

SICOMPONENTS®

---

TsiLang® Components Suite

# Developer's Guide

TSILANG® COMPONENTS SUITE

# Developer's Guide

---

Copyright © 1998-2021 Igor Sitikov, SiComponents <https://www.sicomponents.com>.

TsiLang®, SiComponents® and Resource Builder® are registered trademarks or trademarks of Igor Sitikov.

Other product and brand names are trademarks of their respective owners.

# Table of Contents

<b>INTRODUCTION</b>	<b>6</b>
<b>Welcome!</b>	<b>6</b>
<b>How to Use This Manual</b>	<b>8</b>
<b>Contact Information</b>	<b>9</b>
<b>GETTING STARTED</b>	<b>10</b>
<b>Installation</b>	<b>10</b>
Trial Version	10
Registered Version	11
<b>Components Review</b>	<b>12</b>
Main Components	12
Dispatcher	13
<b>Tutorial</b>	<b>15</b>
Which components to use?	15
Setting properties	15
Using siDialogs	16
Delphi:	16
“Hard-Coded” Strings	16
C++ Builder:	17
Language Switching	17
Delphi:	18
C++ Builder:	18
Editing Translation Data	18
<b>Translations Editor</b>	<b>18</b>
<b>SIL Editor</b>	<b>19</b>
<b>COMMON TASKS</b>	<b>20</b>
<b>Using TsiLang Expert</b>	<b>20</b>
Using Translation Wizard	21
Search for Hard-Coded Strings	22
Using TSI:IGNORE tags	24
Working with External Files	25
Other Functions	25
Checking Identifiers	25
Search and Replace	26
Excluding Properties	26
Clearance Translations	26
TsiLang Type Changing	26
Expert Options	26
General Options	27
Source Strings	27

Exclude & Skip	29
Save & Load	29
<b>Using Translation Editor</b>	<b>30</b>
<b>Using Extended Translations</b>	<b>34</b>
<b>Using ExtendedTranslations under different DPIs</b>	<b>36</b>
<b>Using Translations Stored in External Files</b>	<b>38</b>
<b>Using Exclude from Translations Editor</b>	<b>40</b>
Components to Exclude	40
Properties to Exclude	41
Component's Properties to Exclude	42
<b>FireMonkey Support</b>	<b>44</b>
<b>Linux Support</b>	<b>46</b>
<b>Using Translation Memory</b>	<b>47</b>
<b>EXTERNAL TOOLS</b>	<b>48</b>
<b>Dictionary Manager</b>	<b>48</b>
Dictionary Manager Automation Server	49
<b>SIL Editor</b>	<b>51</b>
Displayed Properties	51
Fixed Languages	51
Automation Server	52
<b>Resource Strings Wizard</b>	<b>54</b>
Translate resource strings by ID	56
<b>COMPONENTS REFERENCE</b>	<b>57</b>
<b>Core Components</b>	<b>57</b>
TsiLang	57
Properties	57
Methods	62
Methods to work with SIL files	63
Methods to work with SIB files	65
Methods to work with streams	66
Methods to work with message boxes	66
Other methods	67
Events	68
TsiLangLinked	68
TsiLangRT	68
Methods	69
Properties	69

TsiLangRTSE	70
Properties	70
Methods	70
TsiLangTLV	70
Properties	71
TsiLangDispatcher	71
Methods	72
Properties	72
Events	74
TsiInternetTranslator	74
Methods	75
Properties	76
TsiLangCombo	78
Properties	79
<b>Dialogs</b>	<b>80</b>
Properties	80
<b>USEFUL INFORMATION</b>	<b>81</b>
<b>Tips and Tricks</b>	<b>81</b>
Exclude not used components and properties	81
Use TSI: tags to skip hard-coded strings that should not be translated	82
Create multilanguage dialog boxes with custom controls	82
Creating Unicode multilanguage dialog boxes	83
Configuring Default Fonts for TsiLang	83
Performing custom modifications during language changing	84
Translating 3 <sup>rd</sup> party forms without sources	85
How to make TsiLang message boxes “styled” when VCL Style applied.	86
<b>TranslationData as Text in DFM</b>	<b>88</b>
<b>Frequently Asked Questions</b>	<b>89</b>
How to translate resource strings?	89
I am using C++Builder. How do I translate string tables coded into .rc and .rh files?	89
Why some of my string constants don’t appear in found strings form when translating sources?	89
How to translate TDBGrid columns?	90
How to translate InfoPower’s DBGrid component?	90
How to translate arrays of strings	90
How to translate TActionMainMenuBar or TActionToolBar	90
How to modify button widths in standard dialogs?	91
Are TsiLang components compatible with IntraWeb?	91
Is it possible to translate menu shortcuts?	92
Why TDBNavigator hints are not translated at start-up?	92
Why Developer Express components translations are displayed incorrect under XP Theme enabled?	92
How to detect OS default language and switch to it?	92
How to properly load file at run-time?	93
Main menu gets white background after language switching	93
<b>Version History</b>	<b>95</b>
Version 7.9:	95

Version 7.8.5:	95
Version 7.8.4:	96
Version 7.8.3:	97
Version 7.8.2:	98
Version 7.8.1:	98
Version 7.8:	98
Version 7.7:	99
Version 7.6.0:	100
Version 7.5.9:	100
Version 7.5.8:	100
Version 7.5.7:	100
Version 7.5.6:	101
Version 7.5.5:	101
Version 7.5.4:	101
Version 7.5.3:	101
Version 7.5.2:	101
Version 7.5.1:	101
Version 7.5:	102
Version 7.4:	102
Version 7.3.3:	102
Version 7.3:	102
Version 7.2:	102
Version 7.1.1	102
Version 7.1	103
Version 7.0	103
Version 6.5.5	103
Version 6.5.4	103
Version 6.5.3	104
Version 6.5.2	104
Version 6.5	105
Version 6.4	105
Version 6.3	106
version 6.2	107
version 6.1	107
version 6.0.3	108
version 6.0.2	109
version 6.0.1	110
version 6.0	111
version 5.3.2	112
version 5.3.1	112
version 5.3.0	113
version 5.2.4	114
version 5.2.3	114
version 5.2.2	114
Version 5.2	114
Version 5.1	115
Version 5.0	116

**LIST OF TABLES 118**

**LIST OF FIGURES 119**





## Introduction

### Welcome!

The problem of globalization is sooner or later arising for all software companies and developers, who intend to distribute their applications worldwide and wish to create localized versions for the different foreign markets. Using the *TsiLang Components Suite* for the applications developed in the framework of Embarcadero RAD Studio, Delphi™ or C++Builder™ is the best solution of the problem in many aspects.

*TsiLang Components Suite* includes a number of highly professional, easy-to-use VCL components, wizards and tools for building multilingual applications. From the one hand our product radically simplifies the entire process of globalization and from the other hand it allows you to provide your customers with elegant, user-friendly applications that can switch from one locale to another on-the-fly. Some *TsiLang Components Suite* features that make it's a distinguished product are the following:

- Our suite brings you the opportunity to localize all that should be localized: string-type properties of components, “hard-coded” string constants, resource strings, system locales’ and standard dialog’s strings, as well as other properties that may affect on visual appearance of controls (for example, geometrical sizes and positions of labels can be made different for each language when it’s necessary).
- You have the choice where to store the translation data: either internally (ideally suits for lightweight applications – single EXE file, no DLLs, databases, etc.) or externally in special files (this approach allows to change translations without recompiling the project so even your end-user may update translations in the run-time).
- Such essential tools as *TsiLang Expert* and *Resource Strings Wizard* included in the suite help you to manage the translation process of you projects by simple and convenient way. So, a large project might be prepared for the globalization literally in a few minutes. The *TsiLang Expert* adds TsiLang components to the project’s forms and data modules, adjusts their properties, and scans the project files for “hard-coded” strings. The *Resource Strings Wizard* allows you to extract and add to translation data the resource strings of your project even they are buried somewhere in a .dcu (.obj) without the source.



- A specialized tool shipped with the suite, namely the *SIL Editor* can be freely redistributed and allows you to separate the translation process from the project development.
- And the *Dictionary Manager* helps to store common translations such as dialog's captions and frequently used phrases that can be repeatedly used in many projects. Also this tool serves as an automation server so its data can be easily loaded by the *SIL Editor* or TsiLang property editors.

Since the first version release in 1998 the TsiLang Components Suite has been continuously developing and improving, so today it represents a mature product with the excellent ratio quality/price. The evidence of growing its popularity and customer satisfaction are the readers' choice awards "Best Globalization Tool" in 2004, "Runner-up Best Globalization Tool" in 2003 and 2002 by the Delphi Informant Magazine.

## How to Use This Manual

If you are using the *TsiLang Components Suite* for the first time we recommend you to read thoroughly the Chapter 2, which contains instructions on the installation of the suite, the brief review of core components, and tutorial on internalization of a sample application.

Chapter 3 describes the most common tasks in the process of working with the Suite: using the TsiLang Expert for automation routine work; editing translation data and using extended translations.

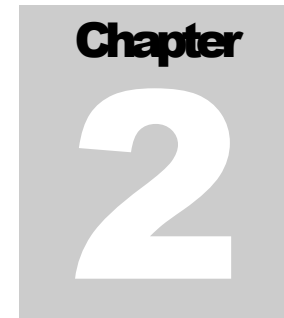
Chapter 4 contains guides on using external tools included in the suite: Dictionary Manager which holds and manages common translations databases; SIL Editor which used for editing translation data in .SIL (.SIB) files; Resource Strings Wizard which helps to import and use resource strings.

Chapter 5 describes in detail all components used in the Suite.

And Chapter 6 contains additional information that could be useful to you: tips and tricks, FAQ and version history.

## Contact Information

If you have any questions, suggestions to improve the Suite, or bug report feel free to contact us via e-mail [support@sicomponents.com](mailto:support@sicomponents.com) or our web site <https://www.sicomponents.com/contacts/>. For the most current information and updates of TsiLang Components Suite visit product's web site: <https://www.tsilang.com>.



## Getting Started

### Installation

The TsiLang Components Suite consists of the following parts:

1. VCL libraries containing all required components, property editors and the *TsiLang Expert*. The trial version includes compiled packages and units whereas the registered version all necessary source files (in case Full Source license purchased) or compiled version of registered sources.
2. The *SIL Editor* is a convenient tool for editing translation data in external files. This tool can be freely redistributed so you can separate the project development and translation processes.
3. The *Dictionary Manager* is a standalone application for storing and managing common translation databases. This program is widely used by other tools and helps automate the translation process.
4. The *Resource Strings Wizard* - tool for importing resource strings from executables.
5. The *INI File Strings Wizard* - tool for importing strings from INI File.
6. The *SI2DFM Wizard* – tool for loading SIL/SIB files into DFM without IDE.
7. Demo Applications - a few sample projects that illustrate using the Suite.
8. Help Files.

#### Trial Version

- Close all running instances of RAD Studio, Delphi or C++ Builder.
- Run installation application called TSILANGTRIALSETUP.EXE and just follows the instructions.
- Once the installation script is completed the *TsiLang Components Suite* will be automatically installed into selected IDEs.

## Registered Version



**Note:** If you have previously installed a trial copy of the TsiLang Components Suite, then uninstall it from your PC. Make sure all its files are deleted completely. Also please be sure to uninstall previous registered version in case of upgrading to newest one.

- Close all running instances of RAD Studio, Delphi or C++ Builder.
- Run installation application called SISETUP.EXE (or SIBINARY.EXE in case of DCU-Only license) and follows the instructions. The installation script will install source files to the destination folder and adjust library path for the selected IDEs. Install script needs license key to complete. This key is included in REGKEY.KEY file located in the same ZIP archive as installation EXE. Please be sure to place this file into the same folder where installation EXE located or just double-click on it to merge it into the Registry.
- Once installation is completed you will find components installed into selected IDEs.



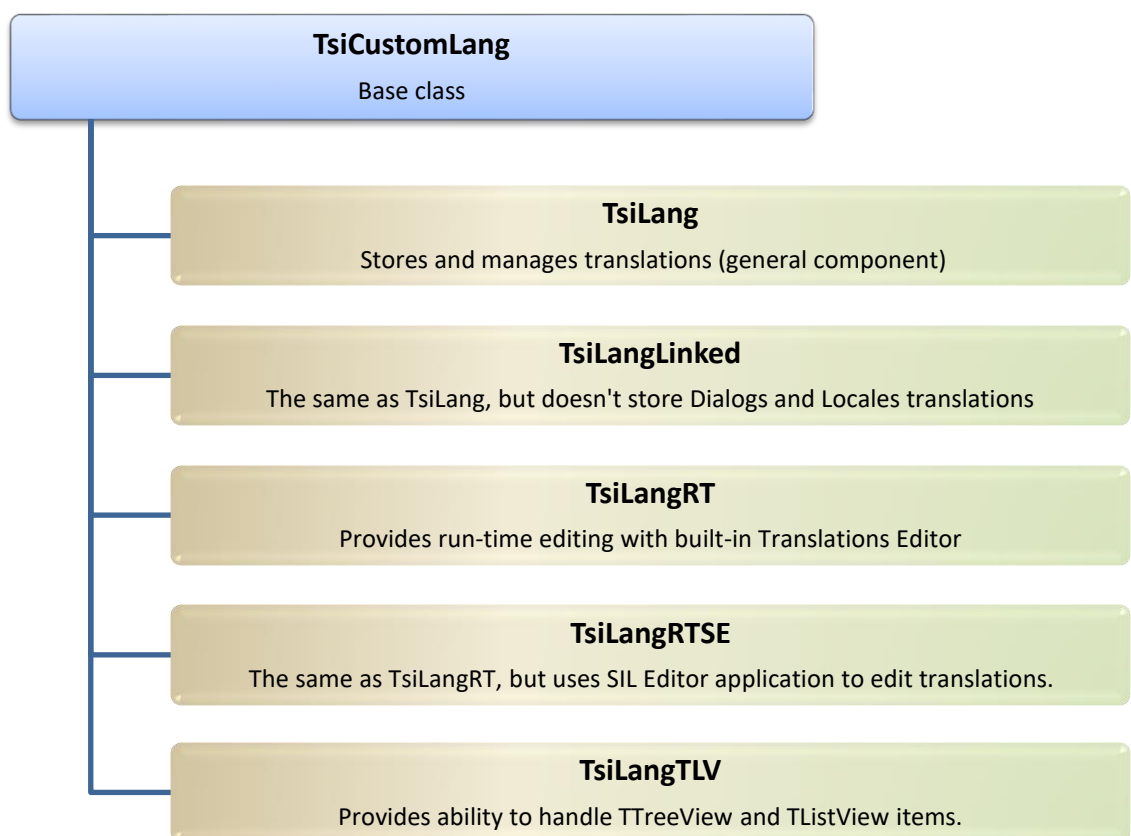
**Note:** In case of error message about loading TsiLang package in IDE startup, please try to copy TsiLang\_XXr.bpl file into [XX]\Projects\BPL folder (where XX- is your Delphi or C++Builder version).

## Components Review

In the base of the *TsiLang Components Suite* there are several classes built on VCL technology. These classes provide the storage for the application's data to be translated as well the methods for managing those translations.

### Main Components

These classes like many other ones are VCL components and are being registered in the IDE palette during the installation (by default on the “SiComponents” page).



**Figure 1** Core components hierarchy

The above figure represents the five main components that are derived from the TsiCustomLang component. Hereinafter we will call all these types “TsiLang” if not specified evidently. All these components have the ability to hold and handle the following data:

- The string-type properties of components such as “Caption”, “Hint”, “DisplayLabel”, and other particular string-based properties like “HelpFile” for TForm or “SimpleText” for TStatusBar.

- The TStrings-type properties of components like “Items” for TComboBox and TListBox, or “Lines” for TMemo, etc.
- “Hard-Coded” string constants in the source.



**Note:** If you explicitly declare string constants in **const** sections, your project should allow assignments to typed constants. This option can be set via “Project | Options | Compiler | Assignable typed constants” or by the conditional `{ $\$J+$ }` in the project source code. Or you can just change declaration from **const** to **var**.

- Strings used in Standard Dialogs and such functions as MessageDlg, InputBox, and other.



**Tip:** *TsiLang Components Suite* also contains the set of dialog components (by default they are installed on the “**SiDialogs**” palette page. These components are identical to standard dialogs except the “**siLang**” property, which links the dialog with translation data storage. We recommend using the SiDialogs components instead of the standard ones, so that your dialogs would display strings according to your application’s active language.

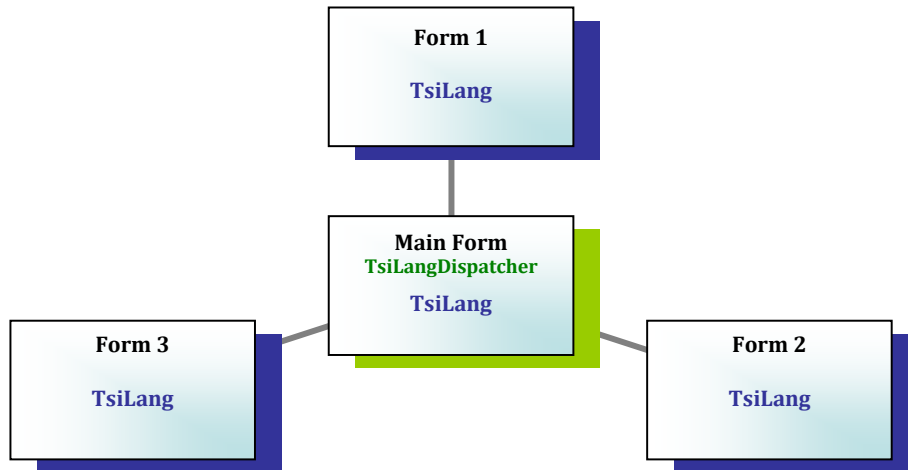
- System-wide locale settings, in particular months and weekdays names.
- All other non-string properties of components that may affect visual appearance of your application. For example, labels on a form might have different positions and sizes for every supported language.
- For the best results fonts and charsets might alter for different languages.

TsiLang components have no limitations on the number of supported languages and give you the choice where to store the translation data: either internally in the executable (since translations are managed as published properties of the TsiLang component, they are stored in the respective .dfm/.nfm file) or in external files (TsiLang components have corresponding methods to save and load external data).

The “LangNames” property of TsiLang components contains the list of supported languages and the “ActiveLanguage” property corresponds to 1-based index of the current language among of supported ones. Therefore, to switch from one language to another, it is enough to assign respective index number to the TsiLang “ActiveLanguage” property (both in design- and run-time).

## Dispatcher

Every form of the application must contain a TsiLang component in order to be properly displayed for each supported language. However, for large projects with many forms it is quite a complicated task to handle all TsiLang components individually. The special component named “TsiLangDispatcher” is intended to simplify multiple TsiLang components management through centralized control.



**Figure 2** Multiple TsiLang components are linked to a single TsiLangDispatcher.

All TsiLang components have “LangDispatcher” property that allows them to be linked with a TsiLangDispatcher component placed on the main (or other “auto-created”) form of the application. Using TsiLangDispatcher reduces the code necessary for language switching in the whole application literally to a single line, like this:

```
...  
procedure TForm1.ChangeLanguageClick(Sender: TObject);  
begin  
    siLangDispatcher1.ActiveLanguage := TMenuItem(Sender).Tag;  
end;  
...  
...
```



## Tutorial

This tutorial takes you through the internationalization process of a sample application with a few forms. In order to minimize the spade-work we suggest using the sample “MDI Application” from the Delphi/C++ Builder repository (Select IDE menu “File | New | Other... | Projects | MDI Application”).

### Which components to use?

1. We should decide where to store translation data: either internally or externally. For the most applications the best results are reached with the usual TsiLang components that do not require additional files and provide fast language switching. So, simply place a TsiLang component on the application's main form.
2. Our application has other two forms. Although we can also use TsiLang-type components, it is often good practice to use TsiLangLinked components on the secondary forms. The latter ones have no such properties as “Dialogs” or “Locales” instead they retrieve the data from a common container (another TsiLang component). Place a TsiLangLinked component on every secondary form.
3. As our application has several forms so for the centralized management of TsiLang components it is convenient to use a TsiLangDispatcher component. Place a TsiLangDispatcher on the main form.

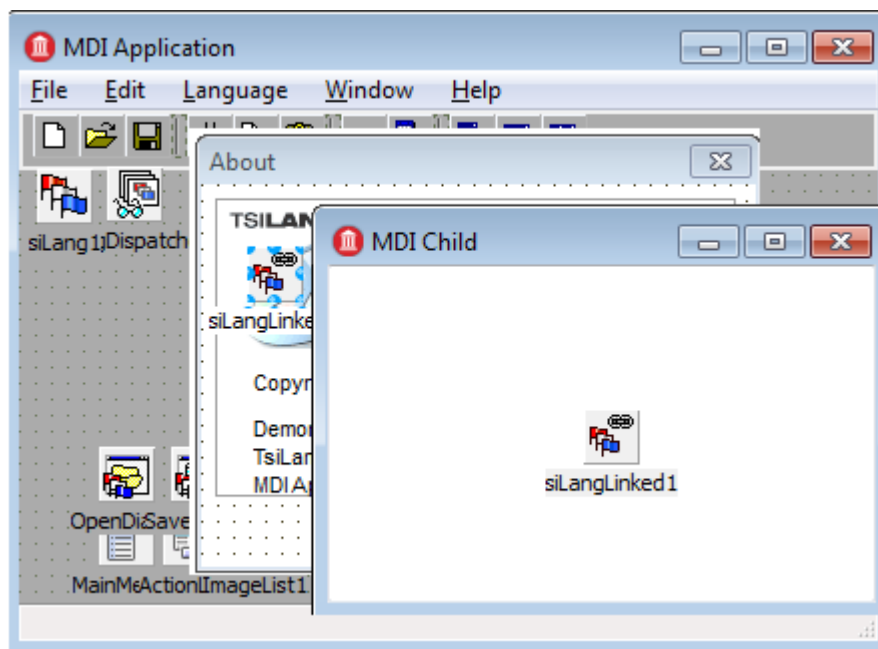


Figure 3 Forms with a TsiLangDispatcher and TsiLang components

### Setting properties

Now our application has a TsiLangDispatcher and a TsiLang component on the

main form, a `TsiLangLinked` on the `MDIChild`, and a `TsiLangLinked` on the `AboutBox`. We need to link the components to each other and set their properties.

1. For the `TsiLangLinked` on the `MDIChild` in the Object Inspector set the property “`LangDispatcher`” to the `MainForm`’s `TsiLangDispatcher`; and the “`CommonContainer`” to the `MainForm`’s `TsiLang` component (unit “`Main`” must be in uses clause of the current unit).
2. Repeat step 1 for the `AboutBox` form.
3. Also set the “`LangDispatcher`” property of the `MainForm`’s `TsiLang` component also set to the `TsiLangDispatcher`.
4. Now we must define supported languages. As all the `TsiLang` components are linked to the `TsiLangDispatcher`, it is enough to adjust “`LangNames`” property only for the dispatcher. Select the `TsiLangDispatcher` and in the Object Inspector open “`LangNames`” property editor. Type names of desired languages in arbitrary form, for example “`English`”, “`German`”, “`French`”. The number and names of languages for the all linked `TsiLang` components will be constituted automatically.



**Tip:** We described all the steps above in order to provide you with the ability to better understand what and how should be done. You can use **Translation Wizard** available in `TsiLang Expert` and it will do all these steps for you automatically. So you will be able to translate your project in few minutes.

### Using siDialogs

If we don’t want our application dialogs depend on the Operating System language all the standard dialogs shall be replaced with their `siDialogs` counterparts.

1. Place on the `MainForm` a `TsiOpenDialog` component and remove the old `TOpenDialog`. Rename the `TsiOpenDialog` to “`OpenDialog`”, set its “`Filter`” property to “`Text Files (*.txt) | *.txt | All files (*.*) | *.*`” and “`siLang`” property to the `MainForm`’s `TsiLang` component.
2. In the application a `TSaveDialog` is absent but we can add the function to save files, so place on the `MainForm` a `TsiSaveDialog` and set its properties: “`Name`” := `SaveDialog`, “`Filter`” := “`Text File (*.txt) | *.txt | All files (*.*) | *.*`”, and “`siLang`” := `MainForm`’s `TsiLang` component.

### “Hard-Coded” Strings

Usually a lot of strings are used immediately in the source code. In the MDI Application a new MDI child window caption is determined by the following code in `Main.pas` (`main.cpp`):

### Delphi:

```
procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
end;
```

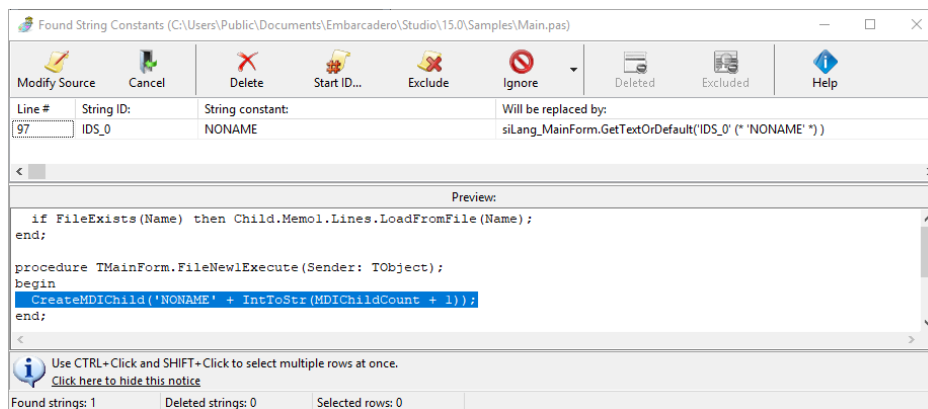
**C++ Builder:**

```
void __fastcall TMainForm::FileNew1Execute(TObject *Sender)
{
    CreateMDIChild("NONAME" + IntToStr(MDIChildCount + 1));
}

```

Such “hard-coded” strings as the above “NONAME” we shall also include to the translation list. The easiest way to do it is to use the *TsiLang Expert*:

1. Launch the *TsiLang Expert* (from the IDE’s menu select “Tools | TsiLang Expert”).
2. From the Expert’s list of forms select the MainForm and choose from the Expert’s menu “File | Source | With Form...”

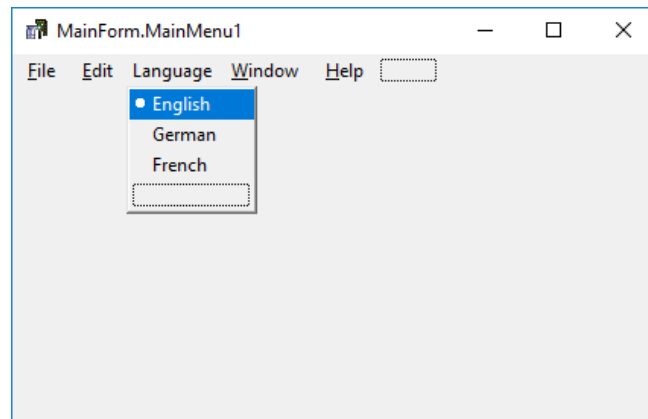


**Figure 4 Strings constants found by the Expert**

3. The above window shows all “hard-coded” strings found in the unit. Push the “Modify Source” button to add “NONAME” to the translation list. All occurrences of this string constant will be automatically replaced with a respective TsiLang method call.

**Language Switching**

In order to switch the current language in design-time just assign to the TsiLangDispatcher property “ActiveLanguage” another value. For run-time we should provide the respective code. We suggest adding corresponding menu items to the MainForm’s menu:



**Figure 5** Menu items for language switching

1. Create menu items like the above in the MainForm' menu.
2. Assign to their "Tag" properties values corresponding the language index (first language has index 1). Make the menu items grouped and set "AutoCheck" property to True.
3. Select all these menu items and create a common OnClick handler, for example "ChangeLanguage". In the event handler type the code:

#### Delphi:

```
procedure TMainForm.ChangeLanguage(Sender: TObject);
begin
    siLangDispatcher1.ActiveLanguage := TMenuItem(Sender).Tag;
end;
```

#### C++ Builder:

```
void __fastcall TMainForm::ChangeLanguage(TObject *Sender)
{
    siLangDispatcher1->ActiveLanguage =
dynamic_cast<TMenuItem *>(Sender)->Tag;
}
```

#### Editing Translation Data

In fact, you have finished all required work as a software developer and now only translator's job is remaining! There are two ways to provide TsiLang components with translation data:

## Translations Editor

First, you can immediately enter and edit translation data in the IDE using the TsiLang component editor (*Translation Editor*) or specialized property editors.

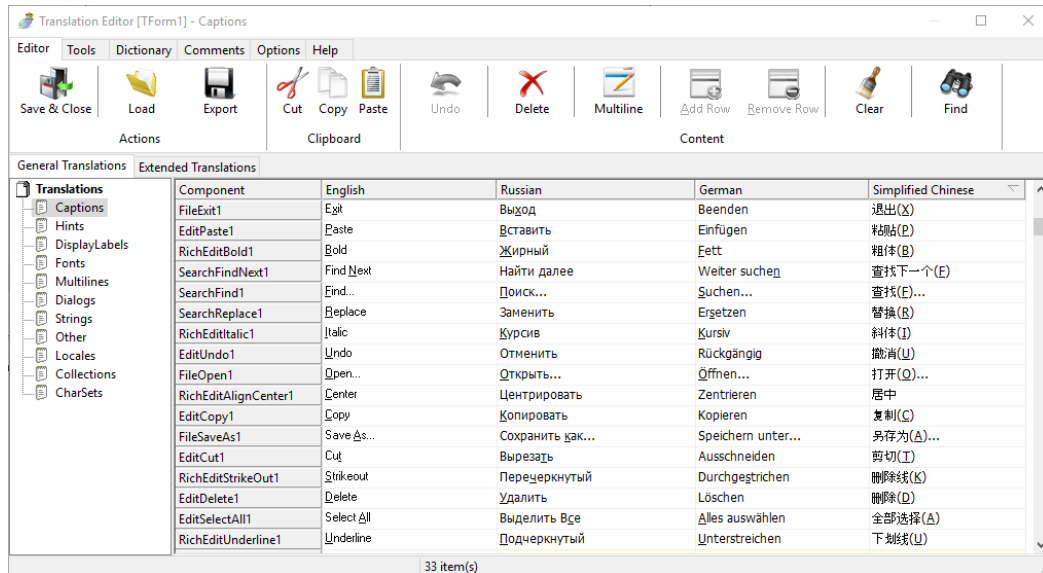


Figure 6 Translation Editor displays “translation-aware” properties

In order to launch the *Translation Editor* double click on a TsiLang component. As you can see, all translation data is divided into several categories that are displayed in the left-pane tree view. You can also open this editor from the Object Inspector as a property editor for the specified property (i.e. “Captions”, “Hints”, etc.). The *Translation Editor* has many useful features and functions that will be considered later in the Section “Using Translation Editor”.

## SIL Editor

You can export all the current data (now it contains only default (“English”) strings) to an external file and transfer this file along with the *SIL Editor* to a third-party translator. The SIL Editor is a light-weight application and can be freely redistributed. It has the convenient user interface and allows entering and editing translations. After the translator returns the file you can import its data back to the project.

- To export the project data to an external file launch the *TsiLang Expert*, from its’ menu select “File | Save/Load Translations | Save Project...” and save the file.
- To import translation data from an external file launch the *TsiLang Expert*, from its’ menu select “File | Save/Load Translations | Load Project...” and select the file.

# Chapter 3

## Common Tasks

### Using TsiLang Expert

One of the most exciting tools of the *TsiLang Components Suite* is the *TsiLang Expert*, an IDE add-in that dramatically facilitates the entire process of application internationalization and carries out most of the routine work. By default, the Expert is installed into IDE under the “Tools” menu.

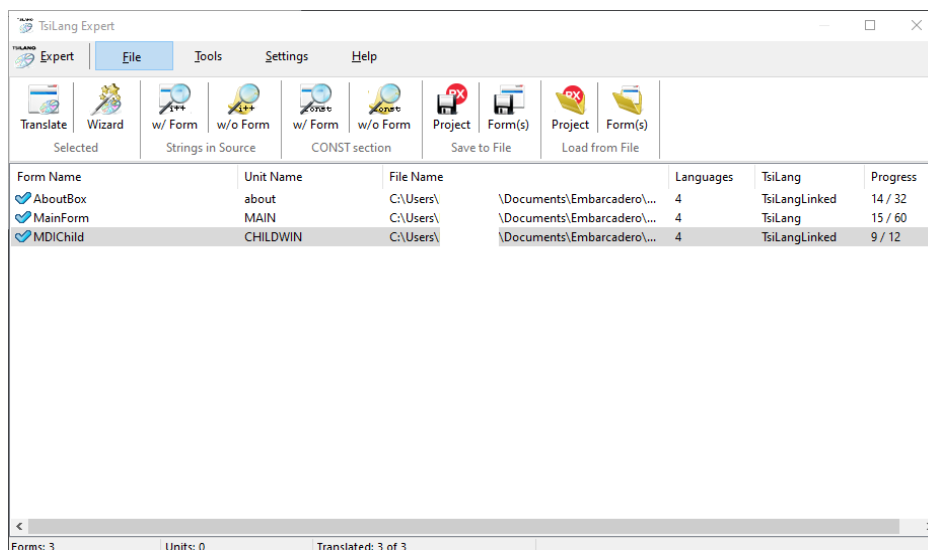


Figure 7 TsiLang Expert

When you launch the Expert it enumerates all forms in the active project and displays their status in the list view. If a form has no TsiLang component you can quickly add a TsiLang to this form via the Expert menu “File | Translate” or by double click on the corresponding list view item.



**Tip:** You can set the default type of a TsiLang component to be added as well as default language names in the Expert Options Dialog (“Tools | Options | General...”). See below for details.

To add TsiLang components to all project forms select all items in the list view and choose from the Expert menu “File | Translate” or just press “Enter” key.

## Using Translation Wizard

TsiLang Expert provides a Translation Wizard in order to simplify and guide you through the translations of your forms and to help you configuring components and settings in the most appropriate way.

To launch the Wizard just select forms you need to translate or configure in TsiLang Expert and select in Expert's menu "File | Wizard".

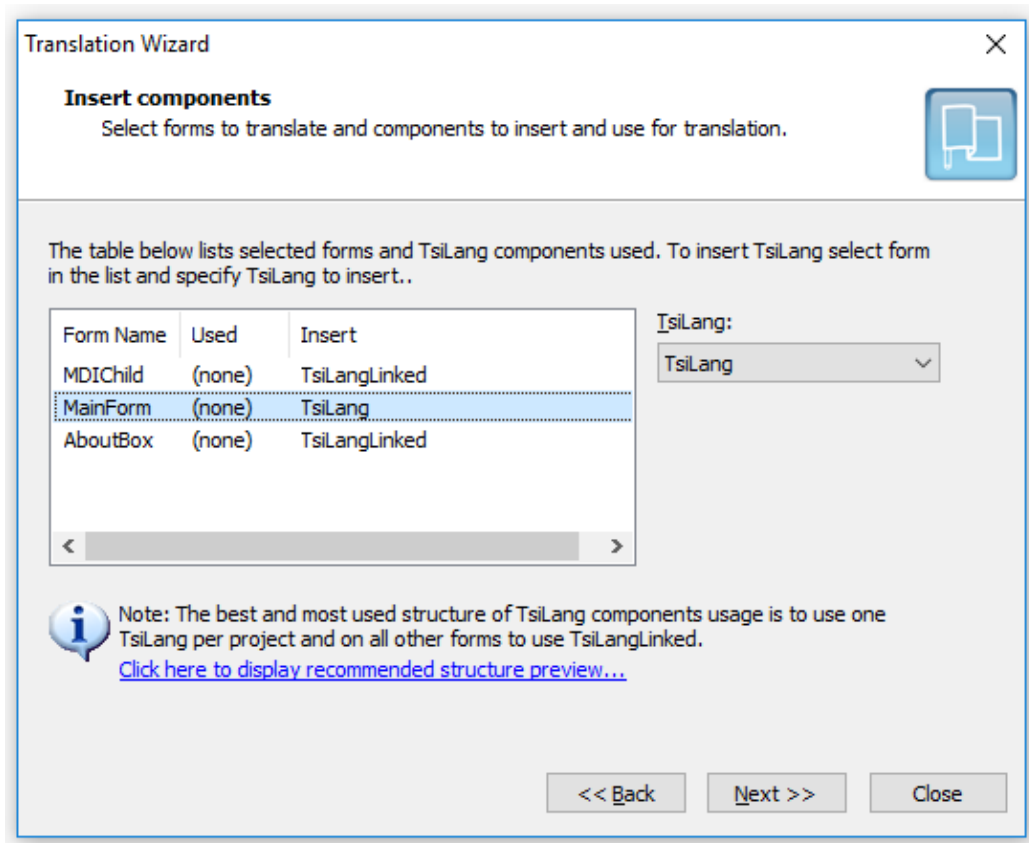


Figure 8 Translation Wizard (First step)

At the first step of the Wizard you need to specify what type of component to use for each form for translations. The most recommended scheme to use is available for preview by clicking the bottom link (highlighted in blue). After finishing selection of components types click **Next** button to go to the next step of the Wizard.

On the second step of the Wizard (see picture below) you need to define languages settings for selected forms and define which TsiLang to use as CommonContainer if you've used any TsiLangLinked on selected forms.

To define languages settings, you can either use TsiLangDispatcher on one of the project forms or configure languages manually. The recommended way is to use TsiLangDispatcher since this will dramatically simplify language switching and management for your project.

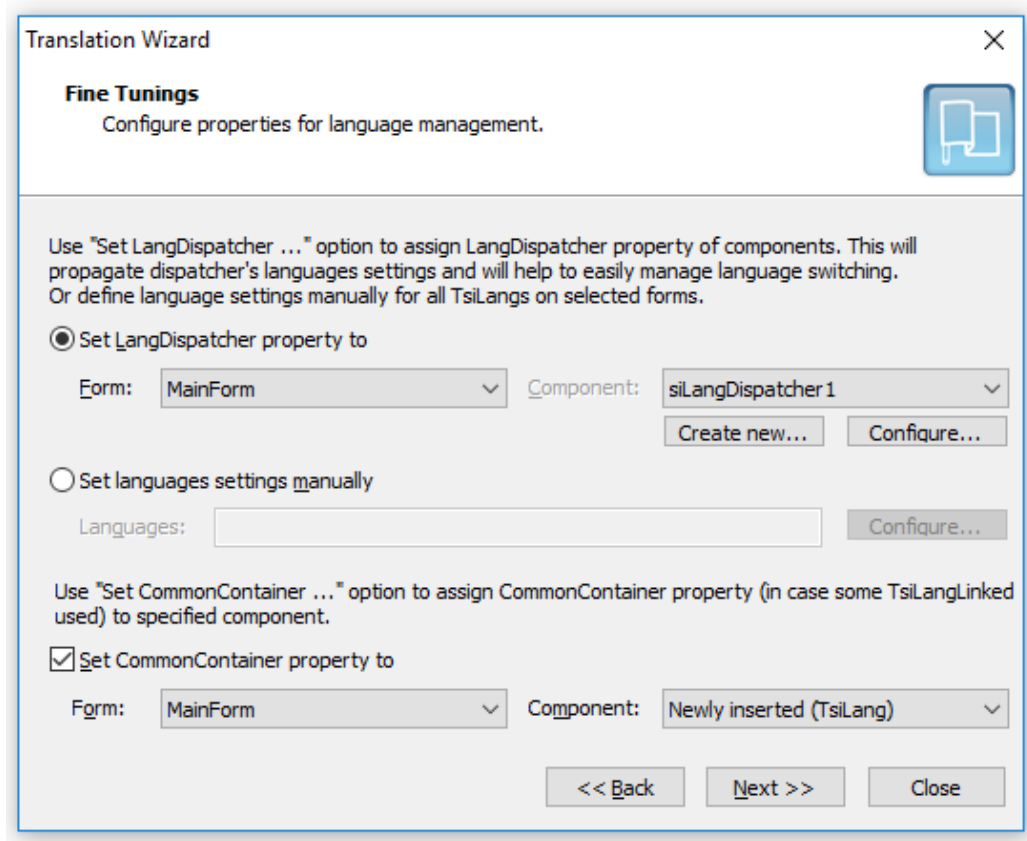


Figure 9 Translation Wizard (Second Step)

If no form in the project holds TsiLangDispatcher you can create one by clicking **Create new** button. If the project already has a TsiLangDispatcher you can configure it by clicking **Configure** button.

To set language settings manually for selected forms activate **Set languages settings manually** options and edit languages by clicking **Configure** button near the option.

After clicking **Next** button the Wizard will provide you with the details of actions to be performed and after your confirmation will configure your forms according to the provided settings.

### Search for Hard-Coded Strings

Besides string properties of components, applications usually contain a lot of “hard-coded” strings that also need translation. The *TsiLang Expert* has an ability of scanning your project in order to find and collect string constants. Selected string constants are added to the “Strings” property of corresponding TsiLang component. When the active language is changed all those strings are substituted with the respective translations.

There are four menu items related with translation of hard-coded strings:

- File | Source | With form...



- File | Source | Without form...
- File | Const section | With form...
- File | Const section | Without form...

The first two commands scan selected unit for strings used immediately in the source, like this:

```
...
  ShowMessage('Hello World!');
...
```

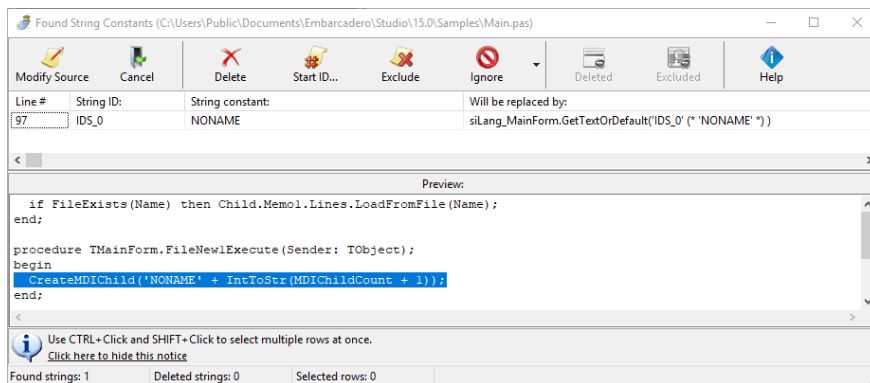
At that the first command treats the unit corresponding to the form selected in the list view, and the second command does any unit of the project.

The second pair of commands is responsible for the treatment of strings declared explicitly as constants, like this:

```
...
const
  sHello = 'Hello World!';
...
  ShowMessage(sHello);
...
```

Like the above pair the first command scans the selected form, whereas the second one does an arbitrary unit.

If the Expert finds any strings in the selected unit the results are displayed in the below dialog:



**Figure 10 Hard-coded strings found by the Expert**

In the above dialog box, you can delete strings that should not be translated (select rows and press the “Delete” button) and change the start number for identifiers (“Start ID” button). If some of the found strings should not be never translated (i.e. ‘.txt’), select these strings and press “Exclude” button. Next time the Expert will ignore all occurrences of ‘.txt’. If all the information is correct, press the “Modify Source” button to add them to the translation data and to replace their occurrences in the unit with TsiLang methods.

## Using TSI:IGNORE tags

You can use **ignore** tags in your source code to mark some lines for skipping while performing scan for hard-coded strings. These tags include:

```
{TSI:IGNORE}
{TSI:IGNORE ON}
{TSI:IGNORE OFF}
{TSI:IGNORE NEXT}
{TSI:TRANSLATE NEXT}
{TSI:IGNORE VALUE}
```

### (Pascal notation)

```
/*TSI:IGNORE*/
/*TSI:IGNORE ON*/
/*TSI:IGNORE OFF*/
/*TSI:IGNORE NEXT*/
/*TSI:TRANSLATE NEXT*/
/*TSI:IGNORE VALUE*/
```

### (C++ notation)

- **TSI:IGNORE** tag could be placed anywhere in a source line to skip this line in scanning.
- **TSI:IGNORE ON** shall be placed on separate line and will mark all source lines below to skip.
- **TSI:IGNORE OFF** shall be placed on separate line and will deactivate **TSI:IGNORE ON** tag.
- **TSI:IGNORE NEXT** shall be placed on separate line and will mark the next line to skip.
- **TSI:TRANSLATE NEXT** shall be placed on separate line and will include the next source line in scanning even if it is inside **TSI:IGNORE ON/OFF** block.
- **TSI:IGNORE VALUE** shall be placed exactly before the single string value that you wish to exclude. String value started right after this tag will be skipped while scanning.

## Sample code:

```
begin
  ShowMessage('This will be skipped!'); {TSI:IGNORE}
  ShowMessage('This won't be skipped!');
  {TSI:IGNORE ON}
  ShowMessage('This will be skipped!');
  {TSI:TRANSLATE NEXT}
  ShowMessage('This won't be skipped!');
  ShowMessage('This will be skipped!');
  {TSI:IGNORE OFF}
  {TSI:IGNORE NEXT}
  ShowMessage('This will be skipped!');
  ShowMessage({TSI:IGNORE VALUE}'This will be skipped!');
end.
```

## Working with External Files

The *TsiLang Expert* has the ability to export the translation data of the whole project or selected forms to an external .SIL (.SIB) files that can be used for editing by third-party translators, or distributed to your end-users which might wish to edit or modify them. And vice-versa, the Expert is able to update current translations through the import data from external .SIL (.SIB) files.

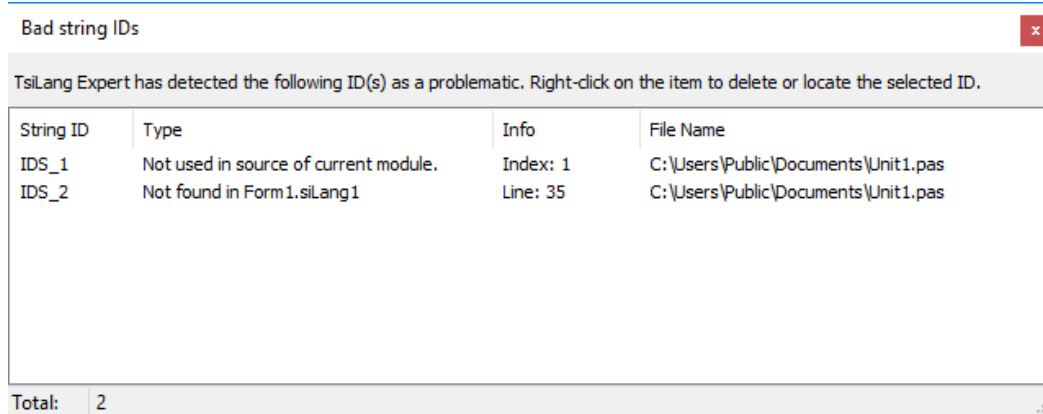
Under “File | Save/Load Translations”, there are the following commands:

- Save form(s): Saves all translation information for the form(s) selected to an external file.
- Load form(s): Loads all translation information for the form(s) selected from an external file.
- Save form(s) properties: Saves selected properties to an external file.
- Save project: Saves all translation information for all project forms to an external file.
- Load project: Loading all translation information for all project forms from an external file.
- Merge form(s): Merges all translation information for the form(s) selected from an external file.
- Merge project: Merges all translation information for all project forms from an external file.

## Other Functions

### Checking Identifiers

Under Expert “Tools” menu, there are two commands “Check Bad String IDs” and “Check Bad String IDs in Unit(s)...” that search selected units for unused string identifiers as well as check whether all used identifiers are included in the translation data.



**Figure 11 Bad string identifiers found by the Expert**

Any illegal or unused identifiers found by the Expert are displayed in the above dialog box. Double click on a line to jump to the relevant place in the source code or Translation Editor.

#### **Search and Replace**

These features are accessible through the Expert menu “Tools | Find Phrase...” or “Tools | Replace Phrase...” and allow finding and replacing any words in the translation data of selected form(s).

#### **Excluding Properties**

The command “Exclude Properties...” under “File” menu allows excluding specified properties from the translation lists of selected form(s).

#### **Clearance Translations**

The command “Clear Translations” removes all translation data from the selected form(s). This feature is useful for applications built with external translation storage. During the development process, you can store translations internally to adjust application layout for different languages in the design-time, but before the deployment, you can quickly remove all unnecessary internal translations to load them from external file or other storage. To clear translations for all project forms, there is a “Clear Project Translations” command.

#### **TsiLang Type Changing**

The command “Tools | Change TsiLang Type to...” allows quickly change one TsiLang type to another on the selected form(s). It can be useful, for example, if you decide to alternate data storage mechanism and change all TsiLang components with TsiLangRT, without translation data losses.

### **Expert Options**

To adjust the Expert's default settings to your needs select the command “Tools | Options”.

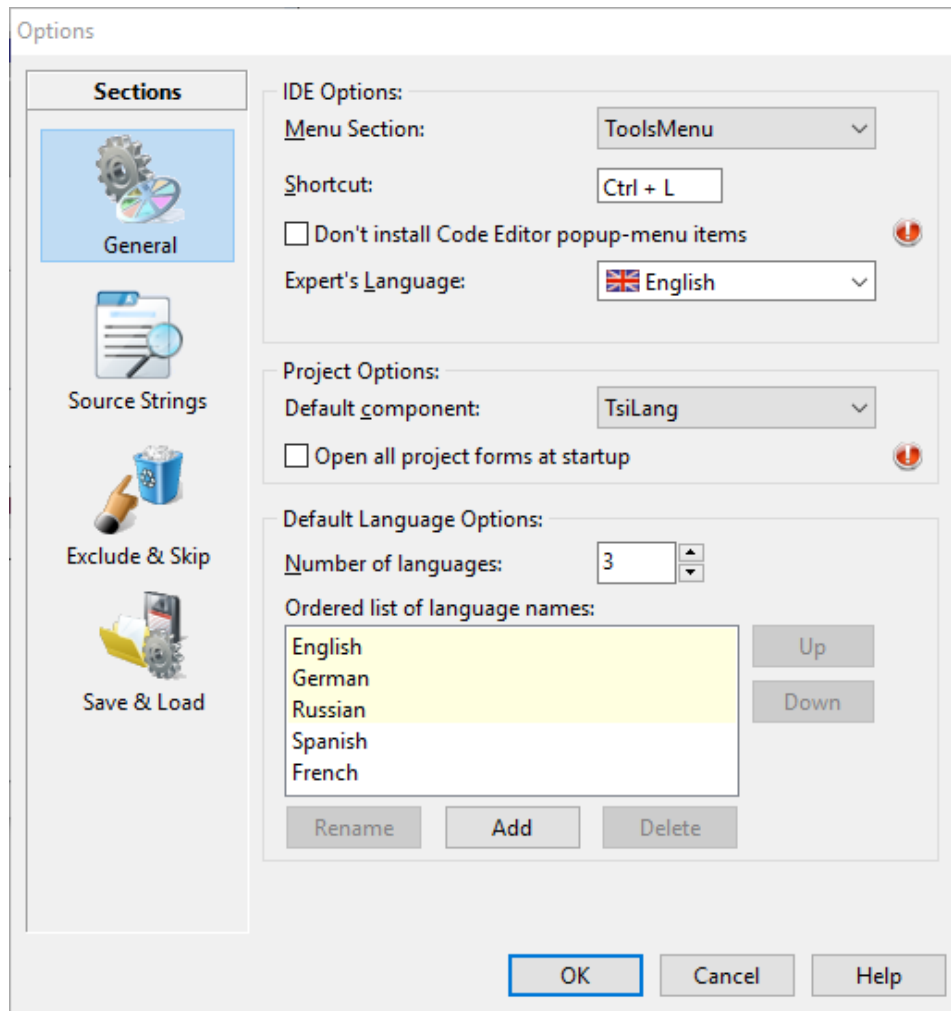


Figure 12 TsiLang Expert options dialog

The Expert Options dialog box is used to modify the following parameters:

#### General Options

- **IDE Options:** By default, the Expert is installed into IDE under “Tools” menu and has an IDE shortcut “Ctrl+L”. You can change these settings. Upon installing TsiLang package into IDE, there will be added Code Editor popup-menu items for translating the selected code. If you experience any problems caused by these items you can activate “Don’t install Code Editor popup-menu items” option and they will be removed.
- **Project Options:** Specifies the default type for auto-created TsiLang components; if the flag “Open all project forms at startup”, the Expert will open all project forms when launched.
- **Default Language Options:** Specifies the default number, names and order of languages for auto-created TsiLang components.

#### Source Strings

- **Replacement Options:**
  - **Get Text method:** indicates the method name of TsiLang that will be inserted into sources for replacing hard-coded strings. Using “GetTextOrDefault” is more preferable since when you have no provided translation for a string you always get the default value, not empty value, as in the case of “GetText” method. To be able to get default values for the “GetText” method too, just set the “TranslateType” property of TsiLang to ttGetDefault. Methods with “xxxC” names are replacements for respective methods without “C” at the end, but return **PChar** (**char \***) as result value. Methods with “xxxW” names are replacements for respective methods without “W” at the end but return **WideString** as result value.
  - **Add form name:** Indicates whether to add or not a form name to “GetText” or “GetTextOrDefault” method calls when replacing hard-coded strings in source. This is useful while using these methods inside “OnCreate” event since the object with such is not yet created and reference to it will cause “Access Violation...” error.
  - **Ignored string length:** Specifies the minimal length of hard-coded strings that should be added to the translation list when scanning a unit. For example; in most cases the string with only one character shouldn't be translated so you can set 1 in this option to skip such strings. To translate all strings just set 0 for this option.
  - **Comments length:** Configures the length of the original string to be placed in place of replacement in comments to preserve the visibility and readability of source code.
  - **No comments at all:** Activate this option if you don't need to place comments in sources after replacement.
  - **Pascal comments use:** Select what comments style to use in Pascal code while doing replacement.
- **Hard-coded strings:** “Default prefix” defines the string to be used as prefix for auto-generated string identifiers when replacing strings in sources.
- **Strings in const sections:** “Default prefix” is the string to be added to the constant name forming the string identifier; “Permanent part options” allows changing constant name to upper case when adding it to string identifier.
- **Empty values after replacement:** activate this option to empty the value of a constant after replacement. Although this will slightly reduce the size of EXE, we highly recommend to not use this option and leave the variables values as is.
- **Source preview lines:** configures amount of source preview lines displayed in Found Strings window when you select any string collected for replacement.

**Exclude & Skip**

- **Exclude strings containing only special characters:** this option allows skipping strings in sources that consist of only special characters.
- **Skip lines containing:** Specifies the phrases that will cause skipping of the source lines by the Expert, even if there is any string to translate. This option is useful, if you need to skip certain common source strings like assignment to TableName or Database property of components.
- **Skip words and phrases:** Specifies list of phrases that must be always skipped in the source by the Expert. Additionally, you can activate **“Use Regular Expression to skip phrases”** option and use Regular Expressions in this option to specify a set of phrases to skip.

**Save & Load**

Use this page to configure template settings for project save/load translations commands from Project Manager popup-menu.

- **Folder:** define folder name to use for project save/load translations commands.
- **File name template:** define file name template. The list of available template tags are listed below of this field.
- **Remember these settings and don't ask for file name:** activate this option to use settings defined on this page and skip file name/template dialog while executing project save/load translations command from the Project Manager popup-menu.

## Using Translation Editor

The Translation Editor is the tool for editing translation data of a component in both, design-time (for all types of TsiLang) and run-time (for TsiLangRT).

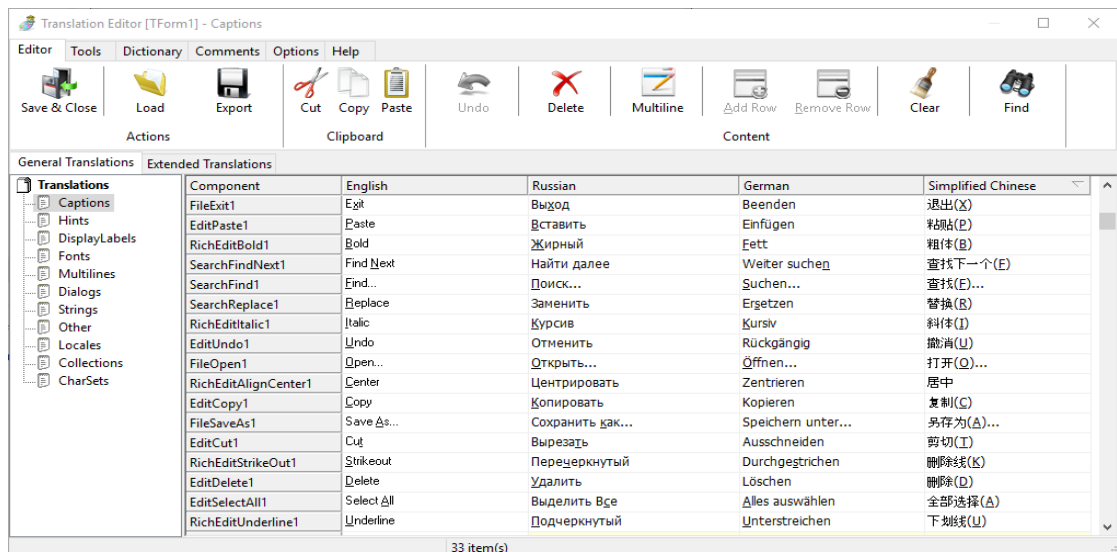


Figure 13 Translations Editor

The main functions of the tool are the following:

- Direct editing of translations in the string grid;
- Working with external .SIL or .SIB files (for example, you can save all translation data into a .SIL file, transfer it to a third-party translator, and after the file is translated, load it back);
- Working with the Dictionary Manager (that is adding and retrieving translations).

Use the **Dictionary** Toolbar for:

- **Show/Hide** Dictionary Manager.
- **Add Row** - adds translations from the selected row to the Dictionary.
- **Add All** translations to the Dictionary.
- **Translate Cell** - finds and inserts translation from the dictionary into the selected cell. The choice is based on the languages that already have translations.
- **Translate Language** - automatically translates selected language only.



- **Translate All** - automatically translates the entire grid. The program will prompt you to select the base language for translation.
- **Suggest** - use this feature in case you can't find an exact translation. TsiLang will analyze the dictionary and give you a list of suggestions according to the sensitivity level you set: the lower the value, the more suggestions you get.



**Note:** These buttons could be disabled if Dictionary Manager is not installed on your system.

When Dictionary Manager is active Translation Editor will try to automatically find a translation for the empty cell when you start editing it.



**Tip:** If you don't want this automation, close Dictionary Manager upon Translation Editor closing, then just open Dictionary Manager manually before opening Translation Editor.



**Tip:** If you specify project's default dictionary in "Project Translations Settings" in TsiLang Expert be sure that TsiLang Expert is open in IDE before opening the Translation Editor. Otherwise, the Translation Editor won't be able to use these settings.

Use **Editor** Toolbar to perform general actions with translations:

- **Load / Export** button will load/save currently selected translations property to/from external file. To load/save whole translation data just select the top-level tree node (**Translations**).
- **Clipboard** group contains actions to cut, copy and paste content.
- **Add row** and **Remove row** commands will add new row or delete the selected one. These commands are only available in the Strings section and for Extended Translations.
- **Multiline** button will open a multi-line editor for the currently selected cell. **Ctrl+Enter** on any cell will open a multiline editor.
- **Clear** button deletes **all** translations in the currently selected list.
- **Find** command will open **Find Dialog** to perform a search for the specified text.

Use **Tools** Toolbar to perform additional actions, such as:

- **Default Fonts** will open the configuration dialog for the default font and charset settings for the different languages.
- **Languages** command allows renaming current languages.

- **Statistics** command displays statistical data for the current translations.
- **Exclude Empty** command adds all component's properties that have no any values in all languages to an exclusion list. This will reduce the size of translations. The excluded component properties are stored into the **SmartExcludeProps**.
- **Delete Duplicates** command will empty translations that are equal to the first (default) language. This is used to reduce the translations size when **TranslateType** property of TsiLang is set to **ttGetDefault**. If translations size is not critical it is not recommended to use this.
- **Find in Source** command will try to find the currently selected string ID in source code. Available only when editing **Strings** section.
- **Check % Strings** command is used to verify that all formatting strings are properly translated.

The **Comments** Toolbar contains commands for working with translations comments:

- **File Name** command allows selecting of file name to be used for storing comments. If you select the existing file it will be loaded and you can view the available comments. This action shall be used at first to enable all other comments functionality.
- **View** command displays comment for the selected item if such comment is available.
- **Edit** command allows you to add or edit the comment for the selected item.
- **Delete** command will delete a comment for the selected item.



**Note:** Translation Comments require **SIL Editor** to be installed! Translation Comments are available in both, the SIL Editor and the Translation Editor. If you use translation comments, be sure to send your comments XML file among with SIL/SIB files to your translators.

The **Options** Toolbar contains configuration settings that will help to adjust look and feel of Translation Editor:

- **Filter** allows to filter current view by:
  - **Translated** - all translations for all languages are done.
  - **Un-translated** - only the base language string entered.
  - **Partially Translated** - you have more than one translation, but still have empty cells.

- **Incomplete**- hides all translated items.
- **Sort** drop-down could be used for sorting by particular column.
- **Highlight Duplicate IDs** – rows in the Strings section that have the same string ID will be highlighted.
- **Highlight Mismatched Multilines** – rows in Multilines section with different amount of lines in translations and original value will be highlighted.
- **Font Size** drop-down menu allows to configure the font size used for the editor. This is useful when working on High DPI monitors.
- **Toolbar Size** drop-down allows to use larger images for toolbar buttons.
- **Auto-use default fonts data** – if checked then editor will automatically insert font name and charset for languages in Fonts and Charsets sections if any cell is empty there.
- **Show widths tooltip** – if checked then editor will display tooltip window with widths of translations for the currently editing cell.
- **String ID prefix** – allows to specify what string ID prefix was used on Strings section. Editor will try to remove it when sorting so IDs could be fixed as integer values.

Some additional functions are available through a pop-up menu (right-click the translation grid to pop-up):

- **Exclude** - exclude translations:
  - a) excludes selected component from current section;
  - b) excludes selected component from all sections;
  - c) excludes all components from current section;

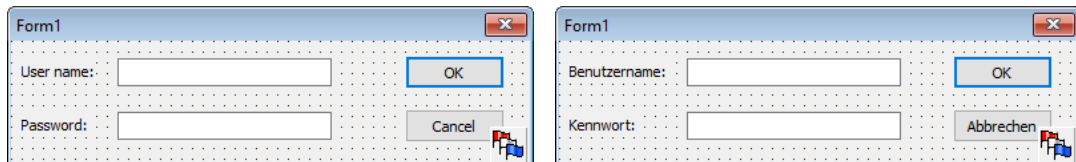


**Note:** The excluded component properties are actually automatically stored into the SmartExcludeProps list. To get the excluded properties back remove them from the SmartExcludeProps list.

Translations Editor allows editing both general and extended translations. Please read **Using Extended Translations** topic to learn the details of working with Extended Translations property.

## Using Extended Translations

Process of application's internationalization sometimes requires translating not only strings and string properties of components but “translating” other properties that can affect on visual appearance of application, first of all size and position of controls. For example, compare these two screenshots:



One can see that lengths of labels “User name:” and “Benutzername” are different so if focused TEdit on the right hand form had the same width and position as one on the left hand form the label would be overlapped. You can either redesign the form layout or use “ExtendedTranslations” property of TsiLang component. Extended Translations can be edited using Translations Editor. This includes components' non-string properties, such as **Left**, **Right**, **Width**, **Tag**, **Align** etc. as well as all properties of any sub-level components such as **TLabelEdit** on Extended Translations tab. This could be very useful, for instance, for repositioning or rescaling controls, when you translate your application into a language with phrase length more than in original language.

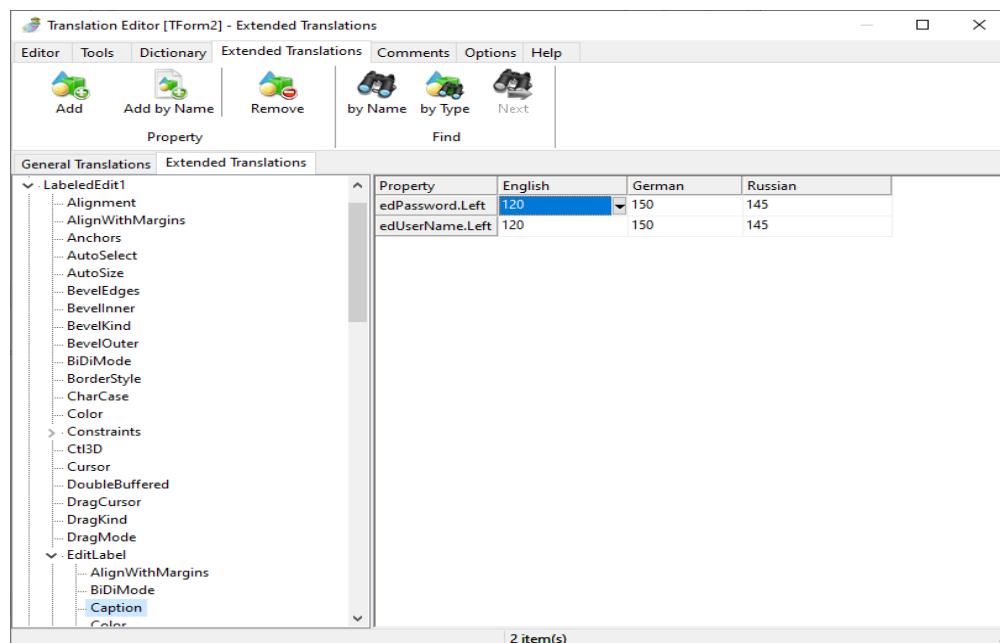


Figure 14 Extended Translations property editor

### Translating

To include a component's property into the list of translated properties:

1. Find the component in the 'Tree (left side of the window), expand the node (pressing the 'plus' sign at the left of the node), then select the property to be translated.
2. Move the property to the *Translations* list (center of the window) clicking *Add* or selecting the appropriate popup-menu item.
3. Select it in the *Translations* list and enter the translations into the cells for every language in a manner you used to do in Delphi Object Inspector.
4. It is very useful to use *Update Translation* in order to populate the values for extended translations. To perform this just follow the next steps:
  - 4.1. Add components' properties that need to be translated into *Extended Translations*.
  - 4.2. Close *Translations Editor*.
  - 4.3. Switch to another language by changing *ActiveLanguage* property of TsiLang.
  - 4.4. Re-design your components as you would like to see them under this language.
  - 4.5. Right-click on TsiLang and select *Update Translations* from appeared popup-menu.
  - 4.6. As you will notice TsiLang will populate the values for selected extended properties with current settings.
  - 4.7. Switching back and forth between languages will show you how your design will look like under it.

To remove a property from the Translations: select it and click *Remove* or select the appropriate popup-menu item.

## Using ExtendedTranslations under different DPIs

### Declaration

```
TExtendedItemChangingEvent = procedure (Sender: TObject; const
NewLanguage:
  Integer; const Item: TsiExtendedItem; var NewValue: string) of
object;

property OnExtendedChanging: TExtendedItemChangingEvent;
```

### Description

OnExtendedChanging event occurs before the change of any extended item is performed due to language switching. **NewLanguage** indicates the language that will be used. **Item** contains information about translation item. You can use **Identifier** field of **Item** to identify the property being updated. **NewValue** contains the new value for property to be set. If you need to change the value to set just modify **NewValue** parameter.

### Tip:

Using of this event may be useful if you use **ExtendedTranslations** to reposition controls under different languages. You need to take care the fact that users may use different font sizes like 100%, 125% or 150% under high DPI displays. In this case the forms will be automatically scaled (if **Scaled** property is set to **True**) and positions/sizes stored in **ExtendedTranslations** will be incorrect. Please see code snippet below that will help you to manage this case:

```
procedure TForm7.siLang_Form7ExtendedChanging(Sender: TObject;
  const NewLanguage: Integer; const Item: TsiExtendedItem;
  var NewValue: string);
const
  // this is the DPI under application developed
  DefaultPPI = 96;
  // this is the Text Height under default DPI you can detect it
  by
  // using Canvas.TextHeight('0')
  DefaultTextHeight = 13;
var
  NewPos: Integer;
  NewTextHeight: Integer;
begin
  if Screen.PixelsPerInch = DefaultPPI then
    Exit;
  if Item.Identifier = 'Edit1.Left' then
    begin
      NewPos := StrToIntDef(NewValue, 0);
      if NewPos <> 0 then
        begin
          NewTextHeight := Canvas.TextHeight('0');
          // using PixelsPerInch won't help because VCL performs scaling
          using TextHeight
          //      NewPos := Round(NewPos * PixelsPerInch / DefaultPPI);
```

```
        NewPos := MulDiv(NewPos, NewTextHeight, DefaultTextHeight);  
        NewValue := IntToStr(NewPos);  
    end;  
end;  
end;
```

---

## Using Translations Stored in External Files

You must decide first which type of file to use as your storage: either SIB or SIL file format. When using SIB file the data will be stored in binary format and loading speed will be very fast. Using SIL file allows you having data stored in text file (SIL is actually INI file) but loading time will be much longer than using SIB file. SIL Editor works perfectly with both file formats as well as components themselves. Using SIL files may be convenient when using any Version Control System (VCS), because many of them can operate only with textual files. Also you may consider using SIL file during development under VCS and convert it to SIB right before release of your project. SIB files provide much more internal data integrity checks and are much more stable than SIL files.

The following ways are available for loading translations from external files during run-time:

1. Set TsiLangDispatcher property FileName to your SIL/SIB file. This is the easiest and the most convenient way. In this case dispatcher will automatically load this file into all TsiLang components “linked” to it when needed.



**Note:** Be sure to check the path settings for FileName property when using this method.

2. Manually load translations at run-time into components. The following methods could be used to load SIL file:
  - LoadAllFromFile () - loads all translations for current form from file.
  - LoadAllFromFileDNC () - loads all translations for current form from file and doesn't update translations in UI controls.
  - LoadFromFile () - loads specified property from file.
  - LoadLanguage () - loads specified language from file.
  - LoadLanguageByExt () - loads specified language into ExtendedTranslations property from file.
  - LoadLanguageByProp () - loads specified language into selected property from file.
  - LoadExtendedFromFile () - loads ExtendedTranslations property from file.

The following methods could be used to load SIB file:

- LoadAllFromBinaryFile () - loads all translations for current form from file.
- LoadPropFromBinaryFile () - loads specified property from file.



- `LoadAllFromBinaryStream()` – loads all translation for current form from binary stream.

## Using Exclude from Translations Editor

It is very good practice to exclude untranslatable components and properties from translations. Please read **Exclude not used components and properties** topic from **Tips and Tricks** section for common tips for excluding components and properties from translations.

Using Exclude from Translations Editor you can exclude any component or component's properties from your form at one place. At the left side of the window you have components tree and properties tree. Properties tree updated each time you select any specific component. Use **Sorted** button to make a list sorted alphabetically; button **Clear All** will clear all lists; **Clear-** clears current list and **Delete-** deletes selected item from current list.

The Editor has three “working modes”:

### Components to Exclude

This mode is used to exclude whole components from translations. Switch to **Components to Exclude** mode by clicking on **Components** tab in the right side of the window. To add component to exclusion list simple select it in components tree or in components combo-box and click **Add Component** button. Additionally, you can add type of the selected component to exclude using **All <Type Name> Components**. This command will force TsiLang to skip all components of specified type. **Add Child Controls** command will add all child controls of the selected component to the exclusion list. Also you can use **Add All having <Property Name>** command to add all components that have property called the same as selected in **Available Properties** tree.

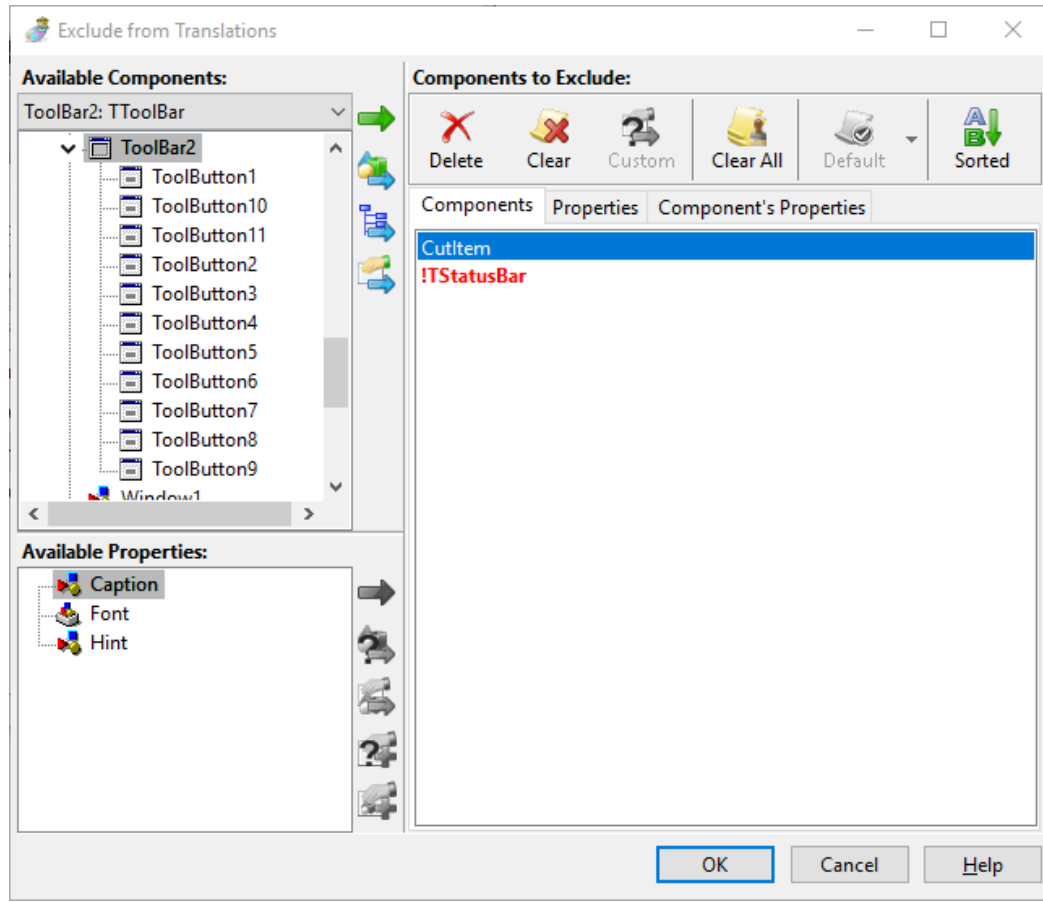


Figure 15 Exclude components from translations

All these commands are also available in components tree's popup-menu.

### Properties to Exclude

This mode is used to exclude properties for all components by property name. Switch to **Properties to Exclude** mode by clicking on **Properties** tab in the right side of the window. To add particular property name to exclusion list select it in the properties tree and click **Add Property** button. When adding property name like **Caption**, **Hint** or others included into TsiLang's standard properties it will be displayed in **red and bold** indicating that the respective list of TsiLang will be empty at all. To add all properties of particular type click **Add by Type** command and specify property type to exclude. You can also use just a part of the type name and Editor will add all properties with type containing provided value. Use **Add all <Type Name> Properties** command to add all properties with the same type as selected one. This could be useful, for example, when it is needed to exclude all properties of **Char** type like PasswordChar and others. Use **Add by Name** command to add all properties with the name containing the specified value. All these commands are also available in the properties tree's popup-menu.

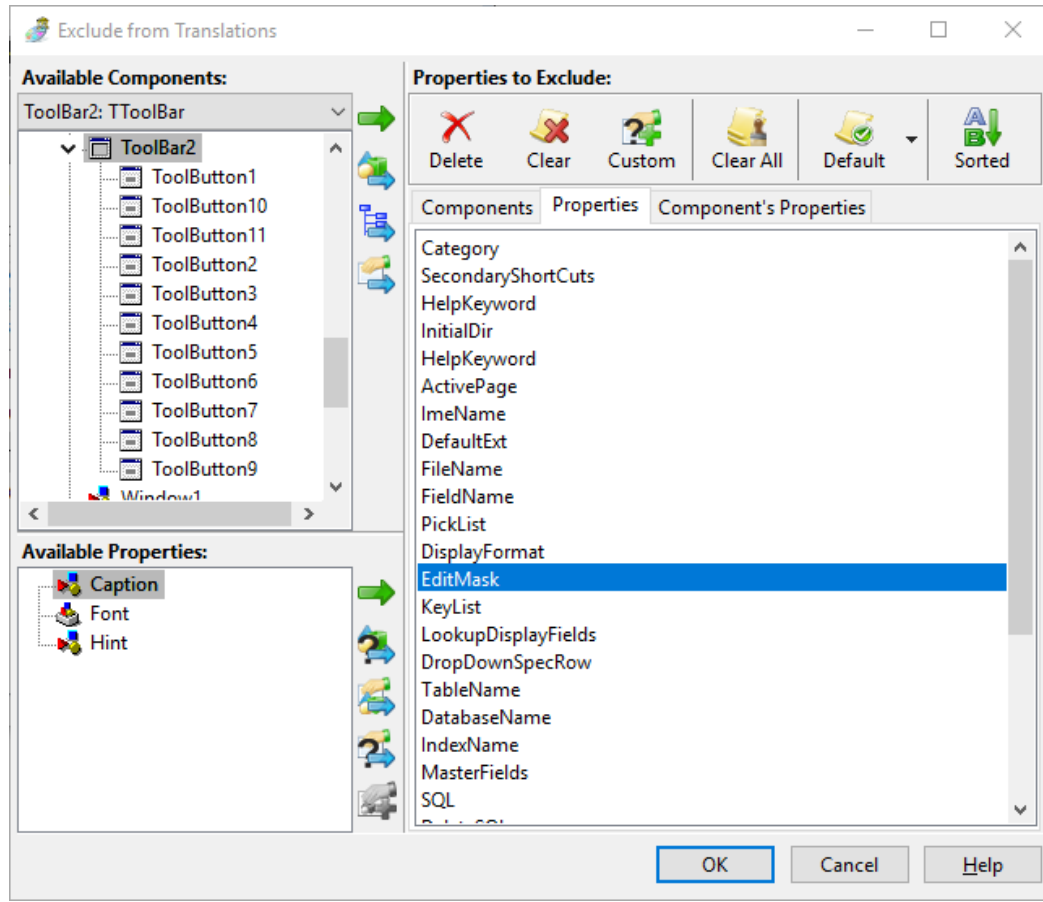


Figure 16 Exclude properties from translations

You can also add some particular properties to **Default** list. **Default** list is stored and can be used later for other forms and projects. **Default** list operations are available through properties list popup-menu and **Default** toolbar button's drop-down menu. Use **Add to Default List** command to add selected property to **Default** list. To insert some property from **Default** list use **Insert from Default List** command and select needed property or use **Insert All** to insert all not yet included properties to the list. Activate **Auto-save Default List** option available in **Default** menu to automatically save **Default** list on any changing. **Save Default List** command saves **Default** list to registry.

### Component's Properties to Exclude

This mode is used to exclude particular properties for specified components. Switch to **Components' Properties to Exclude** mode by clicking on **Components' Properties** tab in the right side of the window. To add particular component's property name to exclusion list select it in properties tree and click **Add Property** button. To add all components' properties of particular type click **Add by Type** command and specify property type to exclude. You can also use just a part of the type name and Editor will add all properties with type containing provided value. Use **Add all <Type Name> Properties** command to add all components' properties with the same type as selected one. Use **Add by Name** command to add

all properties with the name containing the specified value. All these commands are also available in properties tree's popup-menu.

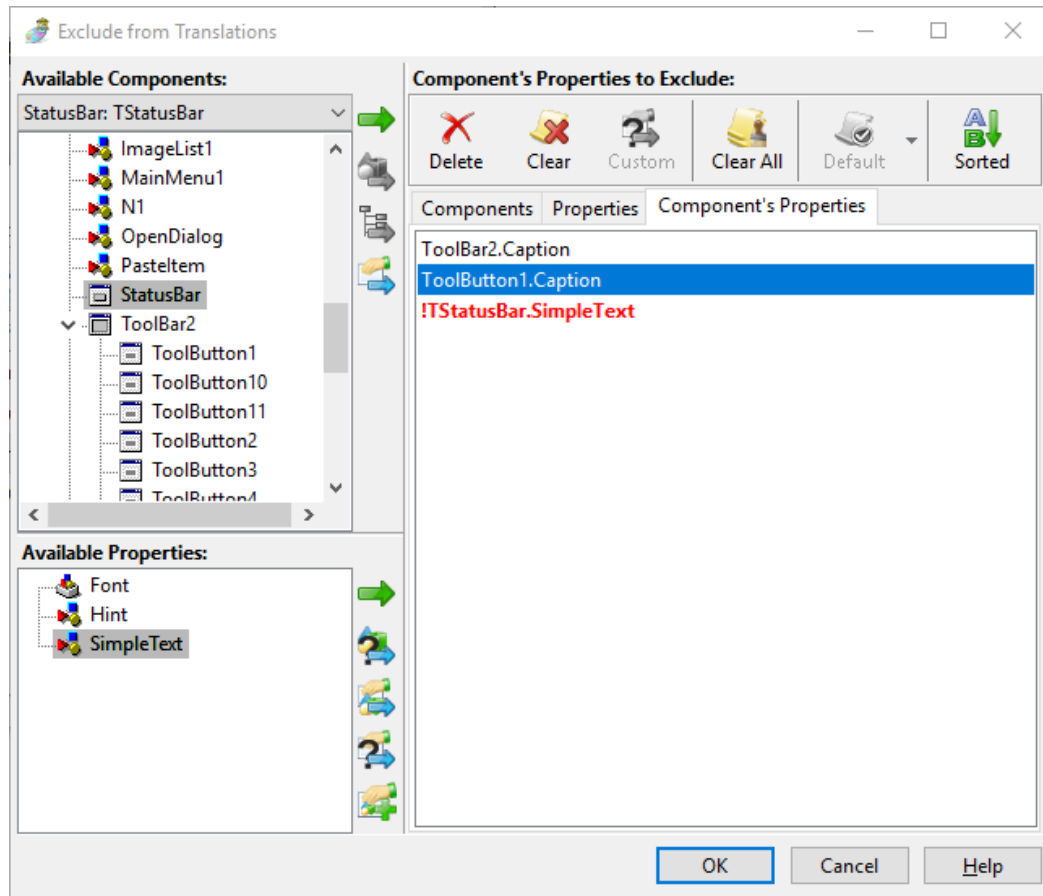


Figure 17 Exclude components' properties from translations

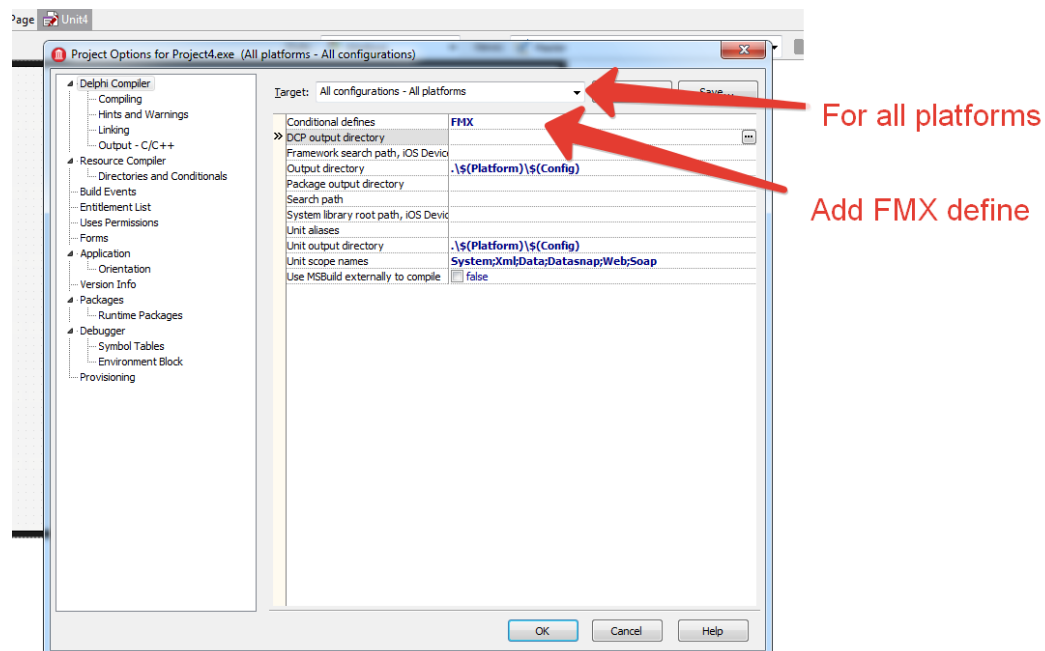
## FireMonkey Support

TsiLang Components Suite version 7.0 introduced FireMonkey (**all versions**) support.



**Note:** Please note the following while using TsiLang components in FireMonkey projects:

- **FMX** global conditional define is required to build TsiLang units under FireMonkey projects. If you just link pre-compiled TsiLang units to your FireMonkey project this define is not required.



**Figure 18 Add FMX define when building Firemonkey projects**

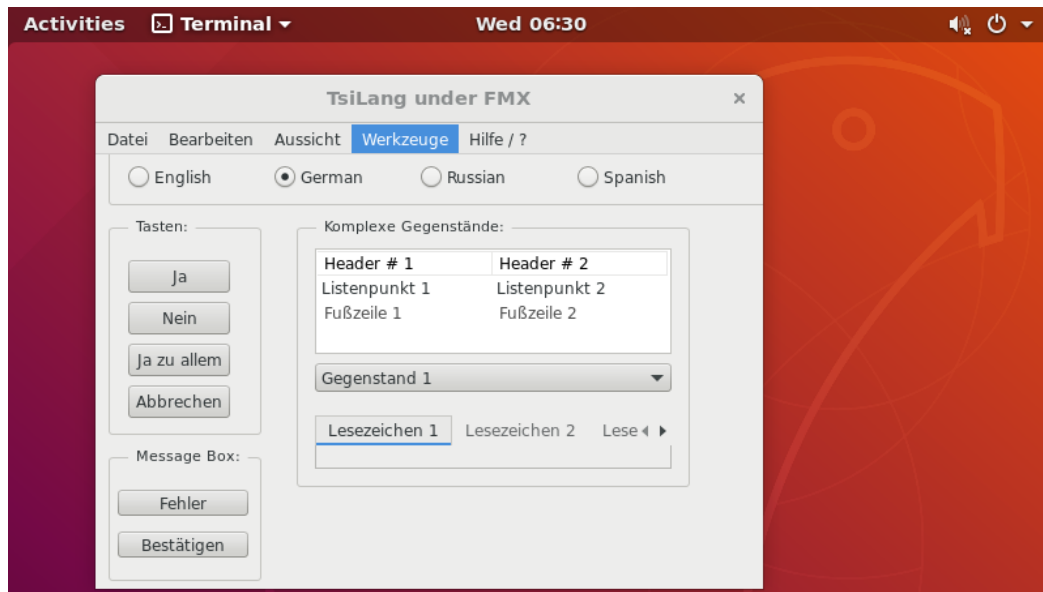
- OS dialogs such as File Open/Save, Print and other can't be translated.
- FireMonkey 2 and later displays message dialog boxes (such as ShowMessage(), MessageDlg()) using OS's API and doesn't allow to interact to UI controls on these dialogs. This is the reason why TsiLang is unable to translate them.
- Only the following components available under FireMonkey applications: TsiLang, TsiLangLinked and TsiLangDispatcher. Since other components designed for specific tasks which are applicable only under Windows OS.
- Compiled units (DCU) are provided **ONLY for Win32, Win64, Android, Android64, Linux64 and OSX32** platforms. If you need to

build your projects for other target platforms you will need to re-build your project **using sources** of TsiLang Components Suite units.

- To compile your FireMonkey projects with compiled units of TsiLang components you need to add path to TsiLang DCUs to your project's search path. For example, while building project for OSX32 platform under Delphi XE 4 add *{TsiLang Folder}\Units\ERS XE4\OSX32* folder. For building Win32 project add *{TsiLang Folder}\Units\ERS XE4\Win32\FMX* folder to project's search path settings.
- Disable **FMX define** if you build for Linux64 target platform. It is not used while building for Linux64.

## Linux Support

Starting since version 7.8 TsiLang Components Suite introduced Linux support for RAD Studio 10.3.2 and later.



**Figure 19 Translated application with TsiLang under Linux**

You can use TsiLang components when building Linux applications in either FireMonkey or regular applications. If you use FireMonkey and FmxLinux to build visual Linux application don't define FMX define for Linux platform in case you use sources of TsiLang units.



---

## Using Translation Memory

Translation Memory helps translating at run-time items that might be translated onto one form but are not translated on other. For example you have translation for "Open" string in Dialogs property of one of the TsiLang components and you have a button on some other form with the same caption but it is not translated on this form. If Translation Memory is active then TsiLang will try to translate the untranslated button caption with the translation from another similar entry.

Translation Memory is accessible only **at run-time**. **TranslationMemoryOptions** property of TsiLangDispatcher configures the behavior for Translation Memory.

- **TranslationMemoryOptions.Active** - set this to True to enable Translation Memory at run-time.
- **TranslationMemoryOptions.AutoUseForComponents** - set this to True to enable auto-translation of untranslated items at run-time.
- **TranslationMemoryOptions.ReturnEmptyForUntranslated** - set this to True if you wish the TranslationMemory() method to return empty for untranslated items. Otherwise it will return the BaseValue.

TsiLangDispatcher's method TranslationMemory() is used to translate items at run-time using Translation Memory.

```
function TranslationMemory(const BaseValue,  
TargetLanguageName: string; var Translation: string):  
Boolean;
```

You can use this method to try to translate items added/created at run-time using Translation Memory.

- **BaseValue** - the value that needs to be translated. BaseValue is value in "base language" (the first one in the list of TsiLangDispatcher's languages).
- **TargetLanguageName** - the name of the target language.
- **Translation** - the result of translation. If there is no translation found then this will be empty if TranslationMemoryOptions.ReturnEmptyForUntranslated set to True or the BaseValue otherwise.
- **Return value** - True if translation was found and False otherwise.



**Note:** using Translation Memory will slow down the first loading of the form at run-time as it will add translations into the memory.

# Chapter 4

## External Tools

### Dictionary Manager

TsiLang Dictionary Manager is a powerful tool, which dramatically accelerates the translation process for several applications with the common vocabulary. TsiLang Translation Editor is connected with TsiLang Dictionary Manager via COM interface and can easily import necessary translations from the TsiLang Dictionary Manager and insert them in the places you want. Necessary translations can be exported to the TsiLang Dictionary Manager as well. The idea of Dictionary Manager is to accumulate all the translations from your different projects and to use them instantly exactly where it is needed.

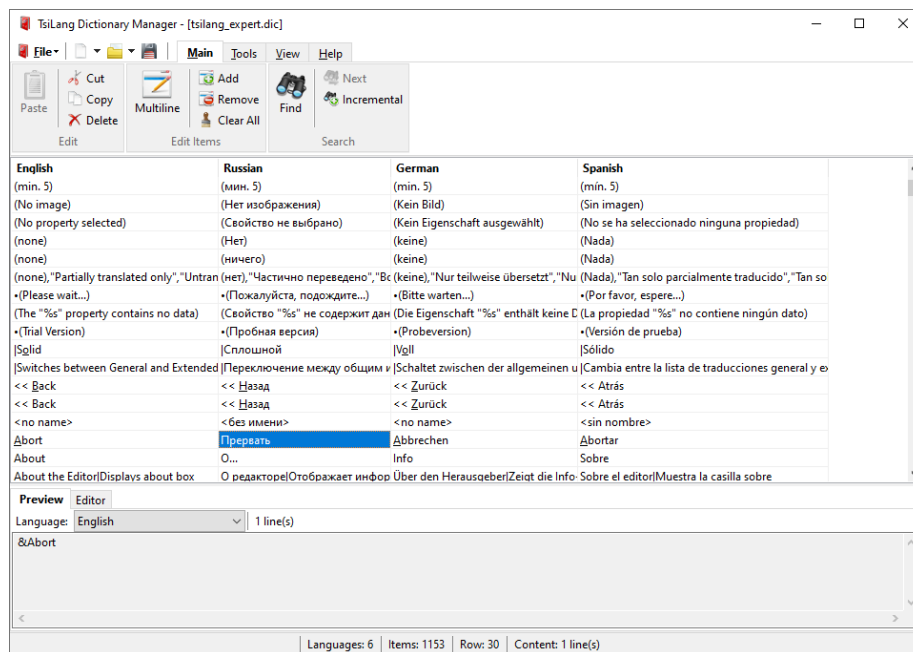





Figure 20 Dictionary Manager

Screenshot above illustrates the Dictionary Manager completed with possible translations.

You can use Dictionary Manager from TsiLang Translation Editor and SIL Editor.

To launch automatic translation of current cell click the  **Find Translation** button. Button **Auto Translate**  will start automatic translations of all empty cells. To add translations to dictionary use  **Add All** button on toolbar.

From the Dictionary Manager you can export dictionary content to the widely spread file formats: (\*.html, \*.htm, \*.xml, \*.csv, \*.doc, \*.xls, or import the dictionary content from the files of all above mentioned formats.

Export/Import features of the Dictionary Manager allow you to organize the work of project translators in the best way making possible usage of convenient widely spread word processors.



**Note:** You can distribute Dictionary Manager to your translators absolutely free.

## Dictionary Manager Automation Server

Some Dictionary Manager methods can be accessed via COM interface. Dictionary Manager's COM server embedded into application provides this possibility. You can import COM server type library using "Projects-> Import Type Library -> dicmgr" from the main Borland C++ Builder or Borland Delphi menu. The files "dicmgr\_TLB.cpp" and "dicmgr\_TLB.h" will be included in your project. The following Dictionary Manager's COM server's methods are available:

**Table 1 Dictionary Manager Automation Server**

Nr	Interface	Description
1.	<pre>virtual HRESULT STDMETHODCALLTYPE OpenFile(   BSTR FileName/*[in]*/,   TOLEBOOL* Value/*[out,retval]*/) = 0; // [1]</pre>	<p>Loads *.dic file with the name FileName into the Dictionary Manager.</p> <p>Value parameter is reserved.</p>
2.	<pre>virtual HRESULT STDMETHODCALLTYPE GetDefaultDict(   BSTR* Value/*[out,retval]*/) = 0; // [2]</pre>	<p>Returns in Value the name of the file, from which the dictionary will be taken by default</p>
3.	<pre>virtual HRESULT STDMETHODCALLTYPE Save(   TOLEBOOL* Value/*[out,retval]*/) = 0; // [3]</pre>	<p>Saves dictionary in the file, from which data were taken.</p>
4.	<pre>virtual HRESULT STDMETHODCALLTYPE Clear(void) = 0; // [4]</pre>	<p>Clears the dictionary</p>

Nr	Interface	Description
5.	<pre>virtual HRESULT STDMETHODCALLTYPE IsStrIncluded(   BSTR AStr/*[in]*/,   TOLEBOOL* Value/*[out,retval]*/) = 0; // [5]</pre>	Searches for the string AStr in the dictionary, Value is set to true if desired string is found, to false otherwise.
6.	<pre>virtual HRESULT STDMETHODCALLTYPE GetTranslation(   BSTR BaseLang/*[in]*/,   BSTR ActLang/*[in]*/,   BSTR Item/*[in]*/,   BSTR* Value/*[out,retval]*/) = 0; // [6]</pre>	The translation of the Item given by the language BaseLang is searched in the dictionary for the language ActLang. The string found is returned in the Value parameter. In the case if the translation was not found, an empty string in Value is returned.
7.	<pre>virtual HRESULT STDMETHODCALLTYPE AddLanguage(   BSTR LangName/*[in]*/) = 0; // [7]</pre>	Adds language with the name LangName into the dictionary
8.	<pre>virtual HRESULT STDMETHODCALLTYPE RemoveLanguage(   BSTR LangName/*[in]*/) = 0; // [8]</pre>	Removes language with the name LangName from the dictionary
9.	<pre>virtual HRESULT STDMETHODCALLTYPE IsVisible(   TOLEBOOL* Value/*[out,retval]*/) = 0; // [10]</pre>	Checks if the Dictionary Manager is visible at the desktop. Returns true if visible, otherwise false.
10.	<pre>virtual HRESULT STDMETHODCALLTYPE SetVisible(   TOLEBOOL Value/*[in]*/) = 0; // [11]</pre>	Makes the Dictionary Manager visible at the desktop.
11.	<pre>virtual HRESULT STDMETHODCALLTYPE IndexOfLang(   BSTR LangName/*[in]*/,   short* Value/*[out,retval]*/) = 0; // [12]</pre>	Returns language index in the Value, based on the language name LangName.
12.	<pre>virtual HRESULT STDMETHODCALLTYPE AddTranslation(   BSTR BaseLang/*[in]*/,   BSTR ActLang/*[in]*/,   BSTR BaseItem/*[in]*/,   BSTR ActItem/*[in]*/,   TOLEBOOL* Value/*[out,retval]*/) = 0; // [13]</pre>	Adds translation ActItem for the string BaseItem into the dictionary with respect to correspondent pair of languages BaseLang and ActLang.

## SIL Editor

SIL Editor is a convenient tool for \*.sil and \*.sib files modification. The tool is designated for convenient work of project translator, who is not obliged to install Borland Delphi or C++ Builder with the only purpose of introducing translations into the TsiLang component and as a consequence he/she is not obliged to pay license for Borland software. Now translator can use only TsiLang files editor in order to work with \*.sil (\*.sib) files exactly in the way it is done in TsiLang component editor.

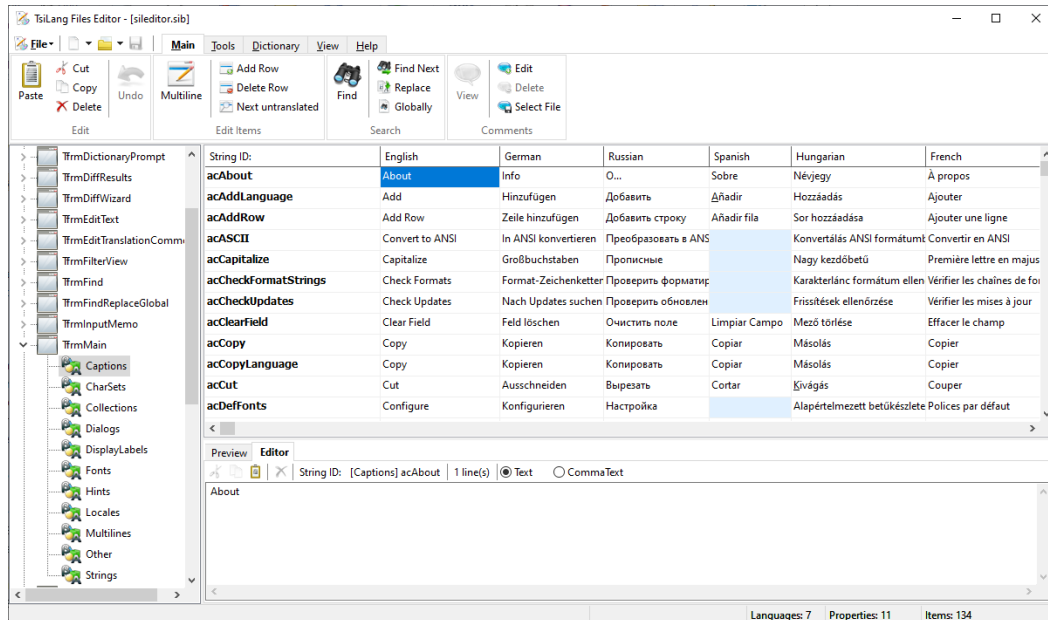


Figure 21 SIL Editor

When the translator finishes his work, he provides the software developer with the \*.sil (\*.sib) file, containing project translation which can be loaded into the project using TsiLang Expert. Standard text editor features are integrated here as well, such as cut, copy and paste, find, replace, save, print, etc.

From the SIL Editor you can export translation information to widely spread file formats: (\*.html, \*.htm, \*.csv, \*.doc, \*.xls), or import the translation information from the files of all above mentioned formats. Additionally, there is realized **Merge Wizard**, so you can combine translations from several SIL/SIB files.

### Displayed Properties

In case you want to work with the selected properties rather than with all of them, you can temporary suppress some properties in order not to be displayed in the editor screen. When saving modified translations in the \*.sil (\*.sib) file all hidden properties will be saved as well however.

### Fixed Languages

In case you want to protect some columns from being occasionally modified you can use “Fixed languages” option of SIL Editor. Columns, belonging to the languages chosen will be displayed with no possibility for being modified.



**Note:** You can distribute SIL Editor to your translators absolutely free.

## Automation Server

Some SIL Editor methods can be accessed via COM interface. This possibility is provided by SIL Editor's COM server embedded into SIL Editor application. You can import COM server type library using "Projects | Import Type Library | SILEditor" from the main Borland C++Builder or Borland Delphi menu. The files "SILEditor\_TLB.cpp" and "SILEditor\_TLB.h" will be included in your project. The following SILEditor COM server's methods are available:

**Table 2 Methods of SIL Editor Automation Server**

Nr	Interface	Description
1.	<pre>virtual HRESULT STDMETHODCALLTYPE <b>EditFile</b>(   BSTR <b>AFileName</b>/*[in]*/,   BSTR <b>PropName</b>/*[in,def]*/,   TOLEBOOL <b>DoPrompt</b>/*[in,def]*/,   TOLEBOOL* <b>Value</b>/*[out,retval]*/) = 0; // [1]</pre>	<p>Loads new file with the name <b>AFileName</b> into the editor. Default property is set to <b>PropName</b>. <b>DoPrompt</b> indicates whether or not to prompt user about modifications performed before exit.</p> <p><b>Value</b> parameter is reserved.</p>
2.	<pre>virtual HRESULT STDMETHODCALLTYPE <b>ShowProperty</b>(   BSTR <b>PropName</b>/*[in,def]*/,   TOLEBOOL* <b>Value</b>/*[out,retval]*/) = 0; // [2]</pre>	<p>Retrieves the translations for the property indicated in <b>PropName</b></p> <p><b>Value</b> parameter is reserved.</p>
3.	<pre>virtual HRESULT STDMETHODCALLTYPE <b>ExportTo</b>(   BSTR <b>FileName</b>/*[in]*/,   Sileditor_tlb::TExportType   <b>ExportKind</b>/*[in]*/,   TOLEBOOL* <b>Value</b>/*[out,retval]*/) = 0; // [3]</pre> <pre>typedef enum TExportType {   etCSV = 0,   etHTML = 1,   etDOC = 2,   etXLS = 3,   etSIB = 4 } TExportType;</pre>	<p>Exports translation information to the <b>FileName</b> with respect to the format given by <b>ExportKind</b>.</p> <p><b>Value</b> parameter is reserved.</p>

---

Nr	Interface	Description
4.	virtual HRESULT STDMETHODCALLTYPE <b>get_EditingFinished</b> ( TOLEBOOL* <b>Value</b> /*[out,retval]*/) = 0; // [4]	When editing is finished <b>Value</b> is set to true, otherwise false
5.	virtual HRESULT STDMETHODCALLTYPE <b>set_EditingFinished</b> ( TOLEBOOL <b>Value</b> /*[in]*/) = 0; // [4]	Set <b>Value</b> to true if you want to inform SILEditor about termination of the editing process.

## Resource Strings Wizard

Since Version 5.1 TsiLang Components Suite allows handling resource strings of your project even they are “hidden” somewhere in a .dcu (.obj) file and you don't have its source. If you want to display language dependent resource strings in your application, first, you need to set TsiLang component's property “HandleResourceStrings” to True and ResourceLanguageIndex to value that corresponds to resource language number in TsiLang LangNames property index.

How it works? Every time a resource string is about to be used in application the one of the following functions **LoadStr()**, **FmtLoadStr()** or **LoadResString()** is implicitly called. But if TsiLang property HandleResourceStrings is **True**, any resource string before loading will be replaced by its translation corresponding to the active language of the TsiLang.

Second, you need add resource strings that should be translated to “Strings” list of TsiLang component. The simplest way to do both steps is to use “Resource Strings” wizard that supplied along with TsiLang Components Suite. This is a standalone application but also can be launched from the TsiLang Expert's menu **Tools | Wizards | PE Import**.

Let's go through the process with “Resource Strings Wizard”. Open the wizard:

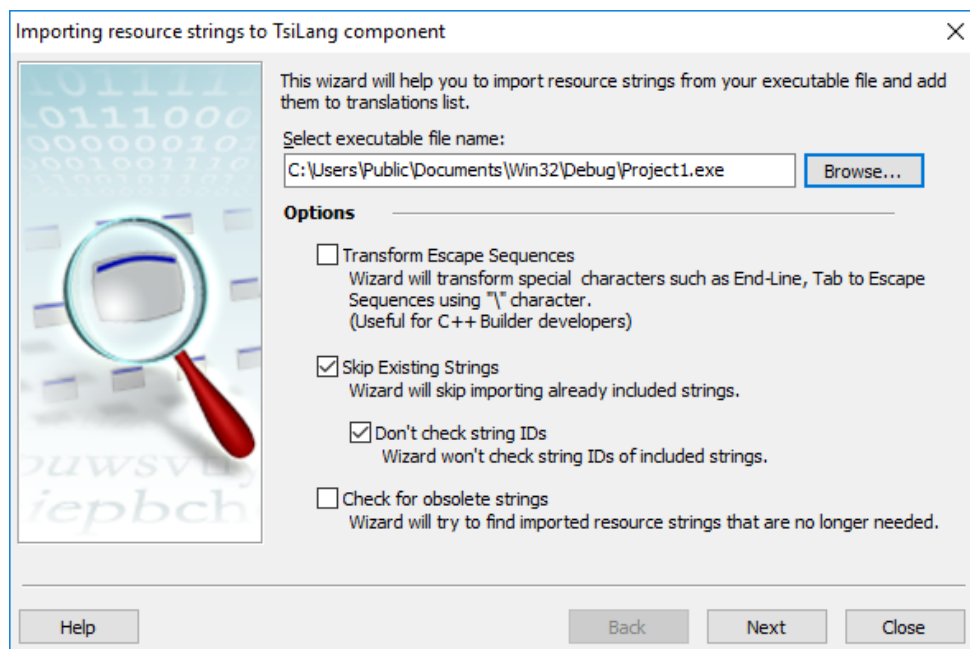
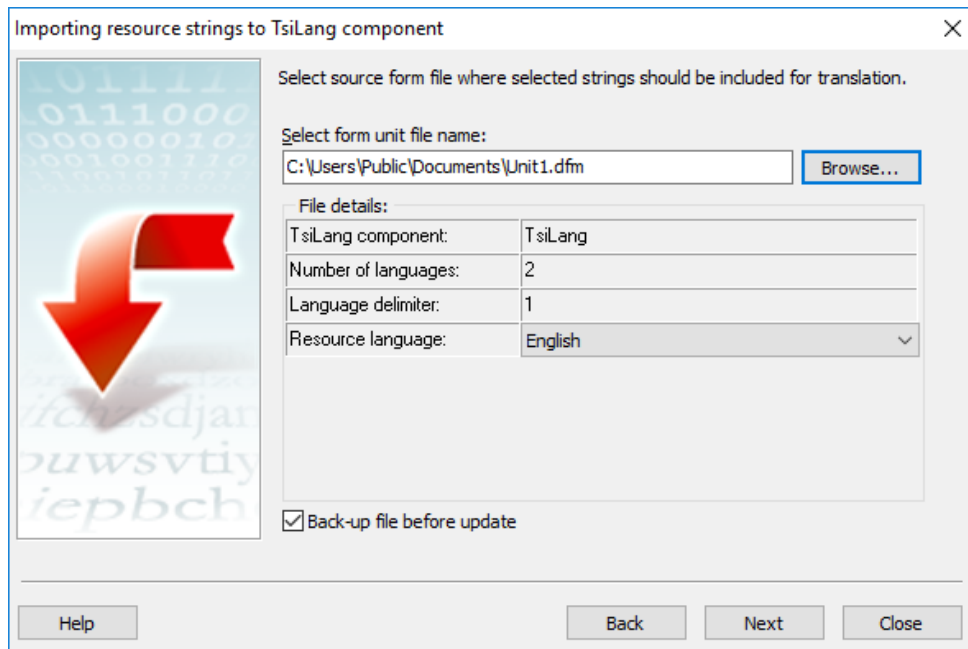


Figure 22 Resource Strings Wizard - Step 1

On the first step you need to select an executable that stores resource strings (if the wizard is launched from TsiLang Expert it is already done).

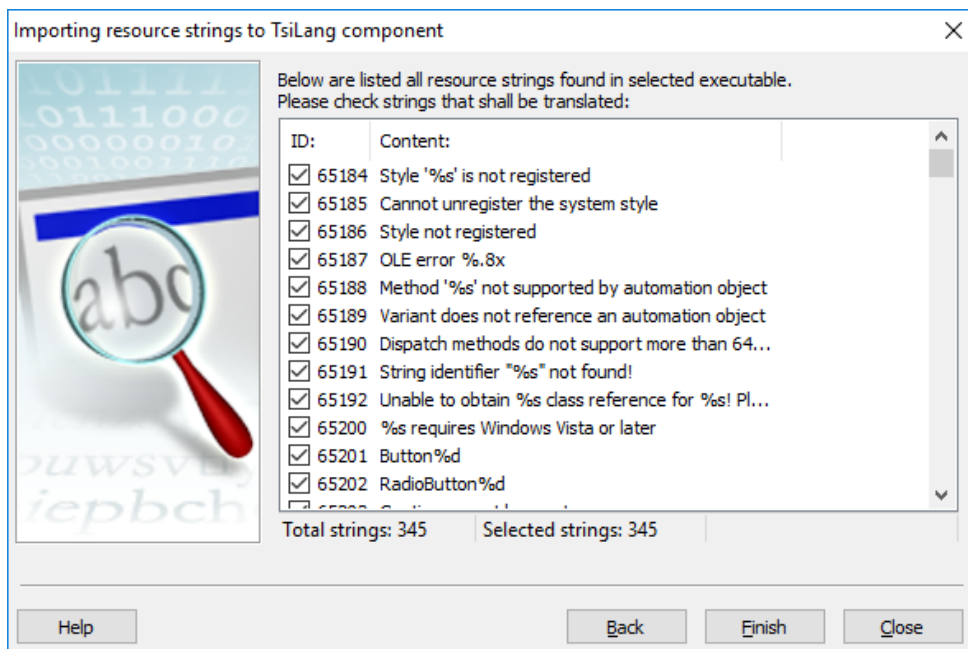
Adjust available options to configure Wizard's behavior.





**Figure 23 Resource Strings Wizard - Step 2**

On the second step you need to specify a unit with a TsiLang component that will hold imported strings. After that you can translate those strings by the same way as other hard-coded strings in TsiLang Translation Editor.



**Figure 24 Resource Strings Wizard - Step 3**

And at last, you select all resource strings that should be handled by a TsiLang component. Most of them are never displayed so we recommend importing only really needed strings. The source code of your application doesn't need any modifications.

## Translate resource strings by ID

Version 6.0 of TsiLang Components Suite has introduced the ability to translate resource strings by their identifier not by their value. This is very useful for resource strings that were added to the project resources not from Delphi or C++Builder units but from your own RC or RES files.

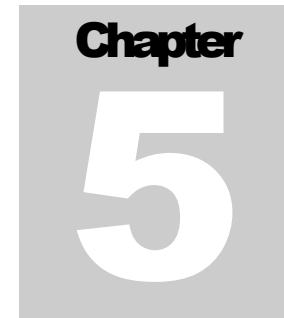


**Note:** This is not applicable for resource strings created with **resourcestring** keyword.

When using **resourcestring** keyword Delphi's compiler changes resource string identifier every time you re-build your application and usually assigned identifiers are at the highest limit of resource string identifiers (the maximum allowed identifier for resource string is **65535** (**0xFFFF** or **\$FFFF** in HEX)). But when you explicitly use the RC or RES files in your application the resource identifiers used will be kept permanently.

Unit **SiComp** has the **HighestInternalResourceID** global variable that helps you to translate resource strings by their identifier instead of value. Initially its value is set to **0** and this indicates to translate all resource strings (if required) by value. But you can set it to the highest value of your resource string identifier in order to translate your strings by identifier. For example, if your highest identifier is **999** set it to this value and all resource strings with identifier less or equal to **999** will be translated by identifier and others by their values as usually.

This is very useful in cases when you use in your resource strings different translations for the same base phrase. For example, in some cases the same English phrase like **File** can be translated differently for menu item caption and for document term or other. Using translation by identifier will help you in such cases but translation by value will always return first occurrence of **File** in translations string.



## Components Reference

### Core Components

The core components group includes the following components:

- TsiCustomLang Descendants:
  - TsiLang
  - TsiLangLinked
  - TsiLangRT
  - TsiLangRTSE
  - TsiLangTLV
- TsiLangDispatcher

Below you will find a comprehensive list of the above components' properties, methods, and events.

#### TsiLang

Most of properties and methods for all TsiLang types are implemented in the base class "TsiCustomLang" and therefore are common. The TsiLang has no additional properties or methods in comparison to the TsiCustomLang but only publishes the relevant ones.

#### Properties

The TsiLang (TsiCustomLang) properties can be grouped as follows:

**Table 3 TsiLang translation-aware properties**

Property Name	Data Type	Description
Captions	TStrings	Keeps translation information for Captions
DisplayLabels	TStrings	Keeps translation information for DisplayLabels

Property Name	Data Type	Description
ExtendedTranslations	TsiExtendedItems	This property allows to include in translation list not only strings but any other properties that can affect on visual appearance of controls, for example, such properties as width or height. From the other hand some composite components such as TLabelEdit contain child components whose string properties cannot be translated by usual way. The ExtendedTranslations property allows to process composite components too.
Hints	TStrings	Keeps translation information for Hints
Strings	TStrings	Keeps translation information for text strings
Fonts	TStrings	Keeps translation information for font names
Charsets	TStrings	This property contains char set values for all components. You shall set them up only in the case if you need to change any char set of the component, when language is being changed.
Multilines	TStrings	Keeps translation information for Memo-like texts
DlgsCaptions	TStrings	Keeps translation information for dialog captions
OtherStrings	TStrings	Keeps translation information for Edit's text, ImeName and other information
Locales	TStrings	Keeps translation information for local settings (date and time format, money format, etc.)
Collections	TStrings	Keeps translation information for collections

**Table 4 TsiLang behavior-aware properties**

Property Name	Data Type	Description
ActiveLanguage	Integer	Range: [1-NumOfLanguages]. By setting this property to the desired value, you force the TsiLang to retrieve necessary text translations from its translation-aware properties and to apply them to all the components in your form for which respective translation-aware properties were created.
AutoSkipEmpties	Boolean	If True TsiLang will automatically skip components which have no translations defined for any language.

Property Name	Data Type	Description
ChangeLocales	Boolean	This property indicates whether system local constants should be changed or not when new user interface language is being selected.
DefaultLanguage	Integer	Use this property to indicate which language shall be used as "default" when you use UseDefaultLanguage set to True. "Default" language is necessary when using methods like GetTextOrDefault or component has empty translation for active language but TranslateType property is set to ttGetDefault.
DoNotTranslate	TStrings	DoNotTranslate property contains the list of component names that shouldn't be translated. It is useful if you want to exclude some components from translation list for decreasing the translation speed
ExcludedProperties	TStrings	This property contains names of properties that should be excluded from translation. For example, here could be included TNotebook.ActivePage property.
HandleResourceStrings	Boolean	If this property is True any resource string of the module is replaced with its translation after string is loaded.
LangNames	TStrings	Language names, ex: English, German, etc.
Language	String	The same as ActiveLanguage but uses the values from LangNames rather than integer constants.
NumOfLanguages	Integer	Indicates the total number of languages for which translation is possible to be done.
SmartExcludeProps	TStrings	This property contains list of specified component properties that should be excluded from translation. For example, it is necessary sometimes to exclude the Text property of some edit components.
TranslateExtendedFirst	Boolean	Indicates to change language for ExtendedTranslations before other properties. This is useful in some rare cases.
UseDefaultLanguage	Boolean	UseDefaultLanguage property indicates if TsiLang component to use translations specified for DefaultLanguage as values if no translation available for current language. This is very useful if your translations for some language are not complete and you wish to use values from other language as translation for missing items.

**Table 5 Other properties of TsiLang**

Property Name	Data Type	Description
ChangingCursor	TCursor	This property allows changing the application's cursor during switching languages.
DoRaiseExceptions	Boolean	Allows raising exceptions on using GetText method with string identifier that is not included in component.
IsInheritedOwner	Boolean	This property is introduced to inform TsiLang component that it is placed on inherited form/module or frame for better handling inherited modules. Set it to True for all TsiLang components that are placed on inherited modules.
LangDelim	Byte	This property indicates the symbol that is used in order to separate translations stored in TsiLang. The default value is 1.
LangDispatcher	TsiLangDispatcher	This property contains a reference to the TsiLangDispatcher component. When this property is not empty then TsiLangDispatcher manages the language changing. So, if you have several forms you can set up this property for each TsiLang per form to the same TsiLangDispatcher component and manipulate the properties of TsiLangDispatcher "NumOfLanguages" and "ActiveLanguage". When you will change "ActiveLanguage" property of TsiLangDispatcher it will change this property for all TsiLang components linked with it.
NeedAllDlg	Boolean	Specifies TsiLang to maintain list for all dialog captions including strings for dialogs like TOpenDialog, TSaveDialog and others.

Property Name	Data Type	Description
Options	TsiLangOption;  TsiLangOption = (loUseExtUDStrings, loUseExtCommonStrings);	<p>Options property indicates how to share translations for specified properties.</p> <p>loUseExtUDStrings- invokes the search for specified user defined string translation through other TsiLangs.</p> <p>loUseExtCommonStrings- invokes the search for translation of Dialogs or Locales properties through other TsiLangs.</p> <p>The search is done through all TsiLangs that have the LangDispatcher property pointed to the same TsiLangDispatcher component. In the case if LangDispatcher property is nil no searching will be performed.</p>
TranslateType	TTranslateType;  TTranslateType = (ttNoChange, ttGetDefault);	<p>This property indicates how to translate non-translated items in user interface. If this property value is ttNoChange then UI item's property such as caption, title, etc. is not changed when no translations are available for the current language. The ttGetDefault indicates that the value for the first language will be used (default value).</p>

Property Name	Data Type	Description
OnlineTranslation	<pre>TOnlineTranslateOptions = class(IPersistent)   published   property AutoTranslateEmptyItems: Boolean;   property BaseLanguageIndex: Integer;   property LanguageNamesAreExts: Boolean;   property      Translator: TsiInternetTranslator; end;</pre>	<p>This property is a published property of TsiCustomLang class which makes it available for all its descendants such as TsiLang, TsiLangLinked etc. Use this property if you wish to get on-line translations for untranslated items in TsiLang translations. Assign Translator property to any configured TsiInternetTranslator component, set BaseLanguageIndex to the index of language that will be used as base for translations and set AutoTranslateEmptyItems to True to force TsiLang to try to get on-line translations for untranslated items during language changing. Language names from LangNames property of TsiLang component will be used to get the on-line translations. So be sure that language names in TsiLang component are matched with names from CurrentServiceLanguages list of TsiInternetTranslator component. In case language names in TsiLang component are just languages' extensions then set LanguageNamesAreExts property to True.</p>

### Methods

❖ **function** AddString(TextID: string; const AStrings: array of string): boolean;

AddString method adds string constants AStrings to Strings property with ID equal to TextID. This method can be used for existing constants editing. The method replaces old constants with new ones in the case if TextID is the same. Returns True if method finished successfully otherwise False.

❖ **function** DeleteString(TextID: string): boolean;

DeleteString method removes constants with ID equal to TextID. Returns True if method finished successfully otherwise False.

❖ **procedure** ReplaceStringValue(AStrings: pStrings; Value, AName: string; ALang: integer); **virtual**;

ReplaceStringValue method is used for run-time translations replacement.

AStrings: pointer to TsiLang TStrings property. For ex. "@siLang1.Captions".

Value: string value that should be placed into the translations list.

AName: string identifier of the translation. For instance, for the following properties such as Captions, Hints, etc. this value represents the name of the component that



should be translated.

ALang: language number the Value (translation) is associated with.

- ❖ **function** `GetText(const TextID: string): string;`  
GetText method returns string constant by TextID depending on ActiveLanguage property.
  
- ❖ **function** `GetTextOrDefault(const TextID: string): string;`  
GetTextDefault method returns string constant by TextID depending on ActiveLanguage property. If there is no translation for TextID in active language the return value is taken for default language (number 1).
  
- ❖ **function** `GetTextC(TextID: string): PChar;`  
  
**function** `GetTextOrDefaultC(TextID: string): PChar;`  
  
**function** `GetTextW(const TextID: string): WideString;`  
  
**function** `GetTextOrDefaultW(const TextID: string): WideString;`  
  
**function** `GetTextOrDefined(const TextID: string; const ADefined: Integer): string;`  
  
**function** `GetTextOrDefinedW(const TextID: string; const ADefined: Integer): WideString;`

Methods with **C** postfix are useful for **C++Builder** programmers since most methods in C++ expect **char \* (PChar)** as input. Methods with **W** postfix are useful for creating **Unicode** applications since they return **WideString** as result.



**Note:** **AnsiString** returned by TsiLang is converted to the **WideString** using **Charset** settings for current language. So be sure if you use these methods you have specified **Charset** settings for each language, otherwise the conversion may be incorrect.

**GetTextOrDefined** works the same as **GetTextOrDefault** but returns the value for **ADefined** language in case no translation for active language available.

- ❖ **function** `GetTextByInt(ID: Integer): string;`  
GetTextByInt method returns string constant by ID from "Strings" property depending on ActiveLanguage property. The difference between GetText and GetTextByInt is so, that the later uses an integer value as an input parameter. Integer IDs for "Strings" property should be introduced prior to method invocation.

**Methods to  
work with SIL  
files**

- ❖ **function** `LoadStringsFromFile(AFileName: string; CanRewrite: boolean): integer;`

LoadStringsFromFile method loads string constants from file AFileName. CanRewrite indicates desired action to be done: append or rewrite string constants. Return value is equal to the count of loaded strings if no error, otherwise it is equal to 1.

- ❖ **function** LoadFromFile (PropType: TStringsType; AFileName: string; CanRewrite: boolean): integer;  
LoadFromFile method loads property defined by PropType from the file named AFileName. In the case if CanRewrite is set to True then the translation information stored in PropType property will be removed. Otherwise, if CanRewrite is set to False then the translation information stored in the file AFileName will be added to the end of the translation list. With CanRewrite set to True you should not obligatory delete all previous translation information if you want to use new translations. Otherwise (when CanRewrite is set to False) you should be aware of the fact that if there are several translation strings for the component with the same name (for ex. "Label1"), then the first translation string found will be used for the following translation. If you want to avoid this situation you should delete old translation information prior to LoadFromFile invocation.
- ❖ **procedure** LoadAllFromFile (AFileName: string; CanRewrite: boolean);  
LoadAllFromFile method loads all TsiLang properties (Captions, Hints, Font etc.) from the file named AFileName and if CanRewrite is set to True then all translation information will be removed before loading translations from file.
- ❖ **function** SaveStringsToFile (AFileName: string; Delimiter: char): integer;  
SaveStringsToFile method saves Strings property in the file AFileName. Delimiter symbol is used in order to separate string constants of different languages. Return value is equal to the count of saved strings or -1 if any error occurs.
- ❖ **function** SaveToFile (PropType: TStringsType; AFileName: string; Delimiter: string): integer;  
SaveToFile method saves property defined by PropType in the file named AFileName and uses the delimiter defined by Delimiter. Delimiter is used in order to separate strings, written in different languages. Beginning from version 4.9.1 the string can be used as a delimiter instead of the single char as it was done in the earlier versions.
- ❖ **procedure** SaveAllToFile (AFileName: string; Delimiter: string);  
SaveAllToFile method saves all TsiLang properties (like Captions, Hints, Fonts and etc.) in the file named AFileName and uses delimiter defined by Delimiter in order to separate strings in different languages.
- ❖ **function** MergeFromFile (PropType: TStringsType; AFileName: string): boolean;  
MergeFromFile method merges the translation information stored in the file named AFileName to the property defined by PropType.  
Method reads string translations from the file and adds them to the end of the

respective list in the PropType property. In the case if the file AFileName does not contain the complete set of translations for any of the components, then missing information is taken from TsiLang.

The return value is set to True if merging was done successfully; otherwise it is set to False.

- ❖ **function** MergeAllFromFile(AFileName: string): boolean;  
MergeAllFromFile method executes MergeFromFile method for all TsiLang properties (Captions, Hints, Font etc.) from the file named AFileName. The return value is set to True if merging was done successfully otherwise it is set to False.
  - ❖ **procedure** LoadAllFromFileDNC(AFileName: string; CanRewrite: boolean);  
LoadAllFromFileDNC method is similar to LoadAllFromFile method except the fact that it does not change the language that is currently active after loading.
  - ❖ **function** MergeAllFromFileDNC(AFileName: string): boolean;  
MergeAllFromFileDNC method is similar to MergeAllFromFile method except the fact that it doesn't change language after merging.
  - ❖ **procedure** LoadExtendedFromFile(const FileName: Tstring; const CanRewrite: boolean);  
LoadExtendedFromFile method loads ExtendedTranslations from the file named AFileName. In the case if CanRewrite is set to True then the translation information stored in ExtendedTranslations property will be removed
  - ❖ **procedure** SaveExtendedToFile(const FileName, Delimiter: Tstring);  
SaveExtendedToFile method saves TsiLang ExtendedTranslations property in the file named AFileName and uses delimiter defined by Delimiter in order to separate translations in different languages.
  - ❖ **procedure** LoadLanguageByProp(PropType: TStringsType; const AFileName: Tstring; const LangName: Tstring);  
Loads specified language LangName into property specified by PropType.
  - ❖ **procedure** LoadLanguageByExt(const AFileName: Tstring; const LangName: Tstring);  
Loads language LangName into extended translations.
  - ❖ **procedure** LoadLanguage(const AFileName: Tstring; const LangName: Tstring);  
Loads specified language from the external file.
- Methods to work with SIB files**
- ❖ **procedure** LoadPropFromBinaryFile(PropType: TStringsType; const FileName: string);  
Loads selected property from binary (SIB) file.
  - ❖ **procedure** LoadAllFromBinaryFile(const FileName: string);  
Loads all translations from binary file.

- ❖ **procedure** `LoadAllFromBinaryStream(AStream: TStream);`  
Loads all translations from binary stream. Can be used to load translation from the different storage like database and so on.
- ❖ **procedure** `SaveAllToBinaryFile(const FileName: string);`  
Saves all translations into binary file.

Methods listed below are the same as above except they use stream to load or save translations. Please note the stream format must be SIL format.

#### Methods to work with streams

- ❖ **function** `LoadFromStream(PropType: TStringsType; AStream: TStream; CanRewrite: Boolean): Integer;`
- ❖ **function** `SaveToStream(PropType: TStringsType; AStream: TStream; const Delimiter: Tstring): Integer;`
- ❖ **procedure** `LoadAllFromStream(AStream: TStream; CanRewrite: Boolean);`
- ❖ **procedure** `SaveAllToStream(AStream: TStream; const Delimiter: Tstring);`
- ❖ **procedure** `LoadExtendedFromStream(const AStream: TStream; const CanRewrite: Boolean);`
- ❖ **procedure** `SaveExtendedToStream(const AStream: TStream; const Delimiter: Tstring);`
- ❖ **function** `MergeFromStream(const PropType: TStringsType; const AStream: TStream): Boolean;`
- ❖ **function** `MergeAllFromStream(AStream: TStream): Boolean;`
- ❖ **function** `MergeAllFromStreamDNC(AStream: TStream): Boolean;`

Methods listed below are the replacements of the standard Delphi or C++ Builder methods, which are generally the same, excluding the fact that button caption values are taken from the translation data.

#### Methods to work with message boxes

- ❖ **function** `InputBox(const ACaption, APrompt, ADefault: string): string;`
- ❖ **function** `InputQuery(const ACaption, APrompt: string; var Value: string): Boolean;`
- ❖ **function** `MessageDlg(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel): Integer;`
- ❖ **function** `MessageDlgPos(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y:`

```
Integer; const DefaultBtn: TMsgDlgBtn = mbOK; const
CancelBtn: TMsgDlgBtn = mbCancel): Integer;
```

- ❖ **function** MessageDlgPosHelp(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer; const HelpFileName: string; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel): Integer;
- ❖ **procedure** ShowMessage(const Msg: string);
- ❖ **procedure** ShowMessageFmt(const Msg: string; Params: array of const);
- ❖ **procedure** ShowMessagePos(const Msg: string; X, Y: Integer);
- ❖ **function** CreateMessageDialog(const Msg: Tstring; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel): TForm;
- ❖ **function** MessageDlgTimeOut(const Msg: Tstring; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel; const TimeOutms: Cardinal = 0): Integer;
- ❖ **function** MessageDlgPosTimeOut(const Msg: Tstring; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel; const TimeOutms: Cardinal = 0): Integer;
- ❖ **function** MessageDlgPosHelpTimeOut(const Msg: Tstring; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer; const HelpFileName: Tstring; const DefaultBtn: TMsgDlgBtn = mbOK; const CancelBtn: TMsgDlgBtn = mbCancel; const TimeOutms: Cardinal = 0): Integer;

**\*TimeOut** methods are the extension of the respective **MessageDlg\*** methods with time-out functionality. The message box will be closed automatically after TimeOutms milliseconds expire. The return value will be respective to the DefaultBtn parameter. If there is no button equal to DefaultBtn then the **mrTimeOut(32000)** value will be returned.

#### Other methods

- ❖ **procedure** MoveLanguage(const FromIndex, ToIndex: Integer);

Moves language from FromIndex to ToIndex.

- ❖ **procedure** InsertLanguage(const Index: Integer; const ALanguage: string);

Inserts new language with name ALanguage into Index position.

- ❖ **procedure** `GetLanguageNamesFromFile(const FileName: string; const LanguageList: TStrings);`  
Gets available language names in SIL or SIB file into LanguageList list.

**Events****Table 6 TsiLang events**

Event Name	Type	Description
OnChangeLanguage	<b>procedure</b> (Sender: TObject)	Fired every time after active language changed.
OnLanguageChanging	<b>procedure</b> (Sender: TObject; <b>const</b> NewLanguage: Integer; <b>var</b> AllowChange: Boolean)	Fired every time before language change. Set AllowChange to False to cancel change.
OnExtendedChanging	procedure (Sender: TObject; <b>const</b> NewLanguage: Integer; <b>const</b> Item: TsiExtendedItem; <b>var</b> NewValue: <b>string</b> )	Fired every time the value of extended item is going to change.

**TsiLangLinked**

In order to achieve better performance and to reduce memory usage as well as the size of application file it is better to use TsiLangLinked component. It is “linked” with the existing TsiLang component and shares with it the translation information that is potentially common for the several forms in your application (see the scheme below).

The difference between “TsiLang” and “TsiLangLinked” is that the properties “DlgCaptions” and “Locales” are run-time only in TsiLangLinked while they are design-time properties in TsiLang class.

**Table 7 TsiLangLinked specific properties**

Property Name	Data Type	Description
CommonContainer	TsiCustomLang	Indicates the component to be used in order to retrieve information about project wide strings such as standard dialog strings and locales information

**TsiLangRT**

You know already that it is enough to use TsiLang in order to internationalize your application, but to do this you must involve a person who knows well the foreign language your project is translated to. Sometimes it is difficult or inconvenient to do and in this case you can delegate the translation of your project to your end-level user. This is

possible to perform using TsiLangRT (TsiLang Run-Time). For allowing end-user to translate text strings from your form you must use **EditAll()** method. Unit **siLangRT** contains two global variables that could be used for providing your end-user the localized version of **Translations Editor**:

**RT\_SILFile- Translations Editor** will load its translations from this SIL file if defined and points to existing SIL file. The initial SIL file for **Translations Editor** is included into standard delivery set and called **RT.SIL**. So you can translate this file and ship it with your application in order to provide your end-users with localized version of built-in editor.

**RT\_ActiveLanguage- Translations Editor** will switch to this language if applicable on activation.

### Methods

```
function EditProperty(PropType: TStringsType): Boolean;
```

“**EditProperty**” method of TsiLangRT is invoked with appropriate property type as one of the following:

```
TStringsType = (stCaptions, stHints, stDisplayLabels, stFonts,
stMultiLines, stDialogs, stStrings, stOther, stLocales,
stCollections, stCharSets, stListView_TreeView_Items);
```

This method calls built-in property editor, providing end-user with the possibility to translate text strings by him. You must only take care about the appropriate internationalization user interface in your application.

```
function EditAll: Boolean;
```

Also you might try to use “**EditAll**” method to perform editing of all translation properties at once. To translate **ExtendedTranslations** property you should use “**EditExtended**” method.

```
function EditExtended: Boolean;
```

All methods above return **True** when user saved translations on close and **False** when user decided to cancel changes.

### Properties

All properties of TsiLangRT are identical to TsiLang's ones, except the following:

**Table 8 TsiLangRT specific properties**

Property Name	Data Type	Description
LoadOnCreate	Bool	LoadOnCreate property indicates whether or not to load the translation information stored in the file named StorageFile on component creating.
StorageDelimiter	String	StorageDelimiter property indicates the delimiter to be used when translation is stored in StorageFile.

Property Name	Data Type	Description
StorageFile	TFileName	StorageFile property indicates the name of the file, which keeps the translation information

## TsiLangRTSE

TsiLang RTSE component is very similar to TsiLangRT. The difference is in the editor that is used for translation information modification and in the way of its invocation. TsiLangRT uses built-in translation editor, whereas TsiLangRTSE uses SIL Editor - stand-alone application and automation server. While using TsiLangRTSE your application is communicating with SILEditor via the COM interface.

TsiLangRTSE “EditStrings” method is used instead of “EditProperty” method (TsiLangRT) in order to invoke automation server (embedded in SIL Editor) for translation information modification. :

### Methods

❖ **procedure** EditStrings(DefPropty: string; FDelim: string);

EditStrings method saves all translation stored in itself into SILFile, then activates automation server object embedded in SIL Editor application for translating it and after that loads translation information from the file named SILFile. The DefPropty indicates which property will be activated in SIL Editor by default. The FDelim parameter indicates the delimiter that will be used for saving translation in the file.

### Properties

All properties of TsiLangRTSE are identical to TsiLang's ones, except the following:

**Table 9 TsiLangRTSE specific properties**

Property Name	Data Type	Description
LoadOnCreate	Bool	Indicates whether or not to load the translation information stored in the file named SILFile on component creating.
SILEditor	string	Indicates the automation server object name for activating SIL Editor.
SILFile	TFileName	Indicates the name of the file, which keeps the translation information

## TsiLangTLV

TsiLangTLV component is an enhanced version of TsiLang component, which provides you with the possibility of TTreeView and TListView items translation.



Since `TTreeView.Items` and `TListView.Items` are objects, derived from `TObject`, they are not supported by RTTI system and `TsiLangTLV` cannot trace Items modification such as Items removal and other changes in Items structure. Thus, you can use this component properly only if you are sure that contents of your `TTreeView` and `TListView` will not be changed at run-time. For editing translation just edit "ListView\_TreeView\_Items" property of `TsiLangTLV` in the same manner as all other properties of `TsiCustomLang` descendants.

**Properties** All properties of `TsiLangTLV` are identical to `TsiLang`'s ones, except the following

**Table 10 TsiLangTLV specific properties**

Property Name	Data Type	Description
ListView_TreeView_Items	TStrings	Contains translation information of <code>TTreeView</code> and <code>TListView</code> items

## TsiLangDispatcher

In the case, when your application contains more than one form, you need to use more than one `TsiLang` component, because `TsiLang` component is aware of text strings to be translated only within the form it is placed at. Thus, each form in your project needs in one `TsiLang` component, i.e. the number of `TsiLang` components in your project should be equal to the number of forms to be translated in it. When the number of forms is big enough then it is rather complicated to manipulate and manage all the `TsiLang`'s manually. For instance, when you have 12 forms, you should write 12 lines of the above code in order to change an active language for all forms.

In order not to do it manually and in order not to keep in mind all your `TsiLang`'s with their properties, you can delegate `TsiLang`'s management process to the `TsiLangDispatcher` component.

The way of `TsiLangDispatcher` using is very simple. You must do the following:

- Place `TsiLangDispatcher` on a form (it should be the Application's main form or some `DataModule` which is "auto-created" and could be accessed from all project forms);
- Link each of your Application's `TsiLang` components with that `TsiLangDispatcher`. It can be done easily by setting `TsiLangDispatcher` property of `TsiLang`.
- Set `LangNames` property of the `TsiLangDispatcher` by the same way as it described for a `TsiLang` component;

- Set `ActiveLanguage` property of the `TsiLangDispatcher`;

Every `TsiLang` component linked with a `TsiLangDispatcher` component reads the values of “`ActiveLanguage`”, “`NumOfLanguages`” and “`LangNames`” properties from the latter one and receives notifications about changing these properties. Therefore it is enough to perform only one assignment like this:

```
...
MainForm.silangDispatcher1.ActiveLanguage := 3;
...
```

In order to all Application's `TsiLang` components consistently change their `ActiveLanguage` property.

### Methods

- ❖ **procedure** `LoadAllFromFile(FileName: string)`: This method allows to load the specified file into all `TsiLang` components linked with this dispatcher.
- ❖ **procedure** `SaveAllToFile(const FileName, Delimiter: string)`: This method allows to save the translation content of all `TsiLang` components linked to this dispatcher.
- ❖ **procedure** `LoadAllFromStream(AStream: TStream)`: This method allows to load translations for the specified stream into all `TsiLang` components linked with this dispatcher.
- ❖ **procedure** `SaveAllToBinaryFile(const FileName: string)`: This method allows to save the translation content of all `TsiLang` components linked to this dispatcher to binary (SIB) file.

### Properties

Table 11 `TsiLangDispatcher` properties

Property Name	Data Type	Description
<code>ActiveLanguage</code>	Integer	Range: [1- <code>NumOfLanguages</code> ]. Automatically sets <code>ActiveLanguage</code> property of all <code>TsiLang</code> components, connected with <code>TsiLangDispatcher</code>
<code>NumOfLanguages</code>	Integer	Indicates the total number of languages for which translation is possible to be done.
<code>LangNames</code>	<code>TStrings</code>	Language names, ex: English, German, etc.
<code>Language</code>	String	The same as <code>ActiveLanguage</code> , but uses the values from <code>LangNames</code> rather than integer constants. Automatically sets <code>Language</code> property of all <code>TsiLang</code> components, connected with <code>TsiLangDispatcher</code>

Property Name	Data Type	Description
Filename	String	This property contains the name of SIL file that should be used for loading into all TsiLang components linked to this dispatcher. It is useful to use this property when you don't want to call LoadAllFromFile method directly. When this property contains value all TsiLang components linked to this dispatcher even placed on dynamically created forms will load translations from the specified file.
SiLangsCount	Integer	Indicates the amount of TsiLang components linked with current TsiLangDispatcher.
SiLangs[Index: Integer]	TsiCustomLang	Allows accessing to specified TsiLang component linked to current TsiLangDispatcher. This property is useful when you need to perform the loading or saving for all TsiLangs at once.
TestModeInfo	TTestModeInfo	Allows to configure and set dispatcher into "Test" mode. This is useful if you wish to test translations before the actual translations done. Please see description of TTestModeInfo type below.
DefaultLanguage	Integer	Use this property to set DefaultLanguage property for all TsiLang components linked to this dispatcher.
UseDefaultLanguage	Boolean	Use this property to set UseDefaultLanguage property for all TsiLang components linked to this dispatcher.
TranslationMemoryOptions	TTranslationMemoryOptions	<p>TranslationMemoryOptions.Active - set this to True to enable Translation Memory at run-time.</p> <p>TranslationMemoryOptions.AutoUseForComponents - set this to True to enable auto-translation of untranslated items at run-time.</p> <p>TranslationMemoryOptions.ReturnEmptyForUntranslated - set this to True if you wish the TranslationMemory() method to return empty for untranslated items. Otherwise it will return the BaseValue.</p>

Table 12 TTestModeInfo fields

Property Name	Data Type	Description
Active	Boolean	Indicates if test mode is active.

Property Name	Data Type	Description
Kind	TTestModeKind	Indicates how the initial translation (default value) will be modified for test mode: <b>tmkExpandWidth</b> - expand the default value by <b>ExpandWidthPercent</b> percent. <b>tmkFlipCase</b> - fLIP cASE for all characters in the default value. <b>tmkExpandChars</b> - the default value will be expanded on <b>ExpandCharsPercent</b> percent using <b>PaddingChar</b> character.
ExpandWidthPercent	Integer	Percent value to expand the width of the default value.
ExpandCharsPercent	Integer	Percent value to expand the width of the default value by <b>PaddingChar</b> .
PaddingChar	Char	Char that will be used for expanding.

## Events

Table 13 TsiLangDispatcher events

Event Name	Type	Description
OnLinkToDispatcher	<b>procedure</b> (Sender: TObject; ASiLang: TsiCustomLang)	Fired every time any TsiLang gets linked to this dispatcher.
OnLanguageChanged	<b>procedure</b> (Sender: TObject)	Fired every time after active language changed.
OnLanguageChanging	<b>procedure</b> (Sender: TObject; <b>const</b> NewLanguage: Integer; <b>var</b> AllowChange: Boolean)	Fired every time before language change. Set AllowChange to False to cancel change.

**TsiInternetTranslator**

Using the TsiInternetTranslator component can provide you with more flexibility to automate translations. With this component, you can set it up to translate your application or anything else, using any of the most popular online translators (for example DeepL or Google Translate) at run time.

In order to use TsiInternetTranslator, you need to drop the component onto your form, and then you can set up the respective properties either using Object Inspector, or through hard code.

**Methods**

❖ **function** GetExtByLanguageName (const LanguageName: string): string;

This method returns language extension by its name from the list of current service's languages. You can use this method to get extensions for the languages to use with "TranslateByLanguageExt" method, which is faster than "TranslateByLanguageName" because it does not perform the search for the language extensions.

❖ **function** IsValidLanguageName (const LanguageName: string): Boolean;

This method returns True if the language specified by the "LanguageName" could be found in the list of current service's languages. You can use this method to perform preliminary check for validity of language names to use for online translations.

❖ **function** TranslateByLanguageName (const Text, SourceLanguage, TargetLanguage: string): string;

This method translates the text, specified by the "Text" property, from the source language, specified by the "SourceLanguage", to the target language, specified by the "TargetLanguage", using the current translation service defined by the "ActiveService" property. As online translator services use language extensions for language translations instead of the language names, this method first finds the matches for the specified languages in the list of current service's languages, and then uses the languages' extensions and "TranslateByLanguageExt" method to perform the translation. If need to perform bulk translations, it is recommended to use "GetExtByLanguageName" method to get language's extension and then use "TranslateByLanguageExt" method. If the returned value is empty and no error was raised, then you can check the "LastError" property to get additional information about the problem.

❖ **function** TranslateByLanguageExt (const Text, SourceLanguageExt, TargetLanguageExt: string): string;

This method translates text, that is specified by "Text" parameter, from the source language, specified by "SourceLanguageExt", to the target language specified by "TargetLanguageExt", using the current translation service defined by the "ActiveService" property. As online translator services use language extensions for language translations instead of the language names, both source and target languages must be specified by their extensions defined in the service. If you do not know the language's extension, you can use the "TranslateByLanguageName" method to translate by language's

name or use “GetExtByLanguageName” method to get the language's extension. If the returned value is empty and no error was raised, then you can check the “LastError” property to get additional information about the problem.

**Properties** Table 14 TsiInternetTranslator properties

Property Name	Data Type	Description
ActiveService	TAutoTranslateService	ActiveService property is used to select which translator service to use. Currently, there are four possible choices: Google, DeepL, BING and MSTerminology
LastError	TTranslateError	Use this property to detect the possible reason for online translation failing. You can find a table that describes the meaning of the error below.
FixFormatStrs	Boolean	When this property is set to <b>True</b> the online translation result will be checked to contain the same format specifiers (%s, %d), as the original value.
BINGSettings	TBingTranslatorOptions	This property is used to set up the settings for BING Translator, if it was chosen as the translation service for the “ActiveService” property. Use this property to provide your BING <b>APIKey</b> so the program can make on-the-go translations via BING.
DeepLSettings	TDeepLTranslateOptions	This property is used to set up the settings for DeepL Translate, if it was chosen as the translation service for the “ActiveService” property. Use this property to

		provide your DeepL <b>APIKey</b> , so the program can make on-the-go translations via DeepL. Additionally, you can modify the <b>UsePro</b> settings to state whether you have the Pro version of DeepL or the standard free one.
GoogleSettings	TGoogleTranslateOptions	This property is used to set up the settings for Google Translator, if it was chosen as the translation service for the “ActiveService” property. If you use <b>Google Cloud Translate</b> then set <b>UseCloudTranslate</b> property to <b>True</b> and provide your Google <b>APIKey</b> . However, it is not necessary and is optional for Google. If you use the free version of <b>Google Translate</b> , then set the <b>UseCloudTranslate</b> property to <b>False</b> . <b>FreeTranslateDelaySecs</b> property sets the delay between translate requests in order to prevent getting banned from Google due to very frequent requests.
MSTerminologySettings	TMSTerminologyTranslateOptions	This property is used to set up the settings for Microsoft Terminology Translator, if it was chosen as the translation service for the “ActiveService” property. You do not need to provide the <b>APIKey</b> and it can be neglected for MSTerminology. Additionally, you can modify the MSTerminology specific settings, such as <b>Sensitivity</b> , <b>OperatorKind</b> , and

		<p><b>SeekSource.</b> <b>Sensitivity</b> determines if the translation will be case sensitive or not. <b>OperatorKind</b> determines if the translation should be made via the exact phrase, exact word, or where it is contained. <b>SeekSource</b> determines where to look for the translation, either in just <b>Terms</b>, <b>User Interface Strings</b> or both combined.</p>
--	--	---

Find the **LastError** values table below:

Value	Meaning
teNoError	There were no errors and all operations completed smoothly.
teServiceUndefined	ActiveService property is set to <b>atsUndefined</b> , so no operation is available
teInvalidSourceLanguage	Source language name was not found in the list of current service's languages.
teInvalidTargetLanguage	Target language name was not found in the list of current service's languages.
teUndefinedSourceExt	Source language extension is empty.
teUndefinedTargetExt	Target language extension is empty.

### TsiLangCombo

This component inherits all standard methods and properties from the TComboBox component, but allows displaying and selecting languages of a corresponding TsiLang (TsiLangDispatcher) component but in an advanced mode. For example, you can specify for each supported language its own graphic image, font and charset (to display language name), as well as custom title.



**Properties** Table 15 TsiLangCombo Properties

Property Name	Data Type	Description
ChangeLanguage	Boolean	TsiLangCombo will automatically change the “ActiveLanguage” property of “siLang” or “siLangDispatcher” assigned component when user changes the selected language, if this property is set to true.
LanguageInfos	TLanguageInfos	It is a collection of TLanguageInfo items that describe display settings for the languages displayed in the combo box.
siLang	TsiCustomLang	Use this property to link the combo-box with a TsiLang component.
siLangDispatcher	TsiLangDispatcher	Use this property to link the combo-box with a TsiLangDispatcher component

## Dialogs

Dialogs are often used components in your application and they are very important for convenient and habitual user interface. It is evident, that powerful multilingual tools, such as TsiLang must provide user with the possibility of translating dialog text elements. TsiLang provide you with such a possibility and contains dialog elements translation information in its DlgCaptions property, but it cannot translate standard dialogs directly because all standard dialogs do not publish their text elements as a set of properties. Issuing a new set of dialog components with included multilingual support solves the problem. The following dialog components are included into the TsiLang Component Suite:


- ❖ TsiOpenDialog
- ❖ TsiSaveDialog
- ❖ TsiColorDialog
- ❖ TsiFontDialog
- ❖ TsiPrinterSetupDialog
- ❖ TsiPrintDialog
- ❖ TsiFindDialog
- ❖ TsiReplaceDialog
- ❖ TsiOpenPictureDialog
- ❖ TsiSavePictureDialog
- ❖ TsiBrowseForFolder

All dialog components listed above (except for TsiBrowseForFolder, which is a component wrapper for Windows API function SHBrowseForFolder) are descendants of standard Delphi's or C++ Builder's dialogs with added siLang property and overridden "DoShow" private method. Just link them to the TsiLang component and type translation of all their strings in DlgCaptions property. All user defined dialog elements will be displayed in language that is currently active in the linked TsiLang.

### Properties

**Table 16 siDialogs specific properties**

Property Name	Data Type	Description
siLang	TsiCustomLang	Specifies a TsiLang to be used for multilingual string constants retrieving while dialogs executing.

Chapter  
6

## Useful Information

If you have any questions about usage the TsiLang Components Suite feel free to send us e-mail on [support@sicomponents.com](mailto:support@sicomponents.com). Also you can visit our Forum at <https://www.sicomponents.com/forum/> where you can share experience with other TsiLang customers and find answers on many questions.

## Tips and Tricks

### Exclude not used components and properties

First, some components (for example, TTable, TQuery) do not need translations at all. Include such components in the “DoNotTranslate” property of corresponding TsiLang component. Second, some properties (for example, “FieldName”) should not be translated for all components. Add these properties to “ExcludedProperties” property of the TsiLang. And at last, if specific properties of a single component should not be translated add them to “SmartExcludeProps” of the TsiLang. Using these rules might significantly improve performance of TsiLang components (especially on Data Modules) and will not confuse your translator with needless data.

TsiLang will add automatically most often and commonly used for exclusion property names, such as TableName, DatabaseName and others to “ExcludedProperties” property. To switch off this functionality you will need to edit manually **SI.INC** file, disable compiler directive `{$DEFINE ADDCOMMONEXCLUDE}` (to disable just change it to `{$DEFINE ADDCOMMONEXCLUDE}`) and rebuild the TsiLang package. In case you need to remove some automatically added property names from this list just select them in the “ExcludedProperties” list and remove.



**Note:** Some components and properties shall be always excluded from translations! The examples of such components are:

1. **TWebBrowser** and its descendants (“translating” it at run-time raises **EOleException**)
2. Properties like **TableName** and **DatabaseName** for data sets (cannot be changed while data sets are active)
3. Usually, properties like **SQL**, **PickList**, **FieldName**, **IndexName**, **LookupDisplayFields** and others.

### Use TSI: tags to skip hard-coded strings that should not be translated

`{TSI:IGNORE}` – ignores single line. Add this tag anywhere in the line with hard-coded string.

`{TSI:IGNORE ON}` – indicates that all lines after this line should be skipped when searching for hard-coded strings. All lines will be skipped till OFF tag found. Note: this tag should be in separate line and there should be no any other text in this line.

`{TSI:IGNORE OFF}` – indicates to stop skipping lines when searching for hard-coded strings. Note: this tag should be in separate line and there should be no any other text in this line.

`{TSI:IGNORE NEXT}` – excludes one following line of code from the translation regardless of `{TSI:IGNORE ON/OFF}`.

`{TSI:TRANSLATE NEXT}` – translates one following line of code from regardless of `{TSI:IGNORE ON/OFF}`.

`{TSI:IGNORE VALUE}` – ignores the string value right after the tag.

### Create multilanguage dialog boxes with custom controls

Sometimes you need to use the modified version of standard message boxes functions in order to display custom controls, for example “Don’t Ask Me Again” checkbox in MessageDlg function. In order to achieve this you can use CreateMessageDlg() method of TsiLang. It returns the instance of created form but won’t display it. So you can use something like the following:

#### Delphi Sample

```
function DontAskMessageDlg(const siLang: TsiCustomLang; const
Text: string; const MsgType: TMsgDlgType; MsgButtons:
TMsgDlgButtons): TModalResult;
var
  MessageForm: TForm;
  CheckBox: TCheckBox;
begin
  Assert(Assigned(siLang), 'No TsiLang instance passed!');
  MessageForm := siLang.CreateMessageDialog(Text, MsgType,
MsgButtons);
  try
    CheckBox := TCheckBox.Create(MessageForm);
    CheckBox.Parent := MessageForm;
    CheckBox.Caption := siLang.GetTextOrDefault('IDS_0');
    CheckBox.Left := 4;
    CheckBox.Top := MessageForm.Top - CheckBox.Height - 8;
    Result := MessageForm.ShowModal;
  finally
    MessageForm.Free;
  end;
end;
```

### Creating Unicode multilanguage dialog boxes



**Note:** This tip applies ONLY to non-Unicode version of IDE such as RAD Studio 2007 or earlier.

If you create Unicode applications in Delphi or C++Builder you will use some 3<sup>rd</sup> party replacement for standard controls. TsiLang Components Suite works fine with Unicode properties so you will be able to translate such controls as usual. But if you will just use message dialog methods of TsiLang it will use standard (ANSI) Delphi controls to build message forms. In order to build 100% Unicode application you will need to use Unicode controls in these methods as well. To achieve this you will need to “tell” TsiLang which controls to use to build forms for message boxes. Unit **siComp.pas** has global class variables that indicate what control classes will be used to build message forms. To change them to Unicode you just change these class variables to Unicode classes. If you use **TNT Controls** the code will look like the following:

#### Delphi Sample

##### initialization

```
InitTntEnvironment;  
MsgDlgFormClass := TTntForm;  
MsgDlgLabelClass := TTntLabel;  
MsgDlgEditClass := TTntEdit;  
MsgDlgButtonClass := TTntButton;
```

#### C++ Sample

```
MsgDlgFormClass = __classid(TTntForm);  
MsgDlgLabelClass = __classid(TTntLabel);  
MsgDlgEditClass = __classid(TTntEdit);  
MsgDlgButtonClass = __classid(TTntButton);
```

### Configuring Default Fonts for TsiLang



**Note:** This tip applies ONLY to non-Unicode version of IDE such as RAD Studio 2007 or earlier.

In order to properly view and edit all available languages in TsiLang Translation Editor and SIL Editor, be sure to configure Fonts and Default Fonts for TsiLang. To configure Default Fonts open Translation Editor and select in menu **Tools | Default Fonts**. In appeared dialog enter language name (**Note:** TsiLang uses language names to detect necessary font and Charset, so be sure to use same language names in the project and in **Default Fonts**), font name (in most cases Tahoma is enough) and Charset for this language.

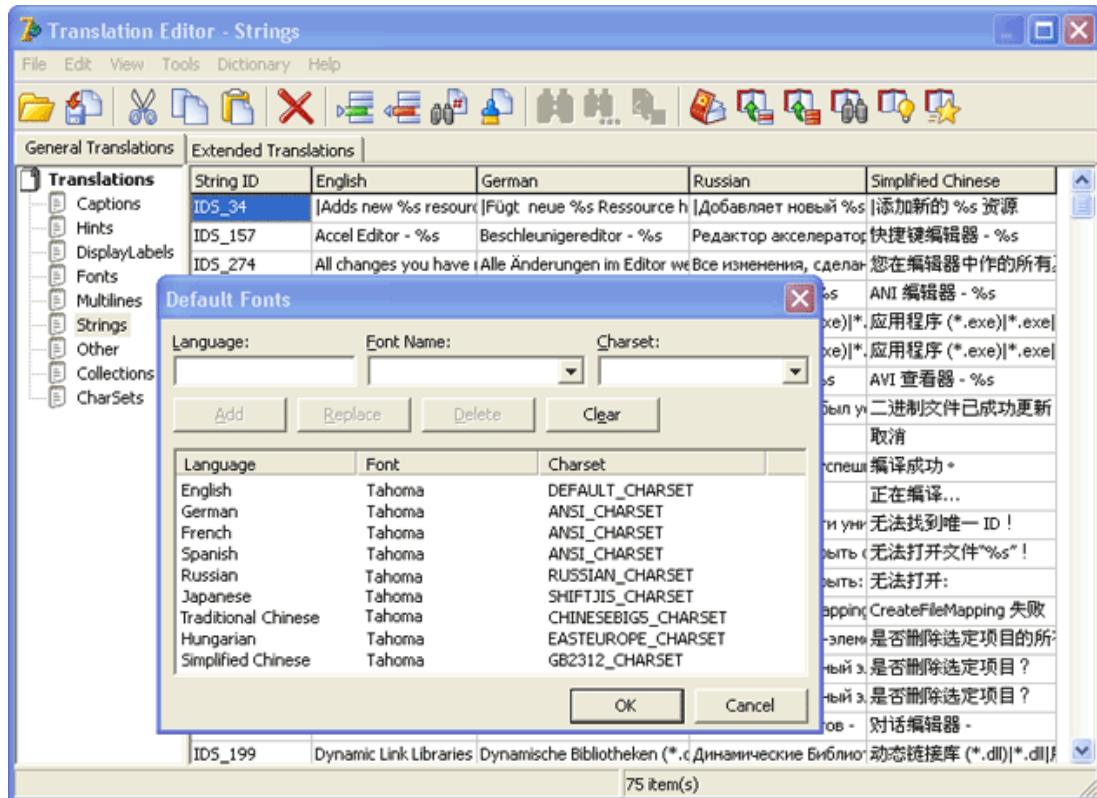


Figure 25 Configuring Default Fonts

After you've set **Default Fonts** you can set **Auto-Use Default Fonts** option in TsiLang Translation Editor (menu **Tools | Settings**) and Translation Editor will automatically fill Fonts and Charsets sections using data from **Default Fonts** (if available).

### Performing custom modifications during language changing

You may need to perform custom modifications on values that TsiLang apply to controls upon language changing. As an example of such case could be some 3<sup>rd</sup> party controls that use UTF8-encoded string properties. To handle such case unit **siComp.pas** introduces global variable **siInterceptStringChange** of a type:

**type**

```
TsiInterceptStringChange = procedure (const AObject: TObject;  
const PInfo: PPropInfo; var APropValue: string);
```

TsiLang will call this procedure if it is assigned and this will allow you to perform custom modifications on the string being applied on control's property. **AObject** parameter indicates the object (or component) whose property will be modified to **APropValue** value. Property could be determined by **PInfo** parameter of a **PPropInfo** type. **PPropInfo** type is declared in **TypInfo** Delphi's unit.

### Delphi Sample

```

procedure ConvertToUTF8(const AObject: TObject; const PInfo:
PPropInfo; var APropValue: string);
var
  WS: WideString;
  S: string;
begin
  if AObject is TControl then // checking if this is UTF8 control
  begin
    if PInfo.Name = 'UTF8Caption' then // checking if this is UTF8
    property
    begin
      WS := AnsiStringToWideStringCP(APropValue,
YOURFORM.siLang1.CurrentCharset); // convert ANSI string from TsiLang to
Unicode
      S := UTF8Encode(WS); // Convert Unicode string to UTF8
      APropValue := S; // Pass UTF8 string back to TsiLang to apply to
control's property
    end;
  end;
end;

initialization
  siInterceptStringChange := ConvertToUTF8; // Assign procedure
variable

```

### Translating 3<sup>rd</sup> party forms without sources

Sometimes if you don't have full source code of 3rd party components but they use some built-in dialogs you won't be able to translate them in "usual" way. To handle such case and be sure your application is 100% multilanguage we can advise you the following way:

1. Detect the type name of this dialog form.
2. Add the following code for Screen.OnActiveFormChange:

#### Delphi Sample

```

procedure TForm1.ScreenFormChange(Sender: TObject);
var
  siLang: TsiLangLinked;
begin
  if Screen.ActiveCustomForm is TYOUR_DIALOG_FORM then
  begin
    if siLang := TsiLangLinked.Create(Screen.ActiveCustomForm);
    try
      siLang.BuildList;
      siLang.SaveAllToBinaryFile('C:\SOMEFILENAME.SIB');
    finally
      siLang.Free;
    end;
  end;
end;

```

3. Run your application and activate this dialog.

4. Translate created SIB file and decide if you will use it as external or built-in into your application.
5. If you decide to use it as external storage then change `Screen.OnActiveFormChange` to the following:

**Delphi Sample**

```

procedure TForm1.ScreenFormChange(Sender: TObject);
var
    siLang: TsiLangLinked;
begin
    if Screen.ActiveCustomForm is TYOUR_DIALOG_FORM then
        begin
            siLang := TsiLangLinked.Create(Screen.ActiveCustomForm);
            siLang.LoadAllFromBinaryFile('C:\SOMEFILENAME.SIB');
            siLang.LangDispatcher := MainForm.siLangDispatcher1;
        end;
    end;

```

6. If you decide to use it built-in into your application then just create RC-file (you can use our Resource Builder <http://www.resource-builder.com>) and place this SIB file as RCDATA resource. Load this translation using **TResourceStream** and **LoadFromStream()** method of **TsiLang**;
7. That's all, now when this dialog appears it will be automatically translated.



**Note:** Example code provided above are for reference only! For your particular case it could be different.

**How to make TsiLang message boxes “styled” when VCL Style applied.**

When you use VCL Styles to apply styles to your RAD studio applications the message boxes displayed by **TsiLang** are not styled in case the **UseTaskDialog** property of **TsiLang** is set to **True**. To prevent this you can just set this property to **False**, but in case you would like to have message boxes based on **Task Dialog** when no application style is active, you can use the following trick:

1. We set the **UseTaskDialog** property to **True** by default.
2. Upon application active form change (inside the `Screen.OnActiveFormChange` event) we check if it is styled and then set this property to **False**.

The following code sample demonstrates this:

**Delphi Sample**

```

procedure TForm1.ScreenFormChange(Sender: TObject);
procedure ProcessActiveForm(AForm: TCustomForm);
var
    I: Integer;
begin
    for I := 0 to AForm.ComponentCount - 1 do
        if AForm.Components[I] is TsiCustomLang then
            begin

```



```
        TsiCustomLang (AForm.Components[I]).UseTaskMsgDlg :=  
StyleServices (AForm).IsSystemStyle;  
        Exit;  
    end;  
end;  
  
begin  
    if Screen.ActiveCustomForm <> nil then  
        ProcessActiveForm (Screen.ActiveCustomForm);  
    end;  
end;
```

---

## TranslationData as Text in DFM

By default the TranslationData property is stored in DFM (FMX) in binary format. This allows to use the same DFM under different versions of IDE, for example under Delphi 7 and RAD Studio XE or others, without losing translations when opening form file in IDE. Although this feature brings some inconvenience when using Version Control System and textual DFM files.

In case you want to use textual representation of TranslationData in DFM and **ONLY** if you don't use your projects with older IDE versions you can do the following:

- **Full Source** edition is required.
- Open **SI.INC** file located in {TsiLang}\Units folder.
- Activate **USETEXTDATA** define by removing the space before \$DEFINE. So the line will look like:

```
{ $DEFINE USETEXTDATA }
```

- Open TsiLang run-time package (the one **with** **\_r** postfix) into IDE and re-build it.
- Open TsiLang design-time package (the one **without** **\_r** postfix) into IDE and re-build it.

Next time you open your DFM files into IDE and save it the TranslationData property will be converted to text and you will be able to effectively use Version Control System.



**Note:** you will need to perform these actions every time upon downloading of a new version of TsiLang Components Suite. This shall be done after installing new version and **BEFORE** opening and saving your project in IDE. Otherwise the TranslationData property will be converted back to binary representation as by default TsiLang packages are built with this option deactivated in order to support the full range of IDE versions.

## Frequently Asked Questions

### How to translate resource strings?

If several strings are declared in **resourcestring** sections of your project the best solution is simply to replace the **resourcestring** with **const** keyword and handle the strings as usually. If the resource strings are declared in external units (for example, in third-party VCL libraries) you need to set “HandleResourceStrings” property of the TsiLang component to “True” and to add these strings to the translation data (see Resource Strings Wizard).

### I am using C++Builder. How do I translate string tables coded into .rc and .rh files?

We recommend you the following workaround:

1. Declare a new function for loading strings from resources, for ex.:

```
AnsiString LoadStr2(int Ident);
```

2. Add the body of the functions as follows:

```
extern PACKAGE AnsiString __fastcall LoadStr2(int Ident)
{
    TResStringRec ResRec;
    ResRec.module = (long *) &HInstance;
    ResRec.ident = Ident;
    return (LoadResString(&ResRec));
}
```

3. Replace LoadStr () calls to LoadStr2 () calls in all your units.
4. Compile your application.
5. Run TsiLang Resource Strings Wizard (available from TsiLang Expert's *Tools|Wizards* menu) and import all strings from your executable that need to be translated.
6. All these strings (selected in wizard) will be imported into TsiLang and will be available for translation.
7. Enter the translations for these strings and re-build the project.

### Why some of my string constants don't appear in found strings form when translating sources?

The following declaration of string constant:

```
const
    constname =
```

```
'stringvariable';
```

should be changed to:

```
const
  constname = 'stringvariable';
```

### How to translate TDBGrid columns?

It is preferable to make fields of your datasets persistent and translate their property “DisplayLabel” in design-time. Then TDBGrid column titles are updated automatically and don't depend on their order (when moved). Additionally, you can use **Collections** property of TsiLang to translate columns.

### How to translate InfoPower's DBGrid component?

For translating InfoPower's DBGrid component: use “**Multilines**” property of TsiLang and translate grid's “**Selected**” property. Note: Be careful with this property translation; be sure to keep the format and places of tabulation **TAB** character.

### How to translate arrays of strings

If you are using data structures like the below

```
const
  str1 = 'My String constant 1';
  str2 = 'My String constant 2';
  str3 = 'My String constant 3';
  StrArr: array[0..2] of string = (str1, str2, str3);
```

You need to replace the StrArr declaration with the following one:

```
StrArr: array[0..2] of PString = (@str1, @str2, @str3);
```

Also, all references in code on the array's elements like

```
...StrArr[Index]...
```

must be replaced with the:

```
...StrArr[Index]^...
```

### How to translate TActionMainMenuBar or TActionToolBar

Usually every TActionClientItem is linked to a corresponding TAction which provides string data for its clients. TsiLang components maintain Action's string properties such

as Caption or Hint, so all visual controls linked to the Action are updated when the active language is changed.

However, those TActionClientItems that not linked with any action, for example top-level items of TActionMainMenuBar, have no published string properties and cannot be handled directly. For such TActionClientItems we would recommend the following trick:

1. For every TActionClientItem without a TAction create a "fake" Action and link them. For example, if your TActionMainMenuBar has a top-level menu item '&File', create a new action FileFile1, set its Caption property to '&File', and link them.
2. Enter translation data for this Action in the Translation Editor.
3. If an action has no event handler it is permanently disabled, so set the OnUpdate event handler of all "fake" actions to a procedure like this:

```
...
procedure TForm1.FakeActionUpdate(Sender: TObject);
begin
    TAction(Sender).Enabled := True;
end;
...
```

### How to modify button widths in standard dialogs?

Unfortunately, there is no easy way to do this. But may be you can hack this by writing something like this in OnShow event of for example siFindDialog:

```
procedure TForm1.siFindDialog1Show(Sender: TObject);
var
    hnd: THandle;
    R: TRect;
begin
    if (siFindDialog1.siLang = nil) or
    (siFindDialog1.siLang.Language <> 'Dutch') then Exit;
    GetWindowRect(siFindDialog1.Handle, R);
    SetWindowPos(siFindDialog1.Handle, 0, 0, 0, R.Right - R.Left
+ 26, R.Bottom - R.Top, SWP_NOMOVE or SWP_NOZORDER);
    hnd := GetDlgItem(siFindDialog1.Handle, 1);
    SetWindowPos(hnd, 0, 0, 0, 100, 23, SWP_NOMOVE or
SWP_NOZORDER);
    hnd := GetDlgItem(siFindDialog1.Handle, IDCANCEL);
    SetWindowPos(hnd, 0, 0, 0, 100, 23, SWP_NOMOVE or
SWP_NOZORDER);
end;
```

### Are TsiLang components compatible with IntraWeb?

Yes, you can use TsiLang components in IntraWeb applications in the same way as in usual VCL applications.

#### Is it possible to translate menu shortcuts?

Yes, although it requires some tricks. Find the details at our forum at <http://www.sicomponents.com/forum/viewtopic.php?t=112>

#### Why TDBNavigator hints are not translated at start-up?

There is small bug (or as designed) in TDBNavigator component. To fix it you may use the following sample code:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    DBNavigator1.Hints.CommaText :=  
        siLang1.GetStringValue(@siLang1.MultiLines,  
            'DBNavigator1.Hints', siLang1.ActiveLanguage);  
end;
```

#### Why Developer Express components translations are displayed incorrect under XP Theme enabled?

When XP Theme enabled Developer Express components convert AnsiString to WideString using application default locale. You can fix this by changing application's thread locale on language changing event. For example:

```
procedure TForm1.siLang1ChangeLanguage(Sender: TObject);  
begin  
    if siLang1.ActiveLanguage = 1 then  
        SetThreadLocale(LANG_ENGLISH)  
    else  
        SetThreadLocale(LANG_JAPANESE);  
end;
```

#### How to detect OS default language and switch to it?

In later version of Delphi and C++Builder could be used SysLocale global variable:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    if SysLocale.PriLangID = LANG_ENGLISH then  
        siLangDispatcher1.ActiveLanguage := 1  
    else if SysLocale.PriLangID = LANG_GERMAN then  
        siLangDispatcher1.ActiveLanguage := 2  
    else if SysLocale.PriLangID = LANG_FRENCH then  
        siLangDispatcher1.ActiveLanguage := 3  
end;
```

```
    else
        siLangDispatcher1.ActiveLanguage := 1;
end;
```

Also you can use WinAPI to detect OS default language:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    LangID: DWORD;
begin
    LangID := GetUserDefaultLangID;
    case Byte(LangID and $03FF) of
        LANG_ENGLISH: siLangDispatcher1.ActiveLanguage := 1;
        LANG_GERMAN: siLangDispatcher1.ActiveLanguage := 2;
        LANG_FRENCH: siLangDispatcher1.ActiveLanguage := 3;
    else
        siLangDispatcher1.ActiveLanguage := 1;
    end;
end;
```

### How to properly load file at run-time?

You must just assign **FileName** property of the dispatcher component and call **LoadAllFromFile()** method:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    siLangDispatcher1.FileName := "YourSILorSIBFileName.SIL";
    siLangDispatcher1.LoadAllFromFile(siLangDispatcher1.FileName);
end;
```

### Main menu gets white background after language switching

Sometimes under Windows XP with themes enabled main menu background becomes white after changing languages. This is known XP bug and as workaround could be used the following:

1. Set **Images** property to **nil** before switching language
2. Switch active language
3. Restore the value of **Images** property

The following **Delphi** code demonstrates this:

```
MainMenu1.Images := nil;
siLangDispatcher.ActiveLanguage := (computed value);
MainMenu1.Images := ImageList1;
```



More FAQ entries could be found online at :  
<https://www.tsilang.com/tsilang-faq/>



## Version History

The history of new features added and improvements in the last versions of TsiLang Components Suite is listed below.

---

### **VERSION 7.9:**

---

#### **Core Components:**

- Embarcadero RAD Studio 11 Alexandria support!
- New component created: TsiInternetTranslator- translates any text using on-line services.
- TsiLang components can now use TsiInternetTranslator to translate terms on-fly.
- Improved work with SIL files.
- Some minor improvements.

#### **TsiLang Expert:**

- Improved stability.
- Fixed error on opening expert when no project is open in IDE and no Welcome page.
- Other improvements.

#### **Dictionary Manager:**

- Added DeepL on-line translation service support.
- Added feature to manage stored language associations for on-line translate services.
- Added Favorites feature.
- Other minor changes and improvements.

#### **SIL Editor:**

- Improved import and export functionality.
- Added feature to auto-translate the selected language for the whole file.
- Added Favorites feature.
- Fixed incorrect deleting of language in SIL file with more than 9 languages.
- Italian translation updated.
- Minor fixes.

---

### **VERSION 7.8.5:**

---

#### **Core Components:**

- Translations Editor improved:
-

- Added replace feature.
- Added Insert Row feature.
- Added Display leading and trailing spaces option.
- TsiLangCombo component improved:
  - Added ChangeLanguage property to automatically switch active language upon selecting it in the combo-box.
  - Other minor improvements.
- Some minor improvements.

**TsiLang Expert:**

- Improved stability for operations with large number of forms.
- Improved source-code scanning feature.
- Other improvements.

**Dictionary Manager:**

- Some UI fixes.
- Improved French language.
- Other minor changes and improvements.

**SIL Editor:**

- Improved import and export functionality.
- Added feature to remove multiple forms from SIB file.
- Minor fixes.

**Wizards:**

- Resource strings Wizard updated.
- INI file strings Wizard updated.

---

**VERSION 7.8.4:**

---

**Core Components:**

- TsiMemIniFile updated.
- Translations Editor improved.
- Some other improvements.

**TsiLang Expert:**

- Minor improvements.

**Dictionary Manager:**

---

- UI re-branding.
- Large toolbar images support.
- Microsoft Terminology Internet translate service support.
- Support for visual themes, including dark and light themes.
- Improved multiline editor.
- Improved Find dialog.
- Spanish UI language.
- Other minor changes and improvements.

**SIL Editor:**

- UI re-branding.
- Large toolbar images support.
- Improved multiline editor.
- Improved source file text editor.
- Improved Find dialog.
- Improved actions to interact with Dictionary Manager.
- Added option to share the last used file path with the IDE's TsiLang Expert.
- Support for visual themes, including dark and light themes.
- Spanish UI language.
- Minor fixes.

---

**VERSION 7.8.3:**

---

**Core Components:**

- Some improvements.

**TsiLang Expert:**

- Improved UI.
- Minor improvements.

**Dictionary Manager:**

- Fixed Clipboard error.
- Other minor changes and improvements.

**SIL Editor:**

- Fixed Clipboard error.

- Improved Find dialog.
- Fixed Pseudo-Translation and Statistics wizards.
- Improved Diff wizard.
- Minor fixes.

---

**VERSION 7.8.2:**

---

**Core components:**

- **Embarcadero RAD Studio 10.4 support!**

---

**VERSION 7.8.1:**

---

**Core components:**

- **Translation Memory** - new feature that allows run-time translating by using existing translations.
- Some improvements in code for FireMonkey support.

**Translations Editor:**

- Improved UI to simplify editing of **Extended Translations**.

**Dictionary Manager:**

- Improved CSV and HTML import\export.
- Added: remember association for Internet Translate language selection.
- Added: preview and life-time editor panel for the selected cell.
- Improved: Internet Translate services.
- Added: Yandex.Translator support for Internet Translate services.
- Improved: Find dialog, added option to find by selected language only.
- Added: Fixed language(s) feature- now you can set any language to be fixed in order to prevent occasional changes.
- Other minor changes and improvements.

**SIL Editor:**

- Support for Yandex.Translator.
- Minor fixes.

---

**VERSION 7.8:**

---

**Core components:**

- **Support for RAD Studio 10.3.3 Rio and Android 64bit target platform!**
- **Linux support! Now you can build multilanguage applications for Linux target platform the same way as for any other FMX target platform.**
- Option to use TranslationData as text in DFM.
- MessageDlgPosHelpTimeout, MessageDlgPosTimeout and MessageDlgTimeout methods.

**Translations Editor:**

- Re-designed UI for the editor.
- Multilanguage: German, Russian and Spanish languages added.

**TsiLang Expert:**

- Added **Configuration Wizard** that will help you to configure step-by-step TsiLang Expert's settings with detailed descriptions.
- Added new **TSI:IGNORE VALUE** tag to skip single string value while scanning.
- TsiLang Expert and all TsiLang editors are now multilingual. German, Russian and Spanish languages added.
- Improved support for national characters in string constants name.
- Improved integration to RAD Studio's 10.3.x Code Editor popup-menu.

**Dictionary Manager:**

- Minor fixes.

**SIL Editor:**

- Minor fixes.

---

**VERSION 7.7:**

---

**Core components:**

- **Support for RAD Studio 10.3.2 Rio and MAC OSX64bit!**

**Translations Editor:**

- Loading speed optimized. No more any delay on loading editor for forms with huge amount of components.
- Added option to increase editor's font size. You can set custom font size when working on High DPI monitors to improve visibility.
- Added large toolbar images for improved support High DPI monitors.

**TsiLang Expert:**

- Fixed possible problem when working with frames in FMX projects.
- Added TsiLang Expert menu items to code editor's popup-menu to perform operation on selected source code.
- Improved saving/loading project to/from external file.

**Dictionary Manager:**

- Added **Merge Wizard**. Now you can smartly merge two dictionaries.
- Added option to configure editor's font size. This will allow to use custom font size on High DPI monitors.
- Improved support for High DPI monitors.
- Some minor fixes.

**SIL Editor:**

- Some minor fixes and improvements.

---

**VERSION 7.6.0:**

---

**Core components:**

- **Support for Embarcadero RAD Studio 10.3 Rio.**
- 

---

**VERSION 7.5.9:**

---

**Core components:**

- Several improvements implemented

**Dictionary Manager:**

- Added support for BING Translator in Internet Automatic Translation Services.
- Improved Excel import/export functionality.
- Some minor fixes.

**SIL Editor:**

- Added feature to translate selected cell using Internet translation services through Dictionary Manager.
  - Improved all import/export functionality.
  - Some minor fixes and improvements.
- 

---

**VERSION 7.5.8:**

---

- **SI2DFM Wizard**– added feature to enforce update of DFM upon opening them in IDE (actual when updating the active language translations).

**TsiLang Expert:**

- Improved source scanning and small fixes.
- 

---

**VERSION 7.5.7:**

---

- **SI2DFM Wizard ANSI** – created ANSI version of SI2DFM tool to load SIL/SIB file to DFM without IDE for DFMs created with Delphi 2007 or earlier.

**Core components:**

- **OnExtendedChanging** event added to TsiLang to help proper resizing and reposition of controls under different DPIs.

**Dialog components:**

- Code updated to fix show hidden controls when application is running under VCL styles.
-

**TsiLang Expert:**

- Added option to configure Pascal style of comments.
- Added options for template file and folder for load and save operations from Project Manager popup-menu.
- Fixed incorrect character encoding while replacing sources through code editor popup-menu.
- Added Clear commands to clear project's translations in Project Manager popup-menu.

---

**VERSION 7.5.6:**

---

- **SI2DFM Wizard** - new tool to load SIL/SIB file to DFM without IDE.

**Core components:**

- Fixed incorrect behavior of ExtendedTranslations under mobile platforms (Android and iOS).

---

**VERSION 7.5.5:**

---

**Core components:**

- **Support for RAD Studio 10.2 Tokyo.**
- Internal fixes.

---

**VERSION 7.5.4:**

---

**Core components:**

- Small fix for FMX and Android.
- Implemented workaround for C++ and UniGUI.

---

**VERSION 7.5.3:**

---

**Core components:**

- Small fix for FMX and Android.
- Implemented workaround for C++ and UniGUI.

---

**VERSION 7.5.2:**

---

**Core components:**

- **Support for Embarcadero RAD Studio XE 10.1 Berlin.**

---

**VERSION 7.5.1:**

---

**Core components:**

- Added small fix for clipboard Unicode support in CreateMessageDlg method.

- 
- Improved translation of File Open\Save dialogs under non-English OS.

---

**VERSION 7.5:**

---

**Core components:**

- Support for Embarcadero RAD Studio XE 10 Seattle.

---

**VERSION 7.4:**

---

**Core components:**

- Support for Embarcadero RAD Studio XE 8.

---

**VERSION 7.3.3:**

---

**Core components:**

- Added "Force empties" option to "Add to Dictionary" dialog to allow adding items without translation to dictionary.
- Added Project Manager extension to Load\Save translations for all projects in project group.

**SIL Editor and Dictionary Manager:**

- Improved UTF-8 support in export/import operations.
- Support for "Force empties" option.

---

**VERSION 7.3:**

---

**Core components:**

- Support for Embarcadero RAD Studio XE 7.
- Fixed problem with UTF-8 encoding and Multilines property.

---

**VERSION 7.2:**

---

**Core components:**

- Support for Embarcadero RAD Studio XE 6.

**Dictionary Manager:**

- Fixed problem with Google Translate.

---

**VERSION 7.1.1**

---

**Core components:**

- Fixed problem with language changing on fly under Android.



## Support for Embarcadero RAD Studio XE 5 Update 2.

### **VERSION 7.1**

#### **Core components:**

**Support for Embarcadero RAD Studio XE 5.**

**Support for Android and iOS** target platforms.

Fixed problem with local characters corruption in units sources when translating CONST section.

Other minor fixes and improvements.

#### **SIL Editor:**

Minor fixes and improvements.

### **VERSION 7.0**

#### **Core components:**

**Support for Embarcadero RAD Studio XE 4.**

**FireMonkey all versions support.** Please add **FMX** global conditional define in your FireMonkey projects that use TsiLang. Please see FireMonkey Support topic for details.

Fixed problem with Unicode conversions under Windows Embedded.

Other minor fixes and improvements.

#### **SIL Editor:**

Fixed bugs on SIL <-> SIB conversions.

Fixed problem with Update Manager.

Other minor fixes and improvements.

#### **Dictionary Manager:**

Fixed several bugs.

Other minor improvements.

### **VERSION 6.5.5**

#### **Core components:**

Support for Embarcadero RAD Studio XE 3 (Win32 and Win64 applications).

Other minor fixes and improvements.

#### **SIL Editor:**

New UI design

#### **Dictionary Manager:**

New UI design

Support for TMX (Translation Memory Exchange) files.

### **VERSION 6.5.4**

**Core components:**

Support for Embarcadero RAD Studio XE 2 (Win32 and Win64 applications).  
Few minor fixes and improvements.

**VERSION 6.5.3****Core components:**

Fixed: bug with Unicode strings for Windows Dialogs controls under Delphi 2009+.  
Other minor fixes and improvements.

**Translation Editor:**

Added information about item length in chars to translation widths tool-tip window.

**TsiLang Expert**

Added an option to prevent insertion of comments with initial string when translating source code.  
Fixed: TsiLang Expert won't start when only a package project opened in IDE.

**SIL Editor:**

**New feature:** Highlight multi-line items with different number of lines in translations.  
Improved SIL file validation speed.  
Fixed: bug with navigation when using global search in SIB files.  
Improved sorting by String ID column.  
Improved Update Manager.  
Fixed: exporting to another format has used the incorrect header value.  
Fixed: Locales section not imported when importing from XML.

**Dictionary Manager:**

Improved Update Manager

**VERSION 6.5.2****Core components:**

Support for Embarcadero RAD Studio XE  
Fixed: bug with UTF-8 and dialog buttons captions.  
Fixed: char corruption issue under Chinese locales.  
Fixed: Collections translations lost if there are trailing spaces.  
Fixed: Help button click in MessageDlg() methods.  
Fixed: Saving and loading UTF8 SIL files and Delphi 2009+.  
Other minor fixes and improvements.

**SIL Editor:**

**New feature:** Diff wizard for comparing two SIL or SIB files.  
**New feature:** Short-Cuts Manager.  
Global Search is much faster now.  
Fixed: Error in export SIL to SIB when incorrect form name in a row.

Fixed: Incorrect SIL to SIB exporting for UTF-8 files.

Fixed: Chinese chars lost when converting to UTF-8 <-> ANSI under Chinese Default Locale.

**Dictionary Manager:**

"Display leading and trailing spaces" option

---

---

**VERSION 6.5**

---

---

**Core components:**

**Embarcadero RAD Studio 2010 support!**

Small fixes and improvements.

**TsiLang Expert:**

Fixed problem with Unicode preview in Skipped Strings window.

Other minor fixes and improvements.

**SIL Editor:**

Fixed several problems and few improvements made.

Updated German translation.

**Dictionary Manager:**

Fixed several problems and few improvements made.

Updated German translation.

---

---

**VERSION 6.4**

---

---

**Core components:**

**OnLanguageChanging event.** New event designed, which will be fired before language changing.

Feature: **exclude property of particular component class.** This will allow you to exclude the specified property for all components of the particular class.

Fixed bug with loading SIB and IsInheritedOwner=true.

Fixed: StoreAsUTF8 and Delphi 2009 conflicts.

**siLang\_Def\_UsedInCpp** global variable to provide ability to handle properly escape sequences under C++Builder projects for BDS2006 and later.

Several other improvements and some bug fixes applied.

**Translation Editor:**

Support for **Project Translation Settings.**

Close button on tool-bar changes to Close Saved when there were made any changes to translations.

Improved pasting of text range from clipboard.

Small fixes and tweaks.

**TsiLang Expert:**

**Project Translation Settings.** This will allow you to define Dictionary and translation related settings for your projects. TsiLang Expert and Translation Editor will use them when working with Dictionary and other wizards.

Found Strings improved: added Ignore button on tool-bar.

Improved speed and algorithm of source scanning.

Fixed bug: ID wasn't renamed in TsiLang after it was renamed in the grid of the found strings form.

Fixed Index Out of Bounds error, which appears occasionally for some units under Delphi(BCB) 6 and earlier.

Fixed AV errors upon exit from Delphi(BCB)6 and earlier.

Other minor fixes and improvements.

**SIL Editor:**

**Copy Language feature.** Now you can copy one language to another with couple of mouse clicks.

String ID column is selectable now, which allows to copy IDs as well.

Improved pasting of text range from clipboard.

Fonts combo-box allows to enter custom font name, like Ms Shell Dlg 2.

**Add language to Dictionary.** New feature allows you to add only the selected language to Dictionary.

Fixed: command-line merge of SIL files didn't add new language from merged file.

Fixed problem with entering Japanese characters.

Hungarian language for user interface.

Other small improvements and fixes.

**Dictionary Manager:**

**Multilanguage Support!** Dictionary Manager now multilingual as well as SIL Editor.

Fixed: Add All to Dictionary incorrectly places item.

<b>VERSION 6.3</b>
--------------------

**Core components:**

**CodeGear RAD Studio 2009 support!**

Single packages for Delphi and C++Builder!

**TaskMessageDlg** support in TsiLang message box methods.

Better support for actions linked to components.

SIL files load speed improved.

Titles of Print, Print Setup, Find, Replace and Color dialogs and Network button in Print Setup dialog now supported and translated by TsiLang.

UTF-8 support for internal translations storage.

CTRL+C support in message dialogs.

**Exclude component** item now available in design-time component's popup-menu.

A lot of improvements and some bug fixes applied.

**Translation Editor:**

**Translations Comments** support.

**Remove duplicates functionality.**

Smart auto-translate of multi-line text with dictionary.

Sorting by string ID improved.

**TsiLang Expert:**

Translation Wizard now able to configure additional properties.

User Interface for some dialogs improved to be nicely displayed under Windows VISTA.

Source scanning for hard-coded strings and strings in CONST section was dramatically improved and now it will handle most of the declarations used.

Found strings form improved in order to provide better preview of source code where string constant is used.

Several improvements implemented into other TsiLang Expert functions.

**SIL Editor:**

**Translations Comments.**

**Fully Unicode!****UTF-8 support for SIL and SIB files.****XML support.****Auto-initialize Fonts and Charsets for languages.****Command-line support for SIB files auto-translation.**

Display of leading and trailing spaces.

**Dictionary Manager:****DIX (XML) dictionaries!****Custom auto-translate services****VERSION 6.2**

- **C++Builder 2007 support.**
- Added global **siInterceptStringChange** procedure that will be called by TsiLang upon changing any UI element. May be used to perform custom modification to translations.
- Several improvements and bug fixes applied to core components.
- **Translation Editor:**
- **Multiform editing for run-time Translation Editor.**
- Few minor improvements and fixes applied.
- Several improvements implemented to TsiLang-TNT components.

**VERSION 6.1**

- **Delphi 2007 support.**
- **Vista Dialogs Support.** New component **TsiTaskDialog** created that provides multilanguage functionality to **TTaskDialog**. **TsiOpenDialog** and **TsiSaveDialog** components now handle translation for new Vista look as well. *Available only under Delphi 2007!*
- **Exclude by Type Name:** It is possible now to add the type name to the exclusion list and TsiLang will skip all components of such type.
- Improved **MergeFromFile** method: it will check the order of languages in merging SIL/SIB file and properly merge the translations.
- Several improvements and bug fixes applied to core components and dialogs.
- **Translation Editor:**
  - **Additional options** when adding to dictionary: **Case sensitive** and **Care of &** options added to **Add to Dictionary** dialog.
  - **Highlight Mismatched Multilines** option added that allows to highlight any entry in Multilines that has improper amount of items in translations.
  - Few minor improvements and fixes applied.
- **TsiLang Expert:**
  - **Regular Expressions** to configure string content that must be skipped. So from now on you can define the smarter rules for skipping strings in sources.

- **View Skipped Strings** feature added that allows to see the strings that **Expert** skipped while searching. This allows you to check if you didn't miss any important string for translation.
- **Const section** scanning bug fixed. This bug incorrectly replaced multi-line string constants in **CONST** section of unit.
- **SIL Editor:**
  - **Vista compatible.** SIL Editor updated with several new functionality and it is now Vista compatible.
  - **Update Manager** will allow to automatically check for updates and update application from our web site. This way you will be sure that your translators have the latest version to use.
  - **Global Search and Replace** functionality will allow you to find and (or) replace any text globally through the whole SIL (SIB) file with user-friendly interface.
  - **German language** added to available UI languages. Unfortunately, German translation is the not 100% complete. If you're native German speaker and wish to help us and other SIL Editor users you can translate German language in **SILEDITOR.SIB** file and send it to us. We will add it and all German speaking users will be able to use it.
  - A lot of minor improvements introduced.
- **Dictionary Manager:**
  - **Vista compatible.** Dictionary Manager updated it is now Vista compatible.
  - Fixed bug with **Add to Dictionary** functionality.
  - Improved **Spell Checking Dialog**.
- **Resource Strings Wizard** improved to handle command-line interface and also create pure console version of such wizard. Now you can use it in automatic build and scripting tools.
- **INI File Import Wizard.** We've created new wizard that allows you to translate strings stored in INI files. This wizard will import strings from INI file into TsiLang and provide you with code templates to translate them easily.

<b>VERSION 6.0.3</b>
----------------------

- **New property AutoSkipEmpties:** allows to automatically skip properties that have no values for translation at all. This will reduce the size of the translations and resource used.
- **Preview** window caption in TsiOpen[Save]PictureDialog translated.
- **Run-time and design-time packages:** we've divided TsiLang package to run-time and design-time only. This will allow you to be able to use run-time packages functionality with TsiLang as well.
- **Translation Editor:**

- **Auto-translate selected language** feature added that allows to automatically translate the selected language only. This is useful when you want to translate only particular language and leave other untouched.
- Fixed bug with incorrect behavior of context popup-menu.
- **TsiLang Expert:**
- Added new menu items to source code Editor's popup-menu. These items include:
  - **Scan selected source-** allows to scan selected source code and extract all found strings.
  - **Mark to skip-** allows to mark selected block of code to be ignored by TsiLang Expert while scanning source.
  - **View translation-** allows to jump from source code to the selected string ID in TsiLang.
- Improved and fixed bug in source code scanning.
- **SIL Editor:**
- **Auto-translate selected language-** allows to automatically translate the selected language only. This is useful when you want to translate only particular language and leave other untouched.
- Fixed problem with SIB files containing TNT and EIPack TsiLang components.
- **Command-line merging for files extended-** additional command-line switch added:

**-mlang [all]** - specifies to merge all languages into original file

- **Dictionary Manager** improved.

## VERSION 6.0.2

- **TNT Controls** and **LMD EIPack** support components- created new components (analogues for all existing TsiLang components) which add support for Unicode TWideStrings and other specific properties of TNT Controls and LMD EIPack controls.
- Improved and fixed bug with file loading. SIB loading is even faster now.
- Improved speed of reading translations for inherited forms.
- Improved source parsing in TsiLang Expert.
- **SIL Editor:**
- **Pseudo Translation Wizard-** SIL Editor now provides you the wizard that will allow to generate pseudo translations for your items in order to be able to test the appearance of you applications for different languages with non-English letters and umlauts, like German, French, East Europe and others.
- **Delete form from SIB file-** it is possible to delete forms directly from SIB file using SIL Editor .
- **Command-line merging for files-** it is possible to use command-line switches to perform merging of different files. To merge files using command-line please use the following switches:

**-morig [file\_name]** - specifies the original file for merging

- mnew [file\_name] - specifies the file that must be merged into original file
- mlang [language\_name] - specifies the name of language that must be merged into original file
- msuperfl - if presents forces to merge superfluous entries into original file

- **Dictionary Manager** improved and fixed small bugs.

<b>VERSION 6.0.1</b>
----------------------

- New methods **GetTextW()** and **GetTextOrDefaultW()** that return **WideString** as result. Useful when building **Unicode** applications.
- New methods **GetTextOrDefined()** and **GetTextOrDefinedW()** that return translation of user defined strings for specified language if no translations for active language available.
- New properties **UseDefaultLanguage** and **DefaultLanguage** allow to use specific language as default while switching languages and no translation available for active language.
- Improved support for inheritance, thanks to Andreas Brodbeck from Mindclue GmbH for his code and help.
- It is possible now to save **ExtendedTranslations** property in Save Properties dialog from TsiLang Expert
- **Extended message dialog functions:**
  - Default button receives input focus if you specify the default button in MessageDlg() functions.
  - Global classes for dialog controls- there is defined global classes to use for dialog controls and initialized with default values like:

```
MsgDlgFormClass: TFormClass = TForm;
MsgDlgLabelClass: TControlClass = TLabel;
MsgDlgEditClass: TControlClass = TEdit;
MsgDlgButtonClass: TControlClass = TButton;
```

This allows you to use additional classes in dialog functions instead of Delphi's standard classes. This functionality is useful also when building fully Unicode application, you can replace standard classes with Unicode and even your message boxes will be fully Unicode.

- **SIL Editor:**
  - Statistics Wizard- SIL Editor now provides you the statistics so you can count how many words and items to translate and other information.
  - Encryption and Decryption- SIL Editor provides ability to encrypt string IDs, in order to "hide" application structure. Also added option to hide first column.
  - Some internal improvements.
- Some minor bugs and improvements.



**VERSION 6.0**

- New user interface for editors and tools.
- **Exclude from Translations Editor**- new editor that handles and helps to manage all exclusions from translations at one place and with very convenient interface.
- Automatic addition of most often used property names for exclusion, like **TableName**, **DatabaseName**, **Category** and others.
- Better translation of **Unicode** components and properties.
- Handling of **LoadStr()** and **FmtLoadStr()** functions when translating resource strings.
- Handling resource strings by identifier instead of value. This would be very useful while translating own resource strings stored and linked as RC file(s) to your applications.
- **ExtendedTranslations** property updates values from components when updating translations. This is very useful for visually designing different layouts for different languages.
- **DefaultBtn** and **CancelBtn** parameters for all **MessageDlg()** methods to provide ability to specify which button to use as default and cancel button.
- **Translation Editor:**
  - Translation Editor now handles **ExtendedTranslations** property as well. This provides full functionality of Translation Editor available while editing **Extended Translations**.
  - Tool tip for width and height of translations while editing content.
  - Navigation directly to first occurrence in source code of string while editing **Strings** property.
  - Improved sorting when sorting by ID under **Strings** section.
  - When translating multi-line contents using **Dictionary Translation Editor** will try to translate line-by-line if no translation for all lines at once available. This is useful when translating combo-box, radio-group and similar items.
  - Translation Editor will try to find a similar translation when translating phrases with *special* symbols at the end, like: **":"**, **"..."**, **" "** and so on.
- **TsiLang Expert:**
  - Improved source scanning.
  - When checking for *Bad String IDs* it is possible to delete strings directly from TsiLang.
  - **Translation Wizard** allows translating selected form(s) with step-by-step detailed instructions and help as well as configures already translated forms.
  - Some internal improvements.
- **Dictionary Manager:**
  - Improved XML import Wizard
  - Added ability to delete multiple selected rows at once.
  - Added ability to delete any language (previously only the last one could be deleted).
  - Fixed some bugs and implemented other minor improvements.
- **SIL Editor:**

- **Multilanguage interface**- SIL Editor now supports multilanguage interface and you can easily translate it into your own language.
- Some internal improvements.

<b>VERSION 5.3.2</b>
----------------------

- TsiLang Expert Improvements:
  - New source scanning tags introduced `{TSI:IGNORE NEXT}` and `{TSI:TRANSLATE NEXT}`. To ignore/translate just next line.
  - Improved source scanning.
  - Fixed some bugs.
- Translations Editor:
  - New filter to show incomplete translations only.
  - Sorting by property name.
  - Automatically handle colon ":" character when translating with Dictionary Manager.
  - Automatically split multiline items into single terms when translating with Dictionary Manager.
- Dictionary Manager:
  - Improved import from Excel
  - Fixed bug whit "Invalid character..." error when using BabelFish feature.
- SIL Editor:
  - Improved error dialog in order to display more user friendly messages.
  - Added support for SIB in ExportTo method of SIL Editor COM server.
- Resource Strings Wizard: added feature to skip checking for existing resource strings IDs.

<b>VERSION 5.3.1</b>
----------------------

- TsiLang Expert Improvements:
  - Ability to mark particular hard-coded strings as untranslatable so it does not show up again in the next source scan.
  - Ability to exclude strings containing specified sub-strings.
  - Ability to skip source lines containing specified words.
  - Ability to skip strings that includes only special characters.
  - Ability to leave existing string constants values when translating CONST section.
- Hijri2Gregorian and Gregorian2Hijri Routines to convert Arabic dates to Gregorian and vice versa.
- New property TestModeInfo for dispatcher: New property allows to use extended functionality when generating translations in test mode. TestMode and TestPercentage properties are deprecated now!
- New feature Statistics: TsiLang Translations Editor now allows to see the detailed translations statistics.

- SIL Editor and Dictionary Manager Improved
- Other Improvements: There were made some minor improvements and bugs fixing.

**VERSION 5.3.0**

- New IDEs support: TsiLang Components Suite includes packages for Delphi 8 (VCL Edition). No any additional changes are required to port your existing projects into Delphi 8.
- Improved Extended Translations support: The Extended Translation property editor is fully redesigned and improved. New functions Save, Load, and Find significantly simplify using extended translations with TsiLang components.
- Property Editors Improvements: All property editors are updated and improved. Exclude Properties from Translation property editor now allows to save and load data to/from external text files.
- New component: TsiLangCombo is a new auxiliary component, which allows displaying all supported languages with specific graphic and font settings for each language.
- SIL Editor Improvements: New command line switch -E added to export file using command line.
- Unicode Clipboard Support: Now Translation Editor allows to copy and paste unicode strings to/from the clipboard.
- Resource Strings Wizard improved: There were made a lot of improvements in wizard so you will be able now to easily skip existing strings from importing, delete obsolete strings which shall no be longer translated and many others.
- Other Improvements:
  - In the Translation Editor a switch added in menu for disabling duplicate IDs highlighting.
  - Added property editor for Language property of TsiLang and TsiLangDispatcher to use combo box with language names listed.
  - Added ability to translate unit's source directly from TsiLang component design-time menu.
  - Locales will be stored only when ChangeLocales=True.
  - Added flag to change language for extended before others (TranslateExtendedFirst property). For fixing Delphi's bug with RightToLeft <-> LeftToRight and menus.
  - New property TestPercentage allows to set length for preview testing.
  - Test mode - new property for TsiLangDispatcher to test translations.
  - Fixed problem with translating QuickReport Font and Charset.
  - Load/Save From/To Stream methods for TsiLangXX components.
  - Improved Found Strings form in TsiLang Expert.
  - SIL Editor and Dictionary Manager improved.

**VERSION 5.2.4**

- Automatic translation using BabelFish web services in the Dictionary Manager is implemented.
- Various improvements and optimizations are implemented in the VCL, SIL File Editor, and Dictionary Manager.
- A few minor bugs are fixed.

**VERSION 5.2.3**

- New command "Check Format Strings" is added to the Translation Editor. This function helps you to verify that all format strings are coincided for all translations. For example, if a format string in the base language looks like "%s - report from %s" then in other languages its translation also should contain two tags "%s", otherwise the "Format" function raises an exception. To launch this command select the "Strings" category and click the menu "Tools | Check Format strings".
- TsiLang Files Editor: exporting .sil-files to .sib format and vice versa.
  - Some minor bugs are fixed.

**VERSION 5.2.2**

- Dictionary Manager is Unicode: The Dictionary Manager is re-designed for Unicode support. Now under Windows NT/2k/XP you can simultaneously edit translations in any language.
- Translation Editor gets Translation: The Translation Editor might be translated itself if you deploy it along with your application. All you need is to enter translation data into a .SIL file (the template file 'RT.SIL' is provided), and before opening Translation Editor in run-time assign appropriate values to the global variables "RT\_SILFile" and "RT\_ActiveLanguage".
- Minor Improvements: A few small bugs are fixed.

**VERSION 5.2**

- New IDEs support: TsiLang Components Suite includes packages for Delphi 7 (both VCL and CLX Editions) as well for Kylix 3 (both for Delphi C++ versions). No any additional changes are required to port your existing projects into above IDEs.
- Translation data streaming: The dfm-streaming method is changed for "translation-aware" properties of TsiLang components. Now you can edit translations in TsiLang components without changing the default locale on your system.
- Other Improvements:
  - Number of optimizations is made to make the library smaller and faster.

- User interface of the Translation Editor is changed so that design-time property editors, run-time Translation Editor and SIL Editor have consistent feel and look.
- Multiple cells selection in the Translation Editor is implemented (for Copy/Cut/Paste/Delete treatment).
- Ignore/Check Removals command is added for TsiLang component editor. This is useful when you move controls on your form via Cut->Paste in design-time. It is good idea before cutting a control to set "Ignore removals" flag for the TsiLang component, so that any references to this control would not lost in the TsiLang properties.
- The new method "ClearTranslations" is introduced for TsiLang components and corresponding command is added to the TsiLang Expert. This removes all translation data from the component.

<b>VERSION 5.1</b>
--------------------

- C++Builder™ 6 support. TsiLang Components Suite successfully compiled under C++Builder™ 6.
- VCL and CLX Editions: For the best support of CLX technology TsiLang Components Suite is divided into two editions: VCL and CLX. The latter one allows you to create true CLX Multilanguage applications under Windows with Delphi™ 6 and C++Builder™ 6 as well as under Linux with Kylix™ 1 and 2.
- Extended Translations: New property "ExtendedTranslations" is introduced for all TsiCustomLang's descendants. Using this property you can internationalize not only strings but any properties of components that can affect an application's appearance after language switching; for example, a label's width and height now may have different values for different languages.
- Binary Storage: Besides the traditional technology for storing external translations in .SIL files the new one based on binary .SIB files has brought into operation. This technology is specifically designed for using with TsiLangRT component and allows loading and saving translations at run-time a few times faster than from/to .SIL files.
- Resource Strings Handling: With this sophisticated improvement a TsiLang component can handle all resource strings of your application even if they are hidden in some .dcl or .obj and you do not have the source. All you need is to set the property HandleResourceStrings to *True*.
- Other Improvements:
  - Checking for duplicate ID when adding new ID in Strings Editor.
  - Renaming ID in sources when changing it in Strings Editor.
  - Possibility to exclude blank values in Translation Editor.

- Suggest translation feature.
- Default language names.
- TsiLang Files Editor (SIL Editor): Support for binary .SIB files; Redesigned user's interface; Suggest translation feature; Improved export/import;
  - Dictionary Manager: Spell checking; Improved export/import;

<b>VERSION 5.0</b>
--------------------

- Kylix™ 2 support! TsiLang Components Suite successfully compiled under Kylix™ 2.
- Improvement: components with ParentFont property set to True are not longer listed in the Fonts and Charsets properties.
- Improvement: it is now possible to exclude particular properties or components directly from Translation Editor.
- Improvement: it is now possible to filter untranslated/partially translated properties from Translation Editor.
- Improvement: When Dictionary Manager is activated from either Translation Editor or SIL Editor, entering an empty cell in Editor will automatically look up and suggest a translation for the given language.
- TsiLang Expert Improvement: it is now possible to check bad string identifiers used in units without forms.
- TsiLang Expert Improvement: it is now possible to configure the minimal length of string in source to be handled by Source Translation feature.
- TsiLang Expert Improvement: TsiLang Expert will now suggest smarter string identifiers under Source Translation feature.
- SIL Editor Improvement: Translation Info tips - when entering a cell an info tip displaying values from all the languages pops up. This is very useful when there is not enough room to display all columns.
- SIL Editor Improvement: Improved HTML import/export.
- SIL Editor Improvement: Columns can now be sorted both ascending and descending.
- Dictionary Manager Improvement: when using Auto-translation feature, strings with accelerators are handled more correctly.
- Dictionary Manager Improvement: Auto-translation feature now uses a base language selected by user.



## List of Tables

Table 1 Dictionary Manager Automation Server .....	49
Table 2 Methods of SIL Editor Automation Server .....	52
Table 3 TsiLang translation-aware properties .....	57
Table 4 TsiLang behavior-aware properties .....	58
Table 5 Other properties of TsiLang .....	60
Table 6 TsiLang events .....	68
Table 7 TsiLangLinked specific properties .....	68
Table 8 TsiLangRT specific properties .....	69
Table 9 TsiLangRTSE specific properties .....	70
Table 10 TsiLangTLV specific properties .....	71
Table 11 TsiLangDispatcher properties .....	72
Table 12 TTestModeInfo fields .....	73
Table 13 TsiLangDispatcher events .....	74
Table 14 TsiInternetTranslator properties .....	76
Table 15 TsiLangCombo Properties .....	79
Table 16 siDialogs specific properties .....	80



## List of Figures

Figure 1 Core components hierarchy .....	12
Figure 2 Multiple TsiLang components are linked to a single TsiLangDispatcher..	14
Figure 3 Forms with a TsiLangDispatcher and TsiLang components .....	15
Figure 4 Strings constants found by the Expert .....	17
Figure 5 Menu items for language switching .....	18
Figure 6 Translation Editor displays “translation-aware” properties .....	19
Figure 7 TsiLang Expert .....	20
Figure 8 Translation Wizard (First step) .....	21
Figure 9 Translation Wizard (Second Step).....	22
Figure 10 Hard-coded strings found by the Expert.....	23
Figure 11 Bad string identifiers found by the Expert .....	26
Figure 12 TsiLang Expert options dialog .....	27
Figure 13 Translations Editor.....	30
Figure 14 Extended Translations property editor .....	34
Figure 15 Exclude components from translations .....	41
Figure 16 Exclude properties from translations .....	42
Figure 17 Exclude components’ properties from translations .....	43
Figure 18 Add FMX define when building Firemonkey projects.....	44
Figure 19 Translated application with TsiLang under Linux.....	46
Figure 20 Dictionary Manager.....	48
Figure 21 SIL Editor.....	51
Figure 22 Resource Strings Wizard - Step 1 .....	54
Figure 23 Resource Strings Wizard - Step 2.....	55
Figure 24 Resource Strings Wizard - Step 3.....	55
Figure 25 Configuring Default Fonts.....	84

# Index

## C

Component's Properties to Exclude .....	42
Components to Exclude .....	40
Contacting Information .....	9

## D

Dialogs .....	80
Dictionary Manager .....	48
Dictionary Manager Automation Server .....	49

## E

Events	
TsiLangDispatcher .....	74
Expert Options .....	26
ExtendedTranslations .....	58, 65
Using .....	34, 38

## F

FireMonkey Support .....	44
Frequently Asked Questions .....	89

## G

GetText .....	28, 63
GetText C .....	63
GetTextByInt .....	63
GetTextOrDefault .....	28, 63
GetTextOrDefaultC .....	63
GetTextOrDefaultW .....	63
GetTextOrDefined .....	63
GetTextOrDefinedW .....	63
GetTextW .....	63

## H

Hard-coded strings	
<b>Prefix</b> .....	28
Search .....	22

## I

Installation .....	10
Registered Version .....	11
Trial Version .....	10

## L

Language switching .....	17
Linux64 .....	45
LoadAllFromFile	
TsiLang .....	64
LoadFromFile	

---

TsiLang .....	64
<b>M</b>	
Methods	
TsiLang .....	62
TsiLangDispatcher .....	72
TsiLangRT .....	69
<b>P</b>	
Properties	
TsiLang .....	60
TsiLangDispatcher .....	72
TsiLangLinked .....	68
TsiLangRT .....	69
Properties to Exclude .....	41
<b>R</b>	
Resource strings	
Importing .....	54
<b>S</b>	
SaveAllToFile	
TsiLang .....	64
SaveToFile	
TsiLang .....	64
SIL Editor .....	51
SIL Editor Automation Server .....	52
<b>T</b>	
Tips and Tricks .....	81
Translate resource strings by ID .....	56
TranslationData as Text in DFM .....	88
TsiCustomLang .....	12, 68
TsiLang.....	10, 11, 16, 18, 28, 57, 72
Properties.....	57
Review.....	12
TsiLang events .....	68
TsiLang Expert	
Using .....	20
TsiLangDispatcher .....	71
Review.....	13
TsiLangLinked .....	68
Properties.....	68
TsiLangRT.....	68
Methods.....	69
Properties.....	69
TsiLangRTSE.....	70
Methods.....	70
Properties.....	70
TsiLangTLV.....	70
Tutorial .....	15
<b>U</b>	
Using Exclude from Translations Editor .....	40
Using ExtendedTranslations under different DPIs .....	36

---

---

Using Translation Editor .....	30
Using Translations Stored in External Files .....	38
Using TSI:IGNORE tags.....	24
Using TsiLang Expert .....	20
<b>V</b>	
Version History .....	95
<b>W</b>	
Welcome .....	6