TCurrentLanguages Component Version 1.0

License:

This component is been published under the GNU (http://www.gnu.org/licenses/gpl.html) public license:

```
TCurrentLanguages Component A component for working with Language Layouts.

Copyright (C) 2003  by Ido Kanner - ik_5@hotmail.com

This component is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Free
Software Foundation; either version 2 of the License, or (at your option)
any later version.

This component is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc., 59
Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

I also request that any bug fixing or additional code that you publish on your own, please send me a copy of that code.

About This Component:

The goal of the component is to make a layout manager and to control all the layouts installed in the user Windows operating system.

This component was designed only for Microsoft Windows, and not for Linux.

Although This component was created with Delphi 7, the component should work with all other versions of Delphi.

The component version is 1.0.

History:

4-11/5/03 - Component created and cleaned from all annoing bugs that was found.
I wish to thanks Uri Hartman for helping me with bug testing...

This component was designed in a way that you can change it and inherited all of it's needed options for tweaking it.

I hope you will find the component helpful for you.

Information about the component:

Files:

LanguageChanger.pas – This file is the main file of the component itslef.
RegLanguageChanger.pas – This file is the registration unit, and a property editor.
LanguageChanger.dcr – This file is the resource file of the component inside the Delphi IDE.
layout.bmp – This file is the bitmap file existed in the .dcr file.
TcurrentLanguages.rtf,pdf – This document.
GNU-Public Licence.txt – A copy of the GNU licence.

Installation:

Options->Library menu. Push the button next to the “library path” field. Select the path where you put all the files mentioned above and press on “ok”. Exit the dialog by pushing on the “ok” button.

In Delphi 3-7 if the package existed then open it in delphi, go to Tools->Enviorment Now, Press on the “Compile” button inside package dialog.

If the package does not existed then go to the main menu and select Components->Install Component->Into New package. Write the same path where the rest of the files are something like this in the “Package file name”: LanguageChangerDXXX.dpk where the XXX stand for the version of your Delphi copy (ie Delphi 4, Delphi 5 etc..). Now Select all the .pas files written above, and press on “ok”.

When the compilation is over press on the “Install” button and go to “File” menu and select “Close All” item remember if the package existed then do not save it, while if it does not exists do save it.
Now search in the Component palate for the “IK - NON Visual” page and you will find the component there.

Help File:

Constants:

MaxLanguaes = 143;

This constant include the maximum number of layouts that existed in Microsoft Windows.

```
LT_ID          = 0;
LT_LOCOLIZED   = 1;
LT_ENGLISHNAME = 2;
LT_SHORTNAME   = 3;
LT_NATIVENAME  = 4;
LT_EXTANTION   = 5;
```

This constants are the integer identifiers witch been uses with the SetLayoutList procedure.

```
LTS_ID          = 'ID';
LTS_LOCOLIZED   = 'LOCOLIZED';
LTS_ENGLISHNAME = 'ENGLISH';
LTS_SHORTNAME   = 'SHORT';
LTS_NATIVENAME  = 'NATIVE';
LTS_EXTANTION   = 'EXT';
```

This constants are the string identifiers witch been uses with the SetLayoutList procedure.

Types:

```
TrecLang = record
    strLongLangID      : string;
    strLangID          : string;
    hklLangID          : HKL;
    intlLANGUAGE       : Integer;
    strSLANGUAGE       : string;
    strSENGLANGUAGE    : string;
    strSABBREVLANGNAME : string;
    strSNATIVELANGNAME : string;
    strExtention       : string;
  end;
```

This record is the main structure that the component uses:

strLongLangID – This is the original outcome that received by windows.
strLangID – This is the hexa number that received by windows.
hklLangID – This variable is the same as the string but in an integer value.
intlLANGUAGE – This variable contains the real identifier of the language.
strSLANGUAGE – This variable is the localized name of the language.
strSENGLANGUAGE – This variable is the English name of the language.
strSABBREVLANGUAGE – This variable is the shorten name of the language (Usually 3 letters).
strNATIVELANGUAGE – This the name of the language in it's own letters and native name (English = English, Hebrew = עברית, Arabic = عربسق etc...).
strExtention – This is only 2 letters of the language (taking from the Abbrev variable).

TLangs = **array**[0..MaxLanguaes-1] **of** TrecLang;

This array contain all of the existed layouts data in Windows.
Note: this is only a structure not the container of the actual layouts data.

TLayouts = **array of** HKL;

This array is a dynamic array structure that contain the existed (installed) language layouts inside Windows.
Note: This is only a structure not the container of the actual layouts.

TBeforeLanguageChangesEvent = **procedure** (Sender : TObject; CurrentLanguage, NewLanguage : TrecLang) **of object**;

This is a structure of the event BeforeLanguageChanges.
The CurrentLanguage and the NewLanguage parameters are from the TrecLang record.
The sender parameter is from the TObject object, and is telling who send this request.
They are containing the information about the current active language layout and the requested layout.

TAftherLangueChangedEvent = **procedure** (Sender : TObject; NewLanguage : TrecLang) **of object**;

This is a structure of the event AftherLanguageChanges.
The sender parameter is from the TObject object, and is telling who send this request.
The NewLanguage is from the TrecLang record, it contains the information about the New layout that just been activated.

TListType = (ltID, ltLocolized, ltEnglishName, ltShortName, ltNativeName, ltExtantion);

This is a structure that is been in use by the SetLayoutList procedure.
Each identifier is represent an information about the layout.

EFileError = **class**(Exception);

This is an exception type. It's been raised when the reading and writing data from/to a file contains errors.

EUnknownIdentifer = **class**(Exception);

This is an exception type. It's been raised when unknown data is passed as a needed parameter.

The main component (TcurrentLanguages = **class**(TComponent)):
procedures:

constructor Create (AOwner : TComponent); **overload**; **override**;
constructor Create (AOwner : TComponent; Layout : HKL); **overload**; **virtual**;
constructor Create (AOwner : TComponent; InFile : **string**); **overload**; **virtual**;

This procedures are the creation constructors and it can contains 3 types of constructors:
1. The original constructor witch initialize the component.
2. A constructor witch initialize the component and then load a layout.
3. A constructor witch initialize the component and then load a layout from an existed file.

Note: the constructor declared in the public section.

destructor Destroy; **override**;

This procedure is a destroying destructor. At the moment it does not contain any code except an inheritance of the original destructor.

Note: the destructor declared in the public section.

procedure MessageHandler (**var** Msg: TMsg; **var** Handled: Boolean); **virtual**;

This procedure receive the message about the layout changing of the current proccess.
This procedure hack into the Application.OnMessage event. But i also execute the OnMessage event if it was assigned.

Note: the procedure declared in the protected section.

procedure InitilizeLanguagesLists; **virtual**;

This procedure initialize the properties data existed in the component.

Note: the procedure declared in the protected section.

procedure MakeListFromLayouts (Layouts : TLayouts; **out** Languages : TLangs); **virtual**;

This procedure makes the list of all installed inside the Layouts array and put it into Languages parameter.

Note: the procedure declared in the public section.

procedure Refresh;

This procedure refreshes all the data that this component uses (i.e. Existed layouts).

Note: the procedure declared in the public section.

procedure SaveToFile (Layout : TrecLang; OutFile : **string**); **virtual**;

This procedure Save a Layout record into a file.
The Layout parameter is of type TrecLang. It is the record that needs to be saved.
Note, that the procedure does not check if the data inside the record is valid or existed, so please make sure by yourself to validate it.

Note: the procedure declared in the public section.

procedure LoadFromFile (InFile : s**tring**; **out** Layout : TrecLang); **virtual**;

This procedure load a saved layout from an existed file. If the file does not exists, then the procedure raises and exception of EfileError.
The InFile parameter is for a file to load.
The Layout parameter is a record that receive the data that loaded from the file.

Note: the procedure declared in the public section.

procedure SetLayoutList (ListType : TListType; List : TStrings); **overload**; **virtual**;
procedure SetLayoutList (ListType : Integer; List : TStrings); **overload**; **virtual**;
procedure SetLayoutList (ListType : **string**; List : TStrings); **overload**; **virtual**;

This procedure put a type of Layout information into a TStringList while the TStrings Is only an abstract.
The procedure takes all the existed layouts that installed inside Windows, thats why It adds the data into a TString.
ListType can be or a TListType parameter, or an integer parameter or a string parameter. In order to know the integer and the string values allowed look at the constant section of this document. In order to know the values that the TListType can have, look at the Types section in this document.

Note: the procedure declared in the public section.

Functions:

Note: All of the following functions existed only in the public section.

function CountLayouts : Integer; **virtual;**

This function counts the existed layouts in Windows.
If there was an error return, then the value of the function will be 0. If no error was found it will return the number of existed layouts even if there is only one layout.

function AvilableLayouts (**out** Layouts : TLayouts) : Boolean; **virtual**;

This function puts the existed layouts in Windows inside a TLayouts paremter.
The function return true if there was no error receiving the layouts. But if there was an error it will return false.

function GetLanguageData(Layout : HKL; Data_Type : Integer; **out** Data : **string**) : Boolean; **virtual**;

This function is an abstract function. It receive a Layout and a DataType information are return the wished data inside the Data parameter.
If the function failles, it will return false, otherwise it will return true.
This function uses the GetLocalInfoA for Windows 9x/Millenum/XP home and GetLocalInfoW for Windows NT/2000/XP Pro etc... .
The Data_Type is a Windows API constant like LOCALE_ILANGUAGE, LOCALE_SENGLANGUAGE and more. For more information look in the Windows API documents about the GetLocaleInfo function.

function GetLanguageID(Layout : HKL; **out** ID : **string**) : Boolean; **virtual**;
function GetLanguageLocolizedName(Layout : HKL; **out** Name : **string**) : Boolean; **virtual**;
function GetLanguageEnglishName(Layout : HKL; **out** Name : **string**) : Boolean; **virtual**;
function GetLanguageShortenName(Layout : HKL; **out** Name : **string**) : Boolean; **virtual**;
function GetLanguageNativeName(Layout : HKL; **out** Name : **string**) : Boolean; **virtual**;
function GetLanguageExtentionName(Layout : HKL; **out** Name : **string**) : Boolean; **virtual**;

All of this functions uses the GetLanguageData function.
They retrieve information about a given layout and put the result into the Name/ID parameter.
If the functions failles then they return false, otherwise they will return true.

function FillArray (Layout : HKL; **out** recLang : TrecLang) : Boolean; **virtual**;

This function receive a layout and put the data of it inside a TrecLang record.
The Layout parameter is the given layout and the recLang parameter is the returned data.
If the function is successful then it return true otherwise it will return false.

function GetActiveLayout : HKL;

This function retrieve the handle of the current active layout in this process.

function ActiveLayout (Layout : HKL) : Boolean; **overload**; **virtual**;
function ActiveLayout (Layout : TrecLang) : Boolean; **overload**; **virtual**;

This function activate a wished layout. The activation set the wished layout to be the active one... i.e. if for example the current active layout is Russion and you wish that English will be the current layout this function does it.
The function can have a layout handle or a TrecLang record.
The function does not check to see if the record is full or with a valid data.
The function return true if the activation of the layout was successful, otherwise it will return false.

function NextLayout : Boolean; **virtual**;
function PreviousLayout : Boolean; **virtual**;

This functions sets the Active layouts to the next or previous layout that existed. It uses the layout list of Windows and it uses that order in order to set the next or previous layout in the list.
The functions return True if the successful, otherwise it will return false.

function GetIDFromRecord(DataType : TListType; Value : **string**) : HKL; **virtual**;

This function will return the layout handle by searching all of the internal array of layouts. The value type is represented by the DataType parameter and the Value parameter is the string to be search (including an handle that you wish to see that is existed).
The function will return the handle of a layout if the layout was found or 0 if the layout was not found.

Properties:

Public properties:

property Handle: HKL;

This is a read only property that will give you the current activated layout handle.

property AutoRefresh : Boolean;

This property determine if every action will refresh the internal layouts array information or not.
Default is false.
Please do not use this property unless you are know what you are doing. It seems that Delphi does not love to refresh the properties information on any action it takes so don't use it. If you wish to refresh the data call to the Refresh procedure.

Published properties:

property LayoutID : **string**;
property LayoutLocolizedName : **string**;
property LayoutEnglishName : **string**;
property LayoutShortName : **string**;
property LayoutNativeName : **string**;
property LayoutExtention : **string**;

This properties are the data of the activated layout of the active process.
All of the information of this properties are string.

LayoutID – return the ID of the active layout (not the handle but the layout number).
LayoutLocolizedName – return the localized name of the active layout.
LayoutEnglishName – return the English name of the active layout.
LayoutShortName – return the short name of the active layout (usually 3 letters).
LayoutNativeName – return the "real" name of the active layout (English = English, Hebrew = עברית, Arabic = عربسق etc...).
LayoutExtention – return 2 letters of the short name of the active layout.

Changing this properties will change the layout language to the selected layout.

Events:

property AftherLanguageChanges : TaftherLangueChangedEvent;

This event happens after the user caused a layout changes to the current process.
To see the structure of the created procedure see the Types section of this document.

property BeforeLanguageChanges : TBeforeLanguageChangesEvent

This event happens when the user wishes to change the layout and the changes of the layout was just sent.
To see the structure of the created procedure see the Types section of this document.

Global Functions:

function MAKELCID (wLanguageID, wSortID : Word) : DWORD;

This function is execute the Windows API macro with the same name.
The function creates a locale identifier from a language identifier.
The wLanguageID parameter Specifies the language identifier. This parameter is a combination of a primary language identifier and a sublanguage identifier and is usually created by using the MAKELANGID function witch does not been created in this code.
The wSortID parameter Specifies the sort identifier.

function GetLocaleDataA (ID: LCID; Flag: DWORD; **out** Size : Integer) : **string**;
function GetLocaleDataW (ID: LCID; Flag: DWORD; **out** Size : Integer) : **string**;

This functions is a tweak of the SysUtils unit internal functions with the same name.
This functions uses NT/9x functions (A for 9x and W for NT).
The functions Query the OS for information for a specified locale.

Global Variables:

LanguagesList : TLangs;

This variable is the exited layout lists that is been used by the component and by the property editors classes ...

LayoutNum : Integer;

This variable return the number of layouts inside the LanguageList array.

Reference

Delphi:

1. Borland -> http://www.borland.com/
2. Google.com -> http://www.google.com/
3. Torry.ru -> http://www.torry.ru/
4. Swiss Delphi Center -> http://www.swissdelphicenter.ch/

Windows API: (http://msdn.microsoft.com)

1. ActivateKeyboardLayout.
2. Locale Information.
3. WM_INPUTLANGCHANGE.
4. MAKELCID.
5. WM_INPUTLANGCHANGEREQUEST.
6. Getkeyboardlayoutlist.
7. GetLocaleInfo.
8. Languages Layout list:

Identifier	Language
0x0000	Language Neutral
0x007f	The language for the invariant locale (LOCALE_INVARIANT).
0x0400	Process or User Default Language
0x0800	System Default Language
0x0436	Afrikaans
0x041c	Albanian
0x0401	Arabic (Saudi Arabia)
0x0801	Arabic (Iraq)
0x0c01	Arabic (Egypt)
0x1001	Arabic (Libya)
0x1401	Arabic (Algeria)
0x1801	Arabic (Morocco)
0x1c01	Arabic (Tunisia)
0x2001	Arabic (Oman)
0x2401	Arabic (Yemen)
0x2801	Arabic (Syria)
0x2c01	Arabic (Jordan)
0x3001	Arabic (Lebanon)
0x3401	Arabic (Kuwait)
0x3801	Arabic (U.A.E.)
0x3c01	Arabic (Bahrain)
0x4001	Arabic (Qatar)
0x042b	**Windows 2000/XP:** Armenian. This is Unicode only.
0x042c	Azeri (Latin)
0x082c	Azeri (Cyrillic)
0x042d	Basque
0x0423	Belarusian
0x0402	Bulgarian
0x0455	Burmese
0x0403	Catalan
0x0404	Chinese (Taiwan)
0x0804	Chinese (PRC)
0x0c04	Chinese (Hong Kong SAR, PRC)
0x1004	Chinese (Singapore)
0x1404	**Windows 98/Me, Windows 2000/XP:** Chinese (Macau SAR)
0x041a	Croatian
0x0405	Czech
0x0406	Danish

0x0465	**Windows XP:** Divehi. This is Unicode only.
0x0413	Dutch (Netherlands)
0x0813	Dutch (Belgium)
0x0409	English (United States)
0x0809	English (United Kingdom)
0x0c09	English (Australian)
0x1009	English (Canadian)
0x1409	English (New Zealand)
0x1809	English (Ireland)
0x1c09	English (South Africa)
0x2009	English (Jamaica)
0x2409	English (Caribbean)
0x2809	English (Belize)
0x2c09	English (Trinidad)
0x3009	**Windows 98/Me, Windows 2000/XP:** English (Zimbabwe)
0x3409	**Windows 98/Me, Windows 2000/XP:** English (Philippines)
0x0425	Estonian
0x0438	Faeroese
0x0429	Farsi
0x040b	Finnish
0x040c	French (Standard)
0x080c	French (Belgian)
0x0c0c	French (Canadian)
0x100c	French (Switzerland)
0x140c	French (Luxembourg)
0x180c	**Windows 98/Me, Windows 2000/XP:** French (Monaco)
0x0456	**Windows XP:** Galician
0x0437	**Windows 2000/XP:** Georgian. This is Unicode only.
0x0407	German (Standard)
0x0807	German (Switzerland)
0x0c07	German (Austria)
0x1007	German (Luxembourg)
0x1407	German (Liechtenstein)
0x0408	Greek
0x0447	**Windows XP:** Gujarati. This is Unicode only.
0x040d	Hebrew
0x0439	**Windows 2000/XP:** Hindi. This is Unicode only.
0x040e	Hungarian
0x040f	Icelandic
0x0421	Indonesian
0x0410	Italian (Standard)
0x0810	Italian (Switzerland)
0x0411	Japanese
0x044b	**Windows XP:** Kannada. This is Unicode only.
0x0457	**Windows 2000/XP:** Konkani. This is Unicode only.
0x0412	Korean
0x0812	**Windows 95, Windows NT 4.0 only:** Korean (Johab)
0x0440	**Windows XP:** Kyrgyz.
0x0426	Latvian
0x0427	Lithuanian
0x0827	**Windows 98 only:** Lithuanian (Classic)
0x042f	FYRO Macedonian
0x043e	Malay (Malaysian)
0x083e	Malay (Brunei Darussalam)
0x044e	**Windows 2000/XP:** Marathi. This is Unicode only.
0x0450	**Windows XP:** Mongolian
0x0414	Norwegian (Bokmal)
0x0814	Norwegian (Nynorsk)
0x0415	Polish

0x0416	Portuguese (Brazil)
0x0816	Portuguese (Portugal)
0x0446	**Windows XP:** Punjabi. This is Unicode only.
0x0418	Romanian
0x0419	Russian
0x044f	**Windows 2000/XP:** Sanskrit. This is Unicode only.
0x0c1a	Serbian (Cyrillic)
0x081a	Serbian (Latin)
0x041b	Slovak
0x0424	Slovenian
0x040a	Spanish (Spain, Traditional Sort)
0x080a	Spanish (Mexican)
0x0c0a	Spanish (Spain, Modern Sort)
0x100a	Spanish (Guatemala)
0x140a	Spanish (Costa Rica)
0x180a	Spanish (Panama)
0x1c0a	Spanish (Dominican Republic)
0x200a	Spanish (Venezuela)
0x240a	Spanish (Colombia)
0x280a	Spanish (Peru)
0x2c0a	Spanish (Argentina)
0x300a	Spanish (Ecuador)
0x340a	Spanish (Chile)
0x380a	Spanish (Uruguay)
0x3c0a	Spanish (Paraguay)
0x400a	Spanish (Bolivia)
0x440a	Spanish (El Salvador)
0x480a	Spanish (Honduras)
0x4c0a	Spanish (Nicaragua)
0x500a	Spanish (Puerto Rico)
0x0430	Sutu
0x0441	Swahili (Kenya)
0x041d	Swedish
0x081d	Swedish (Finland)
0x045a	**Windows XP:** Syriac. This is Unicode only.
0x0449	**Windows 2000/XP:** Tamil. This is Unicode only.
0x0444	Tatar (Tatarstan)
0x044a	**Windows XP:** Telugu. This is Unicode only.
0x041e	Thai
0x041f	Turkish
0x0422	Ukrainian
0x0420	**Windows 98/Me, Windows 2000/XP:** Urdu (Pakistan)
0x0820	Urdu (India)
0x0443	Uzbek (Latin)
0x0843	Uzbek (Cyrillic)
0x042a	**Windows 98/Me, Windows NT 4.0 and later:** Vietnamese

Meaning	Sublanguage identifier	Primary language identifier
Language neutral	SUBLANG_NEUTRAL	LANG_NEUTRAL
User default language	SUBLANG_DEFAULT	LANG_NEUTRAL
System default language	SUBLANG_SYS_DEFAULT	LANG_NEUTRAL