Technical Note

Representing Recursive Relationships Using REP++ TreeView

Author(s): R&D Department

Publication date: May 4, 2006

Revision date: May 2010

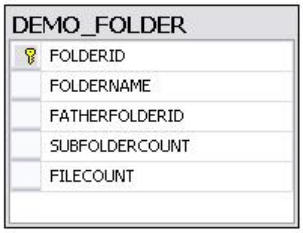
Representing Recursive Relationships Using REP++ TreeView

Overview

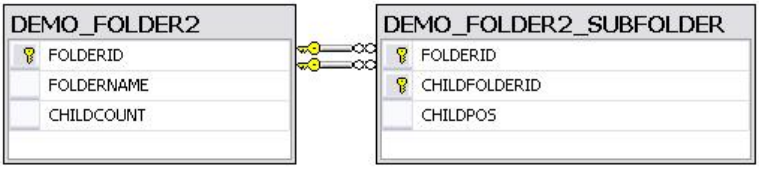
Recursive relationships are quite common in applications and need to be properly presented. To facilitate this process, the REP++*toolkit* for Windows includes a **TreeViewRowsetTree** component (or control) that already supports recursive relationships. **TreeViewRowsetTree** is a subclass of the .NET Windows Forms **TreeView** component that is capable of interacting with the REP++ metadata. This article describes how to use the **TreeViewRowsetTree** component to represent recursive relationships.

Direct vs. indirect recursion

There are two ways to define recursion within a relational database: direct and indirect. A direct recursion is defined in the same table as the recursive entities. For instance, to define a direct recursion between folders, we need to add a column to the folder table so that every folder keeps a reference to its containing folder. The following figure illustrates a direct recursion between folders:



An indirect recursion is defined in a separate table whose sole purpose is to represent the relationship between the recursive entities. For instance, to define an indirect recursion between folders, we need to create a separate table to keep a reference to every folder contained within a specific folder. The following figure illustrates an indirect recursion between folders:



The TreeViewRowsetTree component

To present recursion using a **TreeViewRowsetTree** component, you will follow the same strategy regardless of the type of recursion. The following tasks have to be done:

- Subclassing the **TreeViewRowsetTree** component.
- Retrieving the root entities (i.e. root folders).
- Defining the recursion by telling each node how to retrieve its descendants.

Before you go on with the rest of the document, please make sure that you go over the list of pre-requisites in the appendix (page 8) to:

1. Create and populate the sample database that you will use throughout the document.
2. Create and populate the REP++*repository*.
3. Create a project and generate the **TypedInstances**.

Subclassing the TreeViewRowsetTree component for direct recursion

The first step is to define a new class, **TreeViewRecDirect**, that inherits from the **TreeViewRowsetTree** class.

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using RepPP;
using RepPP.Toolkit.Window;

namespace Recursion {
    public partial class TreeViewRecDirect : TreeViewRowsetTree {
```

Retrieving the root entities

You will define a method that enables the hosting Windows form to initialize your **TreeViewRecDirect** control and have it load the root entities. The **TreeViewRecDirect** control will use the typed RowsetTree **RTCFolder** to load the data:

```
public class TreeViewRecDirect : TreeViewRowsetTree {
    private RTCFOLDER m_rstRoot;

    public TreeViewRecDirect() {
        InitializeComponent();
        m_rstRoot = null;
    }

    public void Init(RepPP.Application app) {
        RowsetTreeDefNode nodeFolder;

        if (m_rstRoot == null) {
            m_rstRoot = RTCFOLDER.Create(app);
        }
        nodeFolder = m_rstRoot.rsFOLDER.RowsetTreeDefNode;
        m_rstRoot.RowsetTreeDef.BuildSqlCommand();
        nodeFolder.SetSelectSqlOrderBy("ORDER BY FOLDERNAME");
        nodeFolder.SetSelectSqlWhere("WHERE FATHERFOLDERID IS NULL");
        Empty();
        m_rstRoot.ReadFromDb();
        FillNodes(m_rstRoot.rsFOLDER, null);
    }

    protected override void Dispose(bool disposing) {
        if (disposing) {
```

```
            if (m_rstRoot != null) {
                m_rstRoot.Dispose();
                m_rstRoot = null;
            }
        }
        base.Dispose(disposing);
    }
…
}
```

As you can see, **Init** uses the REP++ **Application** object to read the sorted list of folders that do not have a parent folder, and then asks the tree to populate its nodes using the result.

Defining a direct recursion

Next, you need to tell the nodes of your **TreeViewRecDirect** control how to recursively get their descendants, and which icon they should display. This task will be accomplished by overriding **OnSetNodeInfo** as follows:

```
public override void OnSetNodeInfo(SetNodeInfoEventArgs e) {
  RepPP.Toolkit.Window.TreeNodeRowset node;
  RCFOLDER                            rsFolder;

  node = e.Node as RepPP.Toolkit.Window.TreeNodeRowset;
  node.NodeType = (int)FindNodeType(node);
  switch ((NodeType)node.NodeType) {
  case NodeType.Folder:

    // Sets the bitmap of the node
    node.ImageIndex = 1;
    node.SelectedImageIndex = 1;

    // Tells the node how to get its descendants and in which order
    node.SetRecursive1Level("WHERE FATHERFOLDERID=:FOLDER.FOLDERID",
                            "ORDER BY FOLDERNAME");

    // This is an optimization that prevents us from displaying
    // an expandable node when it does not have any descendants
    rsFolder = node.Rowset as RCFOLDER;
    if (rsFolder.fldFILECOUNT.TypedValue == 0 &&
      rsFolder.fldSUBFOLDERCOUNT.TypedValue == 0) {
      node.Nodes.Clear();
    }
    break;
  case NodeType.File:

    // Sets the bitmap of the node
    node.ImageIndex = 0;
    node.SelectedImageIndex = 0;

    // File nodes are not recursive, so there is nothing left to do
    break;
  }

  // DO NOT forget to call the base method
  base.OnSetNodeInfo(e);
}
```

Since this example uses direct recursion, **SetRecursive1Level** does not require the name of a different RowsetDef to load the descendants. Instead, it uses a simple WHERE clause to load all the folders whose parent is the folder represented by the node in question.

OnSetNodeInfo uses a helper method named **FindNodeType** and an enumeration named **NodeType** to distinguish between *File* and *Folder* nodes:

```
public class TreeViewRecDirect : TreeViewRowsetTree {

    …
    public enum NodeType {
        Unknown = 0,
        Folder = 1,
        File = 2
    };
    …

    private NodeType FindNodeType(TreeNode node) {
        NodeType eRetVal = NodeType.Unknown;
        TreeNodeRowset nodeRowset;

        nodeRowset = node as TreeNodeRowset;
        if (nodeRowset != null) {
            if (nodeRowset.Rowset.RowsetDef == m_rstRoot.rsFOLDER.RowsetDef) {
                eRetVal = NodeType.Folder;
            } else {
                eRetVal = NodeType.File;
            }
        }
        return (eRetVal);
    }
}
```

The **TreeViewRecDirect** control is now complete.

Subclassing the TreeViewRowsetTree component for indirect recursion

The **TreeViewRecIndirect** component will be almost identical to **TreeViewRecDirect**. The difference is in the way you retrieve root items and the way you load the descendants of a node. Since the two controls are very similar, the entire class definition of **TreeViewRecIndirect** is shown below, with the differences highlighted:

```
public class TreeViewRecIndirect : TreeViewRowsetTree {
 private RTCFOLDER2 m_rstRoot;

 public enum NodeType {
     Unknown = 0,
     Folder = 1,
     File = 2
 };

 public TreeViewRecIndirect() {
     InitializeComponent();
     m_rstRoot = null;
 }

 public void Init(RepPP.Application app) {
     RowsetTreeDefNode nodeFolder2;

     if (m_rstRoot == null) {
         m_rstRoot = RTCFOLDER2.Create(app);
     }
     nodeFolder2 = m_rstRoot.rsFOLDER2.RowsetTreeDefNode;
     m_rstRoot.RowsetTreeDef.BuildSqlCommand();
     nodeFolder2.SetSelectSqlOrderBy("ORDER BY FOLDERNAME");
     nodeFolder2.SetSelectSqlWhere("WHERE FOLDERID NOT IN (SELECT CHILDFOLDERID " +
                                   "FROM DEMO_FOLDER2_SUBFOLDER)");
     Empty();
     m_rstRoot.ReadFromDb();
     FillNodes(m_rstRoot.rsFOLDER2, null);
 }
```

```
 private NodeType FindNodeType(TreeNode node) {
     NodeType eRetVal = NodeType.Unknown;
     TreeNodeRowset nodeRowset;

     nodeRowset = node as TreeNodeRowset;
     if (nodeRowset != null) {
         if (nodeRowset.Rowset.RowsetDef == m_rstRoot.rsFOLDER2.RowsetDef) {
             eRetVal = NodeType.Folder;
         } else {
             eRetVal = NodeType.File;
         }
     }
     return (eRetVal);
 }

 public override void OnSetNodeInfo(SetNodeInfoEventArgs e) {
  RepPP.Toolkit.Window.TreeNodeRowset node;
  RCFOLDER2                           rsFolder2;

  node = e.Node as RepPP.Toolkit.Window.TreeNodeRowset;
  node.NodeType = (int)FindNodeType(node);
  switch ((NodeType)node.NodeType) {
  case NodeType.Folder:
   node.ImageIndex = 1;
   node.SelectedImageIndex = 1;
   node.SetRecursive2Level(m_rstRoot.RowsetTreeDef.FindNode("FOLDER2_SUBFOLD").RowsetDef,
                           "ORDER BY FOLDERNAME",
                           new string[] { "CHILDFOLDERID" });
   rsFolder2 = node.Rowset as RCFOLDER2;
   if (rsFolder2.fldCHILDCOUNT.TypedValue == 0) {
     node.Nodes.Clear();
   }
   break;
  case NodeType.File:
   node.ImageIndex = 0;
   node.SelectedImageIndex = 0;
   break;
  }
  base.OnSetNodeInfo(e);
 }

    protected override void Dispose(bool disposing) {
        if (disposing) {
            if (m rstRoot != null) {
                m_rstRoot.Dispose();
                m_rstRoot = null;
            }
        }
        base.Dispose(disposing);
    }
}
```

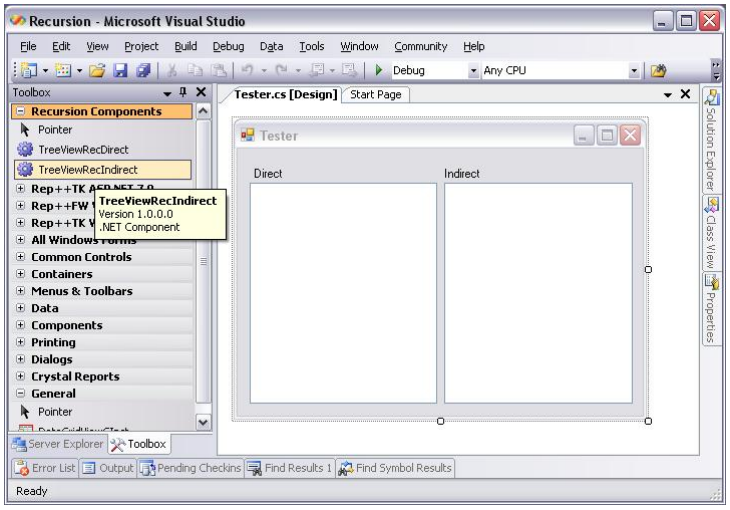
The table DEMO_FOLDER2_SUBFOLDER contains the recursive relation between the folders in DEMO_FOLDER2. Because of this:

- To get the root items, **TreeViewRecIndirect** loads all the folders that are not specified as child folders in DEMO_FOLDER2_SUBFOLDER.
- **SetRecursive2Level** requires a RowsetDef representing DEMO_FOLDER2_SUBFOLDER to get the IDs of the descendant folders. It then uses these IDs to query DEMO_FOLDER2 and populate the descendant nodes.

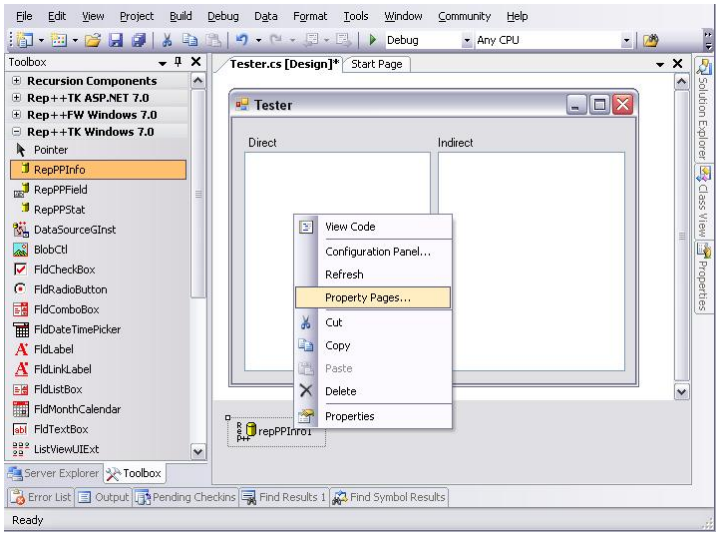
Testing the TreeViewRecDirect and TreeViewRecIndirect components

The final step is to test the two recursive **TreeView** components.

1. Add a new Windows form to your project and name it *Tester.cs*.
2. Add your **TreeViewRecDirect** and **TreeViewRecIndirect** components:



3. Add a **RepPPInfo** component and make sure that you configure your connection properly:

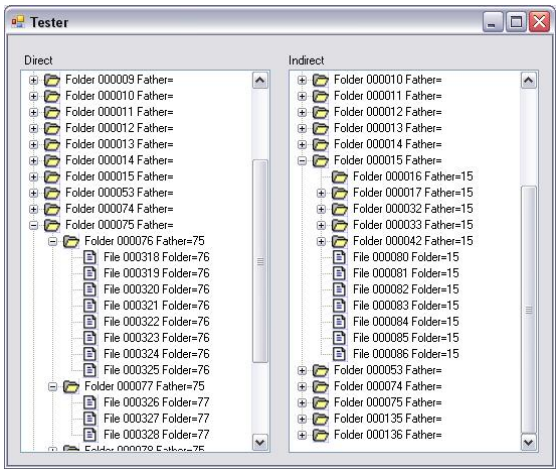


4. Add an **ImageList** control containing two images and associate it with your recursive **TreeView**s.

5. Modify the class of the new Web form as indicated below, and set it as the startup form in *Program.cs*:

```
public partial class Tester : Form
{
    private RepPP.Application m_app;
    public Tester()
    {
        InitializeComponent();
        m_app = RepPP.Application.CreateFromRes();
        m_app.TypedInstance.RegisterAll(true);
        foreach (RepPP.RowsetTreeDef rstDef in m_app.RowsetTreeDefs) {
            rstDef.BuildSqlCommand();
        }
        treeViewRecDirect1.Init(m_app);
        treeViewRecIndirect1.Init(m_app);
    }
}
```

Your form should be similar to the following:

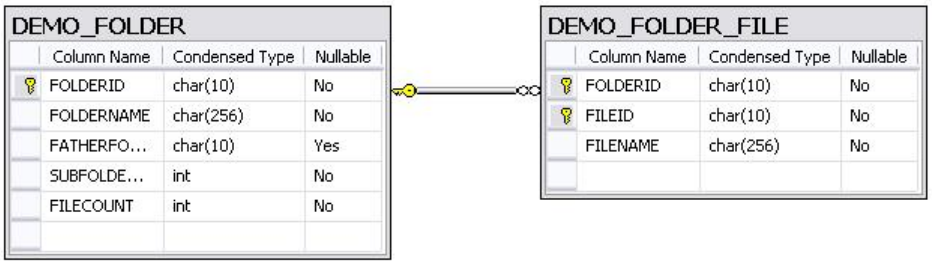


NOTE: TreeViewRowset uses delay reading so that only the displayed nodes are read from the database. The descendants of a node are read only when the node is expanded. This improves performance by reading only the required nodes and enhances the user experience by decomposing the waiting time per expanded node instead of waiting for all nodes to be loaded at startup.

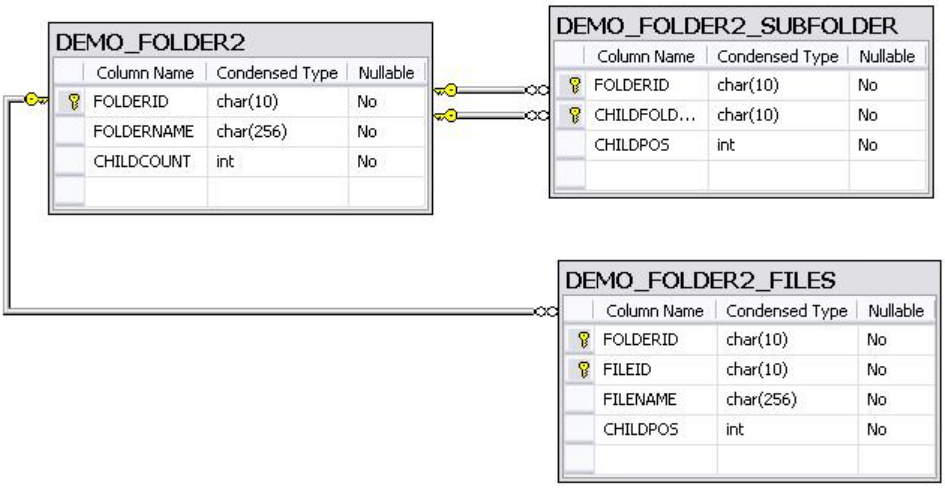
Appendix

Create the database

1. Create a new database and name it *TreeView*.
2. Create the following tables to test the direct recursion:



3. Now, create the following tables to test the indirect recursion:



Populate the Database

To facilitate the population of the database, we have created a script that automates the process. As a result, you can simply add a new button to any Windows form and have it execute the following code: (**butFillTables_Click** handles the click event of the button that populates the database)

```
private int AddFiles(Random rand, string strFolderId,
                     RCFOLDER2_FILES rsFile, ref int iFileId) {
  int iCount;

  iCount = rand.Next(10);
  for (int i = 0; i < iCount; i++) {
    rsFile.AddBlankLine();
```

```
    rsFile.fldFILEID.Value = iFileId.ToString();
    rsFile.fldFILENAME.Value = "File " +
                               iFileId.ToString().PadLeft(6, '0') +
                               " Folder=" + strFolderId;
    rsFile.fldCHILDPOS.ValueAsInt = i + 1;
    iFileId++;
  }
  return (iCount);
}

private int AddFiles(Random rand, string strFolderId,
                     RCFOLDER_FILE rsFile, ref int iFileId) {
  int iCount;

  iCount = rand.Next(10);
  for (int i = 0; i < iCount; i++) {
    rsFile.AddBlankLine();
    rsFile.fldFILEID.Value = iFileId.ToString();
    rsFile.fldFILENAME.Value = "File " + iFileId.ToString().PadLeft(6, '0') +
                               " Folder=" + strFolderId;
    iFileId++;
  }
  return (iCount);
}

private void CreateFolder(RTCFOLDER2 rstFolder2Father, Random rand,
                          ref int iFolderId, ref int iFileId, int iPos) {
  RTCFOLDER2        rstFolder2;
  RCFOLDER2         rsFolder2;
  RCFOLDER2_SUBFOLD rsSubFolder;
  string            strFatherFolderId = "";
  int               iFileCount;
  int               iSubCount;

  if (rstFolder2Father != null) {
    rsSubFolder = rstFolder2Father.rsFOLDER2_SUBFOLD;
    rsSubFolder.AddBlankLine();
    rsSubFolder.fldCHILDFOLDERID.ValueAsInt = iFolderId + 1;
    rsSubFolder.fldCHILDPOS.ValueAsInt = iPos;
    strFatherFolderId = rstFolder2Father.rsFOLDER2.fldFOLDERID.ValueAsInt.ToString();
  }
  using (rsFolder2 = RTCFOLDER2.Create(m_app)) {
    if (rstFolder2Father != null) {
      strFatherFolderId = rstFolder2Father.rsFOLDER2.fldFOLDERID.ValueAsInt.ToString();
    }
    iSubCount = (rand.Next(100) > 75) ? rand.Next(10) : 0;
    rsFolder2 = rstFolder2.rsFOLDER2;
    iFolderId++;
    rsFolder2.AddBlankLine();
    iFileCount = AddFiles(rand, iFolderId.ToString(),
                          rstFolder2.rsFOLDER2_FILES, ref iFileId);
    rsFolder2.fldFOLDERID.ValueAsInt = iFolderId;
    rsFolder2.fldFOLDERNAME.Value = "Folder " + iFolderId.ToString().PadLeft(6, '0') +
                                    " Father=" + strFatherFolderId;
    rsFolder2.fldCHILDCOUNT.ValueAsInt = iFileCount + iSubCount;
    for (int i = 0; i < iSubCount; i++) {
      CreateFolder(rstFolder2, rand, ref iFolderId, ref iFileId, i + 1);
    }
    rstFolder2.UpdateToDb();
    m_app.DataConnection.Commit();
  }
}

private void CreateFolder(RTCFOLDER rstFolder, Random rand, string strFatherFolderId,
                          ref int iFolderId, ref int iFileId) {
  RCFOLDER rsFolder;
  int      iFileCount;
  int      iSubCount;

  iSubCount = (rand.Next(100) > 75) ? rand.Next(10) : 0;
  rstFolder.Empty();
```

```
  rsFolder = rstFolder.rsFOLDER;
  iFolderId++;
  rsFolder.AddBlankLine();
  iFileCount = AddFiles(rand, iFolderId.ToString(),
                        rstFolder.rsFOLDER_FILE, ref iFileId);
  rsFolder.fldFOLDERID.ValueAsInt = iFolderId;
  rsFolder.fldFOLDERNAME.Value = "Folder " + iFolderId.ToString().PadLeft(6, '0') +
                                 " Father=" + strFatherFolderId;
  rsFolder.fldFATHERFOLDERID.Value = strFatherFolderId;
  rsFolder.fldFILECOUNT.ValueAsInt = iFileCount;
  rsFolder.fldSUBFOLDERCOUNT.ValueAsInt = iSubCount;
  rstFolder.UpdateToDb();
  m_app.DataConnection.Commit();
  strFatherFolderId = iFolderId.ToString();
  for (int i = 0; i < iSubCount; i++) {
    CreateFolder(rstFolder, rand, strFatherFolderId, ref iFolderId, ref iFileId);
  }
}

private void butFillTables_Click(object sender, EventArgs e) {
  RepPP.Connection conn;
  RCFOLDER rstFolder;
  int iFolderCount;
  int iFileCount;
  Random rand;

  conn = m_app.DataConnection;
  using (rstFolder = RTCFOLDER.Create(m app)) {
    conn.Execute("DELETE FROM DEMO_FOLDER_FILE");
    conn.Execute("DELETE FROM DEMO_FOLDER");
    conn.Commit();
    iFolderCount = 0;
    iFileCount = 0;
    rand = new Random(1);
    m_app.Tool.SetWaitCursor(true);
    for (int i = 0; i < 20; i++) {
      CreateFolder(rstFolder, rand, null, ref iFolderCount, ref iFileCount);
    }
    m_app.Tool.SetWaitCursor(false);
  }
  conn.Execute("DELETE FROM DEMO_FOLDER2_FILES");
  conn.Execute("DELETE FROM DEMO_FOLDER2_SUBFOLDER");
  conn.Execute("DELETE FROM DEMO_FOLDER2");
  conn.Commit();
  iFolderCount = 0;
  iFileCount = 0;
  rand = new Random(1);
  m_app.Tool.SetWaitCursor(true);
  for (int i = 0; i < 20; i++) {
    CreateFolder(null, rand, ref iFolderCount, ref iFileCount, i + 1);
  }
  m_app.Tool.SetWaitCursor(false);
  MessageBox.Show(iFolderCount.ToString() + " folders and " + iFileCount.ToString() +
                  " files added to the tables");
}
```

The code above requires a REP++ **Application** object. Make sure that you have a REP++ **Application** object that has been initialized properly before **butFillTables_Click** is called:

```
private RepPP.Application m_app;

public YourFormConstructor() {
    …
    …
    m_app = RepPP.Application.CreateFromRes();
    m app.TypedInstance.RegisterAll(true);
    foreach (RepPP.RowsetTreeDef rstDef in m_app.RowsetTreeDefs) {
        rstDef.BuildSqlCommand();
    }
```

```
    …
    …
}
```

Create a REP++repository

1. Create a REP++connection for your **TreeView** database.
2. Create a repository and make sure that your repository can be edited using REP++*studio.*
3. Open your repository in REP++*studio* and create a new system named TREEVIEW.
4. Using the update wizard, import the definition of your tables.
5. Create the fields for every imported column.
6. Create a new program named TV_PRG.

To test the direct recursion

1. In your TREEVIEW system, create the following Rowsets that contain the listed fields:

Rowset	Field
FOLDER	
	FOLDERID
	FOLDERNAME (Check the "User Defined Flag" attribute)
	FATHERFOLDERID
	SUBFOLDERCOUNT
	FILECOUNT
FOLDER_FILE	
	FILEID
	FILENAME (Check the "User Defined Flag" attribute)

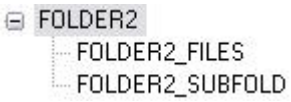
2. Create a new RowsetTree named FOLDER that contains the following Rowsets:

To test the indirect recursion

1. Create the following Rowsets that contain the listed fields:

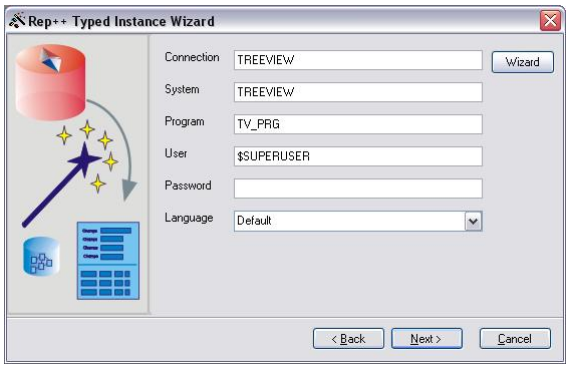
Rowset	Field
FOLDER2	
	FOLDERID
	FOLDERNAME (Check the "User Defined Flag" attribute)
	CHILDCOUNT
FOLDER2_FILES	
	FILEID
	FILENAME (Check the "User Defined Flag" attribute)
	CHILDPOS
FOLDER2_SUBFOLD	
	CHILDFOLDERID
	CHILDPOS

2. Create a new RowsetTree named FOLDER2 that contains the following Rowsets:

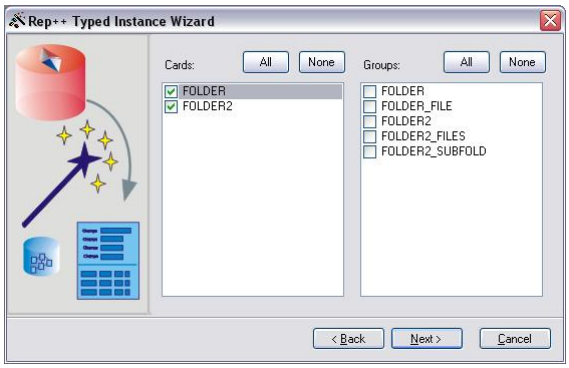


Create a project and generate the TypedInstances

1. In Visual Studio®, create a new Rep++Windows Application project named *Recursion*.
2. In the **Project** menu, choose **Add New Item**.
3. In the *Add New Item* window, select **Rep++ Typed Instance**.
4. When the wizard starts, make sure that you select the TREEVIEW connection and the TV_PRG program.



5. Generate the typed instances only for the FOLDER and FOLDER2 RowsetTrees.



6. Accept all the remaining default values to have the wizard generate a folder named **TypedInstances** that contains *RTCFOLDER.cs* and *RTCFOLDER2.cs*