Technical Note

Automatic Control Translation

Author: R&D Department

Publication date: December 21, 2006

Revised: May 2010

Automatic Control Translation

Overview

Building multi-language applications is easy with REP++, since several of its components can be saved in different languages in the repository. When you login, you simply select the language you want to use and the application will display its interface elements in the correct language. This process is done seamlessly in REP++ with its own user interface elements.

This result can also be achieved with controls and components which are not attached to REP++ visual controls such as menus, toolbars or form name. The REP++*toolkit* provides a utility class, **ControlTextTranslator**, which greatly facilitates the task of translating the textual values of a number of standard controls and components. In particular, the **ControlTextTranslator** utility class translates the ***Text*** and ***Tooltip*** properties for all user interface components that derive from the **Control** class and objects such as **MenuItem**, **ToolStripMenuItem** and **ToolbarButton**.

This article describes how to use the **ControlTextTranslator** class to translate the user interface of your applications and how to extend it to support more controls and properties.

Using the ControlTextTranslator utility class

The **ControlTextTranslator** utility class uses REP++ atoms to store language-dependent text. The atom text must be in the following format:

```
ControlName1.AttributeName1=Text
ControlName1.AttributeName2=Text
ControlName2.AttributeName1=Text
```

For example:

```
BtnClose.Text=&Close
BtnClose.Tooltip=Press this button to close the form
BtnNew.Text=&New page
```

To illustrate how you can use the **ControlTextTranslator** utility class, you will first create a form containing a number of standard controls. You will then use the utility class to set their **Text** and **Tooltip** properties.

Create a new Windows® application project and add the following controls to the default form (make sure that the names of your controls match the names in the figure):

Add a reference to **RepPP.dll** and **RepPP.Toolkit.V1B.dll**.

Change the code of the default form as follows:

```
using System;
using System.Windows.Forms;
using RepPP.Toolkit.Window;

namespace Demo {
  public partial class Form1 : Form {
    public Form1() {
      RepPP.Application       app;
      ControlTextTranslator translator;

      InitializeComponent();
      app     = RepPP.Application.CreateFromRes();
      translator = new ControlTextTranslator(app, new string[]{"MENU_TRANSLATION_ATOM",
                                                               "OTHER TRANSLATION ATOM"});
      translator.TranslateControl(this);
    }
  }
}
```

As you can see, the above code references two atoms that contain language-specific text for the controls. Note that the text was split in two atoms to illustrate the fact that you can provide more than one atom. Use REP++*studio* to create these two atoms in the contact management DEMO system.

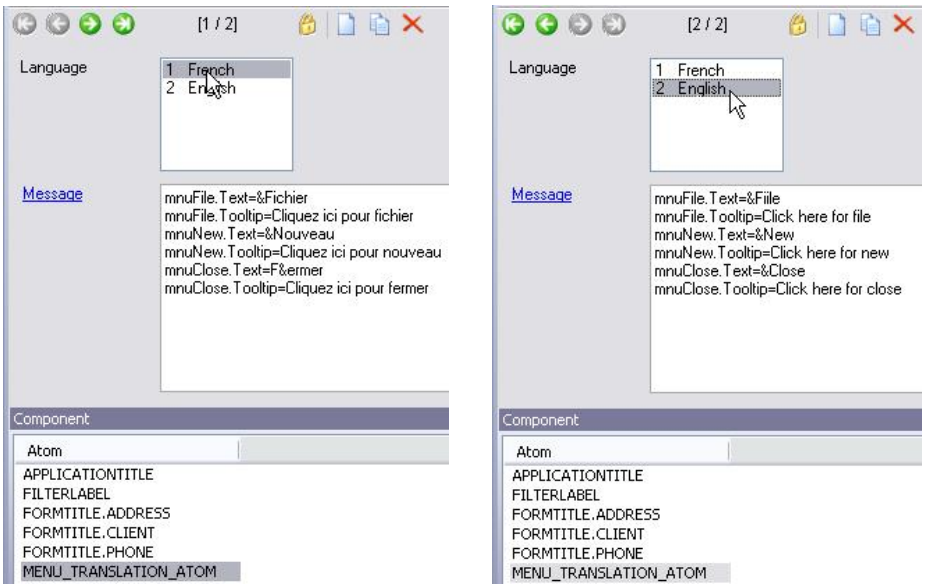


Figure 1. Entering the information in the atom MENU_TRANSLATION_ATOM, in French and in English.

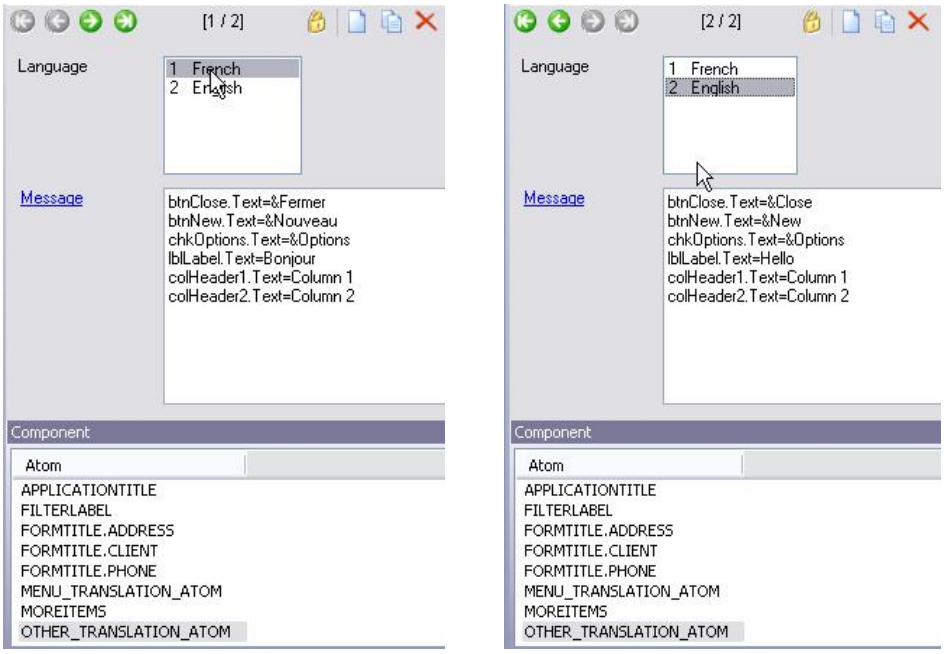


Figure 2. Entering the information in the atom OTHER_TRANSLATION_ATOM, in French and in English.

Build your project and run the application, you should have the following result.

Notice that the column headers were not translated. The **ControlTextTranslator** utility class does not support column headers[1]. To remedy this situation, you can manually translate the column headers by specifying explicitly the control name and attribute to use (in bold):

```
using System;
using System.Windows.Forms;
using RepPP.Toolkit.Window;

namespace Demo {
  public partial class Form1 : Form {
    public Form1() {
      RepPP.Application       app;
      ControlTextTranslator translator;

      InitializeComponent();
      app     = RepPP.Application.CreateFromRes();
      translator = new ControlTextTranslator(app, new string[]{"MENU_TRANSLATION_ATOM",
                                                               "OTHER_TRANSLATION_ATOM"});

      translator.TranslateControl(this);
      colHeader1.Text = translator.GetItemText("colHeader1", "Text");
      colHeader2.Text = translator.GetItemText("colHeader2", "Text");
    }
  }
}
```

Run your application to check the results.

[1] The REP++*toolkit* provides a **ListView** control that can be bound to a Rowset (**ListViewRowset**). The **ListViewRowset** control automatically creates the columns and sets their header text to the language-specific prompt of the field they represent. Thus, you do not need to explicitly translate the column headers of a **ListViewRowset** control.

What you have done so far is one way to extend the behaviour of the **ControlTextTranslator** utility class. In the next section, you will discover a better approach.

Extending the ControlTextTranslator utility class

The standard **ControlTextTranslator** class does not support column headers. In this section, you will create your own **ControlTextTranslator** class capable of doing everything the standard translator does, and supporting column headers too.

First, let us look at the internals of the standard **ControlTextTranslator** utility class to understand what it does and how.

- The **ControlTextTranslator** objects starts translating a control by calling the **SetComponentInfo** method.
- If a control is known to contain components, then the **ControlTextTranslator** objects use reflection to retrieve the list of all fields of the class defining the control.
- Steps 1 and 2 are then repeated for each child control of the current control.

Back with your example:

1. Add a new class that derives from the **ControlTextTranslator** class to your project and name it **MyControlTextTranslator**.
2. Create a new constructor that takes two parameters: a REP++ **Application** object and a string array of atom names.

```
using System;
using System.Windows.Forms;
using RepPP;
using RepPP.Toolkit.Window;

namespace Demo {
  class MyControlTextTranslator : ControlTextTranslator {
    public MyControlTextTranslator(IApplication app,
                                   string[]     arrAtomNames): base(app,
                                                                    arrAtomNames) {
    }
}
```

3. Override the method called to translate each control to enable it to translate column headers.

```
…
namespace Demo {
    class MyControlTextTranslator : ControlTextTranslator {
        …
        protected override void SetComponentInfo(object  obj,
                                                 string  strObjName,
                                                 ToolTip toolTip) {
            if (obj is ColumnHeader){
                (obj as ColumnHeader).Text = GetItemText(strObjName, "Text");
            } else {
                base.SetComponentInfo(obj, strObjName, toolTip);
            }
        }
    }
}
```

Now that you have created a **ControlTextTranslator** class capable of translating column headers, use it in your sample application.

4. Remove the manual translation code from the default form. The code of should look as follows:

```
using System;
using System.Windows.Forms;
using RepPP.Toolkit.Window;

namespace Demo {
  public partial class Form1 : Form {
    public Form1() {
      RepPP.Application       app;
      MyControlTextTranslator translator;

      InitializeComponent();
      app     = RepPP.Application.CreateFromRes();
      translator = new MyControlTextTranslator(app, new string[]{"MENU_TRANSLATION_ATOM",
                                                                 "OTHER_TRANSLATION_ATOM"});
      translator.TranslateControl(this);
    }
  }
}
```

5. Build your project and run the application. You should have the same result as with the previous, manual method.

You can further extend your example to support a **NotifyIcon** component and translate its **Text** and **BalloonTipText** properties.

1. Add a **NotifyIcon** component to *Form1* (make sure that its name is **notifyIcon1**).
2. Modify the code of *Form1* to display the **NotifyIcon** as follows:

```
using System;
using System.Windows.Forms;
using RepPP.Toolkit.Window;

namespace Demo {
  public partial class Form1 : Form {
    public Form1() {
      RepPP.Application       app;
      ControlTextTranslator translator;

      InitializeComponent();
      app     = RepPP.Application.CreateFromRes();
      translator = new ControlTextTranslator(app, new string[]{"MENU_TRANSLATION_ATOM",
                                                               "OTHER_TRANSLATION_ATOM"});
      translator.TranslateControl(this);
      notifyIcon1.ShowBalloonTip(1000);
    }
  }
}
```

3. Add the following lines to the text of OTHER_TRANSLATION_ATOM.

English:

```
notifyIcon1.Text=This is the translated text
notifyIcon1.BalloonTipText=This is the translated Balloon text
```

French:

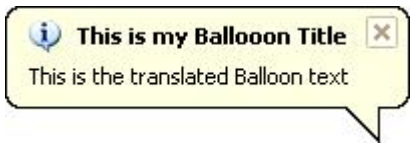
```
notifyIcon1.Text=Ceci est le texte traduit
notifyIcon1.BalloonTipText=Ceci est le texte de l'info-bulle traduite
```

4. Modify the **SetComponentInfo** method to support the **NotifyIcon** component as follows:

```
…
namespace Demo {
  class MyControlTextTranslator : ControlTextTranslator {
    …
    protected override void SetComponentInfo(object  obj,
                                             string  strObjName,
                                             ToolTip toolTip) {
      if (obj is ColumnHeader){
        (obj as ColumnHeader).Text = GetItemText(strObjName, "Text");
      } else if (obj is NotifyIcon) {
        (obj as NotifyIcon).Text = GetItemText(strObjName, "Text");
        (obj as NotifyIcon).BalloonTipText = GetItemText(strObjName, "BalloonTipText");
      } else {
        base.SetComponentInfo(obj, strObjName, toolTip);
      }
    }
  }
}
```

5. Build and run your application.

Your balloon should look similar to the figure below.


This is my Ballooon Title
This is the translated Balloon text