Technical Note

Filtering Using REP++*framework* for .NET — Window® applications

Author: R&D Department

Publication date: January 5, 2007

Revision date: May 14, 2010

Filtering Using REP++*framework* for .NET — Windows® applications

Overview

Applications that read data from a data store often have to provide a user interface that allows the end users to filter the data to be loaded. REP++*framework* for .NET provides objects and forms that automate the dynamic generation of a filter form based on a RowsetDef and the fields it contains.

This article describes the default filter form, how it integrates to the REP++*framework*, and how to use it in a simple application.

Contents

The default filter form 2

Using the default filter form in an application that does not already use REP++framework.....4

Defining a custom filter form 5

Replacing the default filter form in an application that uses REP++framework.....7

The default filter form

The best way to see the default filter form in action is to generate a Windows® application for the contact management application using the REP++ wizard.

1. Create a new Windows project using the REP++ wizard and generate the demo application.
2. Build the project and run the application.
3. From the **Selection** menu, click **Set Filter** and experiment with the default filter form that is dynamically generated for the selection buffer.

Please note that you can include the wild characters "%" and "_" in your text strings to turn the "=" and "<>" operators into "LIKE" and "NOT LIKE" operators respectively.

Look at the code that the framework uses to display the default dynamic filter.

```
//*********************************************************
//
/// <summary>
/// CallFilterDialog:           Popup the filter dialog
/// </summary>
/// <param name="filter">       Filter to adapt</param>
```

```
/// <param name="trans">        Control text translator</param>
/// <param name="configHelper"> Configuration helper</param>
/// <returns>
/// true if succeed, false if not
/// </returns>
//
//*********************************************************

public virtual bool CallFilterDialog(ISBFilter             filter,
                                     ControlTextTranslator trans,
                                     UIConfigurationHelper configHelper) {
    bool        bRetVal = false;
    frmSBFilter frmFilter;

    using(frmFilter = new frmSBFilter(filter as SBFilter, trans, configHelper, false)) {
        if (frmFilter.ShowDialog() == DialogResult.OK) {
            bRetVal = true;
        }
    }
    return(bRetVal);
}
```

The framework uses an **SBFilter** object to send a number of parameters (the RowsetDef used to generate the filter and an option that controls the fields to be displayed) to an **frmSBFilter** form. The **frmSBFilter** form uses the **SBFilter** object to generate the filter user interface dynamically. The **SBFilter** object is also used to return the WHERE clause that represents the filter specified by the end user.

The **frmSBFilter** form delegates most of its work to an **SBFilter** member object. The main methods and properties of the **SBFilter** class are shown below.

*Table 1. Main methods and properties of the **SBFilter** class.*

Property/Method	Description
Between	Property that returns or sets a string used to express the *Between* clause of the filter.
FillPanel	Fills a panel control with controls generated dynamically to build the filter form.
FilterClause	Property that returns the filter clause in a readable format. Usually used to display the active filter (if a filter was applied).
ParamRowset	Property that returns the Rowset containing the values specified for the filter.
ReadRowsetTreeUsingFilter	Reads the content of a RowsetTree using the WHERE clause of the current filter.
ResizeForm	Resizes a form to fit the automatically generated filter controls.
ShowUserFlagField	Property that returns or sets a flag indicating if the fields that set their user flag should be shown as part of the filter. Usually used to control the fields included in the filter.
TemplateRowsetDef	Property that returns or sets the RowsetDef used to generate the filter.
WhereClause	Property that returns or sets the actual WHERE clause of the filter.
WildCharAny	Property that returns or sets the character to be used as a wild character for any number of characters in a LIKE clause.
WildCharOne	Property that returns or sets the character to be used as a wild character for exactly one character in a LIKE clause.

Using the default filter form in an application that does not already use REP++framework

In this section, you will integrate the default filter form in a simple application that up to this point did not use the framework. The application will simply display a filter to let the user specify a search criteria used to read the content of a RowsetTree.

You will build on top of a form that uses the contact management demo system to read the list of clients. The original code of the form is shown below.

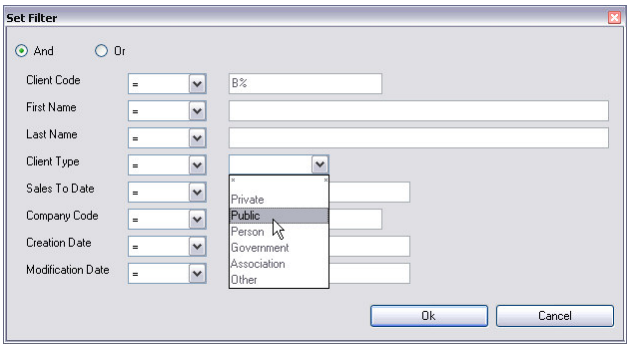
```
using System;
using System.Windows.Forms;

namespace FilterDemo {
  public partial class Form1 : Form {
    public Form1() {
      RepPP.Application       app;
      RepPP.RowsetTreeDef     rstDef;
      RepPP.RowsetTree        rst;

      InitializeComponent();
      app    = RepPP.Application.CreateFromRes();
      rstDef = app.RowsetTreeDefs["CLIENT"];
      rstDef.BuildSqlCommand();
      rst = rstDef.RowsetTrees.Add();
      rst.ReadFromDb();
      MessageBox.Show(rst.RootRowset.LineCount.ToString());
    }
  }
}
```

The code below indicates the steps needed to add a default filter form.

```
using System;
using System.Windows.Forms;
// Step 1 add a reference to the Rep++ framework
using RepPP.Framework.Window;

namespace FilterDemo {
    public partial class Form1 : Form {
        public Form1() {
            RepPP.Application       app;
            RepPP.RowsetTreeDef     rstDef;
            RepPP.RowsetTree        rst;
            SBFilter                filter;
            frmSBFilter             frmFilter;

            InitializeComponent();
            app    = RepPP.Application.CreateFromRes();
            rstDef = app.RowsetTreeDefs["CLIENT"];
            rstDef.BuildSqlCommand();
            rst = rstDef.RowsetTrees.Add();

            // Step 2: Define a selection buffer filter object to
            // specifiy the rowsetdef used to generate the filter and
            // whose fielddefs should be included
            filter = new SBFilter(rst.RootRowset.RowsetDef, true);

            // Step 3: Define a default filter form object
            using (frmFilter = new frmSBFilter(filter, null, null, false)) {

                // Step 4: display the default filter form to get
                // the filter values from the end user
                if (frmFilter.ShowDialog(this) == DialogResult.OK) {

                    //Step 5: Read the clients using the filter specified
```

```
                    // by the end user
                    filter.ReadRowsetTreeUsingFilterV(rst, string.Empty);
                }
            }

            MessageBox.Show(rst.RootRowset.LineCount.ToString());
        }
    }
}
```

Set this form as the current form, then build and run your application. The default filter form displays the list of fields in the template RowsetDef you specified.



To exclude a field of the template RowsetDef from the filter, simply set the user flag of the excluded field, and then set the ***SBFilter.ShowUserFlagField*** property to false. You can set the ***SBFilter.ShowUserFlagField*** property at creation time using the second parameter of the constructor as follows:

```
// When the second parameter equals false, Fields marked with the user flag
// in the Rep++ repository will not be shown in the filter form.
filter = new SBFilter(rst.RootRowset.RowsetDef, false);
```

Defining a custom filter form

In this section, you will create your own filter form by deriving from and customizing the default filter form.

1. Add a new form to your project and name it **frmMyFilterForm**.
2. Derive the newly added form from **frmSBFilter** and add the following constructors. By deriving from **frmSBFilter**, you inherit all the user interface and behaviour of the default filter form.

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

namespace FilterDemo {
  public partial class frmMyFilterForm : frmSBFilter {
    public frmMyFilterForm(SBFilter              filter,
                           ControlTextTranslator translator,
                           UIConfigurationHelper cfgHelper,
                           bool                  bAddFilter) : base(filter,
                                                                    translator,
                                                                    cfgHelper,
                                                                    bAddFilter) {
      InitializeComponent();
```

```
    }
    public frmMyFilterForm() : this(null, null, null, false) {
    }
  }
}
```

3. Open the form designer and change the user interface of the default filter form to match the following figure.

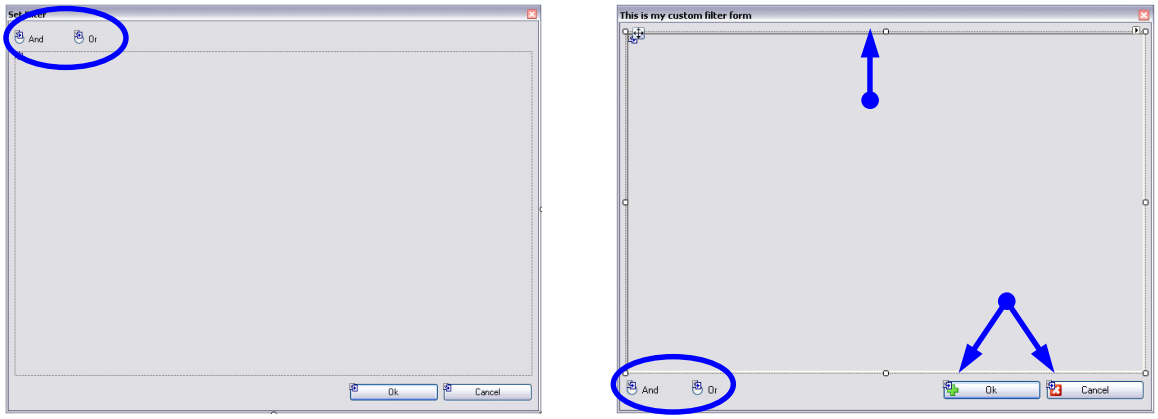
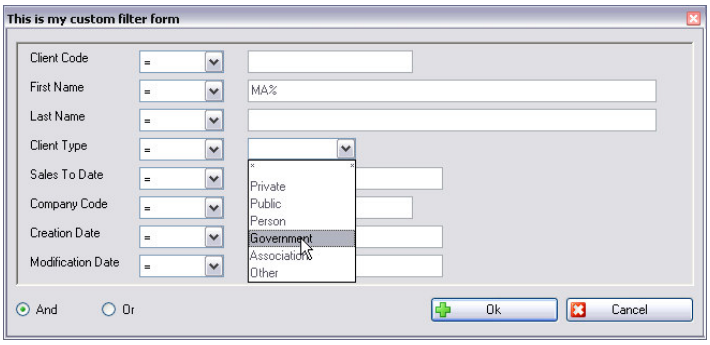


Figure 1. The default filter form user interface (left), and the custom filter form user interface (right). Don't forget to change the Anchor property of the radio button from Top to Bottom.

4. Change the code of *Form1* to use **frmMyFilterForm** instead of **frmSBFilter**.

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

namespace FilterDemo {
    public partial class Form1 : Form {
        public Form1() {
            RepPP.Application       app;
            RepPP.RowsetTreeDef     rstDef;
            RepPP.RowsetTree        rst;
            SBFilter                filter;
            frmMyFilterForm         frmFilter;

            InitializeComponent();
            app    = RepPP.Application.CreateFromRes();
            rstDef = app.RowsetTreeDefs["CLIENT"];
            rstDef.BuildSqlCommand();
            rst    = rstDef.RowsetTrees.Add();
            filter = new SBFilter(rst.RootRowset.RowsetDef, true);
            using (frmFilter = new frmMyFilterForm(filter, null, null, false)) {
                if (frmFilter.ShowDialog(this) == DialogResult.OK) {
                    filter.ReadRowsetTreeUsingFilterV(rst, string.Empty);
                }
            }
            MessageBox.Show(rst.RootRowset.LineCount.ToString());
        }
    }
}
```

5. Build and run your application. You should see the following filter form.



Replacing the default filter form in an application that uses REP++framework

You have seen previously that the framework uses a default filter form along with the selection buffer RowsetDef to display a default filter user interface.

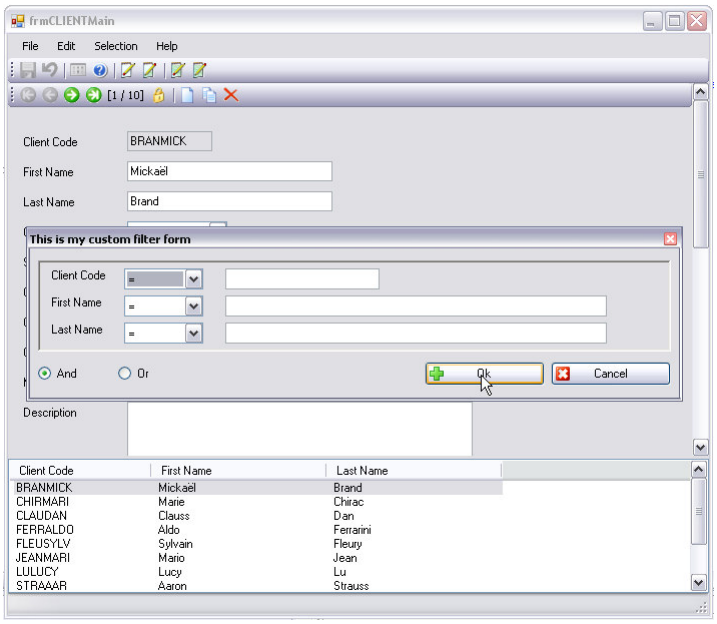
In this section you will learn how to:

- Change the default filter form.
- Change the RowsetDef used to generate the filter form.
- Control which fields should be displayed in the filter form.

You will start by modifying the main form of the application that you have generated earlier so that it uses the custom filter that you have created in the previous section. As you have seen, the framework wraps the creation of the filter form in a virtual (i.e. overridable) method called **CallFilterDialog**. You will override the default behaviour of the **CallFilterDialog** method in the **frmCLIENTMain** form so that it creates and displays your custom filter.

```
…

namespace FilterDemo {
  public partial class frmCLIENTMain : frmSBEditorListView {
…
    public override bool CallFilterDialog(ISBFilter             filter,
                                          ControlTextTranslator trans,
                                          UIConfigurationHelper configHelper) {
      bool            bRetVal = false;
      frmMyFilterForm frmFilter;

      using (frmFilter = new frmMyFilterForm(filter as SBFilter, null, null, false)) {
        if (frmFilter.ShowDialog() == DialogResult.OK) {
          bRetVal = true;
        }
      }
      return (bRetVal);
    }
  }
}
```

Set the **frmCLIENTMain** form as the startup form, then build and run your application. From the **Selection** menu, click **Set filter** and see the results.

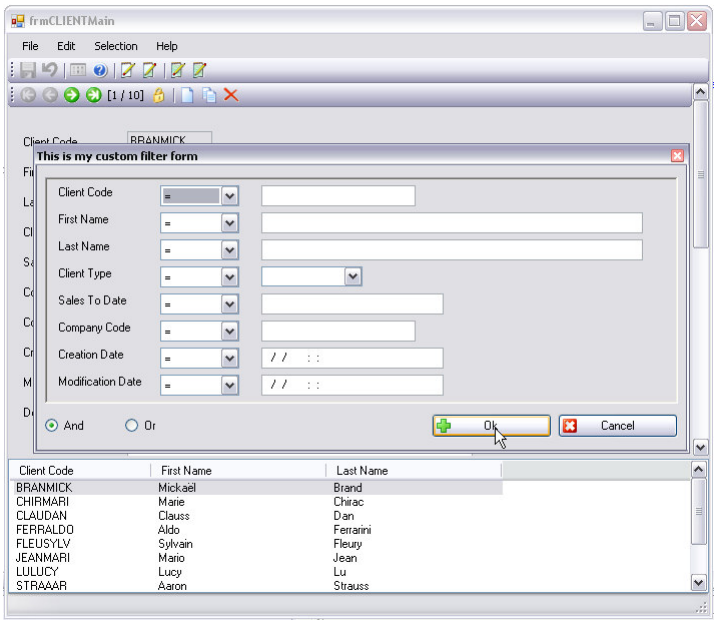


As you can see, you have successfully replaced the default filter form with your own filter form.

The default behaviour of the framework is to use the RowsetDef of the selection buffer to build the filter. What if you wanted to allow the end user to filter on any field of the transaction's root RowsetDef (i.e. CLIENT)? You can easily change the template RowsetDef of the filter as follows.

```
public override bool CallFilterDialog(ISBFilter             filter,
                                      ControlTextTranslator trans,
                                      UIConfigurationHelper configHelper) {
  bool            bRetVal = false;
  frmMyFilterForm frmFilter;

  (filter as SBFilter).TemplateRowsetDef = Application.RowsetDefs["CLIENT"];
  using (frmFilter = new frmMyFilterForm(filter as SBFilter, null, null, false)) {
    if (frmFilter.ShowDialog() == DialogResult.OK) {
      bRetVal = true;
    }
  }
  return (bRetVal);
}
```

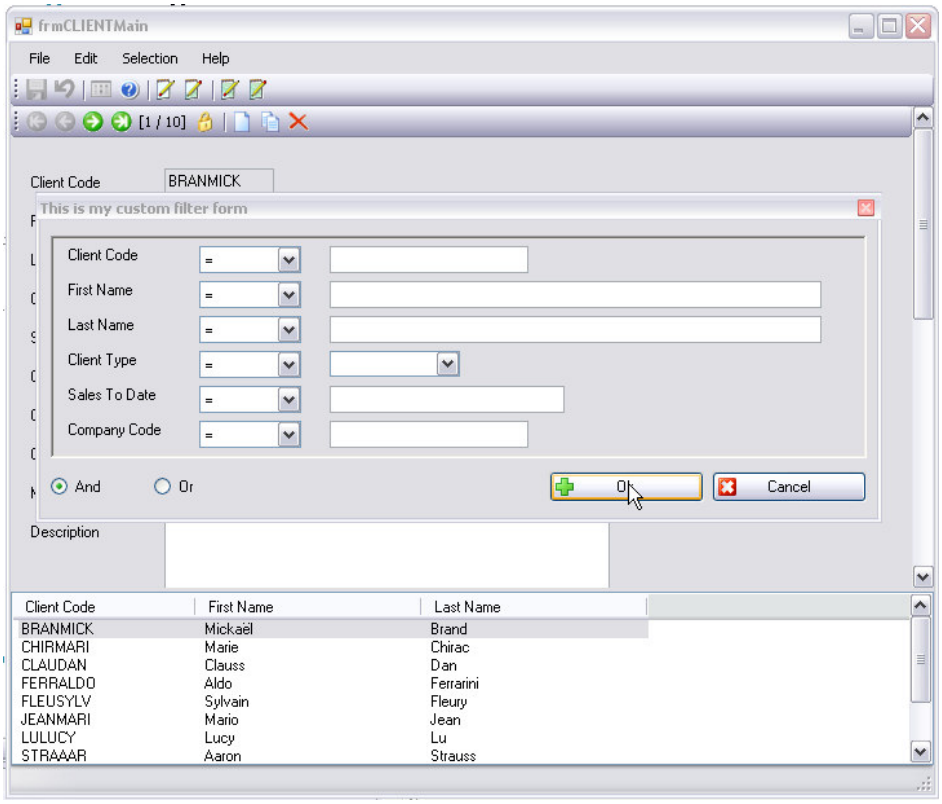
Build and run your application. The following form is now displayed, with all the fields from the CLIENT RowsetDef.



To further customize your filter by excluding the *Creation Date* and the *Modification Date* fields, simply set the user flag for the two fields and instruct your filter not to show fields that have their user flag set.

```
public override bool CallFilterDialog(ISBFilter             filter,
                                      ControlTextTranslator trans,
                                      UIConfigurationHelper configHelper) {
  bool            bRetVal = false;
  frmMyFilterForm frmFilter;

  (filter as SBFilter).ShowUserFlagField  = false;
  (filter as SBFilter).TemplateRowsetDef = Application.RowsetDefs["CLIENT"];
  using (frmFilter = new frmMyFilterForm(filter as SBFilter, null, null, false)) {
    if (frmFilter.ShowDialog() == DialogResult.OK) {
      bRetVal = true;
    }
  }
  return (bRetVal);
}
```

The two fields do not appear any more in the new filter form.



You can also view and modify the WHERE clause resulting from displaying the filter form. The debugger snapshot below shows the WHERE clause of a sample filter run.