Support
support@visualcomponents.com

Community
community.visualcomponents.net

Adding localization

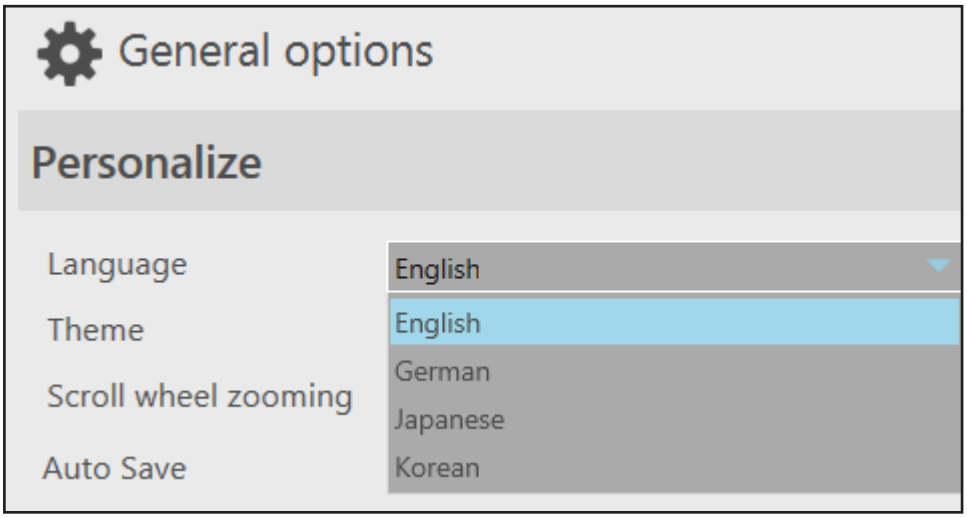
Next Generation | Version: November 20, 2015 | Example: Available upon request

Essentials has a multilingual user interface (MUI) that allows you to localize the user interface and support multiple languages at runtime.

The setup involves mapping the content of controls to keys in resource dictionaries using the ILocalizationService interface. A language resource assembly is named Resource.[Language] and placed in the Visual Components program files in order to be automatically recognized as a supported language. Keys for localizing extensions can be placed in an assembly that is named Resource.[Language].Additional.

The topics covered in this tutorial include:

- Minimal setup to provide localized controls in Essentials.
- Adding and using localization in extensions.
- Using design-time and runtime resources for extension development.
- Using command registry to import and export geometry.



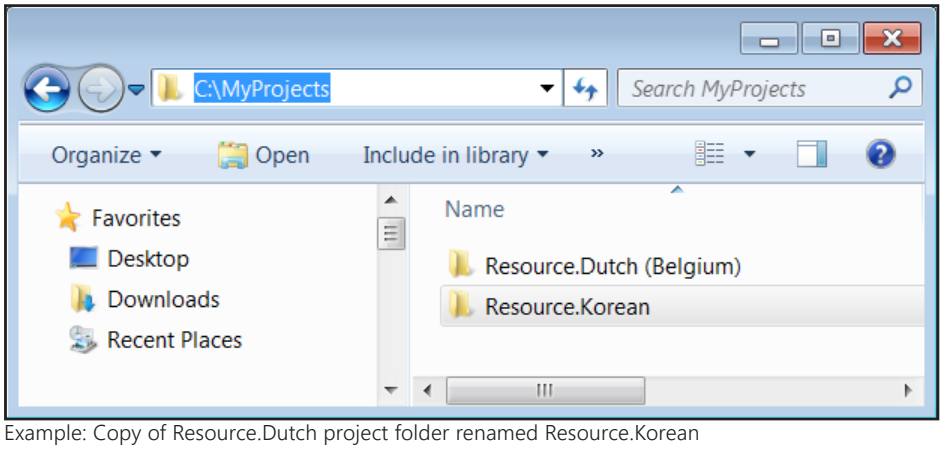
Creating a language resource assembly

Resources for supporting different languages should use a standardized set of keys. That is only the data referenced by resource keys needs to be translated and localized for users.

Request and modify template

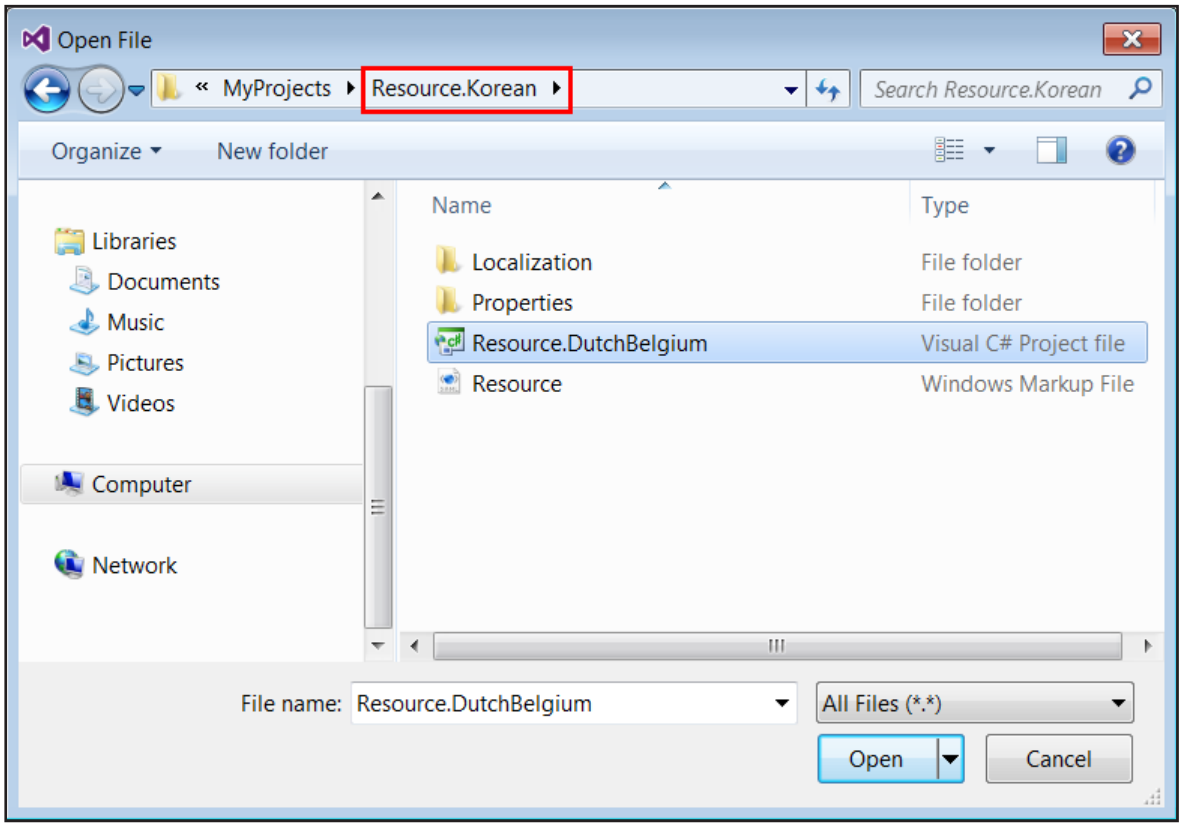
1. Request a **localization project** from Visual Components to use as a template for your target language.

2. Once the requested project has been received, create a **copy** of that project, and then rename that copy to indicate the target language.

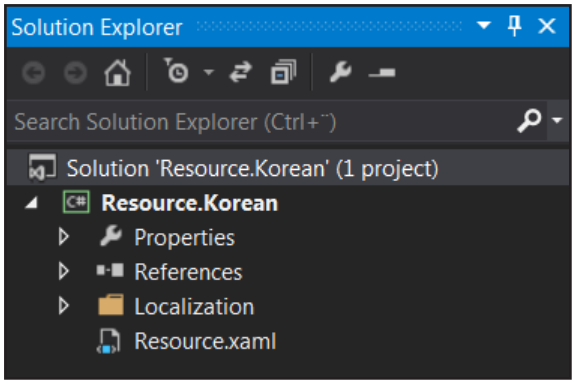


Example: Copy of Resource.Dutch project folder renamed Resource.Korean

3. Run **Visual Studio** as an **administrator**, and then in the copied project folder open the **Visual C# project file**.



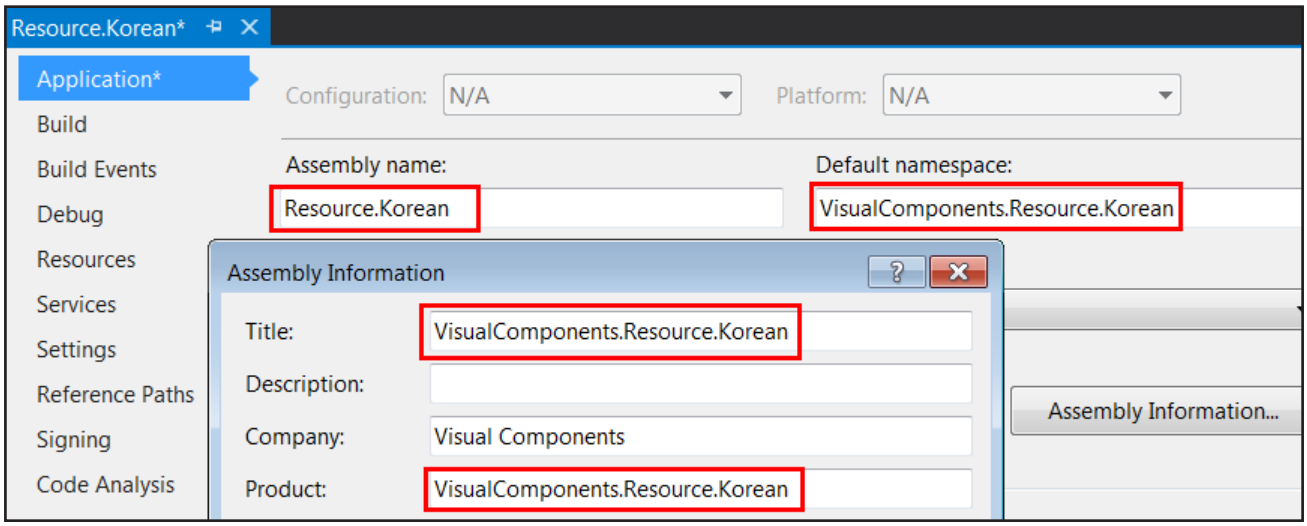
4. In **Solution Explorer**, rename your **solution name** and **project name** to indicate your target language.



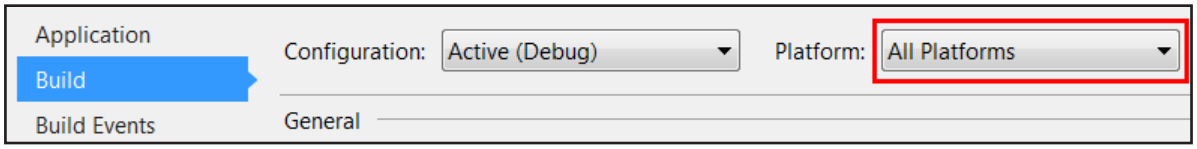
5. Access the properties of your project.

6. In the **Application** tab, set **Assembly name** and **Default namespace** to indicate your target language.

7. Click **Assembly Information**, and then set **Title** and **Product** to indicate your target language.



8. In the **Build** tab, set **Platform** to **All Platforms**.



9. In **Solution Explorer**, double-click **Resource**, and then edit the data for **LanguageCultureName** to be the **Culture Name** of your target language.

```
<!--  Language Culture Name  -->
<!--  see http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx  -->
<system:String x:Key="LanguageCultureName">ko-KR</system:String>
```

Translate data and build assembly

The resource files in the Localization folder can be sent to translation services with a clear indication that the keys should not be translated. You may directly edit the data for string elements in each resource file.

1. In **Solution Explorer**, expand the **Localization** folder, and then double-click **Application**.

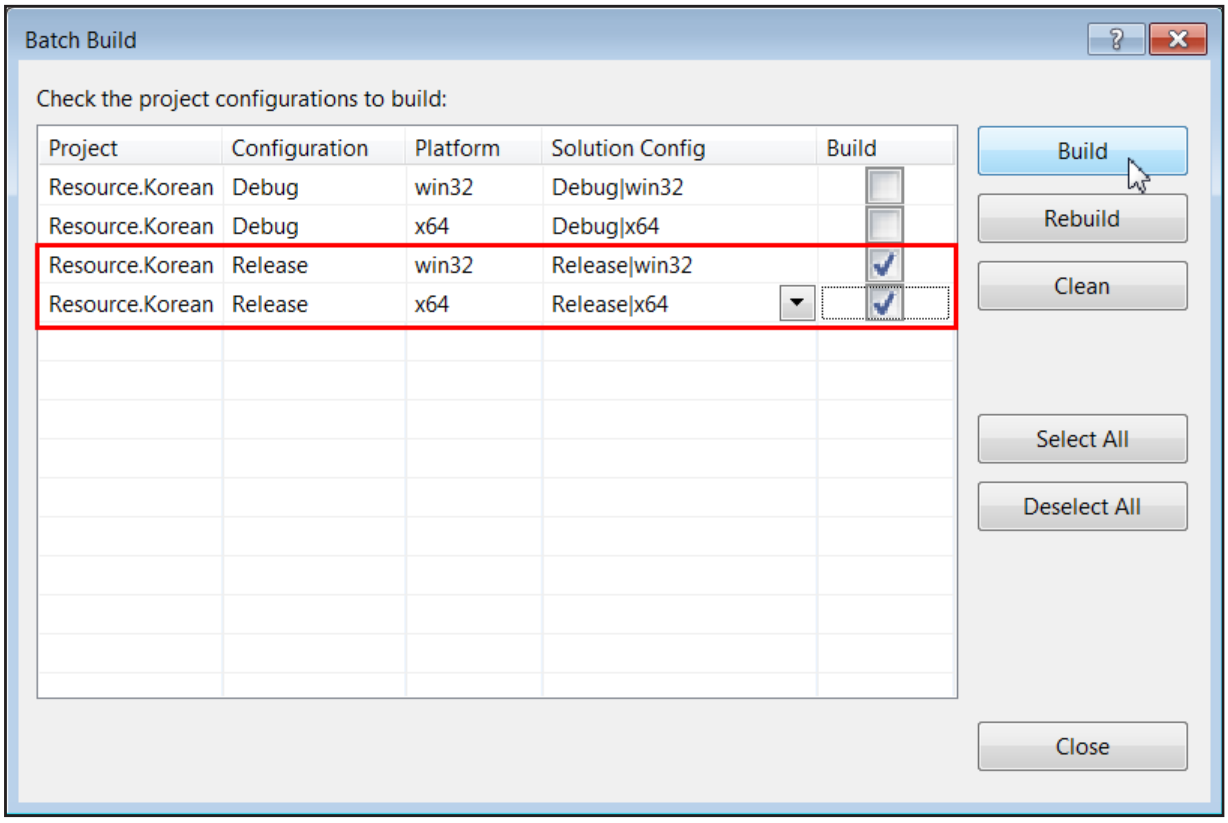
2. Edit the data for **RibbonTabItemFile** to be the name of File in your target language.

```
<!--=================================================================-->
<!--==== Ribbon                                                 =====-->
<!--=================================================================-->

<!--  Ribbon TabItems  -->
<s:String x:Key="RibbonTabItemFile">파일</s:String>

<s:String x:Key="RibbonTabItemHome">HOME</s:String>
<s:String x:Key="RibbonTabItemDrawing">DRAWING</s:String>
<s:String x:Key="RibbonTabItemVisualComponents">VISUAL COMPONENTS</s:String>
<s:String x:Key="RibbonTabItemTeach">PROGRAM</s:String>
<s:String x:Key="RibbonTabItemModeling">MODELING</s:String>
<s:String x:Key="RibbonTabItemImport">IMPORT</s:String>
<s:String x:Key="RibbonTabItemExport">EXPORT</s:String>
<s:String x:Key="RibbonTabItemBaseAndTool">BASE AND TOOL</s:String>
```

3. On the **Standard** toolbar, click **Save All** to save your solution and project.

4. On the **Build** menu, click **Batch Build**.

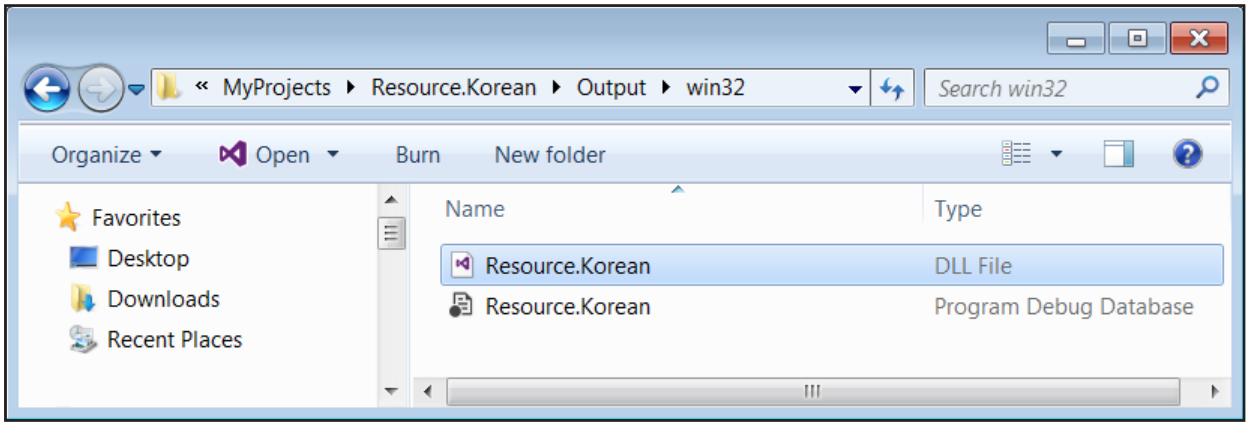
5. In the **Build** column, click the **Release** check boxes for your project, and then click **Build**.



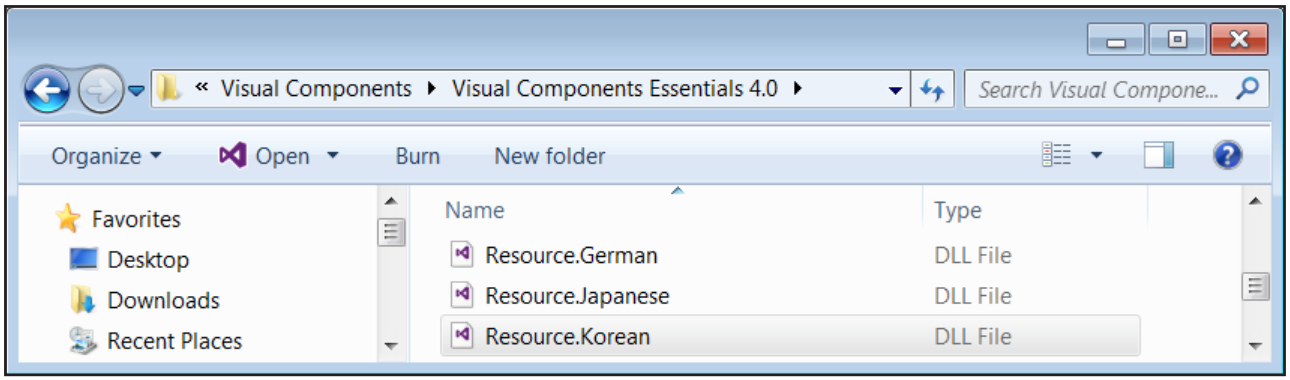
Testing localization

You must place a language resource assembly in your Visual Components program files. If you followed the naming convention of Resource.[Language] the MEF will automatically discover and load assembly at runtime and recognize your assembly as a supported language in Essentials.

1. Browse to the **Output** folder of your project, and then copy either the **32-bit** or **64-bit** version of your assembly.



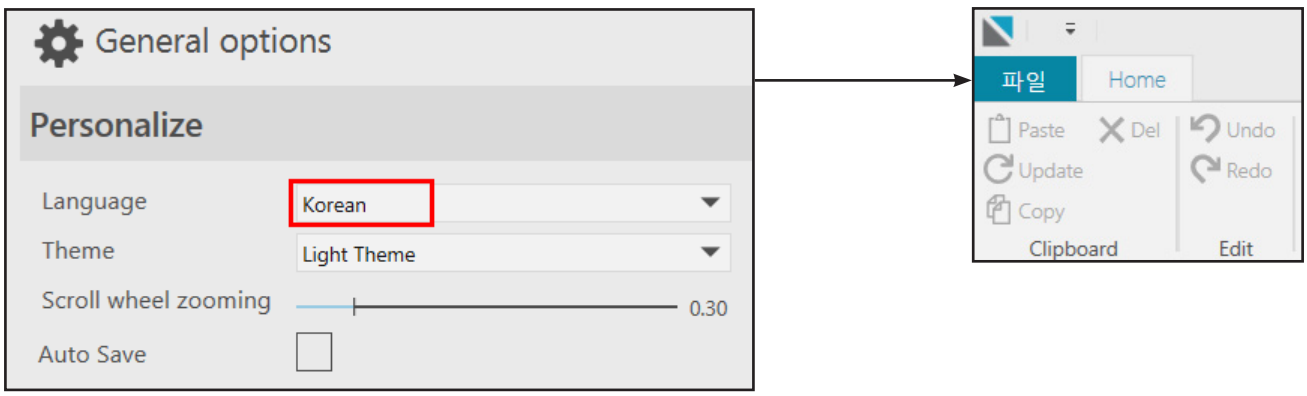
2. Browse to your **Visual Components program files**, and then paste your assembly.



3. Open **Essentials**, and then in the **File** tab click the **Options** tab.

4. In **General**, set **Language** to your target language, and then click **OK**.

5. Restart **Essentials** to verify the File tab is set to your target language.



Creating additional language resources

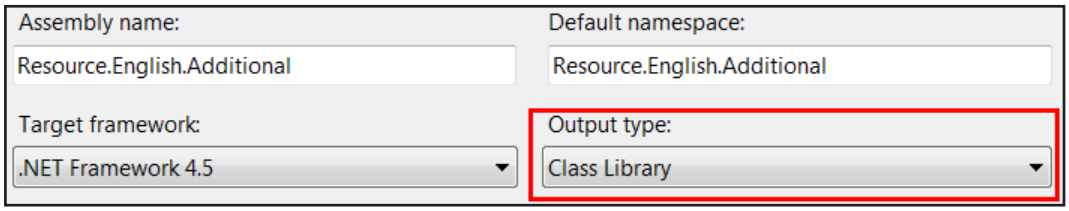
A naming convention of Resource.[Language].Additional allows you to add additional content to language resources to support extensions.

1. In **Visual Studio**, close your **solution**.

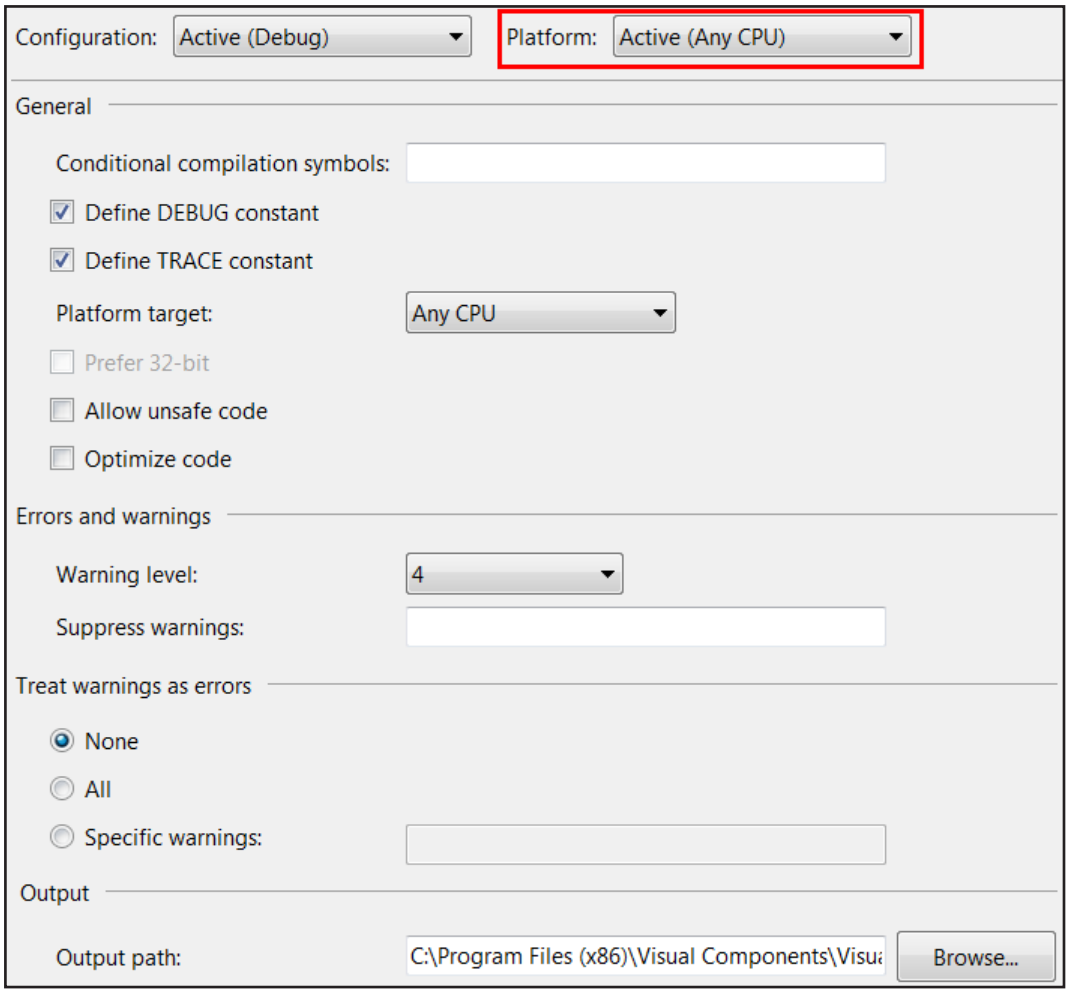
2. Create a new **WPF Application** project and name that project **Resource.English.Additional**.

3. Access the properties of your project.

4. In the **Application** tab, set **Output type** to **Class Library**.



5. In the **Build** tab, set **Platform** to **Any CPU** and **Output path** to be the path to your **Visual Components program files**.



6. In **Solution Explorer**, delete **App.config**, **App.xaml** and **MainWindow.xaml**.

7. In your project, add a **Folder** item and name that item **Localization**.

8. In the **Localization** folder, add a **WPF Resource Dictionary** item and name that item **Additional**.

9. In **Additional**, type the following code to define string elements for user controls in an extension.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:s="clr-namespace:System;assembly=mscorlib">

    <!--  Localized Strings  -->
    <!--  Add Localization comments before the entry (if any)  -->
    <!--  ************************************************  -->

    <!--  Example Controls-->
    <s:String x:Key="Example.PaneHeader">Imports &amp;amp; Exports</s:String>
    <s:String x:Key="Example.ButtonImport">Import Geometry</s:String>
    <s:String x:Key="Example.ButtonExport">Export Geometry</s:String>

</ResourceDictionary>
```

10. In your project, add a **WPF Resource Dictionary** item and name that item **Resource**.

11. In **Resource**, type the following code to add and merge your additional resources into one file.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">

    <!--  In this project Language Culture Name is not needed. It is set in the main language project  -->
    <!--  see http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx  -->

    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Localization/additional.xaml" />
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

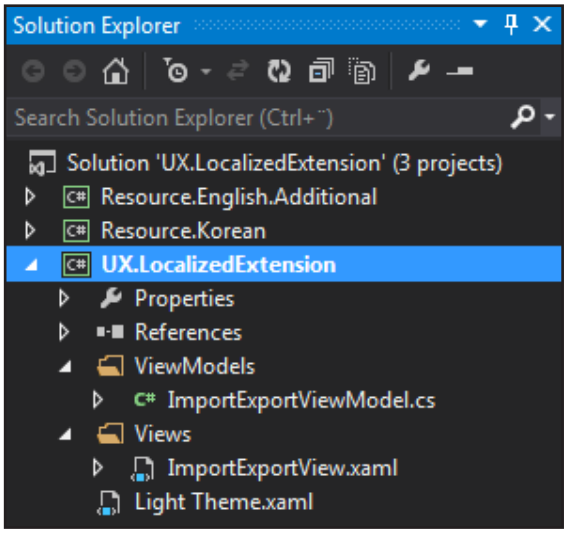
12. On the **Build** menu, click **Build Solution**.

Designing extension using resources

The Designer in Visual Studio supports 32-bit assemblies. Otherwise, you can verify the design of a 64-bit extension during runtime.

Define View Model

1. In **Visual Studio**, close your **solution**.
2. Create a new **Class Library** project and name that project **UX.LocalizedExtension**.
3. Add the following assemblies and namespaces as references:
   - **Caliburn.Micro**
   - **System.ComponentModel.Composition**
   - **System.Xaml**
   - **VisualComponents.Create3D.Shared**
   - **VisualComponents.UX.Shared**
4. Access the properties of your project.
5. In the **Build** tab, set **Platform target** to **x86** or **x64** and **Output path** to be the path to your **Visual Components program files**.
6. In your project, add two **Folder** items: one named **ViewModels** and the other named **Views**.
7. In the **ViewModels** folder, add a **Class** item and name that item **ImportExportViewModel**.
8. In the **Views** folder, add a **WPF User Control** and name that item **ImportExportView**.
9. In your project, add as an existing item the **Light Theme** file located in the **Dictionaries** folder of your Visual Components program files.
10. In your solution, add the existing projects for **Resource.English.Additional** and **Resource.[Language]** that you have created so far in this tutorial.



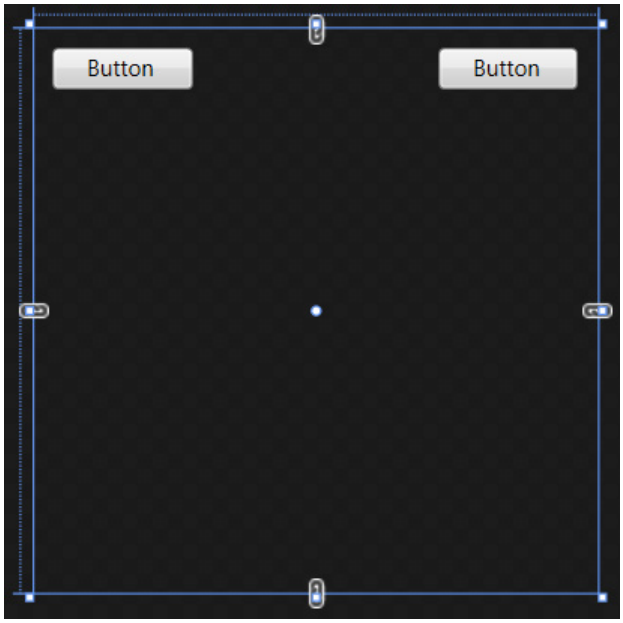
NOTE! The Light Theme file will be used as a design time resource for brushes and will not be copied when you build your assembly.

11. In **ImportExportViewModel**, type the following code to define a basic View Model.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace UX.LocalizedExtension.ViewModels
{
    using Caliburn.Micro;
    using System.ComponentModel.Composition;
    using VisualComponents.Create3D;
    using VisualComponents.UX.Shared;

    [Export(typeof(IDockableScreen))]
    class ImportExportViewModel: DockableScreen
    {
        //use additional resource key for display name
        [ImportingConstructor]
        public ImportExportViewModel([Import] ILocalizationService localizationService)
        {
            this.DisplayName = localizationService.GetText("Example.PaneHeader");
        }
}
```

Design time of View

1. In **ImportExportView**, add two **Button** controls to the Grid element, and then assign your import and export methods to those buttons.




```
<Grid>
    <Button x:Name="importGeometryFileAsComponent"
            Content="Button" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Width="75"/>
    <Button x:Name="exportLayoutAsDrawing3D"
        Content="Button" HorizontalAlignment="Left" Margin="215,10,0,0" VerticalAlignment="Top" Width="75"/>
</Grid>
```

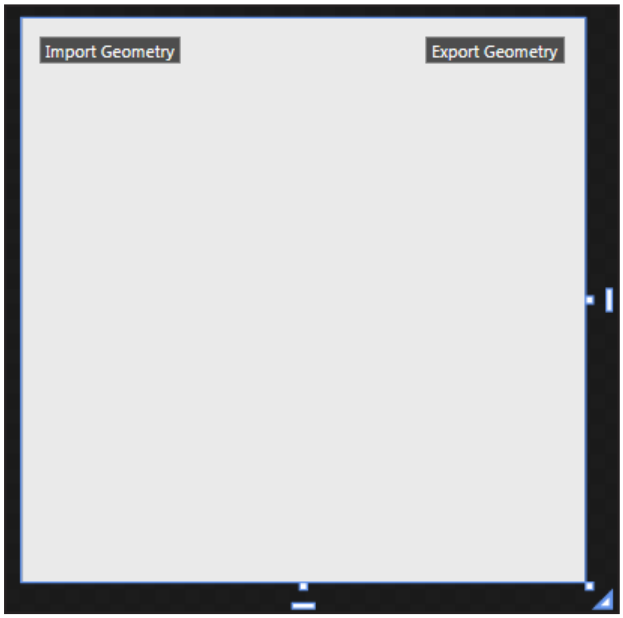
2. In the **UserControl** element, add the following attribute to map to the **VisualComponents.UX.Shared** assembly.

```
        xmlns:shared="clr-namespace:VisualComponents.UX.Shared;assembly=UX.Shared"
```

3. In the **UserControl** element, add the following child elements to add and merge resource dictionaries.

```
<UserControl.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary
                Source="pack://application:,,,/UX.Shared;component/Controls/VCThemes/ThemeDictionary.xaml" />
            <ResourceDictionary
                Source="pack://application:,,,/UX.Shared;component/Controls/VCThemes/VCButton.xaml" />
            <ResourceDictionary
                Source="pack://application:,,,/UX.Shared;component/Controls/VCThemes/VCGrid.xaml" />
            <shared:DesignTimeDictionary
                Source="..\Light Theme.xaml" />
            <shared:LocalizationDesignTimeDictionary
                Source="pack://application:,,,/Resource.English.Additional;component/Resource.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</UserControl.Resources>
```

4. Modify the markup for the **Grid** and **Button** elements to add styles and assign localized content.




```
<Grid Style="{DynamicResource VCSimpleGridStyle}">
    <Button Style="{DynamicResource VCButtonStyle}"
        x:Name="importGeometryFileAsComponent"
        Content="{DynamicResource Example.ButtonImport}" HorizontalAlignment="Left"
            Margin="10,10,0,0" VerticalAlignment="Top" Width="75"/>
    <Button Style="{DynamicResource VCButtonStyle}"
        x:Name="exportLayoutAsDrawing3D"
        Content="{DynamicResource Example.ButtonExport}" HorizontalAlignment="Left"
            Margin="215,10,0,0" VerticalAlignment="Top" Width="75"/>
</Grid>
```

5. On the **Build** menu, click **Build Solution**.

6. Open **Essentials**, and then set **Language** to **English**.

7. Restart **Essentials** to verify the content of your extension matches the language of Essentials.

Importing and exporting geometry

The command registry of Essentials can be used to execute commands that are of type IActionItem.

Retrieve commands

You can create an instance of ICommandRegistry to read and find available commands in Essentials.

1. Exit **Essentials**, and then return to your project in **Visual Studio**.

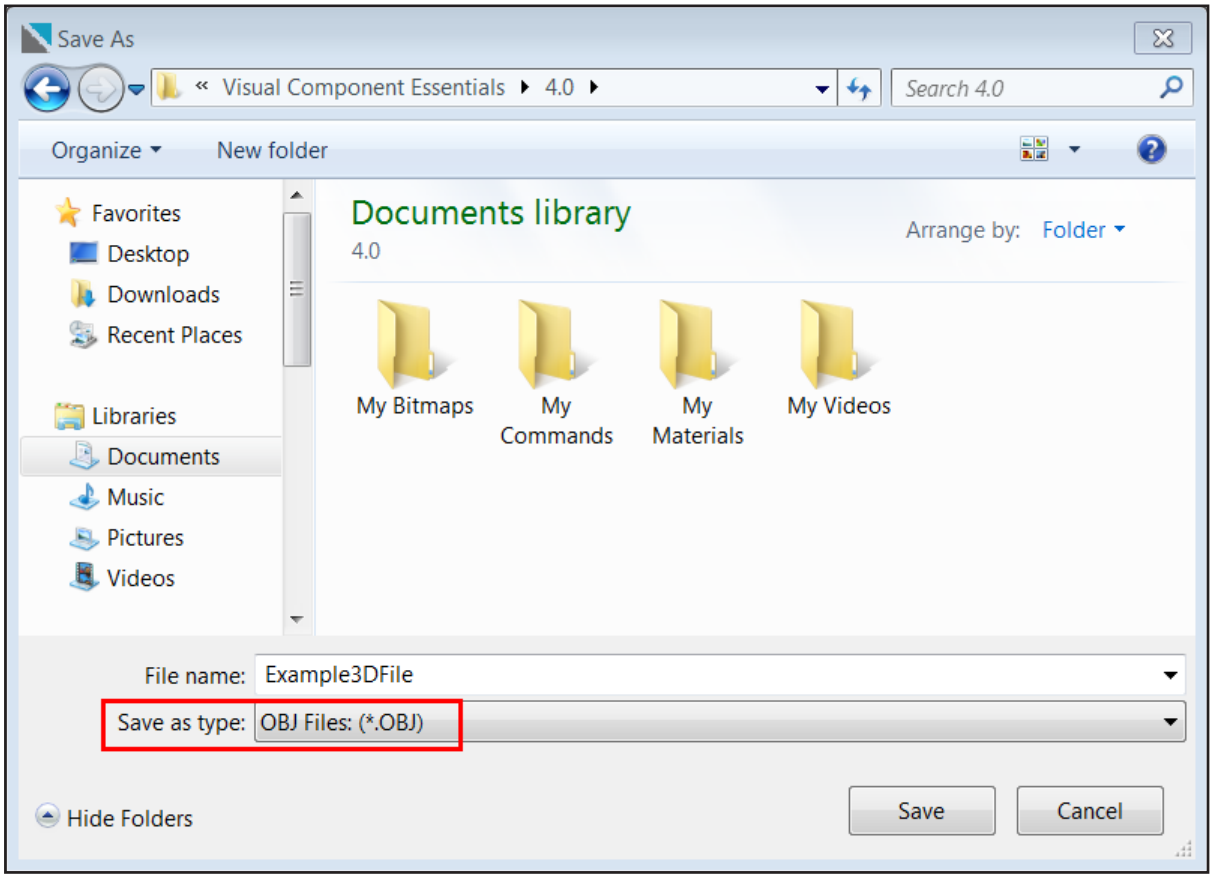
2. In **ImportExportViewModel**, add the following code to define methods for importing and exporting geometry.

```
//import geometry file as new component by referencing IActionItem id
public void importGeometryFileAsComponent()
{
    ICommandRegistry cmdRegistry = IoC.Get<ICommandRegistry>();
    IActionItem import3D = cmdRegistry.FindItem("ImportGeometry");
    import3D.Execute();
}

//export layout as geometry file by referencing IActionItem id
public void exportLayoutAsDrawing3D()
{
    ICommandRegistry cmdRegistry = IoC.Get<ICommandRegistry>();
    ActionItem<bool> export3D = cmdRegistry.FindItem("ExportToVectorDrawing") as ActionItem<bool>;
    export3D.Execute(true);
}
```

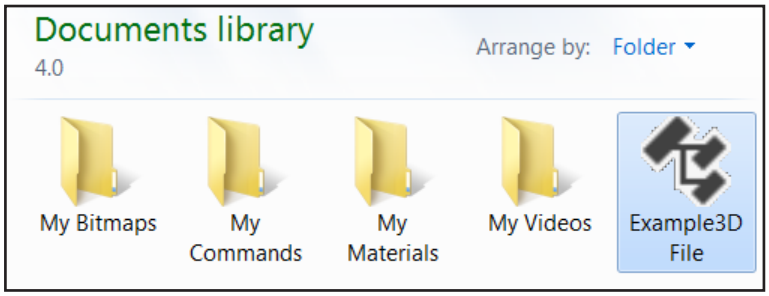
3. On the **Build** menu, click **Clean Solution**, and then click **Build Solution**.

Test command execution

1. Open **Essentials**, and then add a layout to the 3D world.
2. In **Imports & Exports**, click **Export Geometry**.
3. Save the file as an **.obj** file type in the local Documents library for Essentials.



4. Open a new empty layout in the 3D world.
5. In **Imports & Exports**, click **Import Geometry**.
6. Open the **exported .obj file** created from the previous layout.



This concludes the tutorial.