Technical Note

Implementing Drag-and-Drop with the REP++toolkit TreeView Control

Author: R&D Department

Publication date: January 29, 2007

Revision date: December 2010

Implementing Drag-and-Drop with the REP++toolkit TreeView Control

Overview

The **TreeViewRowsetTree** class of the REP++toolkit is a subclass of **System.Windows.Forms.TreeView**. The toolkit's treeview control can be populated automatically by attaching it to a REP++ RowsetTree object. The **TreeViewRowsetTree** class facilitates the implementation of the drag-and-drop function of tree nodes by managing automatically a great portion of the job.

This article describes how to implement the nodes' drag-and-drop function with the REP++toolkit **TreeViewRowsetTree** control.

Note: code samples are given in C#.

Managing automatically the drag-and-drop function

To facilitate the implementation of the drag-and-drop function in a treeview, the **TreeViewRowsetTree** control catches the drag-and-drop user events by overriding some of the methods of the base class. These methods are:

- **OnItemDrag**: Occurs when the user begins dragging a node.
- **OnDragEnter**: Occurs when an object is dragged into the control's bounds.
- **OnDragOver**: Occurs when an object is dragged over the control's bounds.
- **OnGiveFeedback**: Occurs during a drag operation.
- **OnDragDrop**: Occurs when a drag-and-drop operation is completed.

When the drop occurs, the REP++toolkit treeview control calls the **CanDropThere** method to check if the operation can be completed. This method validates the drop operation based on the hierarchical structure of the RowsetTreeDef attached to the treeview. Take the example of a CLIENT RowsetTreeDef, which represents the hierarchical relationship between the CLIENT, ADDRESS and PHONE RowsetTreeDefNodes. When the treeview is populated with data, a user cannot drop an ADDRESS node on another ADDRESS node or on a PHONE node. An ADDRESS node can only be dropped on a CLIENT node. The **CanDropThere** method is a public and virtual method; the user can override it to customize the validation.

When the drop is permitted, the treeview calls the **DropNode** method to move or copy the node in the treeview and to update the data structures in memory (the Rowset objects) related to the treeview. The treeview is not responsible for updating the database. However, the **DropNode** method raises some events that the user can catch to update it. These events are:

- **BeforeNodeMoved**: Occurs before the node is moved.
- **BeforeNodeCopied**: Occurs before the node is copied.
- **AfterNodeMoved**: Occurs after the node is moved.
- **AfterNodeCopied**: Occurs after the node is copied.

Note that the automatic management of the drag-and-drop feature is activated only when the properties **AllowDrop** and **AutoDragDrop** of the treeview control are set to *true*.

Example

In this section, you will build an example to use the automatic management of the drag-and-drop function with the treeview control of the REP++toolkit. This example uses the DEMO system that is provided with the REP++ user training or reference guides.

1. Create a new Windows® application.
2. Add the **RepPP.Toolkit.Window** namespace to form **Form1**.

```
using RepPP.Toolkit.Window;
```

3. In **Form1**, add a **RepPPInfo** component and set the REP++ connection parameters.
4. Add a REP++toolkit treeview control, i.e. **TreeViewRowsetTree**.
5. Set the **AutoDragDrop** and **AllowDrop** properties of the treeview to *true*.
6. Add two members to **Form1** as follows:

```
/// <summary>
/// Rep++ application object
/// </summary>
private RepPP.Application         m_app;
/// <summary>
/// Rep++ RowsetTree object
/// </summary>
private RepPP.RowsetTree          m_rsTree;
```

7. Modify the constructor of **Form1** to load a REP++ **Application** object, load the clients from the database, and populate the treeview with those clients.

```
public Form1() {
    RepPP.RowsetTreeDef rsTreeDef;

    InitializeComponent();
    m_app = RepPP.Application.CreateFromRes();
    rsTreeDef = m_app.RowsetTreeDefs["CLIENT"];
    rsTreeDef.BuildSqlCommand();
    m_rsTree = rsTreeDef.RowsetTrees.Add();
    m_rsTree.ReadFromDb();
    treeViewRowsetTree1.FillNodes(m_rsTree.RootRowset, null);
}
```

8. Modify the **Dispose** method of **Form1** to delete resources.

```
protected override void Dispose(bool disposing) {
    if (disposing) {
        if (components != null) {
            components.Dispose();
        }
        if (m_rsTree != null) {
            m_rsTree.Dispose();
            m_rsTree = null;
        }
        if (m_app != null) {
            m_app.Dispose();
            m_app = null;
        }
    }
    base.Dispose(disposing);
}
```

9. To save the data to the database each time a node is moved using the drag-and-drop feature, you need to catch the **AfterNodeMoved** event of the treeview and write the necessary code to save the data. The following code sample shows an example of how to save the data each time a node of type ADDRESS is moved and attached to a new CLIENT node.

```
private void treeViewRowsetTree1_AfterNodeMoved(object sender,
                                                TreeViewRowsetTree.AfterNodeMovedEventArgs e)
{
     RepPP.Rowset   rowsetNodeParent;
     RepPP.Rowset   rowsetNode;
     RepPP.Field    fld;
     string         strNodeID;
     string         strNodeParentID;
     string         strCmdSQL;
     int            iLineBookmark;

     rowsetNode = (e.MovedNode as TreeNodeRowset).Rowset;

     if (rowsetNode.RowsetDef.Name == "ADDRESS") {
         // Retrieve the address code for the moved node
         fld = rowsetNode.Fields["ADDRESSCODE"];
         iLineBookmark = (e.MovedNode as TreeNodeRowset).LineBookmark;
         strNodeID = fld.GetValue(rowsetNode.LineFromBookmark(iLineBookmark), false);
         // Retrieve the new client code
         rowsetNodeParent = (e.MovedNode.Parent as TreeNodeRowset).Rowset;
         fld = rowsetNodeParent.Fields["CLIENTCODE"];
         iLineBookmark = (e.MovedNode.Parent as TreeNodeRowset).LineBookmark;
         strNodeParentID = fld.GetValue(rowsetNodeParent.LineFromBookmark(iLineBookmark),
                                        false);

         // Update the database to set a new client to the address
         strCmdSQL = "UPDATE DEMO_ADDRESS SET CLIENTCODE='{0}' WHERE ADDRESSCODE='{1}'";
         strCmdSQL = string.Format(strCmdSQL, strNodeParentID, strNodeID);
         if (m_app.DataConnection.Execute(strCmdSQL) != 0) {
             MessageBox.Show(m_app.Tool.DBErrorMessage, "DataBase Error");
         }
     }
 }
```

   In this example, the ADDRESSCODE value of the moved node and the CLIENTCODE value of the new parent node are used to update the DEMO_ADDRESS table by setting the new CLIENTCODE value of the ADDRESS node.

10. Now, execute the program and try to move ADDRESS nodes using your new drag-and-drop feature.

This example only handles ADDRESS nodes. You can similarly implement this feature for PHONE nodes.