Technical Note

New Multiple Document Interface Applications Using REP++toolkit

Author: R&D Department

Publication date: September 13, 2006

New Multiple Document Interface Applications Using REP++*toolkit*

Overview

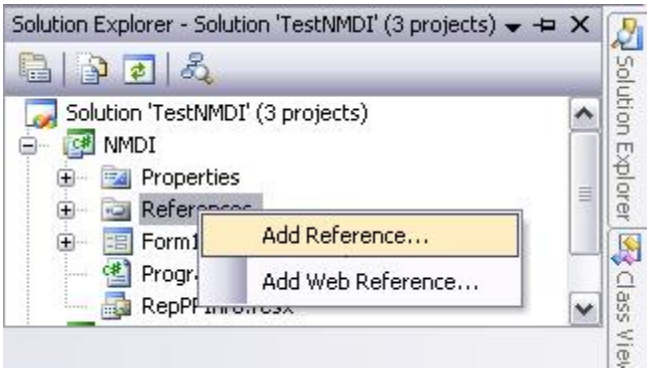
NMDI (New Multiple Document Interface) applications allow you to show multiple documents at the same time within a parent window. Every document is displayed in its individual window and has its own icon in the task bar. The parent window has a **Window** menu that contains the commands for accessing its child windows as well as rearranging the child windows within the workspace.

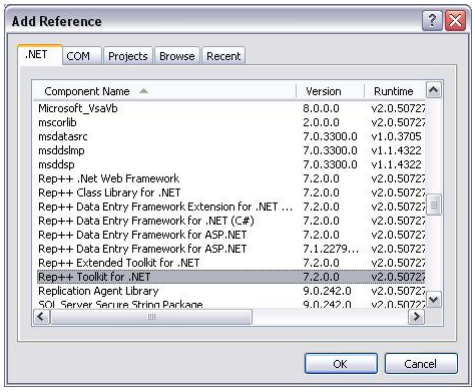
This document describes how to create applications that follow the new NMDI approach.

Creating Parent MDI Forms

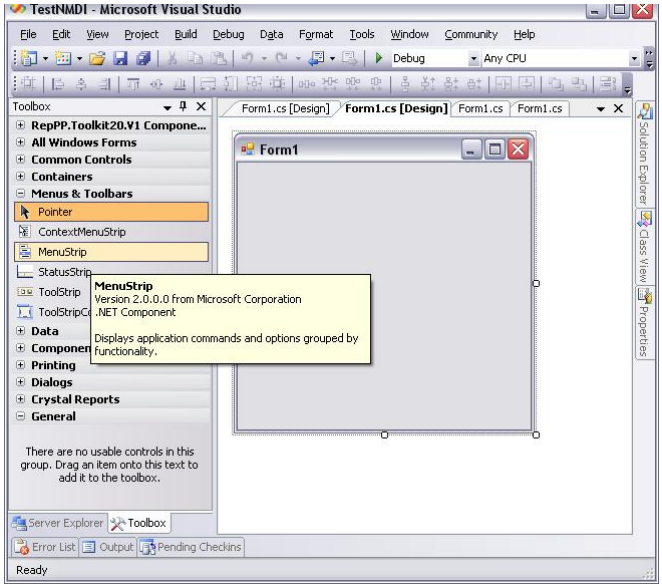
The parent MDI window is the main form that contains all the child forms.

1. In Visual Studio .Net 2005, create a new C# Windows Application Project and name it NMDI.
2. Add a reference to the assembly containing the **Rep++ Toolkit for .NET**.



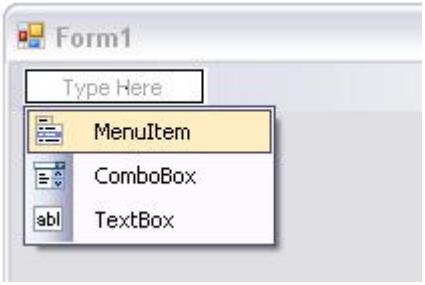


3. In the default form *Form1*, add a **MenuStrip** component.

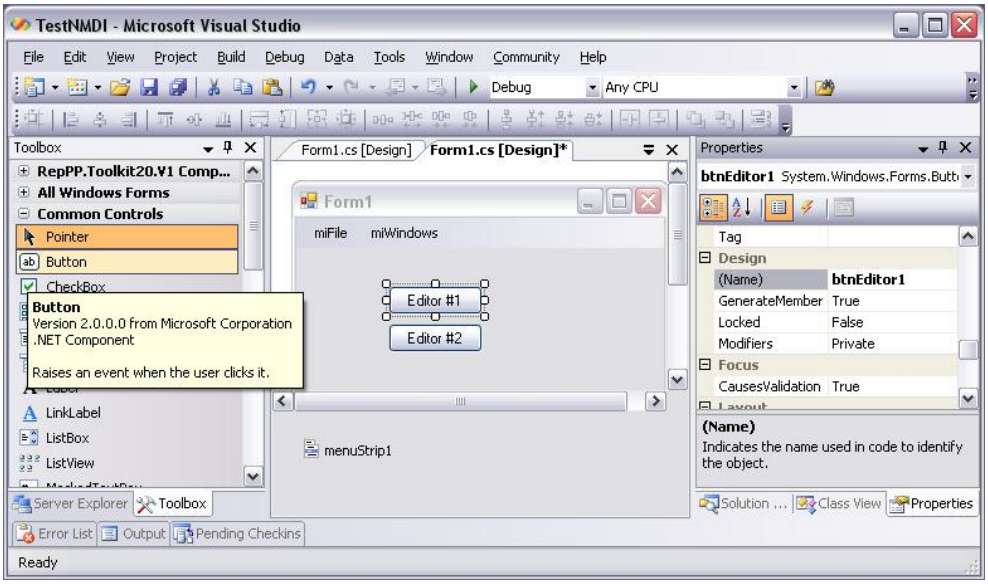


4. In the **MenuStrip**, create the following menus and command using **MenuItem**, as shown below. (Make sure the names match the names in the figure below. As you will see later, these names will allow you to use the built-in toolkit translator to get their text depending on the current language.)

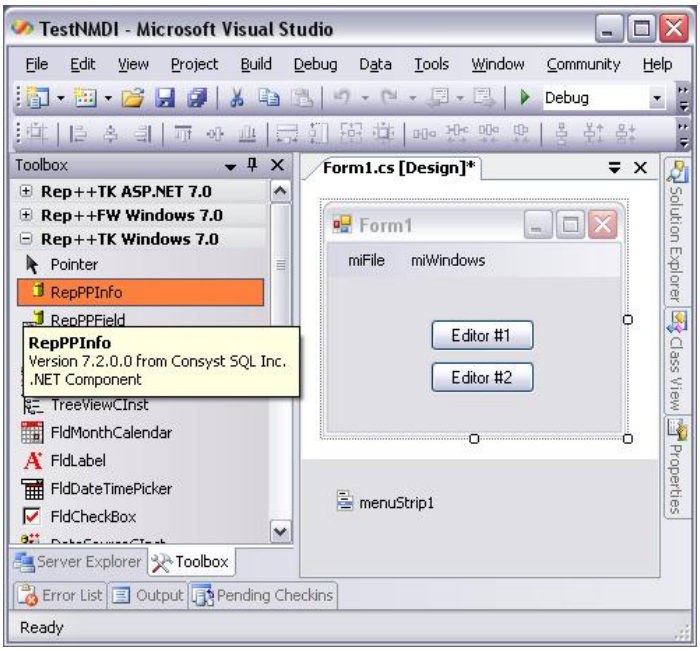
   - Menu name: **miFile**
   - Command (under **miFile** menu): **mnuExit**
   - Menu name: **miWindows**



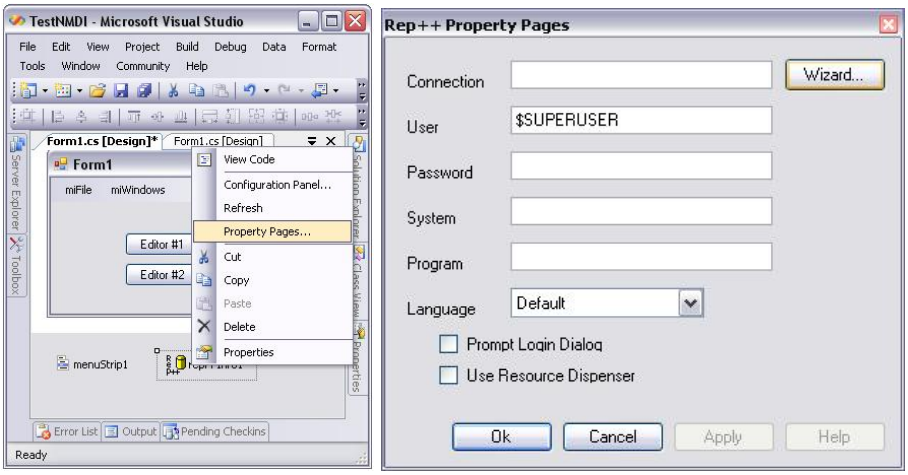
5. Add two buttons to *Form1* and name them **btnEditor1** and **btnEditor2**.



6. Add a **RepPPInfo** component to your form in order to connect your application to the REP++*repository*.



7. In the *REP++ Property Pages* of the **RepPPInfo** component, click on **Wizard**.



8. Make sure that your connection settings look similar to the following figure. (These settings represent a connection to the repository of the Demo Client Management System.)

The connection to the REP++repository is required for automatic translation.

9. Switch to the code view of *Form1* and add a reference to the namespace of the toolkit.

```
…
using RepPP.Toolkit.Window;
```

10. Every New MDI form must implement the interface **NMDIFormHelperBase.INMDIForm** as shown below.

```
public partial class Form1 : Form, NMDIFormHelperBase.INMDIForm
{
    …
    …
    public bool AskBeforeClosing()
    {
        // This is where you would check if your data
        // is dirty and do something about it
```

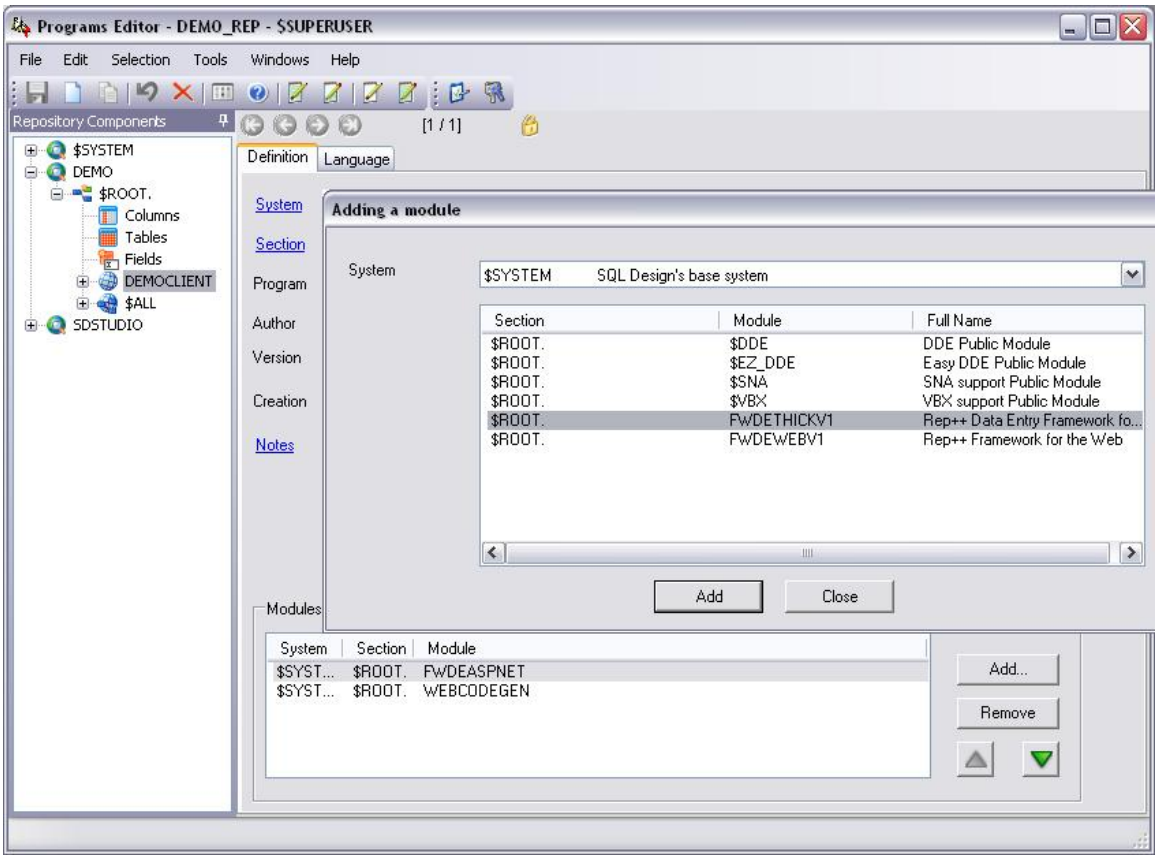
```
            return (true);
        }
    }
```

11. Add the following members to class **Form1**.

```
private RepPP.Application m_app;         // Rep++ Application
private NMDIFormHelperUI m_nmdi;         // New MDI Helper Object
private CmdHolder m_cmdHolder;           // Command holder
private ControlTextTranslator m_trans;   // Control text translator
```

- **m_app** will contain the **Rep++Application** object.
- **m_nmdi** and **m_cmdHolder** will inject and handle MDI related commands in your **miWindows** menu.
- **m_trans** will use reflection to get the text of your form controls according to the current language of the application.

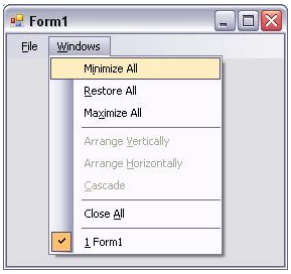
12. The **ControlTextTranslator** uses atoms in the repository to translate the text of many standard controls. In this example, you will simply use the atoms **txtfrmTopBaseEditor** and **txtfrmBaseSBEditor**, which already provide the translated text for all the menu items. These atoms are defined in the public module **FWDETHICKV1** in **$SYSTEM**. You need to import the public module **FWDETHICKV1** into the program **DEMOCLIENT** as follows:



13. Modify the form constructor to add the initialization of the members added in the previous step. Your form constructor should be similar to the following code.

```
public Form1()
{
    InitializeComponent();
    …
    …

    // Create the Rep++ Application
    m_app = RepPP.Application.CreateFromRes();

    // Create the New MDI Form Helper
    m_nmdi = new RepPP.Toolkit.Window.NMDIFormHelperUI();

    // Create the Command Holder
    m_cmdHolder = new CmdHolder();

    // Create the Control Text Translator and set the names of the atoms containing
    // the language specific text of standard controls.
    // NOTE: Make sure that you include the Module FWDETHICKV1 in your program
    m_trans = new ControlTextTranslator(m_app, new string[] {"txtfrmTopBaseEditor",
                                                             "txtfrmBarBaseSBEditor"});
    // Translate the text of all the controls in the form
    m_trans.TranslateControl(this);

    // Register this form as a New MDI
    m_nmdi.RegisterForm(this,           // The form to be registered
                        true,           // True to show the form in the top form list
                        miWindows,      // Menu to be filled with New MDI related items
                        m_cmdHolder,    // Command holder
                        m_trans);       // Translator to set the text of injected items
}
```

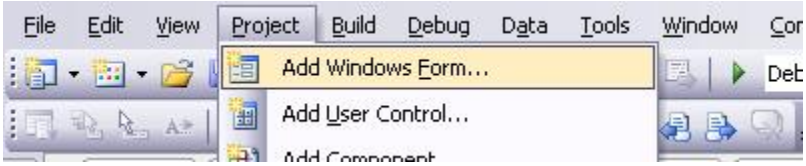
14. At this point, you have created the top form of a New MDI application. You can run your application and see the result.



Creating Child MDI Forms

In this section, you will create the child MDI forms and link them to the top form (*Form1*).

1. Add a new form named *Editor1* to your project. *Editor1* will contain a **MenuStrip** component that is identical to the one you added in *Form1*.



2. Switch to the code view of *Editor1* and make sure that your class is similar to the following class (changes are marked in bold).

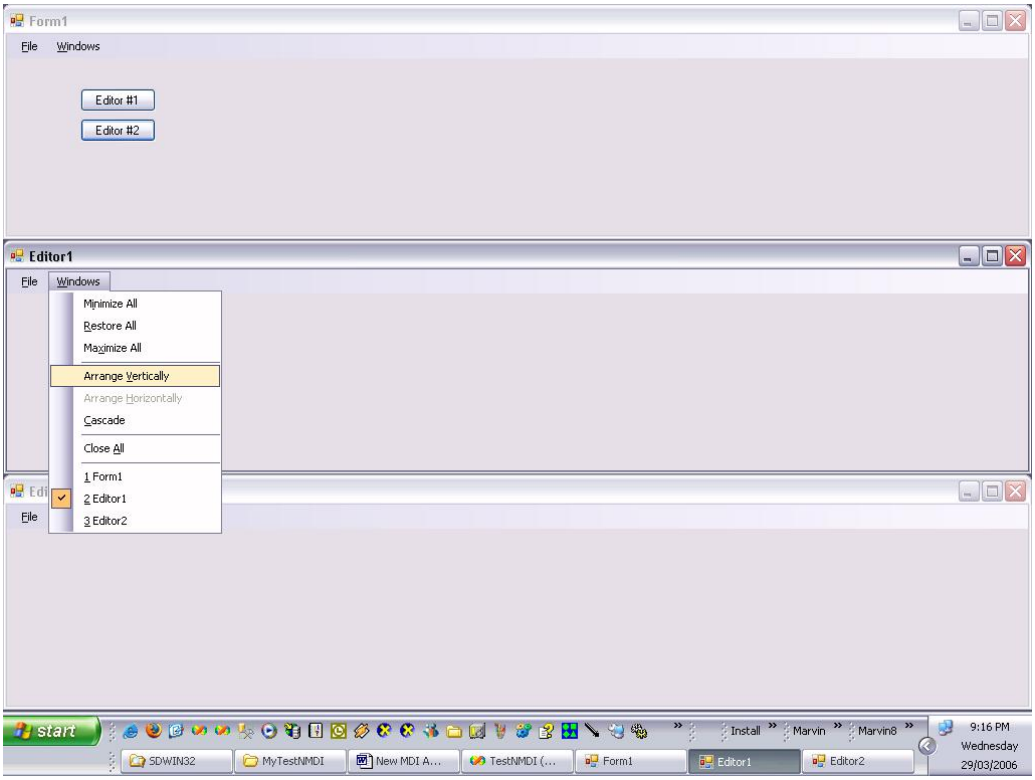
```
…
using RepPP.Toolkit.Window;
…
public partial class Editor1 : Form, NMDIFormHelperBase.INMDIForm {
    private RepPP.Application       m_app;          // Rep++ Application
    private NMDIFormHelperUI        m_nmdi;         // New MDI Helper Object
    private CmdHolder               m_cmdHolder;    // Command holder
    private ControlTextTranslator   m_trans;        // Control text translator

    public Editor1() {
        InitializeComponent();
    }

    public Editor1(RepPP.Application        app,
                   NMDIFormHelperUI         nmdi,
                   ControlTextTranslator    trans) : this() {

        m_app       = app;
        m_nmdi      = nmdi;
        m_trans     = trans;
        m_cmdHolder = new CmdHolder();
        m_trans.TranslateControl(this);
        m_nmdi.RegisterForm(this,
                            true,
                            miWindows,
                            m_cmdHolder,
                            m_trans);
    }

    bool NMDIFormHelperBase.INMDIForm.AskBeforeClosing() {
        // This is where you would check if your data
        // is dirty and do something about it
        return(true);
    }
}
```

3. To link *Editor1* to *Form1*, open *Form1* in design view and double-click on **btnEditor1** to create a handler for its **Click** event. Link it to *Editor1* as follows:

```
private void btnEditor1_Click(object sender, EventArgs e)
{
    Editor1 frm1;

    frm1 = new Editor1(m_app, m_nmdi, m_trans);
    frm1.Show();
}
```

4. Repeat steps 1 through 3 to create another child form called *Editor2* and to link it to *Form1*.

Your New MDI application is complete. Run it and check its behavior.



Conclusion

We have shown you how you can create New MDI Applications using the REP++*toolkit*. In particular, you have seen how the toolkit facilitates the process by encapsulating the common repetitive tasks related to managing MDI forms like the injection of the menu items in the **Windows** menu.