



## TMultLang Component

[See also](#)

[Properties](#)

[Methods](#)

[Events](#)

[Task](#)

[Tips](#)

[Support](#)

### Unit

MultLang

### Description

The TMultLang component will add support for more languages on the form you are designing. Please read the [History](#) and [Structure](#) for more information. Please advice the [Step-By-Step](#) introduction if you are not familiar with this component. See the [Planning](#) before starting and for suggestions and directions of using this component.

This component is inherited from the [TMComboBox](#) component.

By dropping this component on the form you can choose which other properties on the form you want to translate in the [Languages Property Editor](#).

The configured languages are shown in the dropdown box at runtime but can also be adopted on a TMainMenu, TPopupMenu, TMenuItem, TListBox, TComboBox and a TRadioGroup by setting the [AdoptOn](#) property to that control.

With each language you can assign an Icon to be displayed according to [AlignIcon](#) and [ShowIcons](#) properties in the [Language Property Editor](#). The icons can only be displayed in the combo box and will not have any effect if the [AdoptOn](#) property is assigned.

The text will be aligned according to [AlignText](#) property and if there are icons.

The [DefaultLanguage](#) property holds the string of the startup language when the form is first shown. This can be useful when the [Visible](#) property is false, you will then have to call the [Translate](#) method to programatically switch language on the fly.

Date, Time, Currency and Number translations can be done by setting the appropriate property string of the component it self. For ex. the ShortDateFormat property sets the date format for your application so it can be used with the FormatDateTime('dddd', now) function to retrieve the right format depending on the selected language. By including the UpdateWindowsINI property the translations will effect all applications in Windows.

When filtering out languages not to be available in the dop down list set the [LanguageFilter](#) property.

User defined string accessing can be used with [GetString](#) and [SetString](#) methods. This is a useful string holder instead of using TLabels or TMemos. The strings are accessed through your identifier and will automatically be translated when the language changes on the fly. Both user defined strings and other strings can include substrings by using two ampersands characters between the identifier, e.d. "I like my &ComputerName&".

To save the language resources in an external file or to share the languages between forms in your application set the [LanguageFile](#) property. Use your own string editing tool by using the [Export](#) and [Import](#) methods which uses a Windows standard INI text format.

Get the translated term of a special language with the [GetTranslation](#) method.

To translate Windows common dialogs as Open, Save, Find, Print etc. translate the appropriate dialog component from the [MDialogs](#) unit. MDialogs have support for on the fly translations with DBC languages by changing fonts and strings of the dialogs.

To translate Delphi standard message buttons and titles in ShowMessage, MessageDlg, InputQuery etc. drop the [MButtons](#) component on affected forms and translate it with the TMultiLang component.

For more information about advanced features please see the [Advanced](#) topic.


When distributing files maintained by this component see the [Distribution](#).

Author:

Patrik Wang  
PWang@MSN.COM

TMultiLang © 1995 Patrik Wang. All text, information and instructions contained in this helpfile is provided as is, the author disclaims all warranties expressed or implied, including, without limitation. DISCLAIMER AGREEMENT see the About dialog for the TMultiLang package. By accessing, reading or showing this helpfile you agree to all terms stated here, in the About dialog and in the README.TXT file.

## Methods

 key methods

-  AddIcon
-  FindLanguage
-  RemoveIcon
-  AddLanguage
-  GetExternalStrings
-  SaveToFile
-  ClearLanguages
-  GetString
-  SetString
-  DeleteLanguage
-  GetTranslation
-  ShowAbout
-  Edit
-  Import
-  Translate
-  EmptyLanguage
-  IsLanguageFile
-  UpdateLanguages
-  Export
-  LoadFromFile

## **TMultLang Structure**

See also

[Description in detail](#)

Before reading this text please see the [History](#).

This component is a tool for maintaining multi languages in Delphi, primary for developers.

The component operates on an existing application by setting various user defined object properties of controls on the window, for example: the Caption property of Buttons or the Caption property of Labels. Those properties are saved inside the component or in an external file. Translations can be Imported and Exported in text format.

The component can be invisible and translate your forms before they are shown, this way you can select the default language to be generated at design-time before compiling the application.

It can also be useful for many other tasks that manipulates properties on forms. You can have different configurations of layout (colors, fonts etc) of the form.

## TMultLang Tips - Changing values

### Changing values

You may find it inconvenient changing many values by clicking on those every time in the languages property editor. Instead, you can use PgDn and PgUp keys to move between editable properties. When accidentally changing a value it can be restored by pressing ESC. But whenever you press Enter or exiting the editing field or by jumping to the next property, the value will be saved.

## AlignIcon Property

See also

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property AlignIcon:[TOwnAlign](#)

### Description

If [ShowIcons](#) property is true, the icon will be aligned according to this property. If the language does not have an icon associated in the [Languages Property Editor](#), this property do not have any effect. However, you cannot change the horizontal alignment, which is always centred. You can have the icon drawn in the height size of the box by setting property [Resizelcon](#) to true.

## AlignText Property

See also

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property AlignText:[TOwnAlign](#)

### Description

The language text will be aligned according to this property. However you cannot change the horizontal alignment, which is always centred. The language text will also be aligned according to [TextMargin](#). If an icon is shown the text will be aligned beside the icon, if it is not centred.

## DefaultLanguage Property

[See also](#)

### Applies to

[TMultLang](#)

### Declaration

Property DefaultLanguage: String

### Description

The DefaultLanguage property contains the exact name of the language that you want to be default at startup. Since this property is only valid for the first created TMultLang component at design-time, it does not have any effect on the rest of the TMultLang components. However if some of the other TMultLang components does not have the choosen language name of the first created TMultLang component, it will use this property instead.

By setting this property at design time and by having the Visible property to false, this can be useful if you want to distribute one language version of your application. Just change this property for each language compilation.



## IconMargin Property

[See also](#)

### Applies to

[TMultLang](#)

### Declaration

Property IconMargin: Integer

### Description

The IconMargin property shows the distance in pixels from the border according to [AlignIcon](#) property. This property does not have any effect if the icon is aligned in the centre.

This property can be useful in the [OnDrawingItem](#) event.

## LanguageFile Property

See also

### Applies to

TMultLang

### Declaration

Property LanguageFile: String

### Description

The LanguageFile property contains the name of an external data file for all languages and its properties in the form. By leaving this property empty, the component will store its data in the form file instead (\*.DFM).

By pointing this property to a file same as all other TMultLang components in your application, you will share the languages between them. Changing, adding or removing languages will then reflect all forms. This file does not have any special limit and will be a better choice if you have lot of properties and icons that may be too much for the form file to maintain. By having an external file you will also be able to upgrade your languages in the runtime application without recompiling and redistribute the executables.

The external file will be accessed in Read/Share mode. You should make this file Read Only on the shared disk. However if you want your runtime application to be able to modify languages, you must have Read, Write, Delete and Rename access rights for those who should be able to modify the languages. The pathname can be any valid pathname compatible with Windows standard, you could for example have a valid redirected name like '\\FS01\\SYSTEM:APPS\\SHIPMENT\\LANGUAGE.LAN'.

You should name the file with an 'LAN' extension.

If you dont specify any path before the file name it will search for a matching file in the following directories (in this order):

- 1 The current directory.
- 2 The Windows directory (the directory containing WIN.COM), whose path the GetWindowsDirectory function retrieves.
- 3 The Windows system directory (the directory containing such system files as GDI.EXE), whose path the GetSystemDirectory function retrieves.
- 4 The directory containing the executable file for the current task; the GetModuleFileName function obtains the path of this directory.
- 5 The directories listed in the PATH environment variable.
- 6 The list of directories mapped in a network.

It is important to know that if you change this property to an existing file, it will retrieve all properties for the same form name as the one you are editing from the file. Languages will be replaced.

\* NOTE Changes made in the 1.12 version

The property can also hold initialization files if the extension is something else than LAN. The initialization files is compatible with the Import and Export methods. However, by using initialization files you cannot share languages between forms and this property should not be used by more than one form at a time.

## LanguageFilter Property

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property LanguageFilter: TStrings

### Description

To exclude some languages in the dropdown list at runtime, you can include these language names in this string list property. The names must be exact. This property does not prevent the method [Translate](#) to translate a language which is included in this string list.

This property can be useful in runtime when you dont want some users to list all available languages. You can for instance fill this stringlist with language names after the user have logged in, they wont be able to see those languages later.

You have to change this property on each TMultLang component, because it is not saved in the [LanguageFile](#) file, nor will it reflect other loaded forms at runtime.

## Languages Property

**Applies to**  
TMultLang

**Declaration**  
Property Languages: TList

**Description**  
This property should only be edited via the language property editor or the Edit method. It contains all languages, icons and properties.

You will never need to access this property.

## Resizelcon Property

[See also](#)

### **Applies to**

[TMultLang](#)

### **Declaration**

Property Resizelcon: Boolean

### **Description**

If an icon is attached to the language in the [language property editor](#) and the [ShowIcons](#) property is true, this property tells if the icon should be stretched from its original size to the size of an item in the dropdown list.

You will probably only need to set this property at design time, but it can be useful in the [OnDrawingItem event](#).

## ShowIcons Property

See also

### Applies to

TMultLang

### Declaration

Property ShowIcons: Boolean

### Description

If there are icons attached in the language property editor and this property is true, the icons will be shown according to AlignIcon, IconMargin and ResizeIcon properties in the dropdown list. However if the ResizeIcon is false it is centred horizontally.

You will probably only need to set this property at design time, but it can be useful in the OnDrawingItem event.

## TextMargin Property

[See also](#)

### Applies to

[TMultLang](#)

### Declaration

Property TextMargin: Integer

### Description

The TextMargin property shows the distance in pixels from the border according to [AlignText](#) property. This property does not have any effect if the text is aligned in the centre.

You will probably only need to set this property at design time, but it can be useful in the [OnDrawingItem](#) event.

## TMultLang Language Property Editor

### Description

This property editor is the heart of the TMultLang component. It maintains all languages, icons and properties in your forms and also make it possible to reuse string resources.

For a description of the TMultLang component see the [structure](#).

The first time you enter this dialog you will only see the default language ('English'), this language is the same as the form you are designing. this language can only be edited in the Object Inspector. This design language is always there, you cannot delete it but you can rename and attach an icon for it. To change properties values of the default language use the Object Inspector. When adding languages you can choose which properties you want to change for each language. Properties as well as property-values can be copied between languages and other forms.

At design time you can display this editor from the Object Inspector. At run time you will have to call the [Edit method](#) which does exactly the same thing, displays this editor. By clicking on the Help button, this help page is displayed. That means if you want your run time application to modify languages, you will also need to distribute your own help file (MULTLANG.HLP). See [distribution files and rights](#).

### Buttons

In the language section:

- 'Add' Will add a new language with no icon attached. (see [Adding languages](#))
- 'Delete' Will permanently remove the chosen language, icon and properties.
- 'Rename' Change the name of the language.
- 'Load' Attach an icon for the chosen language.
- 'Clear' Removes an icon for the chosen language.

In the properties section:

- 'Browse' Invokes the [browse window](#) for choosing properties.
- 'Copy from' Will copy chosen properties from another language with the [copy from window](#).
- 'Expand All' Expands all objects in the properties outline list.

Other:

- 'OK' Saves changes in the file or form exits.
- 'Cancel' Exits without saving changes to the file or form.
- 'Help' Shows this help window.
- 'Load from' Will load languages, icons and properties from an external file.
- 'Save to' Saves languages, icons and properties to an external file.
- 'Lookup' Does a soundex search for similar words on other forms if and external file is attached.

### Languages

By choosing a language in the drop down list in the language section, that is not the design language, you will be able to alter properties in the property list for that language.

### Property list

The property list is just an outline component which displays all chosen components for the selected language. First you have to choose which component and properties that should be changed for the selected language in the [Browse window](#) by clicking in the 'Browse' button. By double click on those components and properties you can alter the values below. You can see the default value over the editing area. If there are known sets of values they will be displayed in the drop down list of the editing area. It is similar as the Object Inspector.

### Grid Values

Values of the currently selected property of other languages will be displayed in the grid below the property outline list. By clicking on a language it will then display that with bold face, which is good when



there are many languages. By double click on a language you will transfer the value of that language to the value field of the current editing language.

**New Value field**

Contains the value of the currently selected property. This value can be changed according to the type. For editing instructions see [Changing values](#). Also see [Get Values](#).

## **TMultLang TOwnAlign Type**

### **Declaration**

TOwnAlign = (alLeft, alRight, alCenter)

### **Description**

TOwnAlign defines the possible values of the AlignText and AlignIcon property.

## Translate Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Function Translate(Language:String; TranslateApp:Boolean):Boolean
```

### Description

By calling this method with the exact name of the language in the 'Language' string, the component will try to translate all properties that is configured for this language from the [languages property editor](#). It will also try to translate other forms within the same application which have a TMultLang component. The translation may not be successful when objects or properties have dynamically been removed before the translation. However it will try to translate all other selected properties even though one of them may fail. Before translating, the [OnTranslate](#) event occurs. After the translation, the OnTranslated event occurs. After translating all properties on the current form without any error it will return true, otherwise it returns false. The return value does not tell how other forms were translated. To check how other forms were translated you have to attach code in the OnTranslated event handler for each form.

This method can only be used at runtime.

The language string must be the exact name of the language according to design time language name. This cannot be different, even though the language name has been translated by the component itself.

## Example

This example will exclude all configurations for an advanced screen, because it is a casual user.

Some code for logging in to the application

.....  
.....

**With** MultLang1 **Do**

**Begin**

        LanguageFilter.Clear;

**If** (UserName='Joe') **Then**

**Begin**

            LanguageFilter.Add('Manager');

            LanguageFilter.Add('Operator');

            LanguageFilter.Add('Supervisor');

**End;**

**End;**

## Edit Method

**Applies to**  
TMultLang

**Declaration**  
Procedure Edit(ExcludeList:TStrings)

**Description**  
This method will display the Languages Property Editor. The ExcludeList string list contains the language names you do not want to be listed in the dropdown list in the editor. To list all available configurations just pass a NIL value in the Edit method. This method can be called at both design and run time.

## TMultLang Languages Property Editor - Browse

### Description

All available properties will be listed in the outline to the left. Available properties is almost all properties that object inspector can edit. TStrings sub properties are shown only with the Items property and the number of the index. By singel click on properties you can see the design value as a hint below.

To include properties to the selected language just click on 'Insert', to remove properties click on 'Remove' whenever a property is highlighted. You can also move the whole component with all of its shown properties by highlighting the component at level 1 and then click on 'Insert' or 'Remove'.

By default it does not show all available properties in the left outline, you may for example want to change the visible property, then you can check the 'All' choice below in the filter section. It will then redisplay all properties according to the filter section.

The 'OK' button saves all changes and returns to the [languages property editor](#), the 'Cancel' button will cancel all changes and then return to the [language proprty editor](#). The 'Help' button will display this help page.

You can see the name of the language currently editing in the top border of the dialog.

## **TMultLang Language Property Editor - Copy from**

### **Description**

Probably you will configure one language for properties you want to change. Those properties may then be copied to other languages, this dialog specifies which language you want to copy properties from. Properties you already have in the selected language will not be overwritten if the Overwrite option is not checked.

The properties you copy will contain values, according to 'Set property values based on' section. If you have an external file you will be able to copy values from other forms with the option 'Other forms properties ...' set.

## OnTranslate Event

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property OnTranslate: [TTranslateEvent](#)

### Description

This event is called just before translating. Translating occurs whenever the dropdown list changes or another form within the same application is currently translating. If the translation is occurred by a change of this listcontrol the OnChange Event is always called before this event.

The LanguageName string contains the language to translate. You can not change the LanguageName string.

By Setting DontTranslate to true you will prevent the translating of the language and the [OnTranslated event](#) will have its Success variable set to false.



## TMultLang Tips - External files

### Many forms

As default the component saves the configuration in the same form as it is placed on. This behavior may not be what you expect if you have many forms like a MDI application, because it does not share configurations (languages) between forms. Instead, fill in the LanguageFile property to the same file in all TMultLang components. Now all changes in the configuration (languages) will automatically be reflected on all TMultLang components.

### LAN files

All files used by LanguageFile property, SaveToFile method, LoadFromFile method and 'Load from', 'Save to' buttons in languages property editor, is compatible with each other.

### Backup your configurations

The 'Load from' and 'Save to' buttons in languages property editor is a nice way of saving all configurations on external files which can be retrieved later. NOTE: Whenever a property name have changed since it last was saved to a file, you can only get these properties back by changing the name of the property back to what it was before you saved the file. This applies only to files that is not pointed to by the LanguageFile property.

## TMultLang Tips - Adding languages

### Copying properties

Once you are finished with choosing properties for one language, it would take long time to do the same with all new languages you add. You should instead use the 'Copy from' button in the properties section in the [languages property editor](#), which can copy properties from other languages without overwriting any you already have. You can then choose to remove or add properties from the 'Browse' button to make the language easier to maintain.

### Loading properties from files

If you have other forms with properties of same names as you have on the current form you can import languages, icons and properties from that form. The 'Load from' button will only import configurations based on the name of the current form. Instead call the [LoadFromFile](#) method with the name of the form you are loading from as parameter. That form should of course have been saved using the [LanguageFile](#) property in the TMultLang component first. However, if there are languages or properties that you already have configured for this form it should not be used, because all languages, icons and properties with same names will then be overwritten in the current form but all other configurations will be untouched.

### Language names

It can be very useful translating the language names in the component itself. This will enable you to display local language names for each language.

## LoadFromFile TMultiLang Method

[See also](#)

[Example](#)

### Applies to

[TMultiLang](#)

### Declaration

```
Function LoadFromFile(FileName:String; FormName:String):Boolean
```

### Description

Whenever you want to retrieve configurations from an external file you can call this method. It will overwrite all languages, properties and icons which already exists, but will leave other untouched.

The [LanguageFile](#) property and the buttons 'Load from' and 'Save to' in the [languages property editor](#), will always save the configurations with the current name of the form and the component (Form.ClassName+Self.Name). You can specify a form name you want to retrieve from with the FormName string and then add the name of the TMultiLang component.

The \*.LAN files can contain as many form configurations as you like.

The LoadFromFile method will try to open the pathname FileName in Read/Share mode and will close it when finished. If it can read the FormName successfully, it returns true otherwise false.

## SaveToFile TMultiLang Method

[See also](#)

[Example](#)

### Applies to

[TMultiLang](#)

### Declaration

```
Function SaveToFile(FileName:String; FormName:String):Boolean
```

### Description

All configurations of current form is saved in the pathname FileName under the name FormName whenever this method is called. The pathname specified by FileName can be an existing file and will then update the file if it is a language compatible file, otherwise it will create a new file.

The [LanguageFile](#) property and the buttons 'Load from' and 'Save to' in the [languages property editor](#), will always save the configurations with the current name of the form (Form.ClassName). You can specify a form name you want to save with the FormName string.

The \*.LAN files can contain as many form configurations as you like.

The SaveToFile method will try to create a temporary file in a temporary directory and save all configurations in that file even those that already exists in the specified file, then it will delete any existing file name (FileName) and move the file from the temporary directory to the pathname specified by the FileName string. It should the have Read, Write, Delete and Rename access rights in that directory. If it is succesfull saved it will return true, otherwise false.

## **TMultLang TTranslateEvent Type**

### **Declaration**

TTranslateEvent = **Procedure**(Sender:TObject; LanguageName:**String**; **Var** DontTranslate:Boolean) **Of Object**

### **Description**

This is the procedure type for the OnTranslate event.

## OnTranslated Event

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property OnTranslated: [TTranslatedEvent](#)

### Description

This event is always called after a translation and the LanguageName string contains the language just translated and the succesfull is true if everything was OK, otherwise it is false.

## **TMultLang TTranslatedEvent Type**

### **Declaration**

TTranslatedEvent = **Procedure**(Sender:TObject; LanguageName:**String**;  
Succesfull:Boolean) **Of Object**

### **Description**

This is the procedure type for the OnTranslated event.

## ShowAbout TMultiLang Method

**Applies to**  
TMultiLang

**Declaration**  
Procedure ShowAbout

**Description**  
This method will display the About dialog box. This is the same as clicking on the about property in the Object Inspector.



## AddIcon Method

See also

### Applies to

TMultLang

### Declaration

Function AddIcon(Language:String; Alcon:TIcon):Boolean

### Description

This method will attach an icon to the language named in the Language string. The language must be exact, and the icon must contain a valid icon reference. If successful it will make a copy of Alcon and place it in the language specified by the Language variable and return true, otherwise it returns false.

The added icon will be saved together with the language in either the form or an external file. The icon is ignored when using Export and Import methods.

The icon is only shown in the dropdown list when the ShowIcons property is true, and will be shown according to the AlignIcon and IconMargin properties.

You will probably never need to call this method because it can be assigned with the languages property editor.

## AddLanguage Method

[See also](#)

### Applies to

[TMultiLang](#)

### Declaration

```
Function AddLanguage(LangName:String; Alcon:TIcon):Boolean
```

### Description

Adds the language name LangName and the icon Alcon to the component. Alcon must be a valid icon or it can be a NIL value to just add the language. If the language already exists or something else went wrong, it will return false, otherwise true.

The LangName variable must contain the design-time name, this language name will be used when using other methods that needs the language name.

By default, a new added language will not contain any properties. It will be saved in the form or an external file.

**See also**

[AddIcon method](#)

[ClearLanguages method](#)

[DefaultLanguage property](#)

[DeleteLanguage method](#)

[EmptyLanguage method](#)

[FindLanguage method](#)

[LanguageFilter property](#)

[ReadLanguages method](#)

[RemoveIcon method](#)

[UpdateLanguages method](#)

[WriteLanguages method](#)

## ClearLanguages Method

See also

### **Applies to**

TMultLang

### **Declaration**

Procedure ClearLanguages

### **Description**

Clear all languages, including icons and properties. This method should be used with care!

Does not delete the design-time language name but all its properties and also the attached icon if any.

**See also**

[DeleteLanguage method](#)

[EmptyLanguage method](#)

[LanguageFilter property](#)

## DeleteLanguage Method

See also

### **Applies to**

TMultLang

### **Declaration**

Function DeleteLanguage(LangName:String):Boolean

### **Description**

Deletes the language called LangName, including icons and properties that is associated with this language. The LangName must contain the exact name of the language to delete which is in the design-time naming convention. If it finds the language and it is successfully deleted it returns true, otherwise false.

You cannot delete the design time language, it will always return false in that case.

The deleted language will be removed from the form. If an external file exist it does not physically delete the language unless you call the SaveToFile method after.

**See also**

[ClearLanguages method](#)

[EmptyLanguage method](#)

[LagnuageFilter property](#)

## EmptyLanguage Method

[See also](#)

### Applies to

[TMultLang](#)

### Declaration

Function EmptyLanguage(LangName:String):Boolean

### Description

Removes all properties and the icon for the language named LangName. The LangName must contain the exact name of the language to empty. If it finds the language and it is successfully emptied it returns true, otherwise false.

The emptied language will not be removed from the form or the external file, but all its attached properties and the Icon will be removed.



**See also**

[ClearLanguages method](#)

[DeleteLanguage method](#)

[LanguageFilter property](#)

## FindLanguage Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Function FindLanguage(LangName:String; ReturnID:Boolean):Integer
```

### Description

This method will look for the language named LangName and will return the ID, or the Index of the language depending on the ReturnID value. LangName must contain an exact name. If the language is not found it returns -1, regardless of the value ReturnID.

## IsLanguageFile Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Function IsLanguageFile(AFile:String):Boolean
```

### Description

This method will check the pathname of the file specified by the AFile string if it is a language file compatible with TMultLang component. If it is compatible it will return true otherwise false.

The file will be open with Read/Share mode and will check the header signature.

**See also**

[LoadFromFile method](#)

[SaveToFile method](#)

**See also**

[SaveToFile method](#)

[IsLanguageFile method](#)

## Removelcon TMultiLang Method

[See also](#)

### **Applies to**

[TMultiLang](#)

### **Declaration**

Function Removelcon(Language:String):Boolean

### **Description**

Clears an icon attached to the language named in the Language string. The Language string must contain an exact name of the language to clear the icon from. If succesfull it have disposed the memory and removed the icon from the language and returns true, otherwise it returns false.

The removed icon will also be removed from either the form or an external file.

You will probably never need to call this method because it can be removed with the [languages property editor](#).

**See also**

[AddIcon Method](#)

[AddLanguage Method](#)

[AlignIcon property](#)

[IconMargin property](#)

[ResizeIcon property](#)

[ShowIcons property](#)

**See also**

[LoadFromFile method](#)

[IsLanguageFile method](#)



## UpdateLanguages Method

### Applies to

TMultLang

### Declaration

Procedure UpdateLanguages

### Description

Will update all components and refresh their references. This method is internally called whenever a component changes, inserted and removed or if the Edit and LoadFromFile methods is called. The Translate method is using information from this method.

The dropdown list is updated whenever this method is called.

You should never need to call this method because it calls this method internally if whenever you call other methods to maintain languages.

## OnLoaded Event

See also [Example](#)

### Applies to

[TMultLang](#)

### Declaration

Property OnLoaded: [TLoadedEvent](#)

### Description

This event is always called after the component is created and all its data is initiated but before it is displayed.

## **TMultLang TLoadedEvent Type**

### **Declaration**

TLoadedEvent = **Procedure**(Sender:TObject) **Of Object**

### **Description**

TLoadedEvent defines an event handler for the OnLoaded event.

**See also**

[OnTranslate event](#)

[OnTranslated event](#)

**See also**

[OnLoaded event](#)

[OnTranslated event](#)

[Translate method](#)

**See also**

[OnLoaded event](#)

[OnTranslate event](#)

[Translate method](#)

**See also**

[OnTranslate event](#)

[OnTranslated event](#)

**See also**

[AddIcon method](#)

[AddLanguage method](#)

[AlignText property](#)

[IconMargin property](#)

[ResizeIcon property](#)

[ShowIcons property](#)



## Example

This procedure is attached to the OnDrawingItem event of the component. It will align the icon and text for the Chinese language to the right and all the other languages to the left.

```
procedure TForm1.MultLang1DrawingItem(Sender: TObject; Index: Integer);  
begin  
    With MultLang1 Do  
        if (Items[Index]='Chinese') Then  
            Begin  
                AlignIcon:=alRight;  
                AlignText:=alRight;  
            End  
        Else  
            Begin  
                AlignIcon:=alLeft;  
                AlignText:=alLeft;  
            End;  
end;
```

## OnDrawingItem Event

[See also](#)

[Example](#)

### Applies to

[TMultiLang](#)

### Declaration

Property OnDrawingItem: [TDrawingItemEvent](#)

### Description

This event is always called whenever the items in the drop down is to be drawn. This event is little different than OnDrawItem event in that it does not override the drawing. After this event is finished it will start drawing the items in the dropdown. The Index variable contains the index of the item to be drawn. This event is called for each language name item.

## **TMultLang TDrawingItemEvent Type**

### **Declaration**

TDrawingItemEvent = **Procedure**(Sender:TObject; Index:Integer) **Of Object**;

### **Description**

This is the procedure type for the OnDrawingItem event.

OnLoaded event  
OnTranslate event  
OnTranslated event

**See also**

[TextMargin property](#)

**See also**

[LanguageFilter Property](#)

**See also**

[AddIcon method](#)

[AddLanguage method](#)

[AlignIcon property](#)

[AlignText property](#)

[ResizeIcon property](#)

[ShowIcons property](#)

**See also**

[IsLanguageFile method](#)

[LoadFromFile method](#)

[SaveToFile method](#)



**See also**

[DefaultLanguage property](#)

**See also**

[AddIcon method](#)

[AddLanguage method](#)

[AlignIcon property](#)

[IconMargin property](#)

[ShowIcons property](#)

**See also**

[AddIcon method](#)

[AddLanguage method](#)

[AlignIcon property](#)

[IconMargin property](#)

[ResizeIcon property](#)

**See also**

[AlignText property](#)

**See also**

[AddLanguage method](#)

[AlignIcon property](#)

[IconMargin property](#)

[ResizeIcon property](#)

[ShowIcons property](#)

### **Example**

You can check if the language exists with this code:

```
If (MultLang1.FindLanguage('Swedish', False)=-1) Then  
    ShowMessage('Could not find the Swedish language !');
```

## **Example**

To check if a specified file is compatible with the TMultiLang component and then load the contents of it, you could use this code:

**With** MultiLang1 **Do**

**If** IsLanguageFile('C:\SHIPMENT\SHIPPING.LAN') **Then**

        LoadFromFile('C:\SHIPMENT\SHIPPING.LAN', Owner.ClassName+Name);

## Example

To save the current forms configuration (languages, icons and properties) you could use this code:

```
With MultLang1 Do  
  If SaveToFile('C:\SHIPMENT\SHIPPING.LAN', Owner.ClassName) Then  
    ShowMessage('The language configuration was successfull saved');
```



## Example

To translate a language which have been configured in the languages property editor you can use this code:

**With** MultLang1 **Do**

**If** Translate('Swedish', False) **Then**

        ShowMessage('The translation of Swedish was successful');

## **Example**

This code will show a message whenever the form is created:

```
procedure TForm1.MultLang1.Loaded(Sender: TObject);  
begin  
    ShowMessage('This form has multi language support');  
end;
```

### Example

This example event, attached to the OnTranslate event will prevent user Joe to translate to Spanish.

```
procedure TForm.MultLang1Translate(Sender: TObject; LanguageName: String; var  
DontTranslate: Boolean);  
begin  
    If (UserName='Joe') And (LanguageName='Spanish') Then DontTranslate:=True;  
end;
```

### Example

Attach this code to the OnTranslated event and it will show a message on each form that is translated whenever a translation occurs.

```
procedure TForm1.MultLang1Translated(Sender: TObject; LanguageName: String;  
Successfull: Boolean);  
begin  
    If Successfull Then ShowMessage(Owner.ClassName+' form was succesfull translated');  
end;
```

**See also**

[History](#)

[Property Editor](#)

[TCustomComboBox](#)

## **Tips**

[Editing values](#)

[External files](#)

[Adding languages](#)

## TMultLang Structure in Detail

Please see general [Structure](#) before reading this description.

TMultLang is a descendant component from TComboBox which is a descendant component from the [TCustomComboBox](#). The component maintains a list of languages and stores them in the component with the TList object.

Once you have created forms and controls you have probably already edit all captions and stringlists for a dedicated language. It is important to understand that while designing your application you should not change the language you have on all other forms, just continue designing it as you would do to make an application for one language. We will call it the design language.

This is where TMultLang component will come in. The TMultLang component will scan your design language and add more support for other languages. It does not matter if you add the component in the beginning or end of your projects, it will dynamically check for inserted or deleted components on forms. As default the design-time language is called English, this can be changed any time. The design-time language cannot be modified from the property editor of the TMultLang component, just edit those properties as you usually do in the Object Inspector.

All languages you add to the TMultLang component can change properties for other controls on the form (run-time), that is for example the Caption property of buttons or menus. The TMultLang component will easily alter almost all possible properties for Objects, Controls or Components that exists on forms.

Because of the TMultLang component can only recognise other components and controls on the same form, you will have to place the TMultLang component on each form you want to translate.

The component store the data on the form it is placed on, or on an external file. By default it is the form. By having an external language file for the languages, you can use the same file for all forms within the same project. When languages changes on one form it is reflected by all other forms both at design- and run-time.

If you choose to save languages in forms you will have to add the exact name of the languages on all other forms, otherwise it wont translate them automatically.

External files are opened with Read/Share mode.

Icons can be attached to make a more visibleimpression. You could for example attach flags for the languages. The icons will also be saved in the component.

You can filter out languages with the LanguageFilter property. That property is a TStrings Object and the language name must be exact otherwise it will be displayed anyway. This can be useful if you want to hide languages for special users or if you have special buttons which should not be displayed for a few persons.

## History of multiple language support

Below I will briefly describe a few scenarios about multi language support in applications, it is not ment to be a guide or a referense but just give you some ideas of the concerning issues. The situations and tasks described below is not associated with the TMultiLang component in other than just multi language support.

### The dilemma

When you already have a finished application you are glad as loong as the users dont find any intermitent bugs in the software that will take all your time to solve. But when you have come as far as distribution of your application it is time to think about other countries which may be a market for your software. You have to come up with a solution of having your software translated in another language 'Pronto' and you don't have the time to do it. Most of the people I have been in contact with had this situation, if you are, please read on for your best investment of your life! Just imagine having your software in lot of other markets, you may double or even triple your sales.

So a decision needs to be made quickly, we need a strategy for translating the software. The translation itself is not the real problem. The problem is how the source code will be maintained afterwards.

- o Will we simply copy all source files to a new directory, hire a translator, show him/her how to use a text editor and recognize strings and bingo! le tour est joué?
- o Will we create defines for each string?
- o Will we create memory arrays for each string?
- o Will we create a table (Database) or a text file with all strings in it?
- o Will we use conditional compiling and simply duplicate the portions of code containing string?
- o Will we embed the translation directly into on single source?
- o How are we going to change the form layout?

### Single language (hard coded)

Well this method is the most common if you are coming from the DOS world. When it was never foreseen in advance that multilingual support would ever be an issue. The marketing folks suddenly have the brilliant idea of expanding the territory for sales and surprise! surprise! surprise! .... The worst solution to support the need for multiple language is to copy all source code into another directory and simply translate all forms and menus into the desired language. Imagine supporting 3 languages, means three version of the source to maintain at all times; the smallest new release can become a nightmare! There are lot of drawbacks that this method have but it is quick and dirty. One major drawback is the maintaince of source code, another is the problems with common dialogs. You will also have problems with the length of your buttons/menus and other controls that has been designed for a specific size, most other languages have larger strings than the English language.

### Using Windows resources

This is the Windows way of having multi language support, just compile your resources to a DLL and then exchange the DLL with the language version you want. It is neat in most of the cases but it has a few drawbacks. You have to distribute different application versions and there may be easier having new bugs implemented.

As long as there are a few languages to support it is ok, but imagine having more than 3 languages and maintaining string and dialog resources it would be a real pain. You cant reuse string resources because they may be different in other languages. Imagine having massive duplication of menus with short cuts to be added whenever a new language should be implemented. It is a possible way but time consuming! However this method is the probably a better choice than all other methods, you have the ability to



change sizes of controls or even the layout. You'll have to make common dialogs templates as well as recompiling the application for each added language. Of course there are lot of utilities which maintains windows resources and make it more easy but you will never come away from the problem that the Widnows resources are read only and can only be created with your development tools. Another concern of using Windows resources is the different versions of 16 or 32 bit Windows. There could also be problems using API callback functions to point to your own resources on Windows NT or Windows 95, which have some changes in the resource structure loading. You have to concentrate on what you are doing and not the concern of how to maintain different languages.

### **Not only strings and dialogs**

Actually, there is more to consider than just text. There are lot of different date formats. This means we are actually talking about supporting other cultures, not just languages. The other things to consider are units of measure, currency and help files.

### **Dealing with dates...**

While most of the time dates cause no problem, the ones they do cause are often unexpected. Often you may not think of that when programming, many people write routines that reads or writes the date in a special format and some use the format derived from Windows. There may be problems when trying to run those applications on machines with different software versions installed. You may have a Swedish version of a database application but an English version of Windows. By having the date format set to Swedish in the control panel you will probably only be able to enter the date format of that setting even when you have switched to another language in that application.

The biggest problem is the string-to-date conversion. This is often solved by having user options available.

### **Dealing with money...**

For monetary units there are basically three things to consider: the number of significant digits; the character(s) used (e.g. \$); and where the characters are placed. This is the same situation as with dates.

### **Dealing with units of measure...**

This one is often a non issue for most applications. The biggest trick is making sure that any calculations are unitless. However, this doesn't work when validating user input, and some sort of factor needs to be stored. Having a public variable that indicates English, Imperial, or Metric units works extremely well, but requires checking it in each validation of unit specific data.

### **Dealing with Windows help files...**

This is the real dark side when thinking of having lot of complex help files for different languages. Because it is no resources in the helpfiles, we cannot change it on the fly, Ok maybe the names of buttons and some description texts. But the main idea with helpfiles is to describe different tasks and give help. In the reality, helpfiles is the one's Microsoft should have thought of when adding multi language support. Ok they have adressed most of the problems with the new Windows 95, but the help system is still the same, shame on them.

I think you will find the best way to make different help files with the same topic numbers and then

distribute different files depending on language. But what about switching language at run-time, you will have to programmatic make a solution for that. This is usually done by switching the referens name of the help file. Ex: HELPENG.HLP and HELPSWE.HLP can be switched by changing the referens to the file that contains the right language. Another draw back of using different help files is maintaining them, more languages will just add more complexity.

You can include different languages in the same helpfile and then just relate the topic to the running language in the application. This can be a great idea depending on the system you are designing, but it is not a god idea whenever you have more than 3 languages to support, it will eat up all windows resources depending on how complex it is and how much memory available at the client site.

You should also select a tool for making help files which enables you to use a more advanced word processor like Novell WordPerfect or MS Word because of the massive text you may be dealing with. Also you may be able to use grammar and spell checking to complete your translation which will make it more convenient for a quick translation.

Have you had enough ?, It is just the beginning.

**See also**

[Description in detail](#)

[History](#)

[TMultLang component](#)

## AdoptOn Property

Example

### Applies to

TMultLang

### Declaration

Property AdoptOn:TComponent

### Description

The AdoptOn property enables you to display the languages in an anothercontrol. It places a hook to maintain the list and to watch for changes. The AdoptOn accepts controls of type: TMainMenu, TPopupMenu, TMenuItem, TListBox, TComboBox and TRadioGroup. If there are icons assigned they will not be displayed on the assigned control. The TextMargin and AlignText will not have any effect. To Clear the property, just assign a NIL value. When setting this property to a valid control the Visible property will be set to False.

## Step-By-Step Tutorial

### **My first international application**

Before continuing I recommend reading the [Structure](#) topic. And for the one of you who want a quick introduction I have included a quick Step-By-Step.

The tutorials is based on the English Delphi 1.0 version with all its environment options set as when it was first installed(eg. browse gallery on New Project). The TMultiLang component must first be installed on the component palette according to the [Installation](#) topic.

You should already be familiar with the Delphi Design-Time interface. However, the design-time interface have two major parts, The Form Designer and the Component Designer and you will only need to know little about the Form Designer to go through this tutorial.

Please also look at the MDI example project included in this package which contains more than 10 translated forms in both English and Swedish.

The quick tour, show some basic functionality of the component and will take aprox. 5 min.

Launch the [QUICK STEP-BY-STEP](#).

The advanced tour, will demonstrate some advanced functionality and will take aprox. 15 min. Launch the [ADVANCED STEP-BY-STEP](#)

**PropertyVisible**

## Tasks

[A Step-By-Step tour](#)

[Adding a language](#)

[Changing the property values](#)

[Distributing your application](#)

[Getting property values from other forms](#)

[Installation of the TMultiLang Component](#)

[Planning your development](#)

[Un-Install the component](#)

## Planning your international application

### Before you start

Always when you have started develop an application something will come up that changes the rules for your development. It is just the same way the politicians do, they change the rules when the ball game have already started.

Well, I am very aware of those problems and thats why I have done my component as painless as possible. However it is a few guide lines I want you to be aware of that makes it easier in the future. I advice you to at least briefly go through the TECH\*.TXT files wich contains important implementation guidelines.

### Displaying private dialogs with ShowMessage, MessageDlg or InputQuery.

The most worse thing you can do is embedd text inside your code, always try to have the text in user-defined variables in the TMultiLang component.

For instance, whenever you want to display a message with the ShowMessage function you should put the message in a user-defined variable which is the best method. You can then assign names for your message variables and then add them in the Language Editor.

The following information is not recomended anymore but I have kept it here for you. Instead you should use the GetString and SetString methods of the TMultiLang component, which returns or sets a string dependning on the selected language.

*The TLabel is also a good holder for such strings because it is a control which can not have focus in run-time. However there are a few drawbacks, you cannot have more than 255 controls on the same window (Windows 3.1 limit) and you may run out of Windows resources. Whenever you have lot of messages I recommend using a Memo component instead. Because you can access and set lines of text inside it whenever needed. You may in that case make some empty lines in the Memo at design-time to hold enough strings for different languages. You can for example have each new string every 10<sup>th</sup> line, that will enable you to have just one Memo component for all your messages. An example follows:*

*ShowMessage(Memo1.Lines[(10\*5)+MultiLang1.ItemIndex]+AFileName);*

*Above line could be 'Error reading file: C:\AUTOEXEC.BAT' and in Swedish 'Fel vid läsning av fil : C:\AUTOEXEC.BAT'. Each 10:th line of the Memo represent a new string label.*

### Buttons and titles of the Message dialogs

The buttons of the message dialogs is predefined in a resource file in Delphi and cannot be used for on the fly translations. Instead use my TMBButtons component to translate all button captions and titles of those dialogs whenever suitable. This component also supports on the fly translations for the dialogs.

### Short &cuts

You should have a clear and distinct translation of the languages you will add, because it will then be easier to set the short cuts. I suggest you to first translate the languages without short cuts and then run your application and assign the short cuts afterwards. This is a very difficult thing to do because it often hapen that you will assign the same short cut to more then one control and there can happen mysterious things because of a bug in Windows 3.1.

Another thing to do is to have the same short cut on all languages, it will then be much easier to maintan and translate new languages. This is the recommended method.

However when you have menus on different forms which partially have the same structures you can use the 'Look Up' option in the Language Property Editor which does a English Soundex search of other forms. This enables you to choose menu items with short cuts on other forms from a drop down list. You can then more easier pick the words with an already assigned short cut.

### Foreign languages

If you plan to translate a lot of languages on your own computer I suggest you to make your translations on Windows 95 with the Multi Language support installed, which enable you to have different keyboard layouts. It would then be much more easier to find the right characters on the keyboard. To support



different languages that do not have latin or cyrillic scripts, you will have to switch fonts. You may for example use another type of MS Sans Serif to support double character languages. You can also use Right-To-Left editing with the Language Editor, please advice the TECH03.TXT file.

### **Common dialogs**

Use the MDialogs unit components which is descendants of the TCommonDialog components from Delphi but slightly modified to support different languages, even DBC.

Instead of using the TOpenDialog you should use the TMOpenDialog and translate all properties that suits the language as needed.

## Adding a new Language

You can add more languages both programmatically and via the Language Property Editor at both design- and run-time.

To add a new language using the Language Property Editor, just double click on the Languages property in the Object Inspector at design-time, or call the Edit method at run-time. Then click on the 'Add' button in the Language section. Enter the name of the new language and click on OK. That's it, you can attach an icon to be displayed next to the language by choosing the Load icon section. You will also have to choose the properties which this language will translate in the Browse dialog or in the Copy from dialog.

To add a new language at run-time call the AddLanguage method. To choose properties for the new language you have to call the Edit method and select them in the Browse dialog or in the Copy from dialog.

## Changing Values of properties

The values of properties can be changed both at design- and run-time. It is not possible to assign special values programmatically, it must be done via the Language Property Editor.

In the Language Property Editor it is a section called Properties which contains properties of Objects, Controls or Components, the value of those properties can be changed by in the New Value field. Properties can be added or removed in the Browse dialog or added via the Copy From dialog.

When inside the Value field you can navigate through properties by pressing PageUp or PageDown buttons. To choose values in the drop down list when available, press Up- or Down-arrow. To drop down the list click on the down arrow to the right or press ALT-Down arrow when inside the Value field. By pressing ESC you will discard any change to the property.

The drop down list may contain values when you edit properties which contain: Font Name, Colors, Ordinary Name values or when the Lookup check box is checked and it is a Caption or a Text property.

Color values can be entered as identifiers or as numbers.

String values have a limit of 255 characters. The string values can contain non alphabetic characters like CR or LF.

## Get Values of properties of other forms

Whenever you have configured one form with languages and property values translations it is possible to use those when translating other forms. However you can only use properties of other forms that has the same language and objects. For ex. you may have a MainMenu component on two forms which have Edit menus you would like to import.

You can choose the properties and the values to be transfered in the Copy From dialog in the Language Property Editor.

Also the Lookup check box in the Language Property Editor will retrieve values in the drop down list whenever there is a Caption or a Text property including variations of short cuts.

## Distributing your application

If you are a registered user of the [TMultLang](#) component you may distribute as many applications as you want Royalty Free. However, you are not allowed to distribute a derived component based on the TMultLang component or the included help file. If you are designing a component which is descendant from the TMultLang component you will have to purchase a special license agreement which make you be able to sell or distribute your component. See the README.TXT file included in the TMultLang package.

By distributing your application which contain the TMultLang component you will automatically agree the statements for distribution applications.

### Files

Because the TMultLang is a component, you will never need to and must not distribute any of the TMultLang component files. When compiling your application the component will be included in your EXE file. However you may want to distribute the external \*.LAN files that holds your translation strings for languages. If you want your application to be able to add more languages without recompiling you should make a Help file that responds to the actions when clicking on the Help buttons in the [Language Property Editor](#). Refer to the table below when designing help files.

*.EXE	Your application and its associated files.
*.LAN	The TMultLang component language files (Not needed when saving languages in the forms)
*.HLP	Your own Help file for the Language Property Editor. YOU ARE NOT ALLOWED TO DISTRIBUTE THE HELPFILE INCLUDED IN THE TMULTLANG COMPONENT PACKAGE.

### Help file

You have to compile your own helpfiles with your application including the one that can be attached to the Language Property Editor. You dont need this helpfile if you dont want the Language Property Editor displayed at run-time. The name of the helpfile must be MULTLANG.HLP. For futher information see the [Creating Helpfiles](#) documentation.

CONTEXT ID	DIALOG
210	The help button in the <a href="#">Language Property Editor</a> .
260	The help button in the <a href="#">Browse</a> dialog
270	The help button in the <a href="#">Copy From</a> dialog

## Ten easy steps to make your first international application!

This example will demonstrate the basic translating function of the component.

- ▶ 1. Create a new project in Delphi by choosing New Project under the File menu and then choose the Blank Project Icon and click OK.
- ▶ 2. Drop a label and a button on the form from the component palette. Change the Caption property of the label to 'Hello World' and the Caption property of the button to 'Translate' and the Caption property of the form to 'My first international application'.
- ▶ 3. Now, drop the TMultiLang component on the form. As you see it has already named our form language to English, this name can be changed with the DefaultLanguage property.
- ▶ 4. Double click on the Languages property in the Object Inspector, this brings up the language editor for our form we are designing.
- ▶ 5. In the language section click on the 'Add' button and then enter the name of your favorite human language we are going to support, ex 'Swedish'.
- ▶ 6. We will now tell the component which strings it should translate, we will do that in the browse dialog, click on the 'Browse' button in the properties section.
- ▶ 7. To the right we have all components and its properties that is available for translation, we will only translate the caption properties. Click on the 'Expand All' button then click on the first Caption property then click the '>> INSERT >>' button. It was now transferred to our language, do the same thing with the caption property of Button1 and Label1. When finished you should have three folders in the right list, then click the 'OK' button.
- ▶ 8. Now when we have selected all properties we want to translate, we have to change the strings to the appropriate translation. The cursor are already set in the Value edit field, so just translate the string to your language, ex 'My first international application' in Swedish 'Min första internationella applikation'. Then press PageDown button to jump to the next caption. Translate also the next two to your selected language, ex 'Translate' in Swedish 'Översätt' and 'Hello World' in Swedish 'Heja, Sverige'. Now we are finished, Click on the 'OK'.
- ▶ 9. Double click on the 'Translate' button and paste this code in it:  
`MultiLang1.Translate('Swedish', True);`  
Exchange the 'Swedish' name with the exact name of the language you have used instead.
- ▶ 10. Start the application by pressing 'F9'. It will prompt you where to save the unit and the project files, just pick a temporary directory.  
Try playing little bit with the controls and I think you will understand how it works .....  
You can also have the drop down invisible and use the DefaultLanguage property to have your application a pre translated language at startup. This can be useful when you want to distribute your application in a special language version.

Thank you for taking your valuable time for this tour, but I think you will get more productive instead. Think about how easy it can be having your program sold everywhere in the world with this kind of functionality. You can concentrate on what you are doing instead of trying to implement solutions yourself.

For a more advanced tour that demonstrates a little more the way of making your international application please click here: [ADVANCED STEP-BY-STEP](#)

## A MDI application example!

This example will demonstrate how the TMultiLang component communicates between forms and how it shares the languages between them. It also demonstrates the AdoptOn feature which makes a hook on another control and lists the languages in that control instead.

- ▶ 1. Create a new project in Delphi by choosing New Project under the File menu, this brings up the 'Browse Gallery dialog'. Choose the MDI Application Icon and click OK. Choose a temporary directory for the sample project.
- ▶ 2. Display the MDIChild window by selecting Forms in the View menu and select the MDIChild window.
- ▶ 3. Now, drop a Memo component on that form and then set the Align property of the memo to Left in the Object Inspector. Then drop two BitBtn buttons to the right of the memo component and set the Kind property to bkOK on one of them and bkCancel on the other. Also drop a simple button on the right side of the memo and change the caption property to 'Load'. Double click on the Load button and enter this code:  

```
If (Copy(Self.Caption, 1, 6)<>'NONAME') Then Memo1.Lines.LoadFromFile(Self.Caption);
```
- ▶ 4. Double click on the MainMenu in the MainForm window, then click on the empty rectangle to the right of Help (For inserting a new menu item). Enter 'Language' in the Caption property. You can close the menu window when you have entered the new menu item, it should be a new menu item called 'Language' below the title in the MainForm.
- ▶ 5. Now is the time for the TMultiLang component, drop the MultiLang component on both the MDIChild window and the MainForm window. Change the LanguageFile property to MDI.LAN on both forms for the MultiLang component. This ensures that we will share the languages between the forms in an external file. Also change the AdoptOn property on the MultiLang component in the MainForm to 'Language1' and the Visible property of the MDIChild window MultiLang component to False.
- ▶ 6. Ok, we will start translating the MainForm by double clicking on the Languages property in the Object Inspector for the MultiLang Component in the MainForm. This will bring up the Language Property Editor for the MainForm.
- ▶ 7. For simplicity we will only add one language. Click on the Add button in the language section. Then enter the name of your favorite language you want to support( Ex. 'Swedish'). I will refer to Swedish from now on, just exchange that with the name you have entered. Then click on the OK button to set the new language name.
- ▶ 8. We will now tell the component which strings it should translate, we will do that in the browse dialog, click on the 'Browse' button in the properties section of the Language Property Editor.
- ▶ 9. To the left we have all components with the properties that is available for translation, in this case we will only translate the caption properties. Click on the 'Expand All' button then click on the first Caption property then Click the ">> INSERT >>" button. It was now transferred to our language, do the same thing with the caption property of all other items but not those that have a name of Btn at the end of the folders. When finish you should have a lot of folders with items in the right list, then click the 'OK' button.
- ▶ 10. Now when we have selected all properties we want to translate, we have to change the strings to the appropriate translation. The cursor are already set in the Value edit field, so just translate the string to your language, ex 'MDI Application' in swedish 'MDI Applikation'. Then press PageDown button to jump to the next caption, also translate that to your selected language, ex '&File' in swedish '&Arkiv'. Press PageDown and continue until you have translated all captions. You will realize that when translating lot of menu items you should have a planned structure which includes which short cuts keys that will be used for each language (See [Planning](#)). Now we are finished, Click on the 'OK'.

- ▶ 11. Activate the MultLang component on the MDIChild window and double click on the 'Languages' property in the Object Inspector to bring up the Language Property Editor. Then click on the 'Browse' button to add the properties we want to translate. Click on the 'Expand All' button and insert the Caption property of BitBtn1, BitBtn2 and Button1. Then Click 'OK'.
- ▶ 12. Translate the three caption properties as we did before (navigating between properties with PageDown and PageUp buttons). When finished click on the 'OK' button.
- ▶ 13. Then start your application by pressing F9 and play around with the Language menu Item. You will notice that the hints are still in english, but that is not problem, you can translate also those in the Language Property Editor but I did not demonstrate that because it is the same procedures as we already have done.

If you try to select Open you will see that the Open Common dialog is not translated. This can easily be done with the MOpenDialog and MSaveDialog components. It is a good practice to change this project so it will also translate those dialogs. Further, after selecting a file from the OpenFileDialog or selecting New from the Main menu, it will bring up the MDIChild window, you will notice that the translation you select in the main window effects all the client windows you have opened or created with File-Open or File-New menu items.

Of course there are many other advanced features and those are detailed discribed in the help files for the Component package.

Thank you for taking your valuable time for this tour, but I think you will get more productive instead. Think about how easy it can be having your program sold everywhere in the world with this kind of functionality. You can concentrate on what you are doing instead of trying to implement solutions yourself.



## **Installation of TMultiLang component**

Please advice the README.TXT file for installation.

After that I recommend you to try the quick [Step-By-Step](#) tutorial in this help file.

**LanguageProptertyEditorLookUp**

### **Example**

```
MultLang1.AdoptOn:=ListBox1;
```

## Properties

▶ runtime properties

🔑 key properties

🔑 <u>AdoptOn</u>	<u>Font</u>	
🔑 <u>Resizelcon</u>		
<u>Align</u>	<u>Height</u>	🔑 <u>ShowIcons</u>
🔑 <u>AlignIcon</u>		
🔑 <u>IconMargin</u>	<u>ShowHint</u>	
🔑 <u>AlignText</u>		<u>ItemHeight</u>
<u>Ctl3D</u>	<u>Items</u>	🔑 <u>TextMargin</u>
<u>Color</u>	🔑 <u>LanguageFile</u>	<u>Visible</u>
🔑 <u>DefaultLanguage</u>		
🔑 <u>LanguageFilter</u>		
<u>DropDownCount</u>	🔑 <u>Languages</u>	
<u>Enabled</u>	<u>PropupMenu</u>	

Sorted

## GetExternalStrings Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Procedure GetExternalStrings(LangID:Integer; ObjectName, PropertyName:String; Var Value:String;  
Soundex:Boolean; Var Strings:TStrings)
```

### Description

The GetExternalStrings method will do a soundex search for Value in the external file pointed by [LanguageFile](#) property.

When Soundex is true, the function will search for language string values on other forms that has the supplied LangID and have property names of 'Caption' or 'Text'. ObjectName and PropertyName variables is ignored. The strings found on other forms according to above will be soundex compared with the Value. The soundex comparison is based on the english language and excludes characters like & (ampersands) to enable you to get values that have different short cuts. The function then returns the successfully compared values in the String variable.

If Soundex is false, the method will only retrieve the first value from other forms which have the LangID, ObjectName and PropertyName. The Strings is ignored and the function returns the found value in the Value variable.

LangID	Contains the language ID number retrieved by <a href="#">FindLanguage</a> method.
ObjectName	This is the name of the Object, Component or Control the method looks for when the Soundex variable is False.
PropertyName	The Name of the property from the ObjectName that the method looks for when the Soundex variable is False.
Value	When Soundex is true, the method will compare the Value variable and insert the successfully compared values in the Strings object.
Soundex	When true, search for Soundex values of other forms. When false, just retrieve first available value of other forms that has the LangID, ObjectName and PropertyName the same.

Strings When Soundex variable is false this Object will be ignored and should contain a value of NIL. When Soundex variable is true, values will be added to this String object. NOTE: The String object is not owned by this method and it is up to you to Create and Destroy it. If the Soundex variable is true and this String object is not valid, the result is unpredicted and will probably cause a GPF.

## Example

This example show how to retrieve all values that has a soundex value the same as 'Redigera' and have the same language as Swedish. The values found will be listed in the ListBox1 control. If you for example already have translated a form which contain a menu that has an item called '&Redigera' you will see that string in the list box. This can be useful when you want to retrieve the short cuts assigned to other forms.

```
MultLang1.GetExternalStrings(FindLanguage('Swedish', True), "", "", 'Redigera', True, ListBox1.Items);
```

**See also**

[FindLanguage](#)

[GetTranslation](#)

## Un-Install the TMultiLang component

1. First remove the component from the component palette in Delphi via the 'Install Component' menu options.
2. Regenerate the keyword index for helpfiles in the Helpfile installer program (HELPINST.EXE) without the MULTLANG.KWF file.
3. Just delete all files in the TMultiLang package in the directory you choosed when it was installed. Dont forget to delete the helpfile (MULTLANG.HLP) also, which is in the DELPHI\BIN directory if you installed it according to the installation instructions.



**Support**

All registered users will the first 3 month get free support. Support will be given on the address below. After the first 3 month the registered users will still get free support but in a reasonable amount. However, support will also be given to unregistered users with the trial period of the Shareware version.

If you have access to Internet, please take a look at URL site '<http://mailhost.net/~delphi>' for late braking news and updates. Do you use MAPI, MSN or Netscape I suggest you to take a look in the about dialog, that can be accessed from the Object Inspector.

Please see the enclosed README.TXT file for more updated information and the disclaimer agreement. As described in the Disclaimer I reserve the right to change any text at anytime as well as the features of the component in future updates.

Support address:

Patrik Wang  
1/8 Tusculum Street  
Potts Point, N.S.W. 2011, Sydney  
AUSTRALIA  
Fax +61 2 3269032  
100754.231@COMPUSERVE.COM or PWang@MSN.COM

## GetTranslation Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Function GetTranslation(AString:String; Language:String):String
```

### Description

This method will return a translated string in the language chosen by the Language string. AString is compared with the design language for a translation. The scope is only the current form. AString variable contains the string in the design language, and the Language variable contains the exact language name of the returned translation.

The AString variable string must already exist in the design-time language and there must also be an translation for the Language variable name, otherwise the method will return an empty string.

String comparison is made by the installed local language driver and it will skip the '&' characters.

**See also**

[FindLanguage](#)

[GetExternalStrings](#)

[GetMessage](#)

**Example**

```
ShowMessage('A french term of Open is '+MultLang1.GetTranslation('Open', 'French'));
```

## Export TMultiLang Method

[See also](#)

[Example](#)

### Applies to

[TMultiLang](#)

### Declaration

Function Export(FileName:String):Boolean

### Description

The Export method uses the usual Windows INI text format to write translations of languages. This method should work in junction with the [Import](#) method.

For explanation of the output format please see the [Import](#) method.

If FileName does not exist it will be created. If FileName already exist the file is updated in the same manner as Windows INI files. If no path is supplied the file will be created in the Windows directory.

Only properties belonging to the TMultiLang component is output and it does not output other forms properties.

The section language names is in the design-time language naming convention, that means if the language names are translated it wont write the translated language name but the design-time language name.

## GetString Method

[See also](#)

[Example](#)

### Applies to

[TMultLang](#)

### Declaration

```
Function GetString(Identifier:String):String
```

### Description

Returns a string from a user defined identifier. The Identifier is automatically translated to the current language. This method is very useful when displaying short messages with for example the ShowMessage function.

The string contents of the identifier can include substrings. Ex: "I like my &ComputerName&" could return a string like this: "I Like my Compaq".

The string contents can also have string or number substitutes. Ex "Welcome %s, you have %d message(s) waiting" using the [Format](#) function it can return a string like this: "Welcome Patrik Wang, you have 4 message(s) waiting"

## Import TMultiLang Method

[See also](#)

[Example](#)

### Applies to

[TMultiLang](#)

### Declaration

Function Import(FileName:String):Boolean

### Description

The Import method uses the usual Windows INI text format to retrieve translations of languages. The format is as follow:

```
[English]
```

```
Button1.Caption=&OK
```

```
Button2.Caption=&Cancel
```

```
ListBox1.Items%0=Apple
```

```
ListBox1.Items%2=Orange
```

```
[Swedish]
```

```
Button1.Caption=&OK
```

```
Button2.Caption=&Avbryt
```

```
ListBox1.Items%0=Äpple
```

```
ListBox1.Items%2=Appelsin
```

```
.....
```

```
....
```

Where the sections forms the language and the keys are the objects + properties separated by a dot character. The value is the actual value of the property. The TStringItems property are recognised by the index number to the right of a % character. Each row must not exceed 255 characters before the linebreak. The file can be over 64Kb.

This function does not remove other languages or properties but it will overwrite all existant properties and languages. This function can also be used to create new languages or properties and will work best with small files.

All object and properties will be loaded regardless if they exist in the component. If some of the properties does not exist it will be automatically attached to an object if such a object name is created afterwards in the design-time. This is the automatic recovery method used by the TMultiLang component. However when using the [Export](#) method only valid properties will be exported at the time the method was called.

The imported properties does only effect current form. If using external files the newly created languages will not take effect on other forms until the [SaveToFile](#) method is called.

If any problems occur it will stop and return False otherwise if successfull it returns true.

## SetString Method

[See also](#)      [Example](#)

### Applies to

[TMultiLang](#)

### Declaration

```
Procedure SetString(Identifier:String; AString:String);
```

### Description

Sets a user defined identifier. The Identifier will be created if it does not exist. If AString is empty the Identifier will be removed from current selected language.

The AString variable can include substrings. Ex: "I like my &ComputerName&".  
The identifier "&CompuerName&" is the name of another user defined identifier.

AString can also have string or number substitutes. Ex "Welcome %s, you have %d message(s) waiting" using the [Format](#) function on the [GetString](#) method, it can return a string like this: "Welcome Patrik Wang, you have 4 message(s) waiting"



## **Advanced features**

### **Localized language names**

The language names can also be translated by selecting the Items property of the TMultiLang component it self. Whenever you translate to a language you will see available languages in the local language name.

### **Let agents do the translations for you**

A really nice way is having your application distributed to other countries for localized translations without sending out your source code for recompilation. How this can be done is easilly demonstrated in the component it self. The TMultiLang component has a key kombination (CTRL-ALT-E) to Add, Remove or Edit the supported languages of the component. By default the component does not save the languages in an external file and therefore it cannot be edited in run-time. But by setting LanguageFile=c:\myfile.lan in the Default section of the MULTLANG.INI file in the windows directory you will be able to add your new languages to this component without the use of the source code. How does it work ?, well in the Form.Create event enter a conditional code to set the LanguageFile property of the component. You will then have the basic languages you made in the beginning and whenever new languages needs to be added it can be done via the external file option.

### **Component recovery**

If you accidently deleted a component in the design-time interface that had many translations, don't worry just paste or drop it back again and set the name as it was before and every thing is normal again. The deleted component is not removed from the language resource until the next time you load, saves or quit. Invalid components (deleted or not existent) will show up in the property list with four ??? question marks before the object name. It will be removed automatically later if you don't assign a new component with that name.

**See also**

[SaveToFile](#)

[LoadFromFile](#)

[Import](#)

**Example**

All configured language properties with their values will be written to the INI file MYFILE.TXT with this calling method.

```
MultLang1.Export('C:\MYFILE.TXT');
```

**See also**

[SaveToFile](#)

[LoadFromFile](#)

[Export](#)

## Example

This will create a language called Quebec and insert a translation for the Caption property of the OKBtn button on the form.

```
MultLang1.Import('C:\MYFILE.TXT');
```

MYFILE.TXT contents ....

```
[Quebec]
```

```
[Swedish]
```

```
OKBtn.Caption=A&vbryt
```

**See also**

[GetTranslation](#)

[SetString](#)

[Import](#)

## Example

This shows a user defined message in the local selected language. The message of the identifier must first be set up by the Language Editor and 'Browse' dialog or by using SetString and Import methods.

```
ShowMessage(Format(MultLang1.GetString('LoadBtn_Fail'), [MOpenDialog1.FileName]));
```

If the user defined string "LoadBtn\_Fail" contains "Could not load the file %s" in English and "Filen %s kunde inte öppnas" in Swedish, it can show a string like "Could not load the file AUTOEXEC.BAT" in the english and "Filen AUTOEXEC.BAT kunde inte öppnas" in Swedish

**See also**

[GetString](#)




## Example





This will assign a text string to the CheckMsg user-defined variable and then display that message. However the message can vary depending on the current selected language as shown below.

```
MultLang1.Translate('English', False);  
MultLang1.SetString('CheckMsg', 'This is a test !!!');  
MultLang1.Translate('Swedish', False);  
MultLang1.SetString('CheckMsg', 'Detta är en test !!!');
```

```
ShowMessage(MultLang1.GetString('CheckMsg'));  
MultLang1.Translate('English', False);  
ShowMessage(MultLang1.GetString('CheckMsg'));
```

## Events

 key methods

<u>OnChange</u>	<u>OnEndDrag</u>	<u>OnMouseUp</u>
<u>OnClick</u>	<u>OnEnter</u>	 <u>OnTranslate</u>
<u>OnDragDrop</u>	<u>OnExit</u>	 <u>OnTranslated</u>
<u>OnDragOver</u>	 <u>OnLoaded</u>	
 <u>OnDrawingItem</u>		<u>OnMouseDown</u>
<u>OnDrawItem</u>	<u>OnMouseMove</u>	

## TMComboBox Component

[Properties](#) \_\_\_\_\_

[Methods](#)

[Events](#)

This is a descendant component from the [TCustomComboBox](#). It adds some functionality to include icons and to adjust the layout of any Text and the Icon with a few properties.

This component makes it possible to Left, Center or Right justify text in the list.

In addition to the inherited component, you can use this component when you want to display Icons or adjust the text layout of single items in the list.

Use the AttachIcon and DetachIcon with the Index nr of the Item you want to set. The property ShowIcons tells if any contained icons should be displayed.

The IconMargin, TextMargin, AlignIcon and AlignText properties determine the appearance of text and if any Icon.

The AlignText and AlignIcon controls where the Text and or Icon should be placed in the list.

## Properties

▶ runtime properties

🔑 key properties

<u>Align</u>	<u>Font</u>	🔑 <u>ShowIcons</u>	
🔑 <u>AlignIcon</u>		<u>Height</u>	<u>ShowHint</u>
🔑 <u>AlignText</u>			
🔑 <u>IconMargin</u>		<u>Sorted</u>	
<u>Ctl3D</u>		<u>ItemHeight</u>	🔑 <u>TextMargin</u>
<u>Color</u>	<u>Items</u>	<u>Visible</u>	
<u>DropDownCount</u>		<u>PopupMenu</u>	
<u>Enabled</u>	▶	<u>ResizeIcon</u>	

## Methods

▶ key methods

- ▶ GetIcon
- ▶ AttachIcon
- ▶ DetachIcon

## GetIcon Method

### Applies to

TMComboBox

### Declaration

```
Function GetIcon(Index:Integer):TIcon
```

### Description

Returns the attached Icon to the item indicated with the Index parameter. If it does not contain any Icon it will return **nil**. The Index parameter must contain a valid item nr, the first is 0.

## AttachIcon Method

### Applies to

TMComboBox

### Declaration

Function AttachIcon(Index:Integer; Var Alcon:TIcon):Boolean

### Description

This method will attach an icon to the list item indicated by the index parameter. The index parameter must be within the range of its containing strings, the first Index is 0. The icon must contain a valid icon reference. If succesfull it will make a copy of Alcon and place it in the list index and return true, otherwise it returns false.

The added icon will be saved together with the string and is not using the object referenses available in TStrings, therefore you can safely use any Objects properties of the string list for own usage.

The icon is only shown in the dropdown list when the ShowIcons property is true, and will be shown according to the AlignIcon and IconMargin properties.

## DetachIcon Method

[See also](#)

### Applies to

[TMComboBox](#)

### Declaration

Function DetachIcon(Index:Integer):Boolean

### Description

Clears an icon attached to the stringlist at the item indicated by the Index parameter. The Index must contain a valid item, the first is 0. If succesfull it have disposed the memory and removed the icon from the language and returns true, otherwise it returns false.



## Events

▶ key methods

<u>OnChange</u>	<u>OnEndDrag</u>	<u>OnMouseUp</u>
<u>OnClick</u>	<u>OnEnter</u>	
<u>OnDragDrop</u>	<u>OnExit</u>	
<u>OnDragOver</u>	<u>OnLoaded</u>	
▶ <u>OnDrawingItem</u>	<u>OnMouseDown</u>	
<u>OnDrawItem</u>	<u>OnMouseMove</u>	



