# Babylon.NET Help

## Table of contents

# Introduction

### *Welcome to Babylon.NET*

Babylon.NET is a software localization tool specifically designed to localize applications created using MS .NET and Microsoft Visual Studio. Starting with version 2.4.0 Babylon.NET supports Generic Localization projects that can be used to translate almost any type of text content by using so called Resource Providers to import and export the strings to and from Babylon.NET.

When using Babylon.NET the localization of the application is done at the end of each development cycle directly in Babylon.NET. Using its own solution files, Babylon.NET can synchronize with the Visual Studio project whenever required and import new and changed resources marking them as "to be translated" or "changed". Babylon.NET can also be used to edit and review the original (invariant) language of an application making it easy to spot and correct common errors in resource strings.

### *How to get started*

- *Localize a Visual Studio project*

## Editions

Babylon.NET comes in two editions, the **Full edition** and the **Translator edition.**

### *Standard edition*

The full edition is used by the developer to create new localization solutions, synchronize the localization solution with Visual Studio projects, add and remove locales and write the localized resource files. The full edition can also be used to edit the invariant language and to translate to other languages. Once a localization solution has been created, new languages can be added and translation packages generated. Translation packages can be opened directly using the Translator Edition and contain everything a translator needs to translated the given languages.

### *Translator edition*

The translator edition is geared towards translators with no knowledge of Visual Studio, software development or other technicalities. The translator edition offers all the functionality to translate and verify the localization while all the other functionality is disabled. The translator opens the translation package given to him and translates the application saving all the changes into the translation package. When completed the translation package is given back to the developer who can either use it directly to write the resource files or can import the localized strings back into the localization solution.

## Licensing

Babylon.NET is copyrighted and licensed (not sold). By purchasing the program, you are accepting and agreeing to the terms of this license agreement. This license agreement represents the entire agreement concerning the program, between you and Redpin., (referred to as "licensor"), and it supersedes any prior proposal, representation, or understanding between the parties.

1. License Grant. Licensor hereby grants to you, and you accept, a nonexclusive license to use the program in machine-readable, object code form only, for use only as authorized in this

License Agreement. The Programs may be used only on computers owned, leased or otherwise controlled by you. The full edition of the program shall only manage the number of user accounts specified in the purchase agreement. You agree that you may not reverse assemble, reverse compile, or otherwise translate the Program.

2. Translator edition License Grant. The translator edition of the program may be freely distributed to and used by all parties involved in the translation of your software applications.

3. Licensor`s Rights. You acknowledge and agree that the Program is proprietary to Licensor and protected under copyright law. You further acknowledge and agree that all rights, titles, and interest in and to the Program, including associated intellectual property rights, are and shall remain with Licensor. The License Agreement does not convey to you an interest in or to the Program, but only a limited right of use revocable in accordance with the terms of this License Agreement.

4. No Warranty; Limitation of Liability. You acknowledge that the program is provided on an "as is" basis without warranty of any kind. Licensor makes no representations or warranties regarding the use or performance of the program. Licensor expressly disclaims the warranties of merchantability and fitness for a particular purpose. Licensor shall have no liability to customer or any third party for any loss or damage caused, directly or indirectly, by the program, including, but not limited to, any interruption of services, loss of business, loss of data or special, consequential or incidental damages.

5. Severability. Should any term of this License Agreement be declared void or unenforceable by any court of competent jurisdiction, such declaration will have no effect on the remaining terms hereof.

6. No Waiver. The failure of either party to enforce any rights granted hereunder or to take action against the other party in the event of any breach hereunder shall not be deemed a waiver by that party as to subsequent enforcement of rights or subsequent actions in the event of future breaches.

## Settings

### *Google API key*

Sets the Google API key to access the Google Translator service. Visit *https://code.google.com/apis/console/?api=translate* to request a key. The service is not free. See *http://code.google.com/intl/de/apis/language/translate/v2/pricing.html* for pricing.

### *Azure API key*

Sets the Azure API key to access the MS Translator Text service. Visit the *MS Azure Portal* to request a key. The service is free for up to 2m characters a month.

### *Path to Assembly Linker*

Sets the Path to AL.exe. Only set the path in case of errors during Assembly Generation.

### *Next Row On Enter*

Sets the behaviour when pressing the Enter key in the localization view.

### *Enable Spell Checking at startup*

Sets wether the Spell Checker should be active after startup or not.

### *Show Tooltips in main grid*

Sets wether warning and error messages should be shown in a tooltip when hovering over a string.

### *Automatically translate duplicate strings*

Indicates if duplicate strings should be automatically translated when one of the strings gets translated.

### *Set translated duplicates into To Review state*

By default translated duplicates will be set into AutoTranslated state.

### *Source Control Integration*

Source Control Integration allows to specify a command (batch file) executed before writing a ResX file, after creating a ResX file and after writing a ResX file. The commands will be executed in a command shell window and should perform the related actions. They will vary depending on SCM used. Source Control Integration is not supported for Generic Localization Projects.

Babylon.NET calls the commands passing 3 parameters in this order:

- Filename of the file to be created, checked-out or checked in
- Filename of the Visual Studio project file
- The operation to be performed as string: “Create”, “CheckOut” or “CheckIn”

*Example: A batch (.bat) file to automatically check-out a file from Team Foundation Server would contain just the line:* **tf checkout %1**

## Support

### *Sales inquiries*

For questions about products, prices and ordering please write to info@redpin.eu.

### *Technical support*

For technical support please write to support@redpin.eu.

# Translating Projects

Babylon.NET was specifically designed to localize Visual Studio .NET projects in all supported .NET languages.

### *Process to localize an new application for the first time*

1. Prepare the Visual Studio project for localization
2. Create a new localization solution in Babylon.NET and associate Visual Studio projects with it
3. Add the desired target locales
4. Translate all resource strings into all locales
5. Verify the translation using the built-in verifier
6. Write all resource files using the "**Write ResX**" feature in Babylon.NET
7. Add all localized resource files to your Visual Studio project and build the application
8. Check in the Babylon.NET solution file into your source control system

### *Process to localize a new version of an already localized application*

1. Develop your application in the invariant language without considering localization
2. When the new version of the application is ready use the "**Synchronize**" function in Babylon.NET to import new and changed resource strings into the localization solution.
3. New and changed resource strings are clearly marked in Babylon.NET. Translate all new resource strings and verify all changed strings.
4. Verify the translation using the built-in verifier
5. Write all resource files using the "**Write ResX**" feature in Babylon.NET
6. Rebuild the application

## Project Types and supported file formats

Babylon.NET offers two types of translation projects. Both project types can be freely mixed within one Babylon.NET Solution.

### *Visual Studio Translation Projects*

Visual Studio translation projects are used to directly work with Visual Studio projects. When creating a project strings are directly extracting from all .resx file contained in the project. Visual Studio Translation projects offer the best support for .NET projects also providing special features such as Form Preview or direct Satellite Assembly Generation not available in Generic Localization projects.

### *Generic Localization Projects*

Generic Localization projects are used to translate all non .NET projects. Generic Localization projects read and write string resources from a variety of file formats such as JSON or XML by using so called ResourcesProviders. Each ResourcesProvider implements import/export functionality for one specific file format.

Out of the box Babylon.NET offers 3 ResourcesProvider:

JSON ResourceProvider - Reads and writes strings from JSON files.

XML ResourcesProvider -  Reads and writes strings from XML files.

Java Properties File ResourcesProvider -  Reads and writes strings from Java .properties files.

### *Custom Resources Providers*

ResourcesProviders in Babylon.NET are implemented as plugins. New file formats can therefore be supported by simply writing a new ResourcesProvider.

To install a new ResourcesProvider simply copy the assembly to the ResourcesProviders subfolder of your Babylon.NET installation folder.

More information about ResourcesProvider and about writing custom Resources Providers can be found on our website.

## Preparing your Visual Studio project for localization

Babylon.NET will look for resource strings in all .resx files included in a project.

For Babylon.NET to find all resource strings in Windows.Forms projects that need to be localized (and for the localization to work in your application) set the "**Localizable**" property of every form to true before creating the Babylon.NET localization solution.

## String Extraction

String Extraction is a semi automatic tool that finds and extracts string literals from source code and generates string resources in ResX files. String Extraction can be used to refactor applications written without using string resources that have all text in the source code. In this case String Extraction should be used only once to move all the strings to resource files. After this one time effort development should start using resource strings.

String Extraction can also by used for application adhering to localization guidelines to check whether some strings have been accidentally put into the source code.

String Extraction relies on the Microsoft.CodeAnalysis (former Roslyn) namespace to parse the source code and find string literals. It will thus find any string literal in the code with great accuracy.

String Extraction is not supported for Generic Localization projects.

### *Scanning for string literals*

The String Extraction dialog uses a two step approach. The first step is selecting the source code files that should be scanned for strings and filtering the type of string literals that should by searched for. Regular Expressions and Attributes will rarely need translation so by default they will not be included in the scan result. After clicking the Scan button scanning is performed and all string literals matching the search criteria will be listed on the right hand side of the dialog.

### *String Extraction*

During the scan Babylon.NET automatically generates a resource name for the resource string to be generated. In order to generate unique names the class name is used and an incrementing number added. When a string is associated to a variable, a field or a constant the name of the variable, field or constant is added to the class name for greater accuracy. Resource names can be modified by directly editing the names in the grid column. String literals can be included/excluded from extraction one by one using the checkbox in the Extract column. Multiple selection is supported through the grid context menu.

String Extraction supports two ways of generating ResX files: one ResX file for all strings or one ResX file for every source code file. Depending on the selected generation mode, the ResX file name, the ResX destination path and the code snippet to be inserted in source code instead of the string literal need to be set up. String Extraction offers three placeholders

<codefile>                    Will be replaced with the name of the source code file and its relative path in the project.

<resourceFilename>            Will be replaced with the name of the ResX file name. This differs for C# and VB.NET as accessing resource is not handled the same way.

<resourceName>                Will be replaced with the name of the resource string to be generated.

Once everything is set up, clicking on the Extract button will perform the extraction and generate ResX files and resource strings. After each extraction the scan is automatically executed again to reflect the new state. If not all strings of a file are extracted and other strings need to be extracted in a second extraction, automatic resource naming will produce duplicate resource names that will have to be fixed manually.

WARNING: String Extraction will perform multiple modifications on you source code. Make sure you have a backup copy of all the source code before using String Extraction.

### *Including the new ResX files in your VS project*

The newly generated ResX files need to be added to the VS project. This will however not automatically generate the strongly typed class to make resource available from code as static strings. To generate the class the CustomTool property of the ResX file needs to be set to "ResXCodeFileGenerator" and "Run Custom Tool" must be executed from the context menu.

## Creating a localization solution

To create a new localization solution click the "**New**" button in the ribbon bar and then follow the wizard. A localization solution may contain one ore more translation projects. All projects will use the invariant language you select the in the first page of the wizard.

You add translation projects to the solution on the second page of the wizard. For each project you add a translation project is created. For backward compatibility Babylon.NET 1.x projects (*.vslp files) can also be added to the solution.

Visual Studio projects and Generic Localization projects can be freely mixed within one solution.

When completing the wizard, Babylon.NET creates a single solution file with the "*.bsl" file extension. This solution file contains all translation projects, resource strings, translations and status information about the solution and should be checked into the source control system along with the application source code.

## Translating

### *Translation status*

The translation of a resource string indicates the translation progress for the resource. The translation status is automatically set by Babylon.NET. When synchronizing the translation project with the Visual Studio project changed resource strings will be set to translation status "**Changed**". To clear the changed status click the right mouse button in the corresponding cell and select "**Clear changed flag**" from the context menu.

### *Quality status*

The quality status of a resource string is used to manage the review process of the localization. Newly translated or changed resource strings will be automatically set into status "**To review**". Once the resource string has been reviewed the status is changed to "**Approved**" by selecting the "Quality status" item in the cell context menu. If an error is detected during the review process, the status is changed to "Error" to mark the resource string as not correctly translated.

When a project is verified using the Babylon.NET Verifier, the quality status is automatically changed to "Warning" or "Error" according to the severity of the problem that has been detected. Items in status Approved will be skipped by the Verifier.

### *Status scope*

Since most translation work is done by working only on a single language at a time, the display scope of the translation and quality status can be changed from "All locales" to a single locale. This way the translation and quality status columns display the status of just the selected language.

### *Do not translate*

Resource strings can be marked not to be translated using the corresponding item of the cell context menu. Setting the "do not translate" flag for all resource strings that are just placeholders or are otherwise not displayed during runtime will greatly reduce the number of strings that need to be translated.

### *Exclude a resource string*

By excluding a resource string it will be permanently marked as excluded in the Babylon.NET project and not shown in the translation grid. The string will not be synchronized, not written to ResX files and not inserted into Satellite Assemblies. Excluding a string is necessary in special situations such as when a control serializes XML into a resource string.

### *Status backcolors*

To help detect resource strings in a certain translation or quality status the corresponding backcolor function can be selected from the main ribbon bar.

## Custom Cultures

Custom cultures allow to localize to cultures that are not natively supported by .NET. To add a custom culture to a Solution open the Solution properties dialog and define a custom culture.

Once the culture has been defined it can be added as a locale using the Manage Locales menu item. Custom Cultures are treated just as regular cultures for all other functions.

## Verifying

The Babylon.NET Verifier can be activated at any time by clicking on the "**Verify**" button in the main ribbon bar. The Verifier checks for the following problems

- Inconsistent translation. There are more than one identical invariant resource strings that have been translated differently

- Inconsistent translation with Translation Memory.
- Punctuation error. At least one translated resource string ends with a different punctuation than the invariant resource string.
- String.Format error. The invariant resource string contains a different number of string.format indexed placeholders e.g. "{0}" than  the localized strings. Make sure the correct programming language is selected before starting verification. The Verifier only checks the number of string format placeholders but does not check for correct numbering of placeholders or other errors.
- Spacing errors. A spacing error is a resource string starting or ending with a space or a string that contains two or more consecutive spaces.
- Missing Mnemonics in translated strings.
- String length warning. This check compares the string length of the invariant string with the translated strings. If a translated is significantly longer than the invariant string a warning is set.

The Verifier changes the quality status to Warning or Error according to the problem that has been detected.

## Writing resource files

Once the translation has been finished the updated resource strings are written to the resource files by clicking the "**Write Resources**" button on the main ribbon bar. For Visual Studio projects strings are written to .resx files. For Generic Localization projects strings are written using the selected ResourcesProvider.

ALWAYS MAKE A BACKUP OF YOUR RESOURCE FILES BEFORE WRITING TRANSLATED STRINGS TO THE RESOURCE FILES.

Babylon.NET can automatically check-out and check-in ResX files. See *Settings* to setup source control integration.

## Synchronizing a project

By clicking on the "Synchronize" button in the main ribbon bar the selected translation project is synchronized with the Visual Studio project or with the selected ResourcesProvider. Synchronization performs the following steps

- New resource strings in the Visual Studio project or read from the ResourcesProvider will be added to the translation project and set into translation status "to be translated"
- All existing resource strings in the translation project will be updated with the corresponding strings in the Visual Studio project or ResourcesProvider and their translation status set to "Changed". Therefore any changes to invariant or localized resource strings in the Visual Studio project or ResourcesProvider will be propagated to the translation project.
- All resource strings in the translation project that have been deleted from the Visual Studio project or from the ResourcesProvider files will be deleted from the translation project.

Synchronization will never change the Visual Studio project in any way and can therefore be executed at any time.

## Automatic Translation

The automatic translation function uses Google Cloud Translation API or Microsoft Azure Translator API to translate from one locale to another. To translate a locale click on the 'Auto Translate' button on the Automated Translation page of the main ribbon. Select the source locale that should be translated and the target locale. After clicking the Translate button the texts of the source locale are one by one translated using Google/Bing Translator. To use Google or Bing Translator the respective API keys need to be set in the Settings dialog.

### Mnemonics

Mnemonics (&) are not treated correctly by Google/Bing Translator. To avoid wrong translations select from one of the available options how to handle mnemonics. Generally it is best to just strip mnemonics and not reinsert them after translation. This way the Babylon.NET Verifier will warn about missing mnemonics and they can be inserted manually taking care not to duplicate any mnemonic on a form.

### Trailing punctuation

Punctuation is in some cases not treated correctly by Google/Bing Translator. Especially ellipses (...) and whitespace is sometimes eliminated. To work around this problem specify what trailing punctuation should be removed before translation and added back after the translation.

Translated items are set to quality status "Auto Translated" to distinguish them from manually translated or approved items.

### Pseudo Translator

Pseudo translation is used to check the user interface for strings that are not located in resources. Pseudo translation is also used to verify that strings getting longer due to translation still have room in the user interface.

The Pseudo Translator does not translate strings but transforms the strings by adding accents, umlauts, cedillas, tildes etc. The resulting strings can still be read. By starting the application in a pseudo translated language any string not translated, i.e. not located in a resource file, will clearly stand out. The Pseudo Translator also makes the strings 20% longer by stuffing the strings with trailing exclamation points and putting square brackets around the string. This way strings that are truncated in the user interface can be easily identified.

### Setup Google Cloud Translation API in Babylon.NET

- Navigate to *https://cloud.google.com/* and setup an account.
- Use the toaster menu in the upper left  corner to open the menu and open the API Manager
- Select Enable API and choose Translate API
- Open the Credentials page and create a new API key by clicking on New Credentials
- Copy the key and insert it into Babylon.NET using the Settings dialog

**Setup Microsoft Azure Translator API in Babylon.NET**

- Navigate to *https://portal.azure.com* and setup an account.
- Select New->Intelligence + analytics->Cognitive Services API and create a new service
- Select Translator Text API as API type
- From the dashboard navigate to the Translator API
- Open the Keys page, copy Key 1 and insert it into Babylon.NET using the Settings dialog

## Using the Translation Memory

The translation memory is a very useful tool when translating software projects as the same strings are used in many places in the software and need to be translated in a consistent way.

The translation memory stores translation pairs of different source/target locales that are used to automatically translate not yet translated strings. Babylon.NET keeps the translation memory in plain XML files that can be opened at any time when working with Babylon.NET. The file based approach allows the user to create several different translation memories to be used in different ways. For example it may make sense to create one generic translation memory containing basic translations and one translation memory per project containing the specific domain language.

To work with translation memories follow these steps:

1. Open a Babylon.NET project

2. Create a new translation memory

3. Export selected language pairs from the project to the translation memory

4. Whenever the strings in the projects change open the translation memory and translate using the translation memory

5. Review all items in status Auto Translated and translate the remaining items manually

6. Export again to the translation memory to update it with the new strings

The Edit translation memory function can be used to manually edit the content of a translation memory. To edit a translation memory it must contain at least one translation.

### *TMX Export and Import*

Translation Memory eXchange (TMX) is an XML specification for the exchange of translation memory data between computer-aided translation and localization tools. The content of a Translation Memory can be exported to TMX format

Babylon.NET reads and writes TMX format v1.4. Only datatype=PlainText and segtype=Sentence are supported. Content markup, inline elements and level 2 are not supported.

## Generate satellite assemblies

The satellite assembly generator allows you to generate satellite assemblies for one or more selected locales without having to write the ResX files and rebuild the project in Visual Studio. The function will automatically create a new subdirectory for each locale selected and write the satellite assemblies.

To link the satellite assemblies Babylon.NET needs the assembly linker (al.exe) from the .NET

SDK. The .NET SDK (included in Visual Studio or available for download stand alone) must be installed on the system to generate satellite assemblies. The various versions of .NET all have different and incompatible versions of al.exe. Babylon.NET automatically tries to find the correct version. In some constellations however Babylon.NET might use the wrong version. In this case the correct path to al.exe can be set in the settings dialog.

Generating Satellite Assemblies is only supported for Visual Studio projects.

## Removing unused resource strings

In almost any project some resource strings become obsolete over time. Most of the time when a dialog or an error message is removed from code the corresponding resources are not remove from the resource files. Over time these unused resource strings accumulate and cause a lot of wasted time and effort during localization. Removing these strings is problematic as there is no way of knowing if they are still referred to by some code.

The "Remove unused resource strings" function makes identifying and deleting obsolete resource strings easier. The search for unused strings simply loads all source code of the project and searches all the resource names in the source code. Each resource string that cannot be found in any source file is displayed as obsolete. ATTENTION: since the function performs a text search, resource names that are assembled in the source or loaded from an external source will not be found and therefore be wrongly displayed as obsolete.

Only supported for Visual Studio projects.

## Swap locales

The swap function swaps all resource strings between two locales or between a locale and the invariant locale. In the second case the language of the current invariant locale has to be specified.

## Previewing Forms

The "Preview" function can be used to view a Windows.Form in any of the projects locales. By using the Auto Refresh option the preview is automatically refreshed after a string is changed or a string of a different locale is clicked.

To display forms as correctly as possible the Windows.Forms preview recreates the forms by parsing the related ResX and .Designer.cs/vb files. This ensures the greatest accuracy in displaying the form. The preview works correctly also when non Microsoft UI components such as DevExpress or Telerik are used. Babylon.NET will automatically search for the correct assemblies in the global assembly cache. A secondary search folder in which to search for assemblies can be specified in the project settings.

When a Translation Package is opened Babylon.NET will automatically check if a .bpp (Babylon Preview Package) is available for the Translation Package. If a .bpp file exists it is extracted into a folder and the secondary search folder of each project is set to this folder. For more information on Preview Packages please see *Preview Packages*.

Preview is only supported for Visual Studio Windows.Forms projects.

## Quality review wizard

The quality review wizard offers a convenient way of reviewing all resource strings in status other than "approved". The wizard displays one resource string at a time in the invariant language and two locales that can be freely selected. A reviewer simply reviews one item at a time and uses the quality status buttons and the navigation buttons to set the quality status for each string and to navigate between resource strings. Using the keyboard shortcuts the review process can be performance in an accurate and efficient way.

## Integration in build process

Babylon.NET can be run with command line arguments that allow the integration in an automated build process.

### *Command line arguments*

Usage: SolutionFilename | TranslationPackageFilename

Usage: SolutionFilename /wr [Locales] [Projects] [Destination]

Usage: SolutionFilename /ws Destination [Locales] [Projects]

Usage: SolutionFilename /s [Projects]

Usage: SolutionFilename /atm SourceLocale TargetLocale [Overwrite] [Mnemonics] [Punctuation] [Projects]

Usage: SolutionFilename /atg SourceLocale TargetLocale [Overwrite] [Mnemonics] [Punctuation] [Projects]

Usage: SolutionFilename /atp SourceLocale TargetLocale [Overwrite] [Mnemonics] [Punctuation] [Projects]

Usage: SolutionFilename /utp TranslationPackage [DueDate] [CreateBPP]

Usage: SolutionFilename /rtp TranslationPackage

| | |
|---|---|
| SolutionFilename | filename of the Babylon.NET solution or translation package |
| /wr | Write Resources |
| /ws | Write Satellite Assemblies |
| /s | Synchronize project with Visual Studio project or ResourcesProvider |
| /atm | Autotranslate using Microsoft Translator |
| /atg | Autotranslate using Google Translator |
| /atp | Autotranslate using Pseudo Translator |
| /utp | Update Translation Package. To use this option a translation package first has to be created manually using the Babylon.NET user interface. The given Translation Package is updated with new and changed strings in the Solution. After the update a new translation job is added to the Solution. |
| /rtp | Receive Translation Package. After importing the translation package the corresponding translation job in the Solution is deleted. |
| | |
| Locales | A list of locales in the form en-US separated by comma. If not specified all locales will be written. |

| | |
|---|---|
| Destination | The destination file path for the satellite assemblies or resx files to be generated. |
| Projects | A list of project names separated by comma. If not specified all projects will be processed. |
| Version | The version number of the generated satellite assembly in the format X.X.X.X |
| Keyfile | Specifies the key file to generate a signed satellite assembly. |
| SourceLocale | The source language for the translation in the form en-us. Use inv for the invariant language. |
| TargetLocale | The target language for the translation in the form en-us. |
| Overwrite | Flag enum to specify overwrite behaviour during auto translation.<br><br>None=0<br>Overwrite Changed = 1<br>Overwrite Autotranslated = 2<br>Overwrite To Review = 4<br>Overwrite Error = 8<br><br>Defaults to 'None' if not specified. |
| Mnemonics | Enum to specify handling of mnemonics during auto translation.<br><br>None = 0<br>Strip Ampersands = 1<br>Strip Ampersands and reinsert them at the beginning = 2<br>Strip Ampersands and reinsert them at the end in parentheses = 3<br><br>Defaults to 'Strip Ampersands' if not specified. |
| Punctuation | Regular expression indicating trailing characters to strip before automatic translation. Stripped characters will be added back after translation.<br><br>Defaults to '(\.\|!\|\s\|\?\|;\|:\|,)*$' removing trailing punctuation, ellipses and whitespaces. |
| DueDate | The due date for the translation job in the format mm/dd/yyyy. |
| CreateBPP | Indicates if a Preview Package (.bpp file) should be created (1=enabled, 0=disabled). |
| UseSCM | Indicates if source control integration should be used (1=enabled, 0=disabled) |
| [] | Can be used to skip optional parameters. See parameters for default setting. |

## Spell Checker

Babylon.NET offers an integrate, inline spell checker. The spell checker automatically switches to the language of the cell being edited. Babylon.NET uses free OpenOffice dictionaries for the spell checker. Dictionaries for Czech, Danish, German, English, Spanish, French, Hindi, Croatian, Hungarian, Italian, Dutch, Norwegian, Polish, Portuguese, Slovak, Slovenian and Swedish are already included in the standard distribution. Additional dictionaries can be added by simply copying them into the "dictionaries" subdirectory of the Babylon.NET installation directory.

The spell checker can be activated/deactivated using the Spell Checker toolbar button.

## Project Statistics

Shows a dialog with solution and project statistics such as string count, word count, character count and the breakdown of the strings by translation and quality status.

# Outsourcing translations

There are two ways to work with external translators:

### *Create translation jobs*

Create a translation job in your localization solution, generate a translation package and send the package to your translator. Using the free Babylon.NET Translator Edition the translator can open the package and translate it using all the functionality provided by Babylon.NET. When done the translator sends the package back to you and you merge it back into the localization solution.

### *Using MS Excel*

Export all the strings for one ore more languages to an MS Excel file. The translator translates the strings directly in MS Excel or imports them into another system. When the translations are complete they can be reimported in the Babylon.NET solution from the MS Excel file. This approach does not use the build in functions and the translation and quality status tracking system of Babylon.NET and is therefore much slower and more error prone. It should be used only when the translator cannot use the Babylon.NET Translator Edition for some reason.

## Export to MS Excel

Allows you to export all or selected strings of the currently open solution to an MS Excel file. To perform an export first select the destination file, select the locales to export and choose the status of the strings you would like to export. The generated Excel file will contain a first line with column headers and the following columns: filename, resource name, invariant comment, one column for each selected locale.

## Import from MS Excel

Allows you to import all or selected strings from an MS Excel file into the currently open solution. To perform an import first select the source file. All locales contained in the source file will be displayed. Select the locales to be imported, choose which strings to overwrite depending on status and set the quality status of imported strings.

The import function will ignore resource strings present in the source file but not in the project.

## Working with translation jobs

Translation jobs greatly simplify working with external translators. Working with translation jobs is very simple and involves 4 steps:

1. Create a translation job.
2. Generate a translation package (*.btp file) and optionally a preview package (*.bpp file) and send them to your translator
3. The translator translated the package using the Babylon.NET Translator edition and sends it back to you
4. Merge the translation package into your solution and get all the new translations

### *Create a translation job and generate a translation package*

Translation jobs are created by clicking the "**Create Job**" button from the Translation Outsourcing ribbon. For your convenience every translation job has a name. You can also insert the name of the translator and the due date. Check the "**Allow editing of strings in approved state**" checkbox if you would like to allow the translator to change strings already approved.

On the second page of the wizard check all the locales you would like the translator to translate. The invariant language will always be included for reference. When selecting a locale already present in another translation job Babylon.NET warns you about possible conflicts when merging the jobs back into the solution.

On the third page of the wizard you may add notes for yourself and the translator to the translation job. The notes will be displayed to the translator every time he opens the translation package.

On the last page of the wizard you can create the translation package to send to your translator. A translation package contains everything the translator needs for the translation in one single *.btp file.

Once the wizard is completed a new translation job is added to the solution.

### *Managing translation jobs*

Use the "**Manage Jobs**" function to view and manage translation jobs.

### *Translating a translation package*

The translator opens the translator package using the free Babylon.NET Translator edition. The translator edition supports all the functionality to quickly  end efficiently translate the translation package. The translator will not be able to edit the invariant language and he cannot set string to "Do not translate". When the translation is finished the translator sends the translation package back to you.

### *Receiving a translated translation package*

Use the "**Receive Translation**" function to receive back a translation package and merge it into your localization solution. You may select which locales to import and also the status of the strings to import.

## Preview Packages

Preview Packages (*.bpp files) contain all the information for the form preview function to accurately display the preview of a form. Preview Packages can optionally be created when creating a translation package. The preview package must be distributed to translators along with the translation package for the preview functionality to work correctly.

Preview Packages contain the ResX and .Designer.cs/vb files of all the forms in a project. They also contain all the assemblies needed to correctly display the forms on translator PC's. No other files will be added to the Preview Package. In no case are any other source code files added to a Preview Package. The content of a Preview Package can easily be verified by renaming the .bpp file in .zip and by opening it with a standard ZIP compression tool.

The Translator Edition will automatically check for Preview Packages when opening a Translation Package (the bpp file must be in the same folder as the btp file). If a bpp file is found it is automatically extracted and the secondary search folder for assemblies of each project is set to the folder containing the Preview Packages.

Preview Packages are only supported for Visual Studio projects.

## Importing from legacy projects

This import function allows you to import resource strings from a legacy Babylon.NET 1.x project file. It is provided for backward compatibility only. To perform an import first specify the target translation project, than select the source project file, the locale to import and then target locale in the target project.

During the import, resource strings in the source file and target file are compared by name. Resource strings present in the source file but not in the target file will be added to the target file. If resource strings are present in both files the target file will be update with the value from the source file.