

# ***LangMan 1.2***

***Balíček lokalizačních komponent pro Delphi***

(support UNICODE)

MANUAL



**REGULACE.ORG**

**HW & SW Development**



## OBSAH

<b>1</b>	<b>Všeobecné informace</b>	<b>5</b>
1.1	Podporované verze Delphi	6
1.2	Instalace komponent LangMan	6
<b>2</b>	<b>Komponenty balíčku LangMan 1.2</b>	<b>9</b>
2.1	TLangManEngine	9
2.2	TLangManClient	9
2.3	TDesignedLexicon	9
2.4	TProgrammableLexicon	9
2.5	TLangCombo	10
2.6	TLangFlagsCombo	10
2.7	TValuedLabel	10
2.8	TLangManRichEdit	10
<b>3</b>	<b>Třídy balíčku komponent LangMan</b>	<b>11</b>
3.1	<b>TLangManEngine = class (TComponent)</b>	<b>11</b>
3.1.1	<b>Metody třídy TLangManEngine</b>	<b>11</b>
3.1.1.1	Function Translate ( LangName: TLanguage ): TLanguage;	11
3.1.1.2	Function GetLanguagesList: TStrings;	11
3.1.1.3	Function GetLangFilesList: TStrings;	11
3.1.1.4	Procedure ShowLangEditor;	11
3.1.1.5	Procedure ShowLangCreator;	11
3.1.2	<b>Vlastnosti třídy TLangManEngine</b>	<b>12</b>
3.1.2.1	Property CurrentLanguage: String; (read-only)	12
3.1.2.2	Property DesignLanguageName: TLanguage;	12
3.1.2.3	Property DefaultLanguage: TLanguage;	12
3.1.2.4	Property LangSubdirectory: String;	12
3.1.2.5	Property LangFileExtension: String;	12
3.1.2.6	Property LangFileSignature: String;	12
3.1.2.7	Property LangCreatorVisible: boolean;	13
3.1.2.8	Property LangEditorVisible: boolean;	13
3.1.2.9	Property TranslateLangMan: boolean;	13
3.1.2.10	Property LanguageMenu: TMenuItem;	13
3.1.2.11	Property LangMenuFlags: boolean;	13
3.1.2.12	Property DesignLangFlag: TPicture;	13
3.1.2.13	Property LangResources: TStringList;	13
3.1.2.14	Property LangFileEncoding: TLFEncoding;	13
3.1.3	<b>Události třídy TLangManEngine</b>	<b>14</b>
3.1.3.1	OnChangeLangQuery: TContinueQuery;	14
3.1.3.2	OnChangeLanguage: TNotifyEvent;	14
3.1.3.3	OnBeforeEdit: TNotifyEvent;	14
3.1.3.4	OnAfterEdit: TNotifyEvent;	14
3.2	<b>TLangManComponent = class (TComponent)</b>	<b>15</b>
3.2.1	<b>Vlastnosti třídy TLangManComponent</b>	<b>15</b>
3.2.1.1	Property LangManEngine: TLangManEngine;	15
3.2.2	<b>Události třídy TLangManComponent</b>	<b>15</b>
3.2.2.1	OnChangeLanguage: TNotifyEvent;	15
3.3	<b>TLangManClient = class (TLangManComponent)</b>	<b>16</b>
3.3.1	<b>Metody třídy TLangManClient</b>	<b>16</b>
3.3.1.1	Function AddComponent (Component: TComponent; Name: string; Translate:	

boolean): Boolean;.....	16
3.3.1.2 Procedure RecreateTransStruct;.....	16
3.3.1.3 Procedure TranslateComponent(Component: TComponent; Name: string = ");...)	17
3.3.1.4 Procedure Translate;.....	17
<b>3.3.2 Vlastnosti třídy TLangManClient.....</b>	<b>17</b>
3.3.2.1 Property InitAfterCreateForm: boolean;.....	17
3.3.2.2 Property TransStringProp: TTranslateStringProperties;.....	17
3.3.2.3 Property TransTStringsProp: TTranslateTStringsProperties;.....	17
3.3.2.4 Property TransStructuredProp: TTranslateStructuredProperties; .....	17
3.3.2.5 Property TransOtherProp: TTranslateOtherProperties; .....	17
3.3.2.6 Property TransAdditions: TAdditionSet;.....	17
<b>3.4 TLexicon = class (TLangManComponent).....</b>	<b>18</b>
<b>3.5 TDesignedLexicon = class (TLexicon).....</b>	<b>18</b>
<b>3.5.1 Metody třídy TDesignedLexicon.....</b>	<b>18</b>
3.5.1.1 Function CreateItem(Text: string): Integer;.....	18
3.5.1.2 Function CompleteString(const Str: string): string;.....	18
<b>3.5.2 Vlastnosti třídy TDesignedLexicon.....</b>	<b>18</b>
3.5.2.1 Property Item [Index: Integer]: string; (read-only).....	18
3.5.2.2 Property Items : TStringList;.....	19
3.5.2.3 Property Link [Index: Integer]: string; (read-only).....	19
<b>3.6 TProgrammableLexicon = class ( TLexicon ).....</b>	<b>19</b>
<b>3.6.1 Metody třídy TProgrammableLexicon.....</b>	<b>19</b>
3.6.1.1 Procedure DefineItem(ItemNr: Word; Text: string);.....	19
3.6.1.2 Function IsDefined(Index: Integer): Boolean;.....	19
3.6.1.3 Function CompleteString(const Str: string): string;.....	19
<b>3.6.2 Vlastnosti třídy TProgrammableLexicon.....</b>	<b>19</b>
3.6.2.1 Property Item [Index: Integer]: string; (read-only).....	19
3.6.2.2 Property Link [Index: Integer]: string; (read-only).....	20
<b>3.6.3 Události třídy TProgrammableLexicon.....</b>	<b>20</b>
3.6.3.1 OnInitialization: TNotifyEvent;.....	20
<b>3.7 TShadowComboBox = class ( TCustomComboBox );.....</b>	<b>21</b>
<b>3.8 TLangCombo = class ( TShadowComboBox ).....</b>	<b>21</b>
<b>3.8.1 Vlastnosti třídy TLangCombo.....</b>	<b>21</b>
3.8.1.1 Property LangManEngine: TLangManEngine;.....	21
3.8.1.2 Property StyleCombo: TLangComboStyle;.....	21
<b>3.8.2 Události třídy TLangCombo.....</b>	<b>21</b>
3.8.2.1 OnChangeLanguage: TNotifyEvent;.....	21
<b>3.9 TShadowComboBoxEx = class ( TCustomComboBoxEx );.....</b>	<b>22</b>
<b>3.10 TLangFlagsCombo = class ( TShadowComboBoxEx ).....</b>	<b>22</b>
<b>3.10.1 Vlastnosti třídy TLangFlagsCombo.....</b>	<b>22</b>
3.10.1.1 property LangManEngine: TLangManEngine;.....	22
<b>3.10.2 Události třídy TLangFlagsCombo.....</b>	<b>22</b>
3.10.2.1 OnChangeLanguage: TNotifyEvent;.....	22
<b>3.11 TValuedLabel = class ( TCustomLabel ).....</b>	<b>23</b>
<b>3.11.1 Vlastnosti třídy TValuedLabel.....</b>	<b>23</b>
3.11.1.1 Property Value: TCaption;.....	23
3.11.1.2 Property ValueName: TCaption;.....	23
3.11.1.3 Property ValueSeparator : string;.....	23
3.11.1.4 Property ValueSpaces : byte;.....	23
<b>3.12 TLangManStrings = class ( TStringList ).....</b>	<b>24</b>
<b>3.12.1 Konstruktor třídy TLangManStrings.....</b>	<b>24</b>
3.12.1.1 Constructor Create(ControlledStrings: TStrings; Lexicon: TLexicon);.....	24
<b>3.12.2 Metody třídy TLangManStrings.....</b>	<b>24</b>
3.12.2.1 Procedure Translate;.....	24

<b>3.13 TLangManRichEdit = class ( TCustomRichEdit )</b> .....	<b>25</b>
<b>3.13.1 Metody třídy TLangManRichEdit</b> .....	<b>25</b>
3.13.1.1 procedure AssignStyles(LMStringStyles: TLMStringStyles);.....	25
3.13.1.2 procedure ClearStyles;.....	25
3.13.1.3 function GetStyles: TLMStringStyles;.....	25
3.13.1.4 function StylesCount: Integer;.....	25
3.13.1.5 function SetStyle(Style: TFontStyles; Size: Integer = 0; Color: TColor = clDefault; FontName: TFontName = ""; Charset: TFontCharset = DEFAULT_CHARSET; Pitch: TFontPitch = fpDefault; StyleIndex: ShortInt = -1): Integer;.....	26
3.13.1.6 function Format(const Text: String; StyleIndex: ShortInt): String;.....	26
3.13.1.7 procedure Write(const Text: String; StyleIndex: ShortInt = -1);.....	26
3.13.1.8 procedure WriteLn(const Text: String; StyleIndex: ShortInt = -1);.....	26
3.13.1.9 procedure NextLine;.....	26
3.13.1.10 procedure Clear;.....	26
3.13.1.11 function LinesCount: Integer;.....	27
3.13.1.12 function ReadLineText(LineIndex: Integer): String;.....	27
3.13.1.13 function ReadLineFText(LineIndex: Integer): String;.....	27
3.13.1.14 procedure DeleteLine(LineIndex: Integer);.....	27
3.13.1.15 procedure RewriteLine(LineIndex: Integer; const Text: String; StyleIndex: ShortInt = -1);.....	27
3.13.1.16 procedure InsertLine(LineIndex: Integer; const Text: String; StyleIndex: ShortInt = -1);.....	27
3.13.1.17 procedure Translate;.....	27
3.13.1.18 procedure LoadFromFile(const SourceFile: TFileName; Encoding: TEncoding); .....	27
3.13.1.19 procedure LoadFromStream(SourceStream: TStream; Encoding: TEncoding); .....	28
3.13.1.20 procedure SaveRichTextToFile(const DestinationFile: TFileName; Encoding: TEncoding);.....	28
3.13.1.21 procedure SaveRichTextToStream(DestinationStream: TStream; Encoding: TEncoding);.....	28
3.13.1.22 procedure SaveEncodedFormToFile(const DestinationFile: TFileName; Encoding: TEncoding);.....	28
3.13.1.23 procedure SaveEncodedFormToStream(DestinationStream: TStream; Encoding: TEncoding);.....	28
<b>3.13.2 Vlastnosti třídy TLangManRichEdit</b> .....	<b>28</b>
3.13.2.1 property AssignedLexicon: TLexicon;.....	28
3.13.2.2 property AutoFont: Boolean;.....	29
3.13.2.3 Property Link [Index: Integer]: string; (read-only).....	29
<b>4 Jazykové soubory součástí EXE souboru</b> .....	<b>30</b>
<b>5 Dynamické generování textů</b> .....	<b>31</b>

## 1 Všeobecné informace

Balíček komponent **LangMan** slouží pro velice jednoduchou tvorbu multijazyčných aplikací v Delphi. Tyto komponenty vás doslova zprostí od všech starostí s programováním překladů, přepínání jazyků a zachovává plnou přehlednost ve zdrojovém kódu. Na rozdíl od jiných konkurenčních řešení je **LangMan** naprosto unikátním a neocenitelným pomocníkem.

Svou aplikaci tvoříte v libovolném základním jazyce bez ohledu na nutnost budoucího překladu do dalších jazyků. Pouze u řetězců přiřazovaných za běhu programu je nutné se odvolávat na lexikony, jež jsou součástí této série komponent. Při návrhu formulářů je také dobré dbát na proměnnou délku řetězců. Všechno ostatní je plně automatické, proto také **LangMan** při tvorbě multijazyčných aplikací ušetří obrovskou spoustu času.

Výsledné jazykové soubory lze distribuovat jednak současně s aplikací, ale hlavně i samostatně, čímž lze jednoduše opravit nebo přidat nový jazyk v cílové aplikaci u uživatele. Další výhodou **LangManu** je schopnost jazyky dědit a to bez omezení hloubky. Lze dědit jazyk z jazyka a ten z jiného jazyka a tak dále. Dokonce to vřele doporučujeme. Má to tu výhodu, že při opomenutí přeložení nějaké komponenty může například slovenská verze vycházet z české či naopak, nebo americká z britské, rakouská z německé apod. Zároveň to dovoluje překládat některé jazyky pouze částečně, čímž lze zase ušetřit spoustu času.

**LangMan** sám automaticky vytvoří funkční odkazy ve zvoleném menu pro výběr jazyka. Nebo lze pro uživatelskou volbu jazyka do aplikace vložit vizuální komponentu **TLangCombo** případně **TLangFlagsCombo**. To vše je absolutně bez práce a bez složitého nastavování. Jednotlivým jazykům lze přiřadit i grafický symbol či vlajku, která se automaticky zobrazí v menu s výběrem jazyka.

**LangMan** podporuje znakovou sadu UNICODE a umí do libovolného jazyka automaticky překládat **všechny standardní komponenty**, které jsou obsaženy v Delphi do verze 2009 (včetně). Toto platí i pro zděděné vlastnosti od standardních komponent Delphi, tj. pokud vývojář použije pro svou komponentu standardní prvek Delphi, budou automaticky překládány i zděděné vlastnosti jeho komponenty. Cizí nebo vlastní komponenty, které nevycházejí ze standardních komponent lze překládat ručně v události **OnChangeLanguage** pomocí lexikonů nebo lze v jednotce **LMAdditions** s minimální obtížností doprogramovat automatický překlad libovolných jiných komponent.

Pokud je překlad některých vlastností nežádoucí, existuje samozřejmě i několik možností, jak překládání zakázat. Nejdříve bych ještě podotkl, že je **LangMan** dost inteligentní a nezahrnuje do překladu žádné zbytečnosti, nýbrž pouze ty vlastnosti komponent, jejichž překlad je v 99% žádoucí. Do překladů nejsou zahrnovány žádné řetězce, které neobsahují text. Dále nejsou překládány řetězce shodné s názvem příslušné komponenty. Příklad konkrétních vlastností lze navíc v nastavení jednoduše zakázat. Každý formulář může mít jiné nastavení.

## 1.1 Podporované verze Delphi

- Delphi 7
- Delphi 2005
- Delphi 2006
- Delphi 2007
- **Delphi 2009**
- Delphi 2010
- Delphi XE
- Delphi XE2

Komponenty LangMan byly naprogramovány pod **Delphi 2009**. Pod touto verzí je i autor sám stále používá, takže jsou maximálně odladěné a nejsou s instalací žádné problémy. Problémy by neměly být ani ve vyšších verzích tj. **Delphi 2010**, **Delphi XE** ani **Delphi XE2**.

Postupem času jak se rozšiřovala řada uživatelů těchto komponent, přibývali i uživatelé starších verzí a komponenty tak postupně dostali úpravy pro bezproblémovou instalaci a používání ve starších vydání Delphi. Výše uvedený seznam je ale vytvořen pouze na základě zpětné vazby od uživatelů těchto verzí.

Teoreticky by měl jít **LangMan** nainstalovat i na **Delphi 5** a **6**, ale protože autor nemá žádnou zpětnou vazbu od uživatelů těchto dvou verzí, neuvádí je ani v seznamu podporovaných.

Po každé aktualizaci komponent **LangMan** (s každou novou verzí) může opět dojít k tomu, že si **starší verze Delphi** bude při instalaci na něco stěžovat. V takovém případě neváhejte kontaktovat autora, který vždy ve velmi krátkém čase provede potřebnou nápravu.

## 1.2 Instalace komponent LangMan

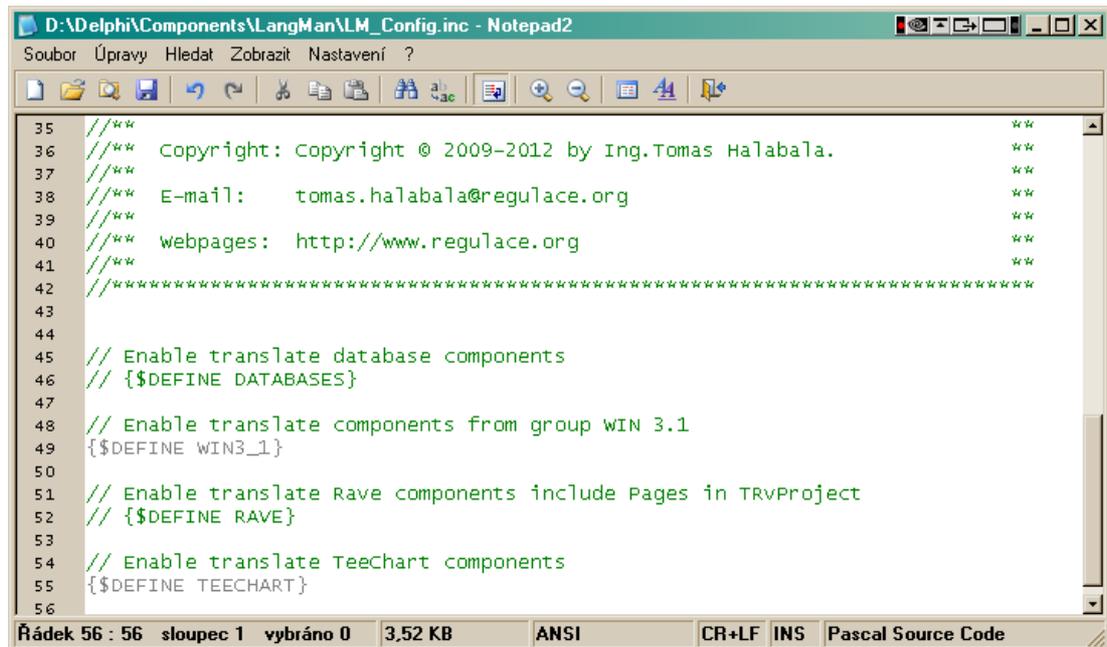
Po stažení balíčku komponent **LangMan** z internetu si nejdříve na pevném disku vytvořte složku, kam umístíte zdrojové kódy komponent **LangMan**. Složku vytvořte nejlépe vedle ostatních komponent, které jste už dříve instalovali nebo zkrátka na místo, kde soubory nebudou překážet, protože je tam budete muset od okamžiku nainstalování nechat po celou dobu jejich užívání.

Takže jsem si tuto složku vytvořil například v následujícím umístění:

```
D:\Delphi\Components\LangMan
```

Do této složky rozbalte a nakopírujte obsah ZIP archivu, staženého z internetu.

Následuje úprava souboru **LM\_Config.inc**. V tomto souboru najdete pár řádků s definicemi některých volitelných knihoven, které nejsou součástí všech verzí a všech edic **Delphi**. Proto u těch knihoven, které vaše **Delphi** neobsahuje musíte příslušné řádky v **LM\_Config.inc** zapoznámkovat. Takže například když vím, že moje **Delphi** neobsahuje knihovny **Rave Reports** a **databázové komponenty**, musím zapoznámkovat tyto řádky následujícím způsobem:



```

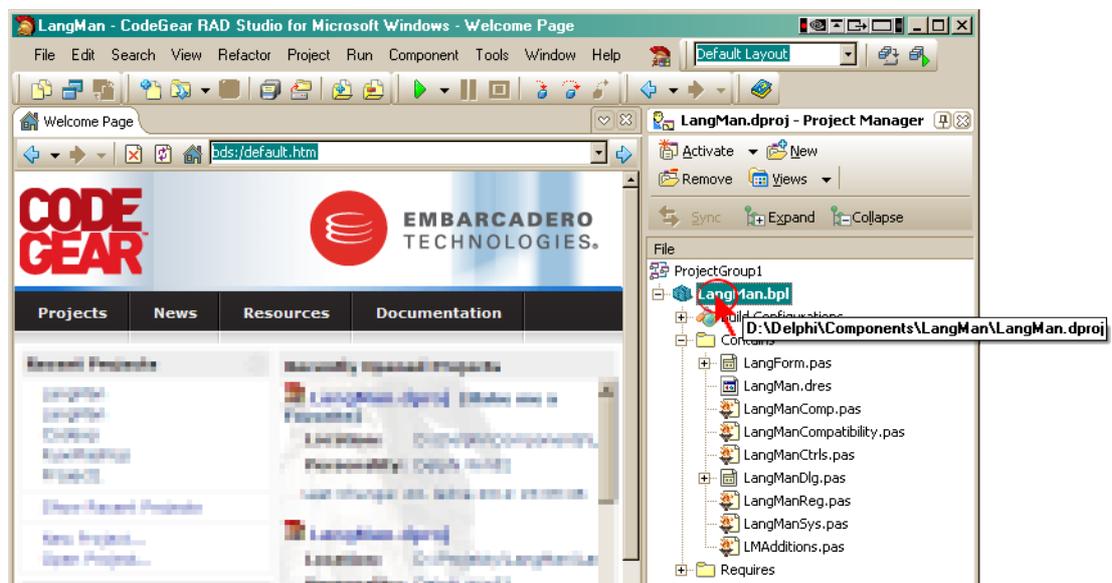
35 //**
36 //** Copyright: Copyright © 2009–2012 by Ing.Tomas Halabala.
37 //**
38 //** E-mail: tomas.halabala@regulace.org
39 //**
40 //** webpages: http://www.regulace.org
41 //**
42 //*****
43
44
45 // Enable translate database components
46 // {$DEFINE DATABASES}
47
48 // Enable translate components from group WIN 3.1
49 {$DEFINE WIN3_1}
50
51 // Enable translate Rave components include Pages in TRVProject
52 // {$DEFINE RAVE}
53
54 // Enable translate TeeChart components
55 {$DEFINE TEECHART}
56

```

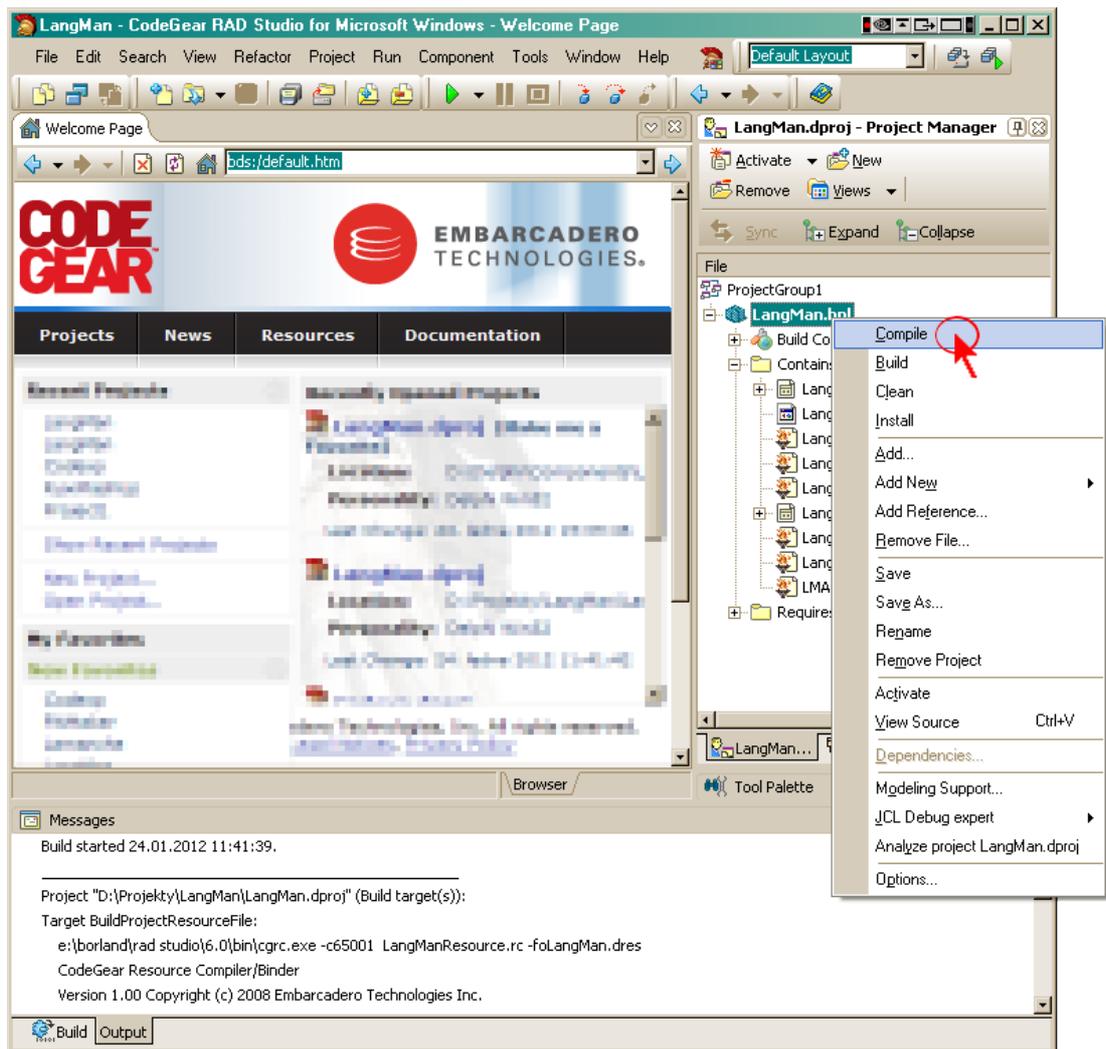
Pokud jste v souboru **LM\_Config.inc** provedli úpravy, nezapomeňte je na závěr uložit.

Následně spusťte své **Delphi** a v menu **File** klikněte na **Open Project**. V dialogovém okně vyberete a otevřete soubor **LangMan.dpk**. Dovolte případnou konverzi souboru projektu.

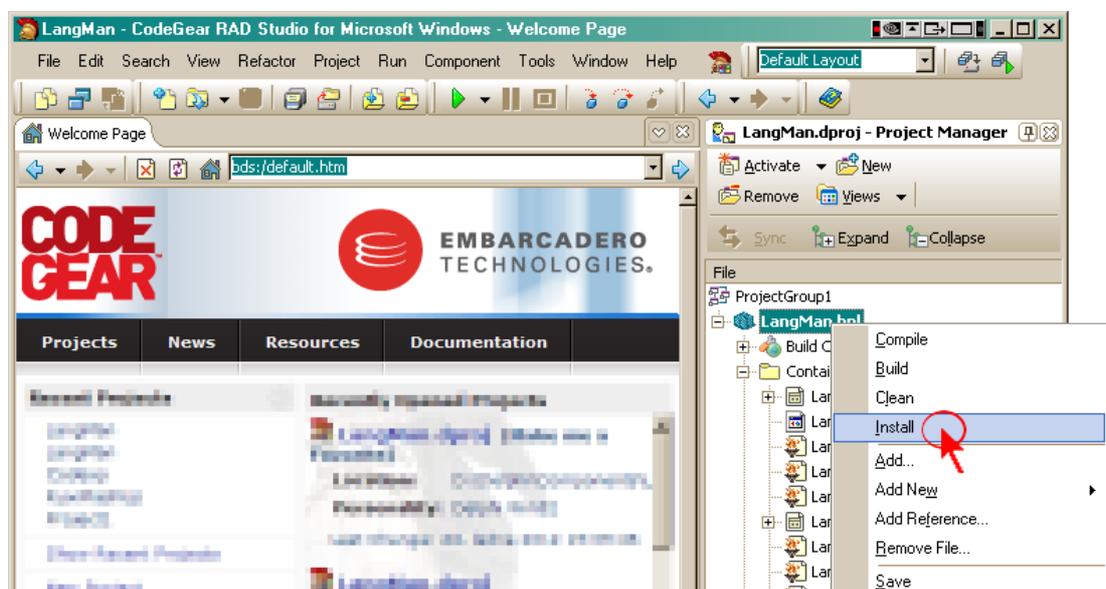
Poté klikněte **pravým** tlačítkem myši na název projektu v **Project Manageru**:



Zobrazí se místní nabídka, ve které klikněte na volbu **Compile**:



a nakonec klikněte ve stejné nabídce na volbu **Install**:



## 2 Komponenty balíčku LangMan 1.2

### 2.1 *TLangManEngine*

Základní stavební kámen, dále jen engine. Zprostředkovává překlad, přepínání jazyků a spravuje jazykové soubory. Tuto komponentu potřebují ke své funkci všechny ostatní jazykové komponenty kromě komponenty **TValuedLabel**. Ve většině případů je jeden jazyk společný pro všechny části programu, takže je zapotřebí umístit engine jedenkrát do té jednotky (na formulář nebo lépe do datového modulu **TDataModule**), která bude přístupná ze všech ostatních formulářů. Potřebujete-li ve svém programu jazykově oddělit dvě a více částí, například chcete-li mít možnost odděleně volit jazyk programu a odděleně jazyk tiskového výstupu, můžete použít tolik enginů kolik je zapotřebí. Jednotlivé enginy se musí lišit v nastavení vlastností **LangFileSignature**, **LangFileExtension** a **LangSubdirectory**. Pro rozlišení stačí jedna odlišnost v některé z uvedených vlastností. Tyto vlastnosti určují základní parametry jazykových souborů, které budou náležet danému jazykovému enginu.

### 2.2 *TLangManClient*

Tato komponenta, dále jen klient, slouží pro zavedení formuláře, na který je umístěna, do překladu vybraným enginem. Kromě toho umožňuje zvolit, které vlastnosti mají být na formuláři a jeho komponentách překládány. Pokud je žádoucí překládat některé vlastnosti jedním enginem a ostatní jiným enginem, lze k tomuto účelu umístit na jeden formulář více klientů a každému přiřadit jiný engine.

### 2.3 *TDesignedLexicon*

Kromě automatického překládání statických komponent, umístěných na formulářích, jsou v programech často zapotřebí řetězce pro různá dynamicky vytvářená dialogová okna, hlášení apod. K těmto účelům slouží lexikony, ve kterých lze potřebné řetězce nadefinovat. Ty jsou poté automaticky překládány, takže při čtení konkrétního záznamu je tento čten vždy už v příslušném jazyce. Řetězce v **TDesignedLexicon** se definují pomocí editoru přímo v object inspektoru.

### 2.4 *TProgrammableLexicon*

Položky programovatelného lexikonu mohou být na rozdíl od **TDesignedLexiconu** definovány až ve zdrojovém kódu programu například v metodě události **OnInitialization**. Tento lexikon najde své uplatnění v případech, kdy jsou jazykové řetězce známy až při běhu programu. Podmínkou při pořizování překladu je, aby se v lexikonu před editací jazyka nacházely všechny položky, které mají být automaticky překládány, jinak nebude možné provést jejich lokalizaci do jiných jazyků.

Další funkcí obou lexikonů je překládání dynamicky generovaných

obsáhlejších textů. Do těchto textů lze při dynamickém vytváření obsahu vkládat namísto jednotlivých řetězců pouze odkazy na řetězce lexikonu. K tomu slouží vlastnost **Link** lexikonu. Pomocí metody **CompleteString**, lze kdykoliv text obsahující tyto odkazy převést do aktuálně zvoleného jazyka.

## 2.5 *TLangCombo*

Jedná se o standardní **ComboBox**, který je po přiřazení k enginu automaticky naplněn existujícími jazyky. Volbou jazyka dojde k přeložení všech klientů a lexikonů. V případě, že je v jazykovém enginu povoleno vytváření či editace jazyků, je nabídka rozšířena i o tyto volby.

## 2.6 *TLangFlagsCombo*

**TLangFlagsCombo** je vylepšený **ComboBox**, jehož položky jsou doplněny o ikony jazyků a případných voleb pro vytvoření a editaci jazyků uživatelem. Kromě těchto ikon je funkce i vzhled shodný s **TLangCombo**.

## 2.7 *TValuedLabel*

**TValuedLabel** je doplňková komponenta podobná standardnímu **TLabel**. Místo vlastnosti **Caption** má vlastnosti **ValueName**, **ValueSeparator**, **ValueSpaces** a **Value**. Tyto jsou ve výsledném zobrazení pospojovány přičemž **ValueSpaces** udává počet mezer za **ValueSeparator**. Trik spočívá v tom, že **LangMan** u této komponenty překládá kromě **Hint** pouze vlastnost **ValueName**. V programu potom stačí zapisovat pouze hodnotu „**Value**“. Využití je z tohoto popisu asi jasné.

## 2.8 *TLangManRichEdit*

Komponenta **TLangManRichEdit** je nejnovější vizuální komponentou balíčku. Je potomkem třídy **TCustomRichEdit** a byla vyvinuta pro generování různých výpisů, logů, protokolů apod. s využitím různých stylů písma. Jedná se o obdobu třídy **TLangManStrings** (viz kapitoly 3.12 a 5 tohoto manuálu). Oproti **TLangManStrings** ale tato komponenta nabízí možnost vytvářet hezčí a přehlednější dokumenty, přičemž je možné dynamicky měnit jazyk obsaženého textu. Zásadní rozdíl oproti komponentě **TRichEdit** spočívá v nemožnosti úprav textu uživatelem prostřednictvím klávesnice. **TLangManRichEdit** má vlastnost **ReadOnly** skrytou a na pevně nastavenou na hodnotu **True**. Není přítomna ani vlastnost **Lines** a veškeré zapisování a úpravy textu mají být prováděny pouze pomocí příslušných metod třídy **TLangManRichEdit**. Výhodou této komponenty je, že si u ní můžete přednastavit sadu stylů (parametrů fontu) pro celý dokument a při zapisování jednotlivých řetězců a údajů už jen odkazujete na styl, který má být pro vypsání příslušného řetězce použit. Samozřejmostí je i možnost následného uložení výsledného dokumentu ve formátu RTF.

## 3 Třídy balíčku komponent LangMan

### 3.1 *TLangManEngine = class (TComponent)*

**Unit:** LangManComp;

Základní stavební kámen, dále jen engine. Zprostředkovává překlad, přepínání jazyků a spravuje jazykové soubory.

#### 3.1.1 Metody třídy TLangManEngine

##### 3.1.1.1 *Function Translate ( LangName: TLanguage ): TLanguage;*

Funkce přeloží všechny připojené komponenty do jazyka **LangName**. Předávaný parametr je typu string a musí odpovídat názvu některého z načtených jazyků. Proběhne-li v pořádku překlad, je volána událost **OnChangeLanguage**. Funkce vrací název aktuálního jazyka po přeložení.

##### 3.1.1.2 *Function GetLanguagesList: TStrings;*

Funkce vrací seznam načtených jazyků. Položky tohoto seznamu jsou platnými jazyky a lze je použít jako parametr funkce **Translate**.

##### 3.1.1.3 *Function GetLangFilesList: TStrings;*

Funkce vrací seznam jazykových souborů, které jsou v enginu načteny. Indexy položek seznamu odpovídají indexům seznamu **GetLanguagesList**.

##### 3.1.1.4 *Procedure ShowLangEditor;*

Procedura spouští editor jazyků. Jazyky v něm lze pouze upravovat, nikoliv však přidávat. Podmínkou spuštění editoru je přítomnost alespoň jednoho jazyka pro editaci. Výchozí jazyk návrhu se nepočítá.

**Poznámka:** Editor jazyků je dostupný i přímo v nabídce jazyků (Menu, Combo), pokud je povolen nastavením vlastnosti **LangEditorVisible** na hodnotu **true**.

##### 3.1.1.5 *Procedure ShowLangCreator;*

Tato procedura spouští editor jazyků v režimu přidání nového jazyka.

**Poznámka:** Přidání jazyka lze spustit i přímo z nabídky jazyků (Menu, Combo), pokud je tato funkce povolena nastavením vlastnosti **LangCreatorVisible** na hodnotu **true**.

### 3.1.2 Vlastnosti třídy TLangManEngine

#### 3.1.2.1 Property CurrentLanguage: String; (read-only)

Vlastnost **CurrentLanguage** vrací název aktuálně zvoleného jazyka. Tato vlastnost je pouze ke čtení. Změnu jazyka lze provést pomocí funkce **Translate**.

#### 3.1.2.2 Property DesignLanguageName: TLanguage;

Vlastnost slouží pro nastavení názvu návrhového jazyka. Jde tedy o hlavní jazyk, který je použit při návrhu aplikace. Tento jazyk lze použít kdykoliv jako výchozí jazyk překladu. Proto se doporučuje použít při návrhu aplikace angličtinu, která je nejvhodnější pro překlad do libovolného jiného světového jazyka.

#### 3.1.2.3 Property DefaultLanguage: TLanguage;

Výchozí jazyk, který má být nastaven po spuštění aplikace. V metodě události **OnCreate** formuláře lze do této vlastnosti zapsat například jazyk operačního systému nebo jazyk, použitý při posledním běhu aplikace. Tento výchozí jazyk se načítá až v okamžiku aktivace aplikace, tedy když už jsou vytvořeny všechny formuláře.

#### 3.1.2.4 Property LangSubdirectory: String;

Název podadresáře pro ukládání jazykových souborů. Za výchozí adresář je považován adresář, v němž se nachází spustitelný soubor aplikace. Bude-li tato vlastnost rovna prázdnému řetězci, budou jazykové soubory ukládány do stejného adresáře.

***Poznámka:** Od verze 1.1.7 lze pomocí této vlastnosti nastavit i úplnou absolutní adresu k jazykovým souborům. Tato možnost je užitečná pouze ve speciálních případech, kdy je zapotřebí jazykové soubory ukládat/načítat odjinud, než do/z adresářové struktury aplikace a tedy nelze použít relativní cestu.*

#### 3.1.2.5 Property LangFileExtension: String;

Koncovka jazykových souborů. Při vícenásobném použití komponenty **TLangManEngine** v jedné aplikaci je nutné nějak odlišit jazykové soubory příslušející této komponentě od ostatních jazykových souborů ostatních enginů. Koncovka jazykových souborů je jednou z vhodných možností odlišení.

#### 3.1.2.6 Property LangFileSignature: String;

Identifikační řetězec jazykových souborů, příslušejících tomuto enginu. Při vícenásobném použití komponenty **TLangManEngine** v jedné aplikaci je nutné nějak odlišit jazykové soubory příslušející této komponentě od ostatních jazykových souborů ostatních enginů. Identifikační řetězec je ukládán do jazykových souborů a měl by odlišovat jazykové soubory náležející různým enginům, ideálně i různým aplikacím.

### 3.1.2.7 *Property LangCreatorVisible: boolean;*

Tato vlastnost určuje, zda má být v nabídkách pro volbu jazyků viditelná i volba pro vytvoření nového jazyka.

### 3.1.2.8 *Property LangEditorVisible: boolean;*

Tato vlastnost určuje, zda má být v nabídkách pro volbu jazyků viditelná i volba pro úpravu jazyka. Tato volba není viditelná v případě, že v programu není zaveden žádný jiný jazyk než hlavní jazyk návrhu.

### 3.1.2.9 *Property TranslateLangMan: boolean;*

Vlastnost **TranslateLangMan** povoluje začlenění jazykového editoru do seznamu překládaných komponent. Jestliže nemá mít uživatel právo používat editor jazyků, tedy v případě, že jsou vlastnosti **LangEditorVisible** a **LangCreatorVisible** nastaveny na **false**, pozbývá smyslu překládání jazykového editoru. V tomto případě nechávejte tuto vlastnost na hodnotě **false**. Jazykový editor momentálně obsahuje tři interní jazyky, do kterých se umí sám přeložit automaticky. Jedná se o **češtinu**, **slovenštinu** a **angličtinu**.

### 3.1.2.10 *Property LanguageMenu: TMenuItem;*

Vlastnost **LanguageMenu** slouží pro automatické generování jazykového menu. Stačí přiřadit libovolnou komponentu třídy **TMenuItem** z hlavní nebo místní nabídky.

### 3.1.2.11 *Property LangMenuFlags: boolean;*

Tato vlastnost určuje, zda mají být v menu zobrazeny vlajky (symboly) jazyků. Toto platí pouze pro menu přidělené vlastností **LanguageMenu**.

### 3.1.2.12 *Property DesignLangFlag: TPicture;*

Grafický symbol jazyka návrhu. Ideálně vlajka země v níž je daný jazyk národním jazykem.

### 3.1.2.13 *Property LangResources: TStringList;*

Této vlastnosti lze v object inspectoru přiřadit ID názvy (resources ID) zdrojových jazykových souborů, přilinkovaných k exe souboru aplikace. Lze tedy pomocí této vlastnosti zabudovat vybrané jazyky pevně do aplikace. Pokud bude uživateli umožněna editace jazyků, bude jazykový soubor z resources před editací uložen na disk.

### 3.1.2.14 *Property LangFileEncoding: TLFEncoding;*

Kódování jazykových souborů. Výchozím nastavením je **Unicode**. Jestliže požadujete z jakéhokoli důvodu jiný typ kódování jazykových souborů, máte na výběr z následujících možností:

*type TLFEncoding = (Unicode, BigEndianUnicode, UTF8, ANSI);*

### 3.1.3 Události třídy TLangManEngine

#### 3.1.3.1 *OnChangeLangQuery: TContinueQuery;*

Událost volaná před změnou jazyka. Pokud je překlad do zvoleného jazyka nežádoucí, lze překlad zastavit zapsáním hodnoty **false** do návratového parametru **Continue**.

#### 3.1.3.2 *OnChangeLanguage: TNotifyEvent;*

Událost volaná po změně jazyka.

#### 3.1.3.3 *OnBeforeEdit: TNotifyEvent;*

Událost volaná před spuštěním jazykového editoru. Při editaci jazyka je nutné, aby byla v paměti nahrána struktura všech prvků, které mají být komponentou překládány. Mají-li být překládány například dynamicky vytvářené formuláře nebo dynamicky vytvářené komponenty formulářů, je třeba všechny tyto komponenty a formuláře alespoň dočasně v paměti vytvořit, aby mohly být přeloženy. To samé platí pro položky programovatelného lexikonu.

#### 3.1.3.4 *OnAfterEdit: TNotifyEvent;*

Po ukončení editoru jazyků je volána událost **OnAfterEdit**. V této události lze komponenty, které byly vytvořené dynamicky jen kvůli překladu, opět z paměti uvolnit. Při opakovaném vytvoření stejného dynamického formuláře se už už automaticky přeloží do aktuálního jazyka. Zatímco po vytvoření dynamické komponenty je nutné zavolat její dodatečné přeložení ručně.

## 3.2 *TLangManComponent = class (TComponent)*

Unit: LangManComp;

Třída **TLangManComponent** je základem pro všechny ostatní jazykové komponenty, jako je **TLangManClient**, **TDesignedLexicon** a **TProgrammableLexicon**.

### 3.2.1 Vlastnosti třídy TLangManComponent

#### 3.2.1.1 *Property LangManEngine: TLangManEngine;*

Vlastnosti **LangManEngine** musí být přiřazen **TLangManEngine**, který jazykové komponentě poskytuje data aktuálního jazyka.

### 3.2.2 Události třídy TLangManComponent

#### 3.2.2.1 *OnChangeLanguage: TNotifyEvent;*

Událost volaná po změně jazyka.

### 3.3 *TLangManClient = class (TLangManComponent)*

**Unit:** LangManComp;

Tato komponenta, dále jen klient, slouží pro zavedení formuláře, na který je umístěna, do překladu vybraným enginem. Kromě toho umožňuje zvolit, které vlastnosti mají být na formuláři a jeho komponentách překládány.

#### 3.3.1 *Metody třídy TLangManClient*

##### 3.3.1.1 *Function AddComponent (Component: TComponent; Name: string; Translate: boolean): Boolean;*

Požadujete-li, aby byly překládány i komponenty formuláře, které jsou vytvářené dynamicky za běhu programu, je nutné každou takovou komponentu dodatečně přidat do seznamu komponent formuláře, který udržuje komponenta **TLangManClient**. Podmínkou pro správnou funkci automatických překladů je, aby byl vlastníkem takových komponent formulář. V případě několika stejných dynamických komponent, které mají být i shodně přeloženy, lze použít společné jméno. Potom stačí zaregistrovat komponentu pomocí funkce **AddComponent** pouze jednou. Při znovuvytvoření stejné komponenty také není zapotřebí ji znovu registrovat, stačí použít stejné jméno jako při předchozí registraci. Podmínkou je, aby byla vždy každá dynamicky vytvářená komponenta, která má být překládána, shodně pojmenována. Funkce **AddComponent** pojmenování provádí za vás.

Předávacím parametrem **Component** musí být přidávaná komponenta, v parametru **Name** je možné předat nové jméno komponenty. Pokud už komponenta má své unikátní jméno přiřazeno ve vlastnosti **Component.Name**, použijte pro parametr **Name** prázdný řetězec. Poslední parametr **Translate** určuje, zda má být po přidání nové komponenty ihned proveden i překlad do aktuálního jazyka. Pro hromadný překlad celého formuláře lze použít metodu **Translate**.

Pro dodatečný překlad jedné komponenty formuláře lze použít metodu **TranslateComponent**.

##### 3.3.1.2 *Procedure RecreateTransStruct;*

Tato metoda provede znovusestavení seznamové struktury komponent formuláře. Do seznamu jsou zahrnuty všechny pojmenované komponenty, které v daném okamžiku patří formuláři, takže lze tuto metodu použít pro hromadné zavedení dynamicky vytvářených komponent formuláře do seznamu překládaných komponent. Po zrušení některé komponenty je tato metoda jedinou možností, jak odebrat zrušenou komponentu ze seznamu překládaných komponent. Přítomnost neexistující komponenty v překladovém seznamu má ale vliv pouze na přítomnost této komponenty v editoru jazyků, což je obvykle spíše žádoucí.

### 3.3.1.3 *Procedure TranslateComponent(Component: TComponent; Name: string = "");*

Metoda **TranslateComponent** slouží pro přeložení jedné komponenty do aktuálně zvoleného jazyka. Přitom nezáleží na tom, která komponenta je vlastníkem komponenty, předané v parametru **Component**. Naopak pro automatický překlad je bezpodmínečně nutné, aby byl vlastníkem formulář.

Podmínkou je, aby měla komponenta v rámci vlastníka své unikátní jméno. Pokud tomu tak není, lze předat nové jméno v parametru **Name**. V opačném případě ponechejte parametru **Name** prázdný řetězec.

### 3.3.1.4 *Procedure Translate;*

Přeloží formulář a všechny jeho pojmenované komponenty. Při změně jazyka enginu proběhne překlad automaticky, avšak po přidání většího počtu dynamicky vytvořených komponent lze tuto metodu použít pro dodatečné hromadné přeložení všech nových pojmenovaných komponent formuláře.

## 3.3.2 Vlastnosti třídy TLangManClient

### 3.3.2.1 *Property InitAfterCreateForm: boolean;*

Tato vlastnost určuje zda má dojít k sestavení seznamu automaticky překládaných komponent formuláře až po volání metody události **OnCreate** formuláře **true**, nebo ještě předtím **false**.

### 3.3.2.2 *Property TransStringProp: TTranslateStringProperties;*

Množina názvů vlastností typu string, které mají být překládány.

### 3.3.2.3 *Property TransTStringsProp: TTranslateTStringsProperties;*

Množina názvů vlastností typu **TStrings**, které mají být překládány.

### 3.3.2.4 *Property TransStructuredProp: TTranslateStructuredProperties;*

Množina názvů vlastností jiných typů, obvykle složitějších struktur, které mají být překládány.

### 3.3.2.5 *Property TransOtherProp: TTranslateOtherProperties;*

Množina názvů vlastností jiných typů, které mají být překládány. V některých případech se může jednat o názvy celých tříd.

### 3.3.2.6 *Property TransAdditions: TAdditionSet;*

Množina názvů uživatelských doplňků, které mají být překládány.

### 3.4 *T*Lexicon = class (*T*LangManComponent)

**Unit:** LangManComp;

Základní třída všech lexikonů. Zprostředkovává propojení lexikonu s enginem. Od verze 1.1 má **T**Lexicon definovány virtuální funkce pro překlad textů, obsahujících odkazy na řetězce lexikonu. Tyto odkazy generuje další virtuální funkce s názvem **GetLink**. Více o funkci překladu textů a odkazů v popisu konkrétních lexikonů.

### 3.5 *T*DesignedLexicon = class (*T*Lexicon)

**Unit:** LangManComp;

Kromě automatického překládání statických komponent, umístěných na formulářích, jsou v programech často zapotřebí řetězce pro různá dynamicky vytvářená dialogová okna, hlášení apod. K těmto účelům slouží lexikony, ve kterých lze potřebné řetězce nadefinovat. Ty jsou poté automaticky překládány, takže při čtení konkrétního záznamu je tento čten vždy už v příslušném jazyce. Řetězce v **T**DesignedLexicon se definují pomocí editoru přímo v object inspektoru.

#### 3.5.1 Metody třídy *T*DesignedLexicon

##### 3.5.1.1 *Function CreateItem(Text: string): Integer;*

Funkce **CreateItem** slouží pro přidání řetězce **Text** do lexikonu za běhu programu. Návrátovou hodnotou je pozice (**Index**) nového řetězce v lexikonu.

##### 3.5.1.2 *Function CompleteString(const Str: string): string;*

Funkce **CompleteString** vrací zadaný řetězec **Str**, u něhož jsou odkazy na řetězce lexikonu nahrazeny řetězci lexikonu v aktuálně zvoleném jazyce. Tyto odkazy generuje vlastnost **Link** lexikonu. Tato funkce najde své využití pro překlad dynamicky generovaných rozsáhlých textů. Místo konkrétních řetězců v konkrétním jazyce použijete při generování textu pouze odkazy na příslušné řetězce lexikonu a až v okamžiku výstupu na obrazovku, tiskárnu apod. řetězec s odkazy převedete do čitelné podoby. Tuto funkci využívá i nový nevizuální objekt balíčku **LangMan TLangManStrings** (viz popis třídy **TLangManStrings**) pro dynamický překlad objektů, odvozených od **TStrings**. Např. **TMemo**, **TRichEdit** apod.

#### 3.5.2 Vlastnosti třídy *T*DesignedLexicon

##### 3.5.2.1 *Property Item [Index: Integer]: string; (read-only)*

Vlastnost **Item** vrací řetězec z pozice **Index** v aktuálně zvoleném jazyce.

### 3.5.2.2 *Property Items : TStringList;*

Vlastnost **Items** slouží pro editaci položek lexikonu v object inspectoru. Řetězce je nutné do tohoto seznamu vkládat v jazyce návrhu, který odpovídá **DesignLanguageName** přiřazeného enginu. V programu lze potom tyto řetězce číst už v konkrétním zvoleném jazyce pomocí vlastnosti **Item**.

### 3.5.2.3 *Property Link [Index: Integer]: string; (read-only)*

Vlastnost **Link** vrací odkaz typu string na řetězec lexikonu na pozici **Index**. Takto získaný odkaz můžete vkládat do textu při dynamickém vytváření nějakých výpisů, logů apod. Pomocí funkce **CompleteString** (viz výše) pak převedete vzniklý text s odkazy do čitelné podoby v aktuálně zvoleném jazyce. Více o využití této funkce najdete také v popisu nevizuálního objektu **TLangManStrings** nebo u popisu vizuální komponenty **TLangManRichEdit**.

## 3.6 *TProgrammableLexicon = class ( TLexicon )*

**Unit: LangManComp;**

Položky programovatelného lexikonu mohou být na rozdíl od **TDesignedLexiconu** definovány až ve zdrojovém kódu programu například v metodě události **OnInitialization**. Tento lexikon najde své uplatnění v případech, kdy jsou jazykové řetězce známy až při běhu programu. Podmínkou při pořizování překladu je, aby se v lexikonu před editací jazyka nacházely všechny položky, které mají být automaticky překládány, jinak nebude možné provést jejich lokalizaci do jiných jazyků.

### 3.6.1 *Metody třídy TProgrammableLexicon*

#### 3.6.1.1 *Procedure DefineItem(ItemNr: Word; Text: string);*

Tato metoda zavede řetězec **Text** do lexikonu na pozici **ItemNr**. Pokud se už na dané pozici vyskytuje jiný řetězec, bude přepsán řetězcem novým.

#### 3.6.1.2 *Function IsDefined(Index: Integer): Boolean;*

Funkce vrací **true**, pokud je položka na pozici **Index** v lexikonu definována. V opačném případě vrací **false**.

#### 3.6.1.3 *Function CompleteString(const Str: string): string;*

Význam této funkce je stejný u obou lexikonů. Viz popis funkce **CompleteString** lexikonu **TDesignedLexicon**.

### 3.6.2 *Vlastnosti třídy TProgrammableLexicon*

#### 3.6.2.1 *Property Item [Index: Integer]: string; (read-only)*

Vlastnost **Item** vrací řetězec z pozice **Index** v aktuálně zvoleném jazyce.

### 3.6.2.2 *Property Link [Index: Integer]: string; (read-only)*

Stejné jako u vlastnosti **Link** lexikonu **TDesignedLexicon**.

## 3.6.3 Události třídy TProgrammableLexicon

### 3.6.3.1 *OnInitialization: TNotifyEvent;*

Tato událost je volána při inicializaci lexikonu. V metodě této události je možné definovat položky lexikonu.

### 3.7 *TShadowComboBox = class ( TCustomComboBox );*

**Unit:** LangManSys;

Deklarace stínové třídy pro **TCustomComboBox**. Významem je pouze rozlišení potomků **TShadowComboBox** od potomků **TCustomComboBox** kvůli automatickému překladu komponentou **LangManClient**. Položky **Items**, třídy založené na **TShadowComboBox**, nejsou **LangManem** překládány.

### 3.8 *TLangCombo = class ( TShadowComboBox )*

**Unit:** LangManComp;

Jedná se o standardní **ComboBox**, který je po přiřazení k enginu automaticky naplněn existujícími jazyky. Volbou jazyka dojde k přeložení všech klientů a lexikonů.

#### 3.8.1 *Vlastnosti třídy TLangCombo*

##### 3.8.1.1 *Property LangManEngine: TLangManEngine;*

Této vlastnosti musí být přiřazen **TLangManEngine**, se kterým má být **TLangCombo** propojen.

##### 3.8.1.2 *Property StyleCombo: TLangComboStyle;*

Tato vlastnost je obdobou vlastnosti **Style ComboBoxu** s tím rozdílem, že zde nelze nastavit styl **csDropDown**.

#### 3.8.2 *Události třídy TLangCombo*

##### 3.8.2.1 *OnChangeLanguage: TNotifyEvent;*

Událost volaná po změně jazyka.

### 3.9 *TShadowComboBoxEx = class ( TCustomComboBoxEx );*

**Unit:** LangManSys;

Deklarace stínové třídy pro **TCustomComboBoxEx**. Význam je stejný jako u **TShadowComboBox**. Slouží pro rozlišení potomků **TShadowComboBoxEx** od potomků **TCustomComboBoxEx** a to kvůli automatickému překladu komponentou **LangManClient**. Položky **Items**, třídy založené na **TShadowComboBoxEx**, nejsou **LangManem** překládány.

### 3.10 *TLangFlagsCombo = class ( TShadowComboBoxEx )*

**Unit:** LangManComp;

**TLangFlagsCombo** je vylepšený **ComboBox**, který má před názvy jazyků navíc zobrazeny i příslušné vlajky (ikony jazyků). Jinak je funkce obdobná jako u **TLangCombo**.

#### 3.10.1 *Vlastnosti třídy TLangFlagsCombo*

##### 3.10.1.1 *property LangManEngine: TLangManEngine;*

Této vlastnosti musí být přiřazen **TLangManEngine**, se kterým má být **TLangFlagsCombo** propojen.

#### 3.10.2 *Události třídy TLangFlagsCombo*

##### 3.10.2.1 *OnChangeLanguage: TNotifyEvent;*

Událost volaná po změně jazyka.

### 3.11 *TValuedLabel = class ( TCustomLabel )*

Unit: LangManCtrls;

**TValuedLabel** je doplňková komponenta podobná standardnímu **TLabel**. Místo vlastnosti **Caption** má vlastnosti **ValueName**, **ValueSeparator**, **ValueSpaces** a **Value**. Tyto jsou ve výsledném zobrazení pospojovány, přičemž **ValueSpaces** udává počet mezer za **ValueSeparator**. Trik spočívá v tom, že **LangMan** u této komponenty překládá kromě **Hint** pouze vlastnost **ValueName**. V programu potom stačí zapisovat pouze hodnotu „**Value**“.

#### 3.11.1 *Vlastnosti třídy TValuedLabel*

##### 3.11.1.1 *Property Value: TCaption;*

Této vlastnosti může být přiřazen libovolný řetězec. Výsledný řetězec, který je zobrazen touto vizuální komponentou má následující formát: '**ValueName**' + '**ValueSeparator**' + **ValueSpaces** x ' ' + '**Value**'. Automaticky je překládána pouze vlastnost **ValueName**.

##### 3.11.1.2 *Property ValueName: TCaption;*

Do překladů je zahrnována pouze tato vlastnost, které by měl být přiřazen název hodnoty **Value** v jazyce návrhu. Výsledný řetězec, který je zobrazen touto vizuální komponentou má následující formát: '**ValueName**' + '**ValueSeparator**' + **ValueSpaces** x ' ' + '**Value**'.

##### 3.11.1.3 *Property ValueSeparator : string;*

Této vlastnosti může být přiřazen libovolný řetězec. Výsledný řetězec, který je zobrazen touto vizuální komponentou má následující formát: '**ValueName**' + '**ValueSeparator**' + **ValueSpaces** x ' ' + '**Value**'. Automaticky je překládána pouze vlastnost **ValueName**.

##### 3.11.1.4 *Property ValueSpaces : byte;*

Tato vlastnost udává počet mezer mezi **ValueSeparator** a **Value**. Výsledný řetězec, který je zobrazen touto vizuální komponentou má následující formát: '**ValueName**' + '**ValueSeparator**' + **ValueSpaces** x ' ' + '**Value**'. Automaticky je překládána pouze vlastnost **ValueName**.

## 3.12 *TLangManStrings = class ( TStringList )*

**Unit:** LangManComp;

Třída **TLangManStrings** slouží pro snadný dynamický překlad objektů, odvozených od třídy **TStrings**, například různých seznamů, celých rozsáhlých textových souborů, vizuálních komponent **TMemo**, **TRichEdit** apod. V okamžiku, kdy v programu vypisujete například nějaký log do komponenty **TMemo** a chcete, aby po změně jazyka byl tento, za běhu programu vytvořený, výpis přeložen do zvoleného jazyka, poslouží vám k tomu objekt **TLangManStrings**, který se o vše postará za vás.

### 3.12.1 Konstruktor třídy TLangManStrings

#### 3.12.1.1 *Constructor Create(ControlledStrings: TStrings; Lexicon: TLexicon);*

Při vytváření objektu třídy **TLangManStrings** je třeba v parametrech konstrukturu předat následující:

V **ControlledStrings** musí být předán objekt odvozený od **TStrings**, který má být novým objektem spravován – z funkčního hlediska se jedná o nahrazení původního objektu nově vytvořeným objektem. Přičemž původní objekt samozřejmě existuje dál, ale slouží pouze pro výstup přeloženého textu ve zvoleném jazyce, kdežto u nového objektu se pracuje pouze z odkazy na řetězce lexikonu.

V parametru **Lexicon** musí být předán lexikon, jehož řetězce mají být pro překlad textů použity.

Od chvíle zavolání tohoto konstrukturu by už v programu nemělo být k textovému obsahu původního objektu typu **TStrings** přístupováno jinak než prostřednictvím tohoto objektu, který je odvozen od třídy **TStringList**, která je také potomkem třídy **TStrings**.

Jeho metody **Add**, **AddObject**, **Insert**, **InsertObject**, **Delete**, **Clear**, **Exchange**, **Sort**, **CustomSort** mají stejnou funkci a stejný vliv na původní objekt jako by se jednalo o metody samotného objektu.

### 3.12.2 Metody třídy TLangManStrings

#### 3.12.2.1 *Procedure Translate;*

Tato procedura provede znovu vypsání celého textu spravovaného objektu v právě zvoleném jazyce. Po změně jazyka příslušného **TLangManEnginu** je tato metoda zavolána automaticky, ale může být zapotřebí například v případě, kdy za běhu programu změníte nějakou položku souvisejícího lexikonu a chcete, aby se změna okamžitě promítla i do příslušného textu.

### 3.13 *TLangManRichEdit = class ( TCustomRichEdit )*

**Unit:** LangManComp;

Jedná se o vizuální komponentu vyvinutou pro stejný účel jako je třída **TLangManStrings**. Tedy pro generování různých výpisů, logů, protokolů apod. ale zde navíc s možností text formátovat. Můžete vkládat řetězce s různými typy písma, různou velikostí znaků, barev, stylů atd. Přičemž jazyk výsledného dokumentu odpovídá právě zvolenému jazyku a při změně jazyka se celý dokument automaticky přeloží.

**TLangManRichEdit** má vlastnost **ReadOnly** skrytou a na pevně nastavenou na hodnotu **True**. Není přítomna ani vlastnost **Lines** a veškeré zapisování a úpravy textu mají být prováděny pouze pomocí níže popsaných metod.

#### 3.13.1 *Metody třídy TLangManRichEdit*

##### 3.13.1.1 *procedure AssignStyles(LMStringStyles: TLMStringStyles);*

Lze použít pro přímé přiřazení pole typu **TLMStringStyles** s definicemi všech stylů písma pro váš dokument. Typ **TLMStringStyles** je definován následovně:

```
TLMStringStyles = array of TLMStringStyle;
```

kde

```
TLMStringStyle = record  
    Color: TColor;  
    Font: TFontName;  
    Charset: TFontCharset;  
    Style: TFontStyles;  
    Size: Integer;  
    Pitch: TFontPitch;  
end;
```

##### 3.13.1.2 *procedure ClearStyles;*

Maže pole s definicemi stylů písma.

##### 3.13.1.3 *function GetStyles: TLMStringStyles;*

Vrací pole definic stylů písma.

##### 3.13.1.4 *function StylesCount: Integer;*

Vrací počet definovaných stylů písma.

**3.13.1.5** *function SetStyle(Style: TFontStyles; Size: Integer = 0; Color: TColor = clDefault; FontName: TFontName = ""; Charset: TFontCharset = DEFAULT\_CHARSET; Pitch: TFontPitch = fpDefault; StyleIndex: ShortInt = -1): Integer;*

Nastavuje styl písma. Každý styl písma vychází z nastavení vlastnosti **Font**. Kromě prvního parametru **Style** jsou všechny ostatní parametry nepovinné. Zadání výchozích hodnot parametrů znamená, že se použije příslušná hodnota z vlastnosti **Font**. Tedy pro **Size=0** bude pro příslušný styl písma použita hodnota **Font.Size** atd. Posledním parametrem je index stylu. Pro výchozí hodnotu **-1** se tento styl přidá jako nový na konec pole stylů. Pokud bude parametr nastaven na hodnotu indexu již existujícího stylu, provede se pouze modifikace příslušného stylu.

Funkce vrací index toho stylu, který byl zavoláním funkce nastaven.

**3.13.1.6** *function Format(const Text: String; StyleIndex: ShortInt): String;*

Tato funkce formátuje řetězec **Text** dle přednastaveného stylu na indexu **StyleIndex**. Výsledek této funkce může být následně použit pro zapsání do dokumentu pomocí metod **Write**, **WriteLn**, **RewriteLine** a **InsertLine**.

**3.13.1.7** *procedure Write(const Text: String; StyleIndex: ShortInt = -1);*

Zapíše řetězec **Text** na konec dokumentu. Pokud v parametru **StyleIndex** předáte proceduře nezápornou hodnotu, bude pro neformátované části řetězce použit vybraný styl. Další dílčí úseky řetězce **Text** mohou mít samozřejmě nastaveny i jiné styly pomocí funkce **Format**.

Poznámka: Funkce **Format** může být volána i vnořeně pro jednotlivé podčásti řetězců.

Například výsledek následujícího

```
Write('Ahoj ' + Format('celý ' + Format('světe',3) + '!',2),1);
```

může vypadat třeba takto: Ahoj **celý světe!**

**3.13.1.8** *procedure WriteLn(const Text: String; StyleIndex: ShortInt = -1);*

Prakticky stejné jako **Write**, pouze je text zakončen skokem na nový řádek.

**3.13.1.9** *procedure NextLine;*

Zapíše pouze skok na nový řádek.

**3.13.1.10** *procedure Clear;*

Maže celý dokument.

### 3.13.1.11 *function LinesCount: Integer;*

Vrací počet řádků dokumentu. Na hodnotu nemá vliv automatické zalomování při **WordWrap** = true.

### 3.13.1.12 *function ReadLineText(LineIndex: Integer): String;*

Vrací řetězec na řádku **LineIndex**. Pro první řádek je **LineIndex** = 0. Automaticky zalomený řádek je vrácen celý a je počítán jako jeden řádek.

### 3.13.1.13 *function ReadLineFText(LineIndex: Integer): String;*

Vrací řádek **LineIndex** v surovém interním formátu včetně pomocných znaků formátování a odkazů na lexikon. Tato data jsou vhodná například pro úpravu a následné vrácení na příslušný řádek metodou **RewriteLine**.

### 3.13.1.14 *procedure DeleteLine(LineIndex: Integer);*

Vymaže řádek s indexem **LineIndex**. První řádek má vždy index 0.

### 3.13.1.15 *procedure RewriteLine(LineIndex: Integer; const Text: String; StyleIndex: ShortInt = -1);*

Nahradí řádek **LineIndex** novým řetězcem **Text**. Stejně jako u metod **Write**, **WriteLn** lze definovat i styl pro daný řádek.

### 3.13.1.16 *procedure InsertLine(LineIndex: Integer; const Text: String; StyleIndex: ShortInt = -1);*

Na řádek **LineIndex** vloží nový řetězec **Text**. Stejně jako u **Write**, **WriteLn**, **RewriteLine** lze definovat styl (**StyleIndex**) pro daný řádek.

### 3.13.1.17 *procedure Translate;*

Tato procedura provede znovu vypsání celého dokumentu v právě zvoleném jazyce. Po změně jazyka příslušného **TLangManEnginu** je tato metoda zavolána automaticky, ale může být zapotřebí například v případě, kdy za běhu programu změníte nějakou položku lexikonu **AssignedLexicon** a chcete, aby se změna okamžitě promítla i v dokumentu.

### 3.13.1.18 *procedure LoadFromFile(const SourceFile: TFileName; Encoding: TEncoding);*

Načte text komponenty ze souboru. Tento text může obsahovat formátovací značky a odkazy na lexikon, takže se ve výsledku dokument zobrazí ve zvoleném jazyce. **Encoding** je nepovinný parametr, kterým lze určit kódování zdrojového souboru.

### 3.13.1.19 *procedure LoadFromStream(SourceStream: TStream; Encoding: TEncoding);*

Načte text komponenty ze streamu **SourceStream**. Tento text může obsahovat formátovací značky a odkazy na lexikon, takže se ve výsledku dokument zobrazí ve zvoleném jazyce. **Encoding** je nepovinný parametr, kterým lze určit kódování zdrojového textu.

### 3.13.1.20 *procedure SaveRichTextToFile(const DestinationFile: TFileName; Encoding: TEncoding);*

Uloží celý dokument ve formátu RTF do souboru **DestinationFile** s kódováním dle parametru **Encoding**. Tento parametr je nepovinný.

### 3.13.1.21 *procedure SaveRichTextToStream(DestinationStream: TStream; Encoding: TEncoding);*

Uloží celý dokument ve formátu RTF do streamu **DestinationStream** s kódováním dle parametru **Encoding**. Tento parametr je nepovinný.

### 3.13.1.22 *procedure SaveEncodedFormToFile(const DestinationFile: TFileName; Encoding: TEncoding);*

Uloží celý dokument v prostém textu s formátovacími značkami a odkazy na řetězce lexikonu do souboru **DestinationFile** s kódováním dle parametru **Encoding**. Parametr **Encoding** je nepovinný. Soubor uložený touto metodou, lze do komponenty opět načíst pomocí metody **LoadFromFile** se zachováním funkčnosti automatických překladů.

### 3.13.1.23 *procedure SaveEncodedFormToStream(DestinationStream: TStream; Encoding: TEncoding);*

Zapíše celý dokument v prostém textu s formátovacími značkami a odkazy na řetězce lexikonu do streamu **DestinationStream** s kódováním dle parametru **Encoding**. Parametr **Encoding** je nepovinný. Stejná data streamu lze použít pro zpětné načtení dokumentu do komponenty pomocí metody **LoadFromStream**. Formátování a funkčnost automatických překladů dokumentu tak bude plně zachována.

## 3.13.2 Vlastnosti třídy TLangManRichEdit

### 3.13.2.1 *property AssignedLexicon: TLexicon;*

Této vlastnosti je nutné přiřadit lexikon, jehož řetězce budou používány při tvorbě dokumentu. Pouze řetězce vložené do dokumentu pomocí vlastnosti **Link** zvoleného lexikonu nebo přímo vlastností **Link** této komponenty budou automaticky vloženy ve zvoleném jazyce a při změně jazyka automaticky v dokumentu přepsány.

### 3.13.2.2 *property AutoFont: Boolean;*

Nastavením této vlastnosti na hodnotu **false**, můžete vypnout automatickou změnu fontu. Tato funkce je standardně zapnutá i u komponenty **TRichEdit**. Některé české znaky s diakritikou ovšem způsobují nežádoucí automatickou změnu fontu i když to ve skutečnosti není zapotřebí. Proto se tvůrce komponent **LangMan** rozhodl přidat vlastnost **AutoFont**, která umožní snadno tuto nežádoucí funkci vypnout.

### 3.13.2.3 *Property Link [Index: Integer]: string; (read-only)*

Vlastnost **Link** vrací odkaz typu string na řetězec **Index** lexikonu **AssignedLexicon**. Po zapsání takto získaného odkazu do dokumentu dojde k vypsání příslušného řetězce lexikonu v aktuálně zvoleném jazyce a po přepnutí jazyka se všechny takto vložené úseky textu automaticky přepíše v nově zvoleném jazyce.

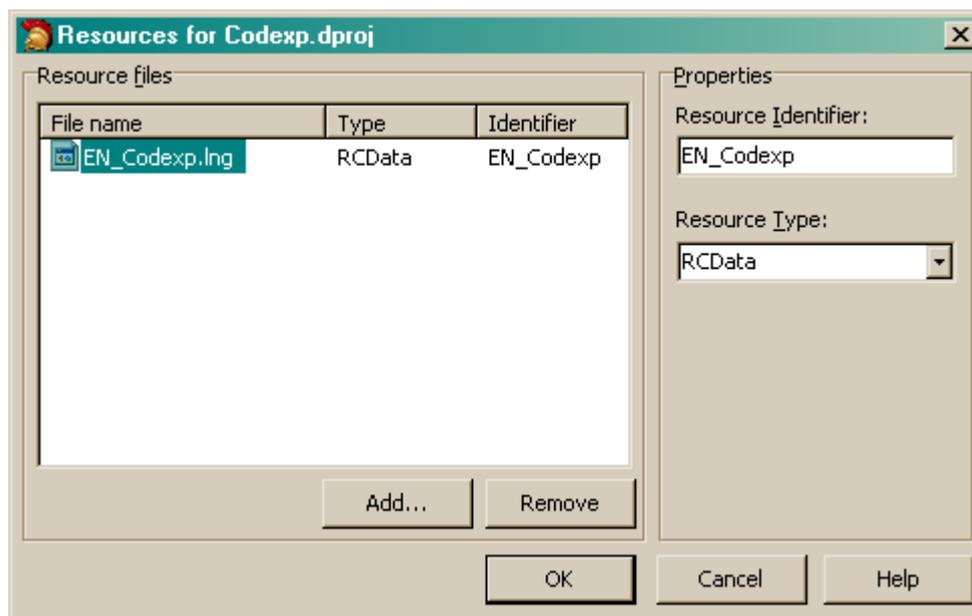
Následující příklad demonstruje vložení řetězce č.1 z asociovaného lexikonu do dokumentu této komponenty. Předpokladem je správně nastavená vlastnost **AssignedLexicon**.

```
Write(Link[1]);
```

## 4 Jazykové soubory součástí EXE souboru

Od verze 1.1 komponent **LangMan** můžou být jazykové soubory součástí spustitelného EXE souboru aplikace. Na následujících řádcích popíšu jak na to v prostředí IDE **Delphi 2009**.

Přes menu **Project / Resources...** otevřete dialogové okno pro editaci zdrojů (resources), které mají být nezávisle na programu aplikace přilinkovány do výsledného EXE souboru.



Obr. 4.1: Dialogové okno pro editaci zdrojů (resources)

Pomocí tlačítka přidat (**Add...**), vložte platný jazykový soubor a v poli **Resource Identifier** zadejte svůj název pro přidání jazyka. Tento název volte pokud možno výstižně a s vědomím, že v případě editace interního jazyka uživatelem, bude tento jazyk z EXE souboru exportován na disk pod stejným názvem a s koncovkou přidělenou dle vlastnosti **LangFileExtension** příslušného jazykového enginu. **Resource Type** ponechte na **RCData**.

Po kliknutí na tlačítko **OK** a sestavení programu (Build) se popsaným způsobem vybrané jazykové soubory přilinkují do EXE souboru vaší aplikace. Při každém dalším sestavení programu se použije vždy aktuální obsah jazykových souborů, takže se nemusíte obávat žádných starostí o neustálé importování jazykových souborů po každé změně – nic takového.

Na způsobu tvorby a úprav jazykových souborů se nic nemění. Kdykoli můžete použít vestavěný editor jazyků **LangMan** a můžete se spolehnout, že hned při následné kompilaci bude každá úprava do nového EXE souboru zakomponována.

Nyní je ale ještě zapotřebí zadat příslušnému jazykovému enginu (**TLangManEngine**) do vlastnosti **LangResources**, které interní jazyky z resources má načítat. Po rozkliknutí **LangResources** v object inspectoru se zobrazí **String List Editor**. Na jednotlivé řádky nyní vypište názvy interních

jazykových souborů (**Resource Identifier**), které jste ke každému jazykovému souboru přiřadily v dialogovém okně **Resources**.

## 5 Dynamické generování textů

Generuje-li váš program za chodu nějaký výpis, log soubor apod., může do tohoto souboru vkládat příslušné řetězce z lexikonu, který je vypisuje vždy v aktuálně zvoleném jazyce. Tento postup má ale jednu slabinu. Při následné změně jazyka, zůstane dosavadní obsah výpisu v původním jazyce, zatímco pokračování výpisu bude v jazyce změněném. Za normálních okolností asi nemá uživatel programu žádný důvod volit jiný jazyk, než ten který si zvolil na začátku a který mu nejlépe vyhovuje. Jsou ale případy, kdy je změna jazyka za chodu programu žádoucí a je i žádoucí, aby se změnil jazyk u celého výpisu od začátku ne jenom u části, která v čase následuje.

Pro tyto případy byla vytvořena třída **TLangManStrings**, vizuální komponenta **TLangManRichEdit** a do lexikonů byla přidána metoda **CompleteString** a vlastnost **Link**.

### Příklad použití třídy **TLangManStrings**(**TMemo**):

Máte-li do své aplikace například vloženou vizuální komponentu **TMemo**, do které program generuje za chodu nějaký výpis, a požadujete, aby se jazyk celého výpisu dal v kterémkoli okamžiku změnit, použijte k tomu třídu **TLangManStrings**. Postup je následující:

V rutině události **OnCreate** formuláře vytvoříte objekt např. **Memo**:

```
Memo := TLangManStrings.Create(Memo1.Lines, MainLexicon);
```

kde **Memo1** je vizuální komponenta **TMemo** a **MainLexicon** je jazykový lexikon (**TDesignedLexicon** nebo **TProgrammableLexicon**), který obsahuje všechny potřebné jazykové řetězce pro dynamické generování vašeho výpisu.

Tento krok lze považovat za jakési nahrazení vlastnosti **Memo1.Lines** novým objektem **Memo**. Takže od této chvíle musíte veškerý textový obsah vlastnosti **Lines** měnit výhradně prostřednictvím nového objektu **Memo** třídy **TLangManStrings**! Jakékoli manipulace s řetězcí ve vlastnosti **Memo1.Lines** tedy provádíte výhradně přes objekt **Memo** a to úplně stejně, jako byste pracovali přímo s **Memo1.Lines**. Například nový řádek vložíte následovně:

```
Memo.Add('Text nového řádku');
```

Výsledkem bude přidání řetězce **'Text nového řádku'** na nový řádek vlastnosti **Memo1.Lines**.

Nyní se dostáváme k tvorbě přeložitelného textu pomocí lexikonu **MainLexicon**, který byl v konstruktoru objektu **Memo** předán jako druhý

parametr. Takže například pokud se na položce s **indexem 2** lexikonu **MainLexicon** nachází řetězec **'Můj text'** (aktuální jazyk je čeština), mohli byste vložit tento řetězec přímo následujícím způsobem:

```
Memo.Add(MainLexicon.Item[2]);
```

Efekt by byl úplně stejný jako v tomto případě:

```
Memo.Add('Můj text');
```

Jedinou výhodou by bylo, že je právě vložený řetězec v aktuálně zvoleném jazyce. V okamžiku, kdy by ale uživatel změnil jazyk u příslušného enginu, již vložený text by se nezměnil a zůstal by v češtině.

Pokud byste ale řetězec vložily následujícím způsobem pomocí odkazu (**Link**):

```
Memo.Add(MainLexicon.Link[2]);
```

bude v daném okamžiku v komponentě **Memo1** na novém řádku vypsán sice stejný text jako v předešlém případě, ale s tím rozdílem, že pokud by uživatel změnil jazyk, převede se automaticky text na posledním řádku **Memo1.Lines** také do nově zvoleného jazyka.

Stejně jako metoda **Add** fungují bez rozdílu od původních metod třídy **TStrings** i další metody jako jsou **AddObject**, **Insert**, **InsertObject**, **Delete**, **Clear**, **Exchange**, **Sort** a **CustomSort** třídy **TLangManStrings**. Pouze je nutné dávat pozor na to, že výsledné přidělení indexů řetězcům vlastnosti **Lines** komponenty **TMemo** se může od objektu třídy **TLangManStrings** lišit. Například, budete-li mít nastaveno u komponenty **TMemo** automatické zalamování řádků, může se tím počet řádků o počet zalomení zvýšit oproti počtu řádků v **TLangManStrings**. I proto je důležité opravdu dbát na to, aby se k textu původního objektu přistupovalo pouze přes náhradní objekt typu **TLangManStrings**.

Odměnou je velmi efektní na čase nezávislá možnost změny jazyka u rozsáhlých textů **TStrings** a opět, jak je tomu u komponent **LangMan** zvykem, bez nároků na čas programátora, který by jinak musel pro dosažení stejného výsledku jít mnohem složitější a pracnější cestou.

Před ukončením programu nezapomeňte uvolnit paměť, přidělenou objektu **Memo** pomocí metody **Free**:

```
Memo.Free;
```

## Revize dokumentu

Datum	Revize	Poznámka
11.2.2012	05	Vydání LangMan verze 1.2.1
24.1.2012	04	Vydání LangMan verze 1.2.0
15.4.2011	03	Vydání LangMan verze 1.1.8
15.8.2010	02	Vydání LangMan verze 1.1.1
17.2.2010	01	Vydání LangMan verze 1.1.0

## **Informace o výrobci**

**Ing. Tomáš Halabala – REGULACE.ORG**

Slunná 848, Luhačovice

CZ-76326

Czech Republic

**Tel.:** +420 728 677 659

**E-mail:** [info@regulace.org](mailto:info@regulace.org)

**Web:** <http://www.regulace.org>