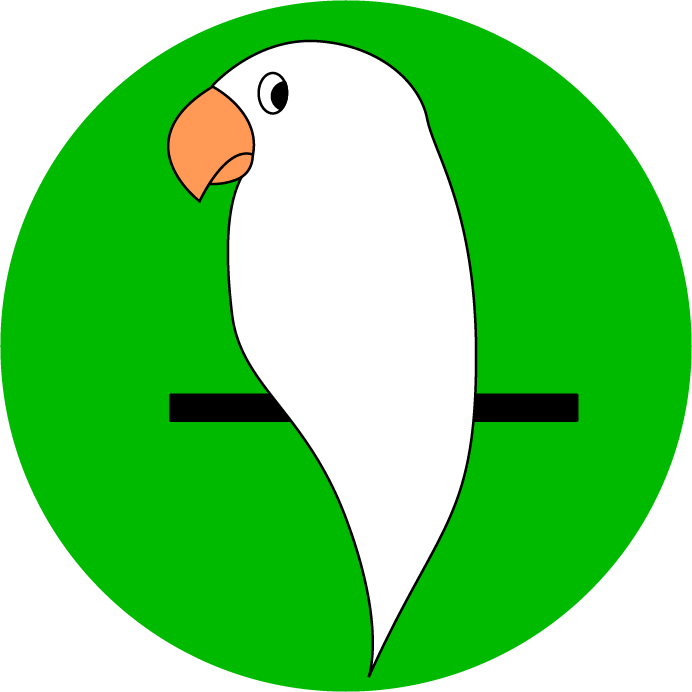
**Polyglot Package for Umbraco**

****

v2.0

by [dimitri](https://our.umbraco.org/member/7174)

[Watch a demonstration](https://youtu.be/yBAIORg9Bpw)

Supported Umbraco Versions 3

Functionality Overview 3

Installation 3

Usage Instructions 5

Creating Translation Document Types 5

Creating Translations 6

An Alternative Way to Work, Using Tabs 6

Adding the Language Selector 7

Setting the Culture 7

Translating to Different Cultures of the Same Language 7

Using Non-Standard Languages and Cultures 8

Further Steps 8

Better SEO with URL Rewriting 9

# Supported Umbraco Versions

Version 2.0 of the package supports Umbraco v7 and up. Previous Umbraco versions (4.5.x - 6.x) are supported by the 1.x version track. Note that, even though Polyglot 1.x comes in two "flavours", one that is based on Razor scripts and another one based on XSLT, Polyglot 2.0 only has one, Razor version. Nevertheless, it works on any kind of Umbraco installation, regardless of the template language selected.

# Functionality Overview

Polyglot provides multilingual support for Umbraco using a 1-1 multilingual site structure. It adds an option to the context menu of the content tree in Umbraco, which allows the automatic creation of translations of a page, placed in a “folder” under that page. A drop down list box, which is also included in the package, can then be used on the front end by visitors to the site to select their preferred language. When a language is selected, the content is retrieved from one of the above-mentioned translations. If the corresponding translation does not exist, or a specific property of a page has not yet been translated, the content is retrieved from the initial page, which contains the default language.

The package also supports creating individual translation properties for each property on a page and placing them on that same page. This way, instead of working with translation folders as mentioned above, languages can be managed via tabs in the back office (one tab per supported language).

The main advantage of using this package, assuming that it is desired, is consistency. The site will have the same structure regardless of the selected language. In the event that a certain piece of information has not been translated at a given time, the visitors to the site will at least be made aware of its existence, since it will be displayed in the default language. Switching between languages is also very easy for the visitor to the site, as he or she can change the language at any time from any part of the site, using the drop down listbox.

# Installation

After installation, the following items will be added to your system:

**Dimi.Polyglot.dll** in bin\: An assembly containing much of the “behind the scenes” functionality of the package

**PropertyTranslation.cshtml** in macroScripts: A corresponding macro will also be created. The macro will be used instead of the Umbraco page fields in the templates, so that the fields entered are translated into the various languages

**PropertyReferenceTranslation.cshtml** in macroScripts: A corresponding macro will also be created. This script is similar to PropertyTranslation.cshtml, and it can be used in other scripts developed for the site. Being given the id of an Umbraco node, the alias of one of its properties and a language code (optionally, otherwise it picks up the language selected by the visitor), it displays the translation of that property or its value from the basic page, if no translation exists. An example of how it works can be found in TranslatedNavigation.cshtml.

**SelectedLanguage.cshtml** in macroScripts: This script renders the ISO code of the language which the user has selected, for example en, fr, de, etc. It is used by the other scripts but it could also be used independently, if needed.

**SetPageCulture.cshtml** in macroScripts: This script sets the culture of the page each time a visitor accesses the site, according to the selected language.

**Translation template**: This is the template that must be assigned to all translation document types. It ensures that the preview button will also work for translations, by redirecting to the default language page, passing the language of the translation as a parameter.

**TranslationRedirect.cshtml** in macroScripts: A corresponding macro will also be created. This is used by the above-mentioned Translation template.

**TranslatedNavigation.cshtml** in macroScripts: A corresponding macro will also be created. Translating the site involves more than just translating the content of each page. Other functionalities need to take translations into consideration as well, and for those some coding might be required. The most common of those is the navigation of the site and this script offers a head-start for that. It creates a “translation aware” navigation list (ul).

**LanguageSelector.ascx** in usercontrols\Dimi.Polyglot: This is the language selection drop down list box that will be used in the front end. A corresponding macro will also be created. It is recommended that the macro is placed in the master page of the site, so that users can always select their desired language.

**LanguageSelectorForWebForm.ascx** in usercontrols\Dimi.Polyglot: A corresponding macro will also be created. This is an alternative to LanguageSelector.ascx. If the pages of the site are structured like Web Forms (meaning that each page is a form in its entirety and the language selector will be placed within the form), it is suggested that this macro is used instead of the simple Language Selector, because the former was made to work independently of web forms and it creates its own form. It is best avoided to place forms within forms when writing html.

**TranslationCreation.aspx** in umbraco\plugins\Dimi.Polyglot: This page is used in the back office by the context menu option that creates the translations of each page.

**Polyglot.config** in config: A configuration file, which can be used to make the package use languages and cultures which are not ISO standard.

# Usage Instructions

## Creating Translation Document Types

After the package has been installed, the first thing to do is to make sure that the languages that will be used have been declared under “Languages” in the Settings section of Umbraco. It is not necessary to have them all declared before starting to use the package. More can be added whenever it is needed, but it is good to have at least two, so that the package can be tested.

Following that, some work needs to be done on infrastructure for each of the document types that will be translatable.

Suppose that one such document type has the **alias** TextPage. First, a document type is needed that will serve as the “folder” for the translations of TextPage. Its name does not matter but its alias has to be TextPageTranslationFolder (notice the difference between Polyglot v2.x and Polyglot v1x: No dashes or underscores are used in v2.x because they are not allowed by Umbraco 7). “Create matching template” should not be checked while creating the document type. TextPage also needs to be configured to allow TextPageTranslationFolder as a child node type. This can be done by clicking on TextPage, selecting the Structure tab and then checking the TextPageTranslationFolder as an allowed child node type and saving.

It is suggested that a true/false property be added to the translation folder, with the alias “umbracoNaviHide”. This will allow the Polyglot package to mark nodes of this document type not to appear in the navigation of the site. If, by convention, another alias is used for such fields on a specific site, that alias can also be used here, but the following key needs to be added to appSettings (in Web.config or Config/appSettings.config, depending on your Umbraco setup):

<add key="PolyglotHideFromNavigationPropertyAlias" value="xxxx"/>

xxxx: the alias used

That is all about the translation folder. It does not need to have any other properties declared.

Following that, the translation document type for the TextPage needs to be created. It is suggested that it be given the alias TextPageTranslation for clarity, but this is not as important as the alias of the translation folder. Again, “Create matching template” should not be checked while creating the document type. All the properties of TextPage, which will be translatable, need to be declared in TextPageTranslation as well, with the exact same aliases which they have in TextPage. An easy way to do this is not to create a new document type for TextPageTranslation but to make a copy of TextPage and then give it the alias TextPageTranslation.

**Most importantly**, an additional property needs to be declared in TextpageTranslation, of type “Label”, with the alias “language” (all lower case). This will hold the ISO code of the language of each translation node. Then, the template “Translation” (installed by the Polyglot package) needs to be assigned as the default template of TextPageTranslation. This same template should be used by all translation pages. On the site, when a language is selected, the parameter lang=xx is added to the URL. The Translation template ensures that the preview of the translations works by redirecting to the main page (TextPage in this case) and adding the lang parameter. Also, TextPageTranslationFolder needs to be configured to allow TextPageTranslation as a child node type. Finally it is suggested that the true/false “umbracoNaviHide” property is declared in TextPageTranslation, just like it has been declared in TextPageTranslationFolder.

## Creating Translations

From here on the system is ready to manage translations of nodes of type TextPage. In the content section, when "... > Do something else" is clicked on a TextPage, there will exist the option "[Polyglot] Create translations" (If this does not appear, logging out and back into Umbraco should solve the problem. Sometimes it may not appear right after the package has been installed). If clicked, a form will appear on the right, with a list of checkboxes for the available language translations to be created. By clicking on the save button, a TextPageTranslationFolder will be created under TextPage and, for each language checked, a TextPageTranslation item will be created in the folder. The content of each translation can then be edited. Of course, each time the "[Polyglot] Create translations" option is clicked, it keeps track of which translation nodes already exist, so that they do not get recreated.

The first language on the list is not checked on the translation creation form. This is because the first language declared in the Settings section is assumed to be the default one (in order to specify another language as the default, the key <add key="uPolyglotDefaultLanguage" value="xx" /> - where xx is the language - can be added to the web site's appSettings). That means that the content in the basic page will be in that language, so there will probably not be a need to create a translation. However, another language could serve as the default language for a certain page. The system is quite flexible so there is no problem if one decides to write German for example in one basic page, while all the others contain English. Then, when an English version of the content is available an English translation node can be created. Of course, in that case, there is no point in creating a German translation.

In cases when the above risks causing confusion, the system can also be configured not to display the default language on the list of checkboxes at all, by entering the following to the appSettings:

<add key="PolyglotHideDefaultLanguageCheckbox" value="true" />

## An Alternative Way to Work, Using Tabs

In the event that using the translation document type as described above is not the desired way to provide translations, there is also another way to do so.

The package also supports creating "translation properties" for each property on a page that is to be translated. So if a property has the **alias**"bodyText" for example and it has to be available in a second language, other than the main one, say, French, a second property of the same type should be created, with the alias "bodyTextfr". If it should also be translated into Italian a third property should be created with the alias bodyTextit.

It is suggested that these properties be given a name that is the same as that of the original property ("Body Text" for example) and placed in separate tabs, one tab per language. For example the default language tab could be named "Content" and the rest "Content (FR)", "Content (IT)", etc.

The rest of the usage instructions apply unchanged. The drawback of using this method is that it takes more work to maintain, and it can make the back office slow if there are too many supported languages. But for say five or six languages it should work without any performance issues, depending of course on the amount of information contained in each page.

The two sections above (*Creating Translation Document Types* and *Creating Translations*) can be ignored if this method is chosen, unless of course a combination of the two methods is used, which is also possible.

## Adding the Language Selector

After following the steps above, the site will contain multilingual content. In order to enable this on the front end, the macro “Language Selector” which has been installed as part of the Polyglot package needs to be entered in a template. The master template of the site would be the most appropriate in most cases.

(NOTE: If the pages of the site are structured like Web Forms, it is suggested that “Language Selector for WebForm” is used instead of “Language Selector”. Please see the Installation section for more information.)

Also, the template of TextPage needs to be changed a bit. For multilingual content, instead of using Umbraco page fields, those should be replaced by the “Translated Property” macro, also installed as part of the Polyglot package.

After following these steps, the site can be tested. By accessing a node of type TextPage in the front end and switching languages using the drop down list box (Language Selector) which has been inserted in the appropriate template, the content will change to reflect the change.

The same operations that have been performed for TextPage can now be performed for the rest of the document types of the site, to provide full multilingual content.

In order to prevent the language ISO codes from appearing next to each language description, if that is preferred, the key <add key="uPolyglotAppendLanguageCodes" value="false" /> can be added to the web site's appSettings.

## Setting the Culture

In addition to offering multilingual content using the facilities offered by the package, the Umbraco Dictionary is also useful for adding localised strings to various templates. A site might also contain other customised items that utilise localisation. In order for these to work, the culture of each page needs to be set according to the selected language when a visitor accesses the site.

This can be done by adding the “Set Page Culture” macro at the top of each template. It is suggested that it is entered right under the <body> tag. If a master template is used, things are much simpler because the macro can just be added there, and it will take effect on all templates under the master template.

## Translating to Different Cultures of the Same Language

Polyglot also supports using multiple cultures per language. In the past, it would only identify each language by its two-character ISO code (e.g. "en", "fr", "de", etc). It now also supports culture identifiers. So for example, it is possible to identify separate languages like en-US (United States English), en-GB (British English), fr-BE (Belgian French), etc.

This allows for translations of the same language in different cultures to be used on a web site. In order to activate this feature, the setting <add key="uPolyglotUseCultureInLanguageCode" value="true" /> needs to be entered into the web site's appSettings. Then, the languages need to be declared under “Languages” in the Settings section of Umbraco, as is the case anyway. Other than that, the package will function just like in the case of using a single culture per language. Note however that, if tabs are used for translations (please see *An Alternative Way to Work, Using Tabs*), the suffix of each property has to include the culture (e.g. bodyTextengb).

Note that the suffix for the culture en-GB, used here as an example, only needs to be entered without a dash in Umbraco aliases, due to Umbraco's not accepting dashes there. When used in the lang=xxxxx URL parameter it needs to have its proper format (en-GB).

It is best if it is decided whether or not this feature will be used before any translations have been created. For switching from a non-culture aware installation to a culture-aware one, there are instructions in the Polyglot 1.x version track documentation, for Umbraco versions prior to 7, but they have not been adapted to the current version at this point.

## Using Non-Standard Languages and Cultures

Polyglot supports the use of language and culture codes that are not ISO standard, hence, not supported by the .NET framework. The needed languages and cultures must be entered in the file Polyglot.config in the config folder of Umbraco. There are examples there of the required syntax, in comments.

This feature has been implemented following certain user requests. It can be very helpful for systems that need to fulfil certain requirements, but it should be used with caution, since the customised codes will not function in the exact same way as the standard ones.

Polyglot will recognise the non-standard entries in the abovementioned configuration file, thus including them in the created translations and in the language selection drop down menus, and changing the language of the front end content as per user selection. However, neither Umbraco, nor the .NET framework will do the same. This means that the Umbraco dictionary will not work with these languages, thus not changing the language of the dictionary items. Other .NET culture-aware controls (e.g. calendars) will also keep being displayed in the same language, regardless of the site visitor's language selection. Of course, it is possible that neither the dictionary nor such .NET controls are used for a web site.

## Further Steps

As mentioned earlier, apart from translating the contents of each page, other items will probably also need translation. Coding would be needed to do this, but some assistance is offered.

One such case is ensuring that the menus of the site also appear in multiple languages. Most sites will have a list containing a set of links for navigation. The Polyglot package also installs a macro called “Translated Navigation” to provide such a navigation list with multi-lingual support. The macro can be added to a template. It accepts the following parameters:

UlCssId: The id that the <ul> (list) will have for style

Level: The level in the content tree from which the navigation will be created.

NaviHideProperty: The alias of the true/false property that indicates if a specific node will be hidden in the navigation. If no value is given for this parameter, it is assumed that the alias is “umbracoNaviHide”.

CurrentItemCssClass: The css class assigned to the <li> (item) which is currently selected. Having a specific class for this allows it to be differentiated from the non-selected items.

TitlePropertyAlias: This is very important. It is the alias of the property of each page, which will serve as the title of the page. This should also be entered into the translation nodes, so that the navigation is actually multilingual.

The “Translated Navigation” macro creates the links to each page contained in the list, using the title translated in the currently selected language and adding the lang=xx parameter in the end, to ensure that, once a link is clicked, the site will continue to have the same language selected.

Of course, this particular macro may not serve all navigation needs 100%. If not however, the source code contained in the associated script (TranslatedNavigation.cshtml) can serve as a basis for developing an appropriate navigation, or even other needed macros.

## Better SEO with URL Rewriting

Polyglot was developed as a means to offer an alternative way than that provided by Umbraco by default to handle multilingual content. Multilingualism in Umbraco is managed by default by means of creating multiple sites, one for each supported language. This is a nice way and it is also optimal for Search Engine Optimisation (SEO), since every page is located at a unique URL, which can very well indicate the language of the page at the same time.

In some cases however, one needs to provide multilingualism in a different way. There are instances when a web site needs to be available, unchanged in its entirety, in all languages. The multi-site solution might not display a translation of a page before it has been made available. There are workarounds but defaulting from a page that does not yet exist to the corresponding one that does can get pretty complicated in this case. Also, there could be a requirement for defaulting even between parts of a page. For example there may be two pieces of text at one location, and if only one has been translated, a client may wish for both of them to appear, the translated one in the selected language and the other one in the language in which it has originally been written.

This kind of requirement can be served by what is often called a 1:1 multilingual strategy for structuring a web site. Polyglot has been developed as such. It places translations of each page in a "folder" under that page, and all content is shown on the front end in the visitor's selected language, if it exists, or in its original version if not.

In the URL of the page, the preferred language is indicated by the URL parameter "lang=xx", where xx is the ISO code of the language. This system has worked well for me and for other users. However, one of the questions I had received already upon the initial release of the package was "what about SEO?" Admittedly, I would imagine that search engines and indexers would handle parameter-differentiated pages a bit worse than ones with distinct URLs. Given the benefits offered by the package however, there seemed to be a good tradeoff between this disadvantage and the flexibility offered by the 1:1 solution to publish translations or even parts of translations as soon as they become available, without changing the site structure at any moment.

Recently however, I have noticed that this SEO thingy is taking a bit of a toll. Google used to group search results related to my site and present them in a nice two-column block. They now appear to be scattered as individual entries. Also, looking into the Google Webmasters Tools, there seems to be an issue in displaying preview texts for my pages, because a different "lang" parameter will lead to a different piece of text being displayed for the same URL.

Many of the users requesting support for Polyglot were mentioning that they were using Umbraco's URL rewriting feature in combination with the package, in order to form unique URLs and get rid of the language parameter. So I have thought that it would be a good idea to do the same on my web site, thus benefiting from the best of both worlds.

There are many ways to do this, so I have not implemented this solution within Polyglot, at least not yet. I would not want to dictate the URL structure for every web site that uses the package. Also, if I were to force URL rewriting over the "traditional" use of the package, that could cause a problem for existing installations.

So this is how I did it in my case. It is quite simple. This method removes the "lang" parameter and places the language code right after the domain name. So an address that used to look like "/home?lang=en" will now be "/en/home".

First, the following URL rewriting rule has to be placed in Umbraco's config/UrlRewriting.config file:

<add name="PolyglotRewrite" virtualUrl="~/([A-z]{2})/(.\*)" rewriteUrlParameter="ExcludeFromClientQueryString" destinationUrl="~/$2?lang=$1" ignoreCase="true" />

The second step has to do with this part of the code of Polyglot's TranslatedNavigation.cshtml:

string link = item.Url;

if (!string.IsNullOrEmpty(selectedLanguage))

{

link += "?lang=" + selectedLanguage;

}

It has to be changed to:

string link = item.Url;

if (!string.IsNullOrEmpty(selectedLanguage))

{

link = "/" + selectedLanguage + link;

}

That's it! If you do the above, your web site will basically have nice clean URLs without the language parameter.

One little caveat, if you are using the LanguageSelector macro of an older version of the package, used to be that it would not detect the selected language from the URL in order to have it preselected (it would still change languages correctly though). For that reason, I have added support for this type of URLs, so please make sure you upgrade to the latest version.

Of course, if you have hardcoded links referring to your pages, they will still be pointing to the old URL format. I have dealt with this by correcting the links wherever I can, but also by introducing this code snippet at the top of my master template:

@if (Request.RawUrl.ToLower().Contains("lang="))

{

var lang = Request.Params["lang"];

if (lang.Contains(","))

{

lang = lang.Split(',').Last();

}

Response.RedirectPermanent("/" + lang + Request.Path);

}

It will make a permanent redirect to the corresponding URL in the new format.