileOrUrl(),false,
"//item");
        }

        public override ValidationResult Validate(IProviderOptions providerOptions)
        {
            var result = new ValidationResult();
            try
            {
                using (var datareader = GetData(providerOptions))
                {
                    if (!datareader.Read())
                    {
                        result.ErrorMessages.Add(“No data in file”);
                    }
                }

            }
            catch (Exception ex)
            {
                result.ErrorMessages.Add($"Error validating the Rss feed:
{ex.Message}");
            }

            return result;
        }

        public override IProviderOptions GetDataProviderOptions()
        {
            return  new RSSFeedDataProviderOptions();
        }
    }
```

9.5.2 Dataprovider UI

To create UI for the Data provider you need to implement a class that derives from the IProviderOptions interface. In the sample Below we created UI for the RSS Data provider. In this sample we create UI that allows the user to Upload, or select a Url Containing the RSS Data.

To make a connection between the Provider and UI you need to specify the **ProviderAlias**. **Renderfield** can be used to enable/disable certain options (fieldAlias is property name as lowercase).

In the sample below we specify the FileOrUrlPickerModel and Decorate the Property with the DataSourcePicker Attribute which will generate the UI for the property. See Section UI Fields for a complete list of all attributes.

```
public class RSSFeedDataProviderOptions : IProviderOptions
    {
        public RSSFeedDataProviderOptions()
        {
            Datasource = new FileOrUrlPickerModel();
        }

        [DatasourcePicker(Name = "Select RSS File")]
        [FileModelValidation]
        public FileOrUrlPickerModel Datasource { get; set; }

        public string Alias => “Rss File”;
        public bool RenderField(string fieldAlias)
        {
            return true;
        }
    }
```

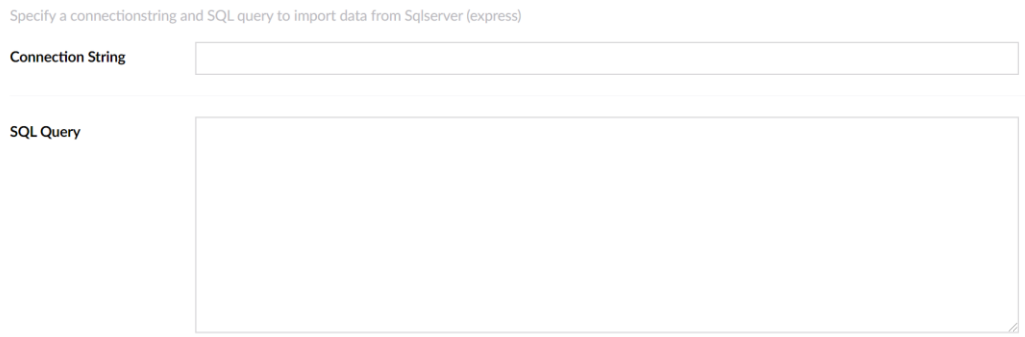
9.6 Possible UI Fields

As seen in the Advanced Settings and Dataprovider section, CmsImport uses Scaffolding to generate the UI for UI properties.

Sample:

In the sample below we have added a Textbox option to the ConnectionString property

```
        [TextBoxField(Name = "[Connectionstring")]
        public string ConnectionString { get; set; }
```



Below the list of all options CMSImport supports:

Attribute	Applies to	Description
BooleanField	Boolean	Allows you to use a Toggle option in the UI
ContentPicker	String	Allows you to use a Comtentpicker in the UI
DatasourcePicker	FileOrUrlPickerModel	Allows you to select, or upload a File in the UI.
DateFormatField	String	Allows you to uses a DateFormat option in the UI.
DropDownField	DropdownModel	Allows you to select an option from a Dropdown in the UI.
LabelField	String	Renders a Label in the UI.
MediaPicker	String	Allows you to use a Media picker in the UI.
MemberGroupOrColumnPickerField	List<String>	Allows you to select mebergroups in the UI.
SmallTextBoxField	String	Allows you to use a small textbox in the UI.
TextBoxField	String	Allows you to use a textbox in the UI.
TextAreaField	String	Allows you to use a textarea in the UI.

9.6.1 Add validation to UI

You can use the default validation attributes to validate Property values.

Sample:

```
[Required(ErrorMessage = "ConnectionString  is required")]
[TextBoxField(Name = "[sqlDataProvider/connectionStringTitle]")]
public string ConnectionString { get; set; }
```

For the Datasourcepicker you need to use **[FileModelValidation]**

9.7 Notifications

Using events it is possible to hook into certain parts of the Import process. The following Events are supported:

Event	Description
BulkImporting	Fired when the complete Import process starts.
BulkImported	Fired when the complete Import process is finished.
Importing	Fired when a single Import definition starts.
Imported	Fired when a single Import definition is finished.
RecordImporting	Fired when a single record starts importing.
RecordImported	Fired when a single record is imported.

9.7.1 Sample

Below the class that logs every event to the Umbraco log file.

First we register our notification handlers in our composer

```
public class CMSImportTestProviderComposer : IComposer     {
        public void Compose(IUmbracoBuilder builder)
        {
            //Content
            builder.AddNotificationHandler<BulkImportingNotification, ContentImportLogEvents>();
            builder.AddNotificationHandler<BulkImportedNotification, ContentImportLogEvents>();
            builder.AddNotificationHandler<ImportingNotification, ContentImportLogEvents>();
            builder.AddNotificationHandler<ImportedNotification, ContentImportLogEvents>();
            builder.AddNotificationHandler<RecordImportingNotification<IContent>,
ContentImportLogEvents>();
            builder.AddNotificationHandler<RecordImportedNotification<IContent>,
ContentImportLogEvents>();
            builder.AddNotificationHandler<RecordSkippedNotification<IContent>,
ContentImportLogEvents>();
        }
    }
```

Then we create a class that can handle the notifications

```
public class ContentImportLogEvents  : INotificationHandler<BulkImportingNotification>,
                                       INotificationHandler<BulkImportedNotification>,
                                       INotificationHandler<ImportingNotification>,
                                       INotificationHandler<ImportedNotification>,

INotificationHandler<RecordSkippedNotification<IContent>>,

INotificationHandler<RecordImportingNotification<IContent>> ,

INotificationHandler<RecordImportedNotification<IContent>>
    {
        private Lazy<ILogger<ContentImportLogEvents>> _logger;
        public ContentImportLogEvents(Lazy<ILogger<ContentImportLogEvents>> logger)
        {
            _logger = logger;
        }
```

```
        public void Handle(BulkImportingNotification notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("BulkImporting content {stateId}",
notification.State.StateId);
            }
        }

        public void Handle(BulkImportedNotification notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("BulkImported content {stateId}",
notification.State.StateId);
            }
        }

        public void Handle(ImportingNotification notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("Importing content {stateId}",
notification.State.StateId);
            }
        }

        public void Handle(ImportedNotification notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("Imported content {stateId}",
notification.State.StateId);
            }
        }

        public void Handle(RecordSkippedNotification<IContent> notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("RecordSkipped content {primaryKey}",
notification.PrimaryKeyValue);
            }
        }

        public void Handle(RecordImportingNotification<IContent> notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("RecordImporting content {primaryKey}",
notification.PrimaryKeyValue);
            }
        }

        public void Handle(RecordImportedNotification<IContent> notification)
        {
            if (notification.ProviderAlias == "ContentImportProvider")
            {
                _logger.Value.LogInformation("RecordImported content {primaryKey}",
notification.PrimaryKeyValue);
            }
        }
    }
}
```

10 Manual Installation/Configuration

If you can't give the installer sufficient rights to create tables in the database you need to install CMSImport (PRO) database tables Manually.

10.1 Manual configuration of Database

Download the SQL script from our website
https://soetemansoftware.nl/downloads/cmsimportv4dto.sql.txt

Rename the file to cmsimport.sql and execute from SQL management studio.

11 Troubleshooting

11.1 I don’t see the CMSImport package in my developer section

Make sure you have sufficient rights to install the package. See chapter 2, otherwise perform a manual installation see chapter9.

11.2 I don’t see my column names when importing from a CSV file.

Make sure that your csv file contains column names

11.3 I get weird column names when importing from a CSV file.

Make sure that you set the correct csv options to display the CSV file. For example, choose **;** as the delimiter and **“** as a string indicator. Also make sure csv files are saved as UTF-8

11.4 Email is not send when importing a member

Make sure you have configured your smtp server in your appsettings.json file. Also check the Umbraco log file for SMTP errors.

11.5 I get an Invalid License exception.

Make sure your license file exists in the bin folder and you’ve bought the correct license. Contact support@soetemansoftware.nl for help.