LangLib instructions (MultiDB)

Table of Contents

Differences between LangLib

The library itself supports SQLite database as default so no code changes are required except for the *DBLangEngine.ShoudConfigure()* and the *DBLangEngine.RunConfigurator();*.

These mentioned additions are required only if you wish to use MySQL server, PostgreSQL server or Microsoft SQL server database engines.

The database configuration software (SecureDatabaseSetting) is included in both of the release and the source package of the library.

The SecureDatabaseSetting adapts to the calling software:

- Uses the icon of the calling software.
- Mentions the calling software product name in it's title bar, so no code changes are required for this little piece of program.

Command line parameter changes

The *–dbLang* or *–dbLang=[culture],* e.g. *--dbLang=fi-FI* command line parameter causes the program to "dump" the language items into a configured or not configured database. This will also cause the library to "dump" a list of cultures into a database so the library won't insert a culture list into the database unless this command line parameter is defined.

A command line parameter *--configureLang* causes the library to execute the SecureDatabaseSetting software to configure a database.

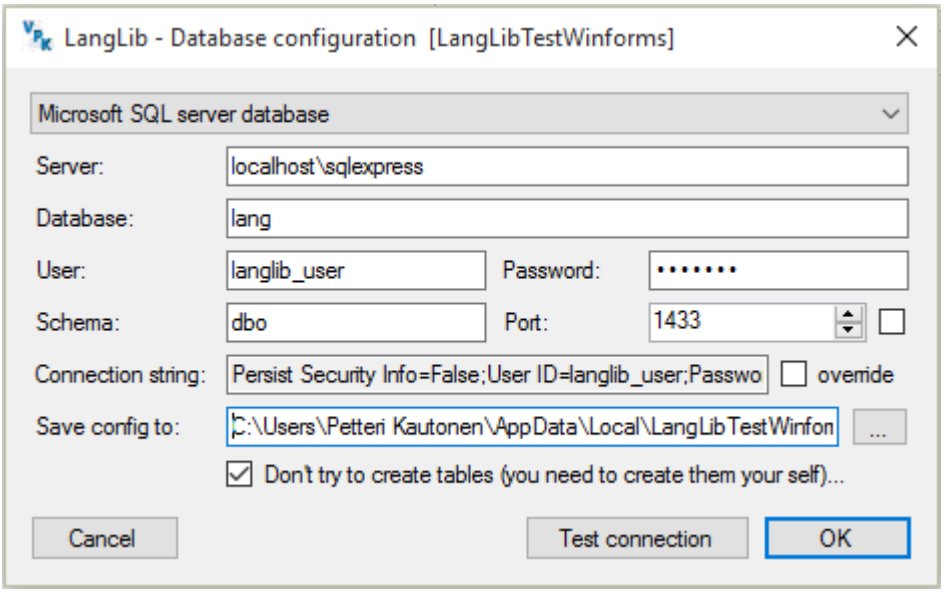
The SecureDatabaseSetting

This program should be located in the same directory as your product's executable. This configures a database connection.

The only thing you shouldn't change in this window is the *"Save config to:"* option, as the LangLib automatically set the path on application initialization.

The "*Don't try to create tables...*" option prevents the LangLib to try to create non-existent database tables on application start. Manual creation scripts are included in the downloads.

You should test the connection before accepting the database configuration.



The logic how this LangLib MultiDB chooses it's database settings is as follows:

- If there is no file called *[...]\AppData\Local\[Assembly product name]\dbconfig.sqlite* the library resets to default SQLite connection, otherwise the connection parameters are read from the previously mentioned file and a database connection is based on those settings in the file.

Other differences

The size of the library is of course larger as 3 more databases are supported. I won't be killing the original LangLib library with only SQlite database support.

The DBLocalization software supports these new databases and a script can be made to share these localizations through all the four supported databases.

The rest is quite the same as with the SQLite-only LangLib.

Windows Forms

Inheritance from a System.Windows.Forms.Form should be changed as follows:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Reflection;
using VPKSoft.LangLib;

namespace LangLibTestWinforms
{
    public partial class FormMain : DBLangEngineWinforms
    {
        public FormMain()
        {
            InitializeComponent();

            DBLangEngine.DBName = "LangLibTestWinforms.sqlite";

            if (Utils.ShouldLocalize() != null)
            {
                DBLangEngine.InitalizeLanguage("LangLibTestWinforms.Messages",
Utils.ShouldLocalize(), false);
                return; // After localization don't do anything more.
            }


            DBLangEngine.InitalizeLanguage("LangLibTestWinforms.Messages");
```

WPF / XAML

With WPF the inheritance is a bit more complex procedure, but once done it stays..

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Reflection;
using VPKSoft.LangLib;

namespace LangLibTestWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : VPKSoft.LangLib.DBLangEngineWPF
    {
        public MainWindow()
        {
            InitializeComponent();

            DBLangEngine.DBName = "LangLibTestWPF.sqlite";
            if (Utils.ShouldLocalize() != null)
            {
                DBLangEngine.InitalizeLanguage("LangLibTestWPF.Messages", Utils.ShouldLocalize(),
false);
                return; // After localization don't do anything more.
            }

            DBLangEngine.InitalizeLanguage("LangLibTestWPF.Messages");
```

After this we need to modify the xaml:

```
<VPKSoft:DBLangEngineWPF x:Name="WindowMain" x:Class="LangLibTestWPF.MainWindow"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VPKSoft="clr-namespace:VPKSoft.LangLib;assembly=VPKSoft.LangLib"

        Title="LangLibTestWPF" Height="309" Width="460" Icon="images/VPKSoft.ico">
```

So we link the Langlib library to the xaml by giving it a a xml namespace. NOTE: The namespace does not need to be VPKSoft.

xmlns:**VPKSoft**="clr-namespace:**VPKSoft.LangLib**;assembly=**VPKSoft.LangLib**"

After that we inherit the change the Window1: Window to

VPKSoft:**DBLangEngineWPF**

Just remember to keep those VPKSoft's as in VPKSoft == VPKSoft..

Program.cs

We need to modify this file so that the localization "dump" or database configuration may take place.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
using VPKSoft.LangLib;

namespace LangLibTestWinforms
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);


            if (DBLangEngine.ShoudConfigure()) // configure database connection and exit
            {
                DBLangEngine.RunConfigurator();
                return;
            }

            if (Utils.ShouldLocalize() != null) // Localize and exit.
            {
                new FormMain();
                new FormAbout();
                return;
            }
            Application.Run(new FormMain());
        }
    }
}
```

So we don't let the application to start if a command line parameter –dbLang or –dbLang=cu-RE (ISO 639-1) or –configureLang was given.

App.xaml.cs

We need to modify this file so that the localization "dump" or database configuration may take place.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;
using VPKSoft.LangLib;

namespace LangLibTestWPF
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        private void Application_Startup(object sender, StartupEventArgs e)
        {
            if (DBLangEngine.ShoudConfigure()) // Configure database and exit
            {
                DBLangEngine.RunConfigurator();
                System.Diagnostics.Process.GetCurrentProcess().Kill(); // self kill after configuration
                return;
            }

            if (Utils.ShouldLocalize() != null) // Localize and exit.
            {
                new MainWindow();
                new AboutWindow();
                System.Diagnostics.Process.GetCurrentProcess().Kill(); // self kill after localization
                return;
            }
        }
    }
}
```

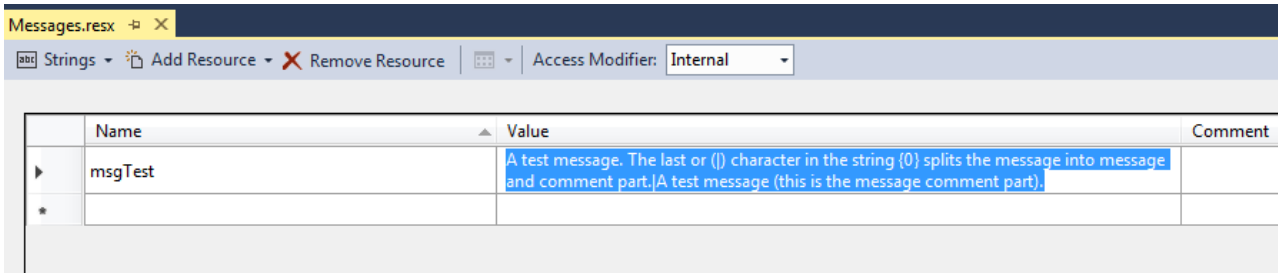
So with WPF we instruct the application to "kill self" if a command line parameter –dbLang or –dbLang=cu-RE (ISO 639-1) or –configureLang was given.

One embedded resource

As the was some talk about not needing any resources, a one is required.

This is to store the inside application messages such as a message shown in a MessageBox.

You may name it anything you want. Here is a sample:



Just remember to call the LangLib with the right **resource name**. If your application namespace is for example WindowsFormsApplication1

you should tell the application to initialize with a the resource name:

DBLangEngine.InitalizeLanguage("**WindowsFormsApplication1**.**Messages**");

That's it

I hope this help file was instructional enough to start localizing.