**Polyglot Package for Umbraco **

v1.5

[Watch a demonstration](http://www.youtube.com/watch?v=Q2rlusfbxP4)

**Supported Umbraco Versions**

The Polyglot package supports Umbraco v 4.5.x and up with the new xml schema. For Umbraco v 4.7.x or higher, a version implemented using razor scripts is also available.

The package supports all databases supported by Umbraco: SQL Server, MySQL and SQL CE.

**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**

The first step in using the package is of course installing it :). 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** (or .**xslt**) in macroScripts or xslt: 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** (or .**xslt**) in macroScripts or xslt: This script is similar to PropertyTranslation.cshtml (or xslt), but 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, it displays the translation of that property or its value from the basic page, if no translation exists. In TranslatedNavigation.cshtml or TranslatedNavigation.xslt, an example of how it works can be found. Please note that, most likely, it will have to be used in combination with SelectedLanguage.cshtml (or xslt).

**SelectedLanguage.cshtml** (or .**xslt**) in macroScripts or xslt: 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** (or .**xslt**) in macroScripts or xslt: 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** (or .**xslt**) in macroScripts or xslt: A corresponding macro will also be created. This is used by the above-mentioned Translation template.

**TranslatedNavigation.cshtml** (or .**xslt**) in macroScripts or xslt: 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 however 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: A corresponding macro will also be created. 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.

If the xslt version is installed, the script **LanguageParameter.xslt** will also be installed in the folder xslt. This is a script that picks up the language parameter from the http request and is used by other scripts.  It can also be used independently, if needed.

***If you are using Umbraco 4.7.0 or earlier and the Polyglot package razor script version, please remember to apply the razor script patch (RazorScriptLegacyPatchForUmbraco4.7.0\_xxxx.zip), which is available for download. You just need to unzip it and copy the three files it contains to your macroScripts folder, overwriting the existing files.***

**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.

So 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 TextPage\_TranslationFolder. “Create matching template” should not be checked while creating the document type. TextPage also needs to be configured to allow TextPage\_TranslationFolder as a child node type. This can be done by clicking on TextPage, selecting the Structure tab and then checking the TextPage\_TranslationFolder 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 the AppSettings section of the web.config file:

<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 TextPage\_Translation 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 TextPage\_Translation 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 TextPage\_Translation but to make a copy of TextPage and then give it the alias TextPage\_Translation.

**Most importantly**, an additional property needs to be declared in Textpage\_Translation, 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 TextPage\_Translation. 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, TextPage\_TranslationFolder needs to be configured to allow TextPage\_Translation as a child node type. Finally it is suggested that the true/false “umbracoNaviHide” property is declared in TextPage\_Translation, just like it has been declared in TextPage\_TranslationFolder.

**Creating Translations**

From here on the system is ready to manage translations of nodes of type TextPage. In the content section, if a TextPage is right clicked, there will exist the option [Create translations] (If this does not appear, logging out and back into Umbraco should solve the problem. Sometimes it will 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 TextPage\_TranslationFolder will be created under TextPage and, for each language checked, a TextPage\_Translation item will be created in the folder. The content of each translation can then be edited. Of course, each time the [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 application settings section of the site’s web.config or appSettings.config file). 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 under the AppSettings tag in web.config:

<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 "bodyText\_fr". If it should also be translated into Italian a third property should be created with the alias bodyText\_it.

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. In case the xslt version is used, the initlang variable in the script LanguageParameter.xslt needs to be set to the ISO code of the default language of the site (e.g. en, fr, de, etc.) This step is not needed for the razor version.

(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 application settings section of the site’s web.config or appSettings.config file.

**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**

As of version 1.5 Polyglot 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 AppSettings section of the web.config file. 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, the suffix of each property has to include the culture, in capitals (e.g. bodyText\_en-GB).

It is best if it is decided whether or not this feature will be used before any translations have been created. Nevertheless, for a system which has already been running without this feature activated and which needs to start using cultures, the procedure mentioned in the next section can be followed.

**Migrating to Using Cultures**

This section can be skipped if you are making a fresh installation of Polyglot.

It is strongly encouraged that the following steps be executed initially on a test environment because they involve massive updates to the database.

As a reference example, we will assume that a system has been running with the languages en-US, fr-FR, de-DE declared in the Umbraco back office, with Polyglot running without the cultures feature activated.

First we need to update the configuration and the database to work with cultures, without adding or changing the existing languages declared in Umbraco. As mentioned in the previous section, we start by adding the <add key="uPolyglotUseCultureInLanguageCode" value="true" /> needs to the AppSettings section of the web.config file.

We execute the following script successively, each time replacing the parts marked in yellow with each corresponding culture-less and culture including language code. This will update the names of the translation nodes on the content tree:

update [umbracoNode] set text = 'fr-FR'

where text = 'fr' and

id in

(select [cmsContent].nodeId

from [cmsContentType], [cmsContent]

where [cmsContentType].nodeId = [cmsContent].contentType

and [cmsContentType].alias like '%\_Translation')

In the same manner we execute the following script successively. This will update the names of the translation nodes in their properties tab.

update [cmsDocument]

set text = 'fr-FR'

where

text = 'fr' and

newest = 1 and

[cmsDocument].nodeId in

(select [cmsContent].nodeId

from [cmsContentType], [cmsContent]

where [cmsContentType].nodeId = [cmsContent].contentType

and [cmsContentType].alias like '%\_Translation')

We repeat the same for the following script. This will update the language label of the translation nodes:

UPDATE [cmsPropertyData]

set dataNVarchar = 'fr-FR'

WHERE

dataNVarchar = 'fr' and

propertytypeid in

(

SELECT id

FROM [cmsPropertyType]

WHERE contentTypeId in

(SELECT nodeId

FROM [cmsContentType]

WHERE alias like '%\_Translation') and alias = 'language'

)

and exists (select \* from [cmsDocument]

where [cmsDocument].nodeId = [cmsPropertyData].contentNodeId

and [cmsDocument].versionId = [cmsPropertyData].versionId and

[cmsDocument].newest = 1)

As mentioned in the previous section, if tabs are used for translations (please see section An *Alternative Way to Work, Using Tabs*), the suffix of each property has to include the culture, in capitals, so we now need to make that change (e.g. bodyText\_fr to bodyText\_fr-FR). The safest way to do this is through the Umbraco interface, by modifying the necessary document types in the Settings session.

Finally, we need to rebuild the xml cache of our instance. This can be done by logging into the back office and accessing the following URL (localhost needs to be replaced with the domain of the website):

http://localhost /umbraco/dialogs/republish.aspx?xml=true

At this point, the web site will have been migrated to being culture aware. It is best at this point to make sure that it works properly as it used to. Custom built scripts that assumed language codes in the format "fr" instead of "fr-FR" will need to be modified. Once that has been done, the additional languages can be added to the Settings section of the Umbraco back office at any time, just as if Polyglot had been configured to use cultures from the beginning.

**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.