Localizing .NET applications

Sisulizer
9 November 2011

Introduction

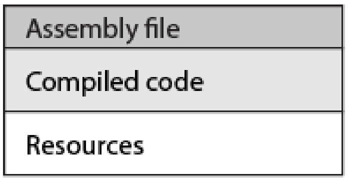
This document describes the steps that you should perform when localizing your .NET applications. Localization means translating your application into another language. In the other words it means translation of user interface elements, resource strings, images and help files. For example if you have written your original application and help files in English and you want to start selling your application in Germany you have to translate your application and help files to German. Before you can start translating your application it must be internationalized. This means preparing your source code and user interface such way that it does not contain any language depend data such as hard coded strings. Internationalization is very important because it is necessary for localization and if it is not properly done then localization will be slower and more expensive.

The first part of the document deals with internationalization and common concept of localization. The second part describes localization architecture of different .NET technologies such as WinForms, WPF and Silverlight. The last part is Sisulizer specific and it shows the localization process and explains how to take .NET's satellite assemblies in use. We finally show how to perform runtime language switch.

This document is written for Visual Studio 2008 and C# using WinForms, WPF and Silverlight. Silverlight samples are written in Visual Studio 2010. If you use an older Visual Studio, Visual Basic or Delphi Prism you can perform same steps but you may have to modify them.

.NET Localization Architecture

.NET applications contain one or more assembly files. The file extension of an assembly file is either .exe or .dll. Assembly files contain compiled code and resource data.



Resource data is important for localization. It contains all forms, strings, images, etc. In order to localize an application we have to translate the resource data. This is where .NET uses its own unique approach. Let's first look at how native Windows applications (WIN32/WIN64) are localized. It is most often done such way that a localization tool creates one EXE for each language. This means that if you application is MyApp.exe the German application is also MyApp.exe but the localization tool creates the file into another location (e.g. de\MyApp.exe) and replaces the resource with German resources. When you deploy a localized native Windows application you deploy the localized EXE instead of original EXE. This does not work in .NET. Microsoft has chosen a different method for .NET localization. In .NET the localized data must be in a satellite assembly files. They are DLL files that contain only localized resource data, no code at all.

Satellite assembly file
German resources
Satellite assembly file
French resources
Satellite assembly file
Japanese resources

If your original application is MyApp.exe then the German assembly file is de\MyApp.resources.dll. Note that the satellite assembly file must locate on a locale specific sub directory of the original application. The name must be the same as the original but instead of having .exe (or .dll) extension the satellite assembly file uses .resources.dll file extension. Having a satellite assembly file on right sub directory with right name is not enough. The actual resource name must also be localized. .NET uses named resources where each resource has a string name. For example if we have MyApp.exe the name of main form could be MyApp.Form1.resources. The German satellite assembly file must use MyApp.Form1.**de**.resources name. If the resource name is not right .NET cannot load the resource even if the file name and directory are right. There is one more pitfall. It is signing. If you have signed your original assembly file and your localized satellite assembly files are not signed .NET runtime does not load the resources even if the pathname and resource names are right.

The first half of the document deals with internationalization and common concept of localization. The second half is Sisulizer specific and it shows the localization process and explains how to handle some advanced issues such as runtime language switch.

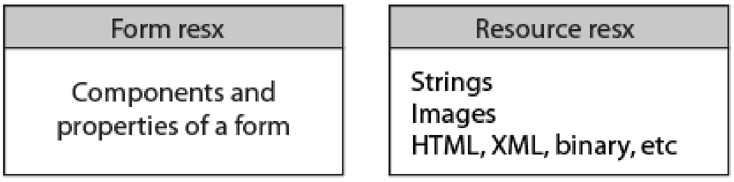
There are two tools that are useful when viewing assembly files. The first is ILDASM.exe that comes with .NET SDK and Visual Studio. It can be used to view the resource names. It cannot show the resource data. If you want to view both resource name and data I recommend using Reflector (http://reflector.red-gate.com/). It is very useful application to view .NET assemblies.

.NET has two kinds of user interface libraries. .NET 1.0 introduced Windows Forms library that uses traditional pixel based layout where every component has position (Left and Top properties) and size (Width and Height properties). .NET 3.0 introduced new kind of user interface library that does not use pixel based layout but uses container components that dynamically layout components on the container on run time according the layout rules. This kind of layout mechanism is similar to Java' s Swing classes. This new library is called WPF. Silverlight is a subset of the library. WinForms also contains some layout controls but most of the user interface is pixel based. WPF also contains pixel based container (CanvasPanel) but it is not that commonly used. Generally Windows Forms use pixel based layout and WPF/Silverlight dynamic layout.

Resources

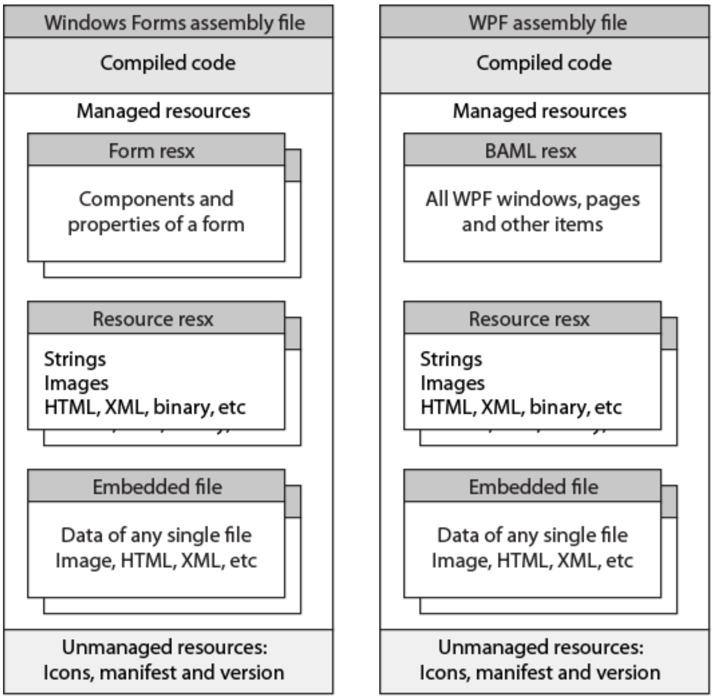
In order to be localizable a .NET application should store all localizable resource into resources. There are several different resource types. The most important resource types are ResX, XAML, image, and XML resources.

ResX is an XML based flat resource format that contains one or more resource items. Each resource has a name, value and optional command and type information. The value contains the actual resource data. There are two kinds of ResX files. One contains properties and components of a form. Another contains any kind of resource data such as strings, images, HTML data and XML data.



Both Windows Forms and WPF use ResX resources. Windows Forms uses both form and resource ResX. WPF uses only resource ResX files. Visual Studio can automatically store all properties of Windows Forms form into ResX file. See the next chapter learn more. WPF added a new resource format called XAML. It is also based on XML but unlike ResX the file format is not flat but is structured. There is one XAML file for each page or window. Each XAML contains the properties and events of the root components and all the components it contains. Silverlight uses same XAML resource files as WPF.

In addition of ResX and XAML resources any .NET application can also contain plain resource. Most often they are image, HTML or XML files that are embedded into resource block of an assembly. However it is possible to add any file as plain resource data inside the .NET assembly. It is up to the developer to choose if to store images and files inside ResX or as plain resource data. A .NET assembly file can also contain some unmanaged resources such as icon, manifest and version resources.

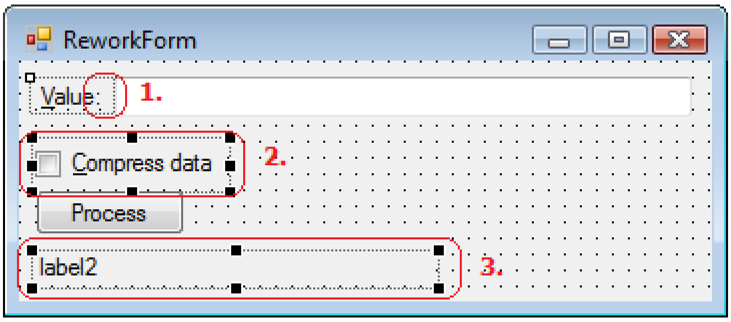


User interface

Before we can start localizing a .NET application it must be prepared for localization. This process is called internationalization (I18N). Depending on the user interface library the internationalization is done little bit different way.

Windows Forms

Let's start with a Windows Forms sample. NET\Tutorial\WinForms\Original sample directory contains a project that requires internationalization. The project is very simple and does not do anything meaningful but the project shows most of the internationalization issues. The following picture contains ReworkForm form that needs preparing for localization.



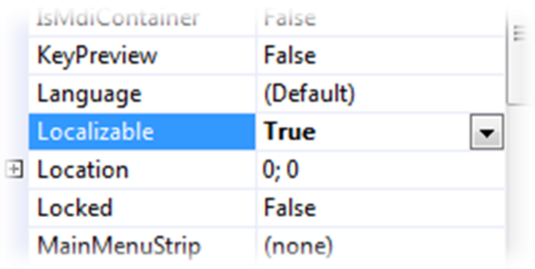
The form contains three cases where we have to change it in order localize the form easier. Each case is marked with red bounds and a number. Let's go through each step.

In the case #1 another control is following a control. The Value label is immediately followed by a text box. This will cause you problems because it is very likely that the translation of the Value to other language will be longer even much longer that the original value. This will make label and text box to overlap each other. The translator can move the edit control more right to make room for the longer label but this will take some time and will cost. Remember if you localize to 10 different languages most likely every single translator has to do the same modification that will take a lot of time and even more important make your UI to look different on each language. A better approach is to design the original UI such way the translators need hardly ever relocate controls. In this case we can place the label and text box on different lines - label above the text box. This gives label possibility to expand very much without overlapping the text box control.

In the case #2 control's AutoSize is set to false and the width is too short. Keep the AutoSize property always set to true.

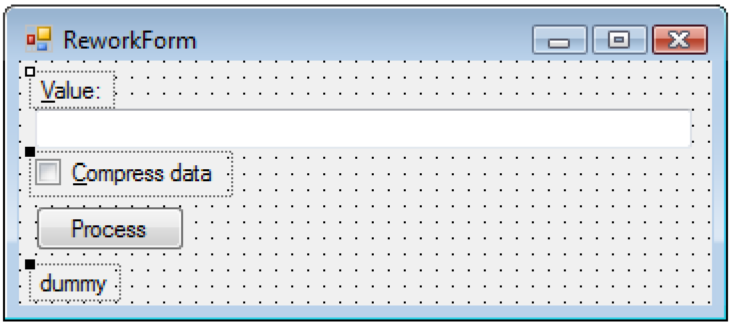
In the case #3 AutoSize property is not set true. Keep the AutoSize property of Label turned on. This makes sure that longer translation of the Text property is not cut when drawn on the form. Label2 control needs another fix. Label2.Text is set to Label2 on design time. On run time this is replaced with "Click the above button to process data" string. So the original string is not used at all. Keeping it will only make your localization project larger and more expensive. A good practice is to set all these Text property some fixed value such as "dummy". After creating Sisulizer project you can easily exclude all strings having "dummy" value.

The first step in internationalizing a form is to turn Localizable property true. This will make Visual Studio to store all property values of the form and its components to the resource file (.resx) instead of hard coding them to the source code.



IsMdiContainer	False
KeyPreview	False
Language	(Default)
Localizable	**True**
Location	0; 0
Locked	False
MainMenuStrip	(none)

The following picture contains the same sample form after it has been prepared for localization. Label and text box are not on the same line any more. Check box’s and Label's auto sizing has been turned on. Unused strings have been replaced with *dummy* word.



WPF

To be written.

Silverlight

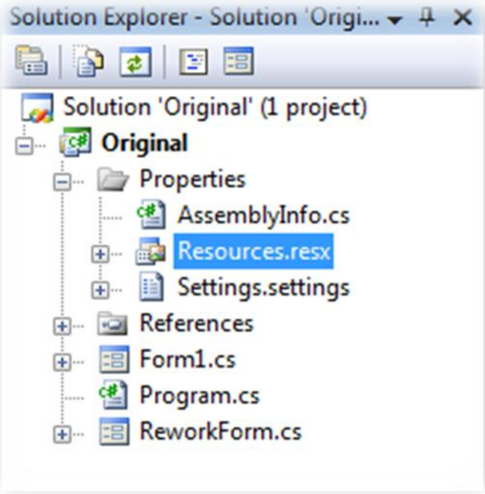
To be written.

Hard Coded Strings

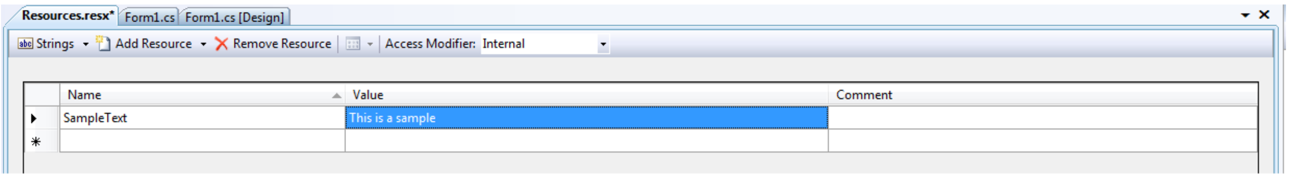
Most applications have hard coded strings inside the code. Some of them must not be translated. Some of them should be translated. If you want to localize such a string you have to remove them and replace them by resource strings. This is called resourcing. Take a look at the following code. It contains a hard coded string that is used to set Caption property of label.

```
private void Form1_Load(object sender, EventArgs e)
{
  label1.Text = "This is a sample";
}
```

To resource this string open Resources.resx file in the Solution Explorer.



A resource editor appears. Enter SampleText in the Name field and copy the text from the source code to the Value field.



Access Modifier: Internal

Name	Value	Comment
SampleText	This is a sample	

Now you have added the string in to string resource table. The final step is to remove the hard coded string from your code and replace it with resource string. Each item in the resource table gets a symbolic name that is a combination of assembly name, resource table name and item name.

```
private void Form1_Load(object sender, EventArgs e)
{
  label1.Text = Original.Properties.Resources.SampleText;
}
```

Code enabling

To be written.

Using Sisulizer

We have now internationalized our .NET project. It is time to localize it. The rest of the document is used to describe how to use Sisulizer to localize a .NET application. There are three different methods to localize a .NET application using Sisulizer. They are *project localization*, *assembly localization* and *resource file localization*.

Project Localization

When using project localization you add either Visual Studio project file (.csproj or .vbproj) or Visual Studio solution file (.sln) into Sisulizer project. Sisulizer scans the project or solution file to find out what resources files it uses. When performing build process for the Sisulizer project Sisulizer creates localized resource files and optionally compiles localized satellite assembly files. You need to have full project source file in order to use project localization. Let's have an example. We have Visual Studio C# project file *Converter.csproj*. Add that file to a Sisulizer project. When Sisulizer scans the project it founds that the project contains one form resource, *Form1.resx*, and one string resource *Properties\Resources.resx*. Sisulizer scans data from these resource files. If you add a new resource into your Visual Studio project there is no need to add that resource into Sisulizer project because the Visual Studio project is already there and it knows what resource files belongs to the project. Let's suppose that you added German and Japanese as target languages into the Sisulizer project. When Sisulizer performs build process it first creates the localized resource files that are *Form1.de.resx*, *Form1.ja.resx*, *Properties\Resources.de.resx*, and *Properties\Resources.ja.resx*. Finally Sisulizer uses the localized resource file to create localized satellite assembly files: *bin\Debug\de\Converter.resources.dll* and *bin\Debug\ja\Converter.resources.dll*.

Assembly Localization

If you don't have project source code you can use assembly localization where you add the assembly file or files of the application into Sisulizer project. When building the project Sisulizer creates localized satellite assembly files. Both project and assembly localizations produce the same output files. Project localization also creates localized resource files (.resx and .xaml). When using assembly localization you need to have all the assembly files used by the assembly file that you localize. For example if you application uses two assembly DLL files you need to have them on the same directory as the assembly that you plan to localize. Let's have an example. We have assembly file *Converter.exe*. Add that file to a Sisulizer project. When Sisulizer scans the project it reads the form resource, *Converter.Form*, and string resource, *Converter.Properties.Resources*. When Sisulizer performs build process it firstly creates localized resource data and then uses them to create localized satellite assembly files: *de\Converter.resources.dll* and *ja\Converter.resources.dll*.

When using assembly localization you need to have the assembly file that you want to localize (e.g. Converter.exe) and all assembly files (.dll) that the assembly file uses.

Resource Localization

If you don't have project source code files and you don't have compiled assembly files either but you have original resource files (either .resx or .xaml) you can use resource file localization. Add the resource file into the Sisulizer project. When Sisulizer builds the project it creates localized resource file. Then it is up to the developer of the application to compile localized satellite assembly files. Let's have an example. We have a resource file Form1.resx. Add that file to a Sisulizer project. When Sisulizer scans the project it reads the form resource file. When Sisulizer performs build process it creates the localized resource files that are *Form1.de.resx*, *Form1.ja.resx*.

I recommend using the project localization. If you do not have source codes then use assembly localization. Use resource file localization only as the last option in the case you do not have project source codes or assembly files but only resource files.

Silverlight Localization

When you deploy a Silverlight application you package assembly file(s) and manifest file into a XAP file. It is a zip-compressed file that contains assembly files of the application and a manifest file that contains information about the application. XAP file can also contain satellite assembly files. It can even contain satellite assembly files for several languages. When Visual Studio compiles a Silverlight application it creates the original XAP file. Add that file into Sisulizer project file. When Sisulizer performs build process it creates either one single multilingual XAP file containing satellite assembly files in several languages, or several XAP files each containing satellite assembly files in one language.

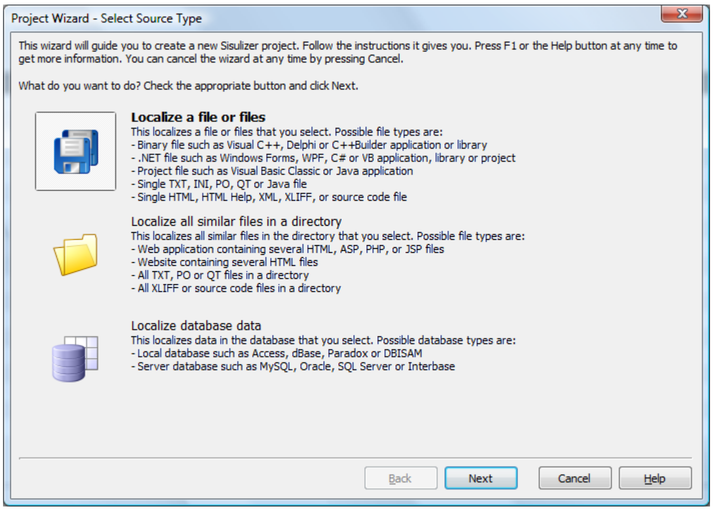
Framework and SDK

Your translator does not need to have .NET framework installed. However you must have at least .NET framework installed if you plan to use either project or assembly localization. If you use assembly localization you must have also .NET SDK installed. If you use project localization and you want that Sisulizer to create satellite assembly files you also must have .NET SDK installed. If you have Visual Studio you have both framework and SDK. If you do not have SDK you can download it from Microsoft's web pages.

Now we can through step by step how to create a new Sisulizer project containing a Visual Studio project.

Create project

Start Sisulizer and choose File | New to start Project Wizard. Click *Locale a file or files* and click Next.



This wizard will guide you to create a new Sisulizer project. Follow the instructions it gives you. Press F1 or the Help button at any time to get more information. You can cancel the wizard at any time by pressing Cancel.

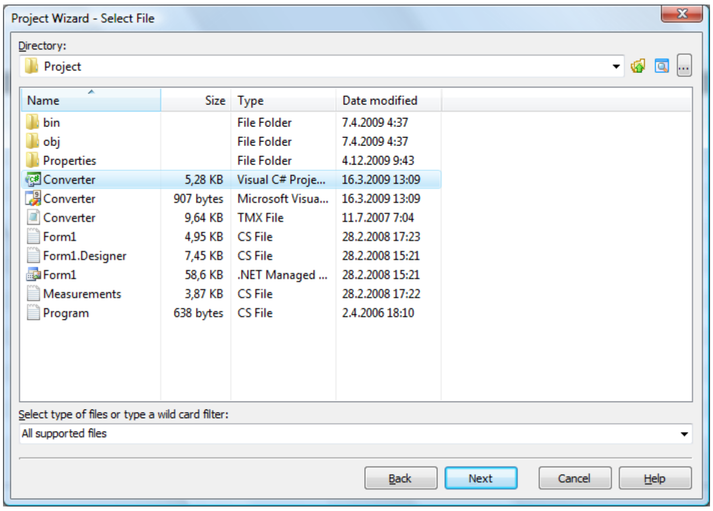
What do you want to do? Check the appropriate button and click Next.

Localize a file or files
This localizes a file or files that you select. Possible file types are:
- Binary file such as Visual C++, Delphi or C++Builder application or library
- .NET file such as Windows Forms, WPF, C# or VB application, library or project
- Project file such as Visual Basic Classic or Java application
- Single TXT, INI, PO, QT or Java file
- Single HTML, HTML Help, XML, XLIFF, or source code file

Localize all similar files in a directory
This localizes all similar files in the directory that you select. Possible file types are:
- Web application containing several HTML, ASP, PHP, or JSP files
- Website containing several HTML files
- All TXT, PO or QT files in a directory
- All XLIFF or source code files in a directory

Localize database data
This localizes data in the database that you select. Possible database types are:
- Local database such as Access, dBase, Paradox or DBISAM
- Server database such as MySQL, Oracle, SQL Server or Interbase

Select File page appears. Browse the directory where your Visual Studio project file (.csproj) is located and select that file. If you use Visual Basic select .vbproj file. You can also select Visual Studio solution file (.sln). Click Next.

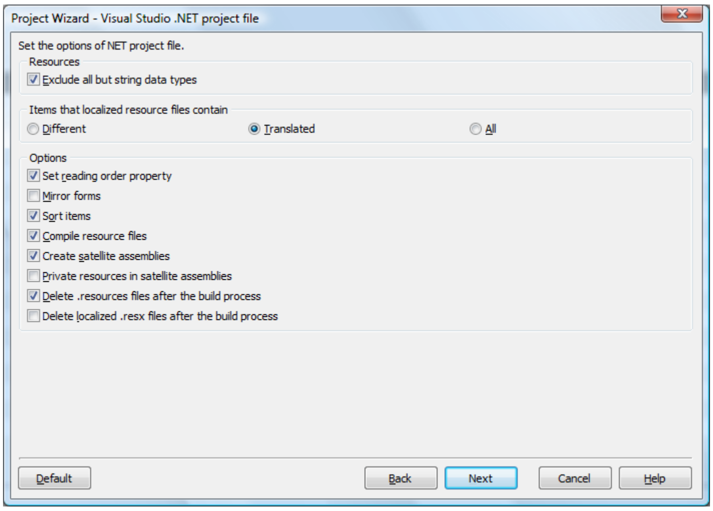


Directory: Project

Name	Size	Type	Date modified
bin		File Folder	7.4.2009 4:37
obj		File Folder	7.4.2009 4:37
Properties		File Folder	4.12.2009 9:43
Converter	5,28 KB	Visual C# Proje...	16.3.2009 13:09
Converter	907 bytes	Microsoft Visua...	16.3.2009 13:09
Converter	9,64 KB	TMX File	11.7.2007 7:04
Form1	4,95 KB	CS File	28.2.2008 17:23
Form1.Designer	7,45 KB	CS File	28.2.2008 15:21
Form1	58,6 KB	.NET Managed ...	28.2.2008 15:21
Measurements	3,87 KB	CS File	28.2.2008 17:22
Program	638 bytes	CS File	2.4.2006 18:10

Select type of files or type a wild card filter: All supported files

Visual Studio .NET project file page appears. This page contains Visual Studio project related options. Click Next.



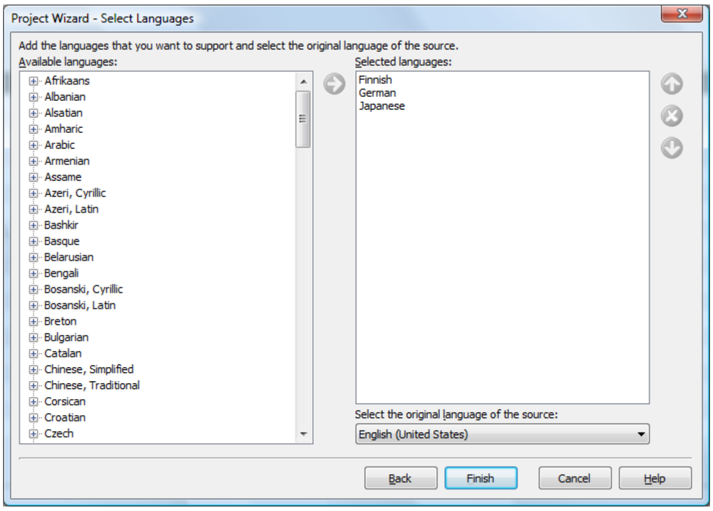
Set the options of NET project file.

Resources
- ☑ Exclude all but string data types

Items that localized resource files contain
- ○ Different
- ● Translated
- ○ All

Options
- ☑ Set reading order property
- ☐ Mirror forms
- ☑ Sort items
- ☑ Compile resource files
- ☑ Create satellite assemblies
- ☐ Private resources in satellite assemblies
- ☑ Delete .resources files after the build process
- ☐ Delete localized .resx files after the build process

Select Languages page appears. Use this page to select the language of the original file (in most cases Sisulizer detects it) and to select the target language(s). You can later change the original language and add more target languages or remove existing ones.

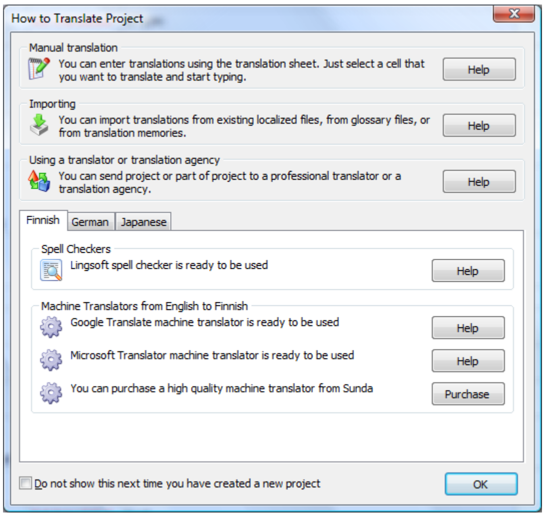


Add the languages that you want to support and select the original language of the source.

Selected languages:
- Finnish
- German
- Japanese

Select the original language of the source: English (United States)

Click Finish to finish the wizard. Sisulizer creates a new project, add the application file into the project and finally scans the project to find resource items from the application file. Before show the project Sisulizer shows the following dialog that gives you more information about how to translate the project.



Manual translation

You can enter translations using the translation sheet. Just select a cell that you want to translate and start typing.

Importing

You can import translations from existing localized files, from glossary files, or from translation memories.

Using a translator or translation agency

You can send project or part of project to a professional translator or a translation agency.

Spell Checkers

Lingsoft spell checker is ready to be used

Machine Translators from English to Finnish

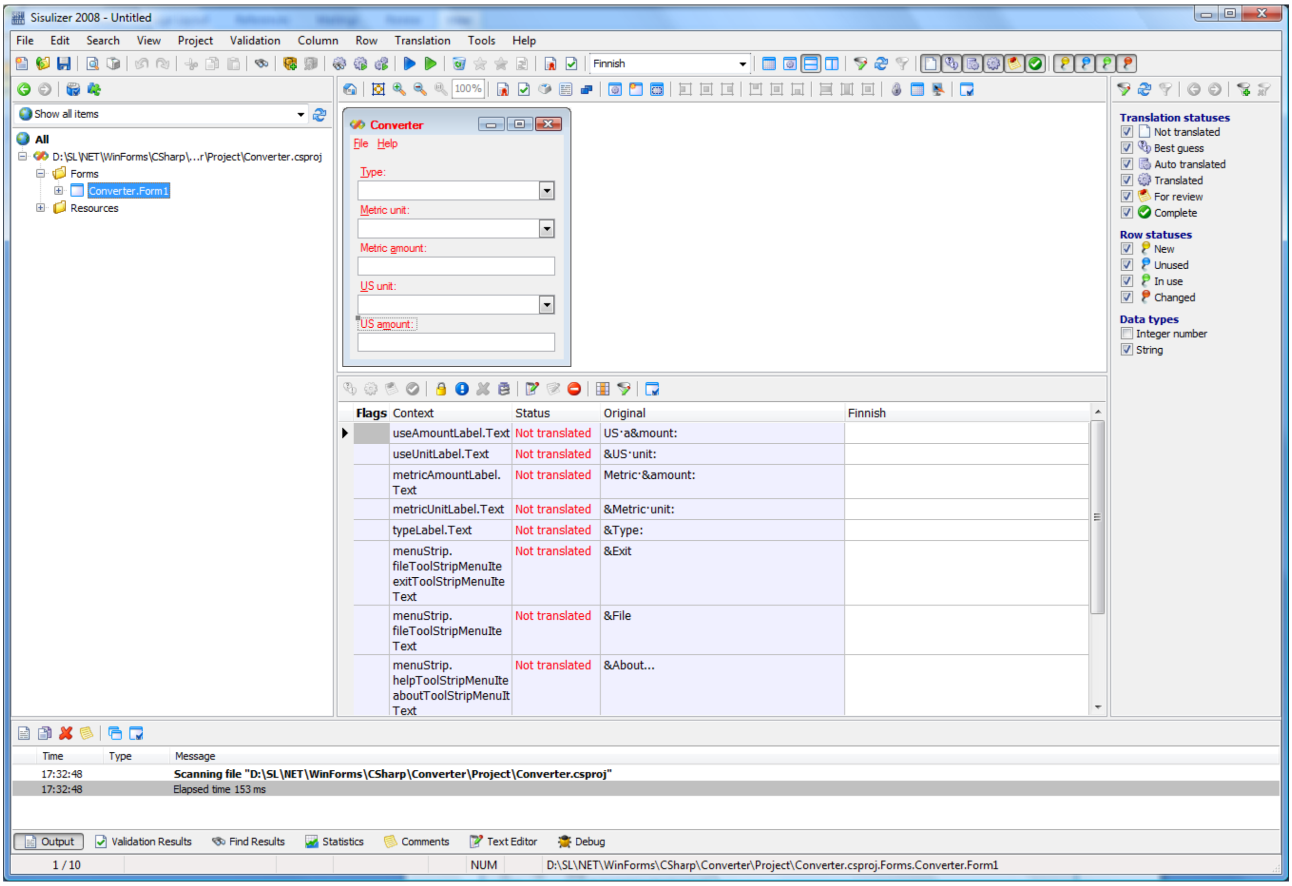
Google Translate machine translator is ready to be used

Microsoft Translator machine translator is ready to be used

You can purchase a high quality machine translator from Sunda

☐ Do not show this next time you have created a new project

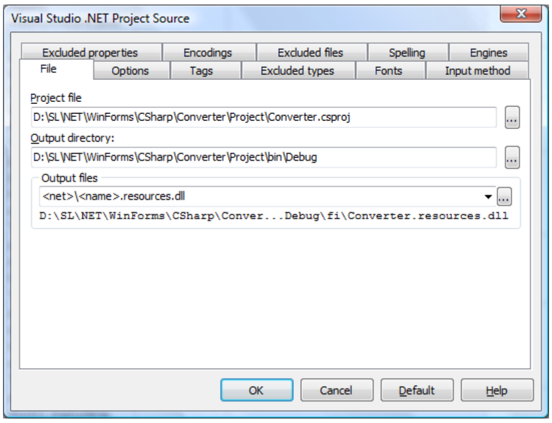
Click Ok to close the dialog and to show the project.



Flags	Context	Status	Original	Finnish
	useAmountLabel.Text	Not translated	US·a&mount:	
	useUnitLabel.Text	Not translated	&US·unit:	
	metricAmountLabel.Text	Not translated	Metric·&amount:	
	metricUnitLabel.Text	Not translated	&Metric·unit:	
	typeLabel.Text	Not translated	&Type:	
	menuStrip.fileToolStripMenuIte exitToolStripMenuIte Text	Not translated	&Exit	
	menuStrip.fileToolStripMenuIte Text	Not translated	&File	
	menuStrip.helpToolStripMenuIte aboutToolStripMenuIt Text	Not translated	&About...	

```
Time      Type   Message
17:32:48         Scanning file "D:\SL\NET\WinForms\CSharp\Converter\Project\Converter.csproj"
17:32:48         Elapsed time 153 ms
```

Sisulizer project can contain any number of files. A file that has been added to the project file is called *source*. To add a new source right click All in the project tree and choose *Add Source*. You can configure the source by right clicking the file name in the project tree and choosing Properties. The following dialog is shown.



Project file: D:\SL\NET\WinForms\CSharp\Converter\Project\Converter.csproj

Output directory: D:\SL\NET\WinForms\CSharp\Converter\Project\bin\Debug

Output files: <net>\<name>.resources.dll

D:\SL\NET\WinForms\CSharp\Conver...Debug\fi\Converter.resources.dll

Use the dialog to set the localization options such as output directory, resources to be localized, etc. To get detailed information about the dialog select the sheet and click Help. Now we have created a project for our application. The next step is to translate it.

Translate

Sisulizer provides several ways to translate projects. You can translate manually by entering in the project sheet. The sheet works in the same way as Excel so if you are familiar to Excel you learn very quickly to use it. If you are not familiar to Excel it is easy to learn how to use the sheet. Just use mouse select a cell where you want to enter translation and start typing. You can also use arrow key to select a cell.

Another powerful way to translate projects is to import translations. Sisulizer can import translation from several sources such already localized files, glossary files (TMX, Excel, TXT, CSV) or from databases. You can import data to project by choosing File | Import. It starts Import Wizard that lets you to select the source and import options. You can also import to single language column by choosing Column | Import.

Sisulizer has build in support for translations memories. They are storages that keep existing translations. You can save existing translation from project to translation memory by choosing File | Save to Translation Memory. You can make translation memory to translate the project by choosing Project | Translate Using Translation Engine.

Final way to translate a project is to send it to translator. This is very easy in Sisulizer. Choose Project | Exchange. It lets you to create a file to be sent to translator. When translator has translated the file he or she will send it back to you and you can import translation from the file by choosing Project | Import.

You can find more information about translating the project from Sisulizer online help.

Build

Last step in the translation process is building localized files. Sisulizer takes care about this. It reads the original files merges the translations from the project file to create localized resource files and finally creates localized satellite assembly file. You have two ways to build the files. The first one is to open Sisulizer project and to choose Project | Build All. This makes Sisulizer to build localized files for all languages in the project file. If you want to build files for one language select the language and choose Project | Build Item. If you have Sisulizer Enterprise edition you can also use Sisulizer’s command line tool, SlMake.exe, to build localized files. Go to Sisulizer directory and type SlMake on command line and press Enter to learn about SlMake. By using SlMake you can integrate Sisulizer into your make process.

Now we have covered the basic Sisulizer localization process. The remaining chapters will cover some advanced topics about Sisulizer.

Component mapping

To be written.

Comments and other additional information

To be written.

Runtime language switch

To be written.

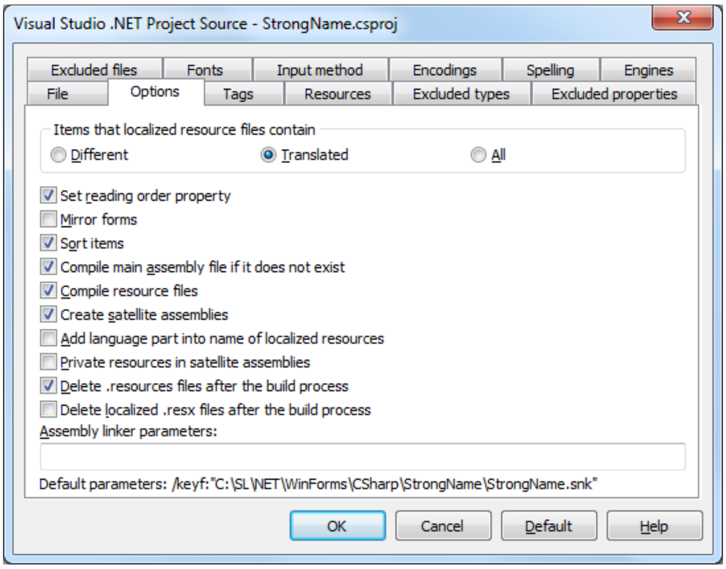
Signing

If an original assembly is signed then also the satellite assembly files must be signed with the same key in order to make them work. There are two kind of key files: SNK and PFX.

You can create a SNK file using SN.exe tool

```
sn -k keyPair.snk
```

Our sample, <sisulizer-data-dir>\NET\WinForms\CSharp\StrongName\StrongName.csproj, has been signed using a SNK key. Sisulizer automatically detects if the assembly has signed and uses the same key to sign the satellite assembly files. To check that Sisulizer correctly detected your key open your Sisulizer project and right click the application project file (e.g. StrongName.csproj) in the project tree. Select Options sheet and check if Default parameters contains /keyf value.



Items that localized resource files contain

○ Different ● Translated ○ All

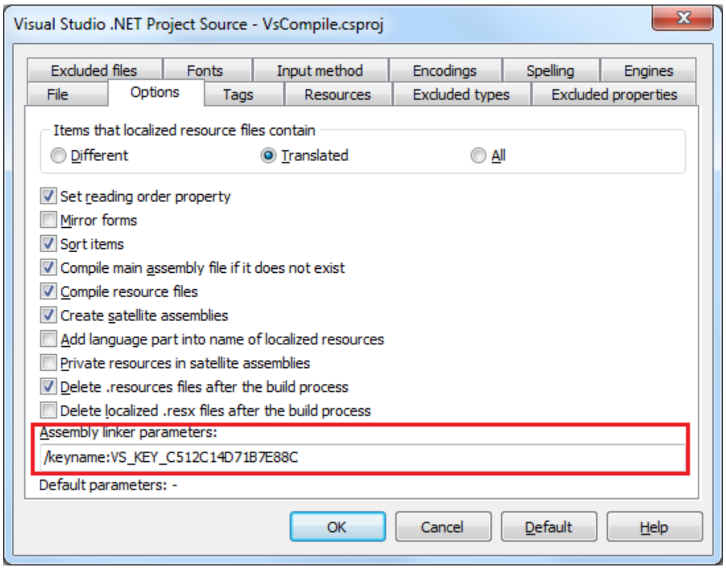
☑ Set reading order property
☐ Mirror forms
☑ Sort items
☑ Compile main assembly file if it does not exist
☑ Compile resource files
☑ Create satellite assemblies
☐ Add language part into name of localized resources
☐ Private resources in satellite assemblies
☑ Delete .resources files after the build process
☐ Delete localized .resx files after the build process

Assembly linker parameters:

Default parameters: /keyf:"C:\SL\NET\WinForms\CSharp\StrongName\StrongName.snk"

If the /keyf value is not there right click Assembly linker parameters edit. Choose *Add key file* and browse the key file (e.g. StrongName.snk).

If you use a PFX key Sisulizer cannot automatically pick the parameter. Instead you have to add the key name parameter into the assembly linker parameters. The following picture shows how to give a key name.



Items that localized resource files contain

○ Different ● Translated ○ All

☑ Set reading order property
☐ Mirror forms
☑ Sort items
☑ Compile main assembly file if it does not exist
☑ Compile resource files
☑ Create satellite assemblies
☐ Add language part into name of localized resources
☐ Private resources in satellite assemblies
☑ Delete .resources files after the build process
☐ Delete localized .resx files after the build process

Assembly linker parameters:

/keyname:VS_KEY_C512C14D71B7E88C

Default parameters: -

You can get the name of the key by browsing key container of your PC.

Once you have configured the key settings Sisulizer always signs all files it creates. This also happens on Sisulizer command line tool, SlMake. Add following line to your makefile to create and sign the localized satellite assembly files.

```
SlMake create StrongName.slp
```

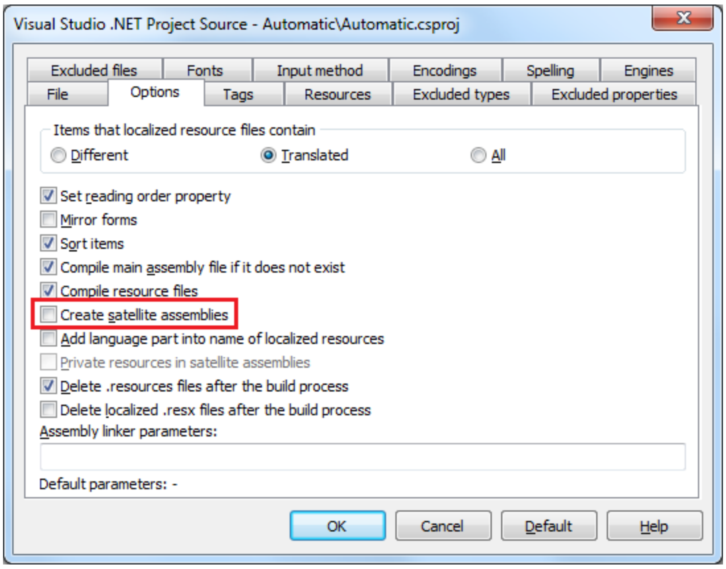
If you localize an assembly file instead of Visual Studio project file Sisulizer cannot pick the key name. You must enter either key file (/keyf) or key name (/keyn) parameter.

ClickOnce deployment

ClickOnce is a .NET technology that enables the user to install and run a .NET application by clicking a link in a web page. Visual Studio supports ClickOnce. In Solution Explorer, right click the application project and choose Publish. The Publish Wizard appears. The wizard asks the deployment options. By default the deployment package contains the application assembly and any library assembly it uses. However it does not contain satellite assembly files created by Sisulizer. If you want to include satellite assembly files you have two choices.

1. Turn off Sisulizer's satellite assembly creation option and let Visual Studio to compile the satellite assembly files.
2. Do not use Visual Studio to create the deployment files but use Mage.exe or MageUI.exe tools to create them.

Let's first look at the option #1. Open Sisulizer project. In Project Tree right click the application project and choose Properties. The Source dialog appears. Select Options sheet and uncheck *Create satellite assemblies.* Now Sisulizer create only the localized .resx and/or .xaml files but does not create the satellite assembly files (e.g. de\Automatic.resources.dll).



Items that localized resource files contain

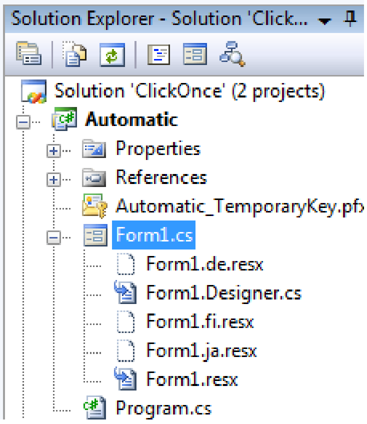
○ Different ● Translated ○ All

☑ Set reading order property
☐ Mirror forms
☑ Sort items
☑ Compile main assembly file if it does not exist
☑ Compile resource files
☐ Create satellite assemblies
☐ Add language part into name of localized resources
☐ Private resources in satellite assemblies
☑ Delete .resources files after the build process
☐ Delete localized .resx files after the build process

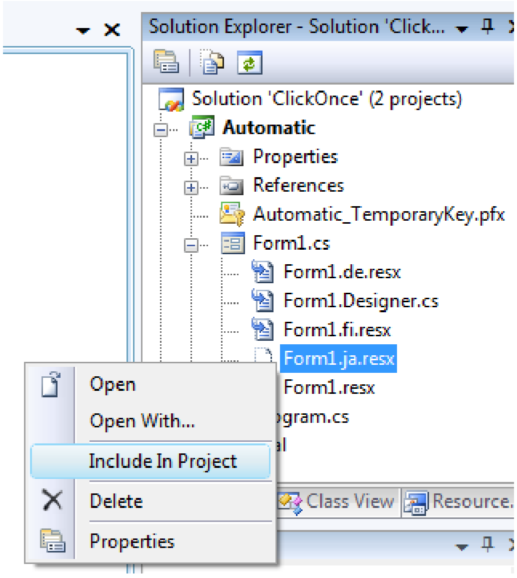
Assembly linker parameters:

Default parameters: -

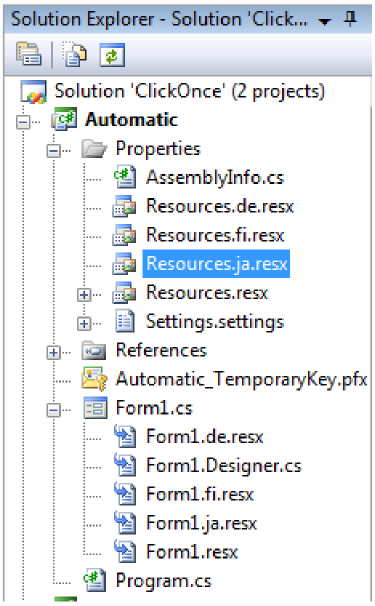
The next step is to configure Visual Studio to compile the satellite assembly files. Once you have used Sisulizer to create the localized resource files you can see them from Visual Studio's Solution Explorer. By default they are not included into the project t(icon is white page).



Select a resource file, right click it and choose *Include In Project*. Visual Studio includes the resource files into the project. When included the icon of the file turns in to normal.



This also makes Visual Studio to compile the satellite assembly file. You have to remember to include every single resource file created by Sisulizer. If the localized resx file is not visible in the Solution Explorer click *Show all Files* button on the Solution Explorer toolbar. If you don't the satellite assembly is not created or it misses the resources of the resource file. The following picture shows Automatic sample where all Finnish, German and Japanese resource files have been included into the project.



This is all you have to do. Next time you publish your project the package contains satellite assembly files. *<sisulizer-data-dir>\NET\WinForms\CSharp\ClickOne\Automatic* directory contains a sample that uses this approach to deploy itself.

Method #1 seems easy but gets complicated if the amount resource files get high. In that case we recommend manually create the deployment files (method #2). MSDN contains a detailed article about this. Search "Walkthrough: Manually Deploying a ClickOnce Application" article form MSDN. However the article does not cover satellite assembly files. We will now go through the steps needed to manually deploy your project. *<sisulizer-data-dir>\NET\WinForms\CSharp\ClickOne\Manual* directory contains a sample that uses manual deployment. You will need Manifest Generation and Editing Tool. There are two version: Mage.exe is a command line tool, and MageUi.exe is a graphical client tool. These instructions are for the command line tool, Mage.exe.

1. Create a new directory where you put the files you want to localize. A good location is the output directory of your sample. In most cases it is *bin\Debug* or *bin\Release* directory. Create *bin\Debug\Deploy* directory.
2. Create a sub directory that contains your application assembly files, satellite assembly files, and other files that are needed. A good practice is to use the version as the directory name. Create *bin\Debug\Deploy\1.0.0.0* and copy there *Manual.exe*, *de\Manual.resources.dll*, *fi\Manual.resources.dll* and *ja\Manual.resources.dll*. Manual.exe is the application assembly file created by Visual Studio. The dll files are the satellite assembly files created by Sisulizer.
3. Open command line and go to the bin\Debug\Deploy\1.0.0.0 directory. Enter
   ```
   mage -New Application -ToFile Manual.exe.manifest -name "Manual"
   -Version 1.0.0.0 -FromDirectory "."
   ```
   This creates the application manifest file, Manual.exe.manifest.
4. Sign the manifest file by entering
   ```
   mage -Sign Manual.exe.manifest -CertFile ..\..\MyKey.pfx
   ```
   Now you have application manifest file done for your deployment version. You still have to create deployment manifest.
5. Go one directory up to bin\Debug\Deploy and enter
   ```
   mage -New Deployment -Install true -Publisher "Sisulizer" -
   AppManifest 1.0.0.0\Manual.exe.manifest -ToFile Manual.application
   ```
   and
   ```
   mage -Sign Manual.application -CertFile ..\MyKey.pfx
   ```
   This will create and sign the deployment package.
6. Finally deploy bin\Debug\Deploy directory into your deployment location.

See *<sisulizer-data-dir>\NET\WinForms\CSharp\ClickOne\Manual\bin\Debug* to see how the files are created and copied.

Online help, databases, data files and web pages

To be written.