User's Guide – TreeViewX

Table of Contents

Introduction

TreeViewX is a control derived from TreeView provided in Microsoft .Net Framework. It inherits all the features from TreeView and added following new features:

- Create shortcuts – a special type of nodes is defined as shortcuts to other nodes. A shortcut node looks and behaves the same as the node it points, but it appears under a different parent node. Many shortcuts may be created for one node.
- Attach data to nodes – a list of data can be attached to a node. Each piece of data appears as a property of the node.
- Node Edit UI – At both design time and runtime the user may edit the nodes via following UI:
    - Context menu – right-click TreeViewX, a context menu allows creating root node, sub-node, attach data to selected node, delete selected node, delete selected data
    - Drag/drop – drag a node and drop it to another node, it allows the user to move the dragged node to the drop target or create a shortcut to the dragged node under the drop target.
- New Properties
    - ReadOnly – Gets or sets a Boolean value indicating whether it allows the user to modify the TreeView at runtime
- New Methods
    - `bool SaveToFile(string filename)` – save nodes to an XML file
    - `bool LoadFromFile(string filename)` – load nodes from an XML file
    - `TreeNodeX AddRootNode(string text)` – Add a root node. The new node is selected. Returns the new node.
    - `TreeNodeX AddSubNode(string text)` – Add a sub node to the selected node. The new node is selected. Returns the new node. If there is not a selected node then this method does nothing and returns null.
    - `TreeNodeValue CreateNewValue()` – Attach a new value to the selected category node and return the new node representing the new value. A dialogue appears asking

for the type of the new data. The default value for the data type is used as the new value. If there is not a category node selected then this method does nothing.

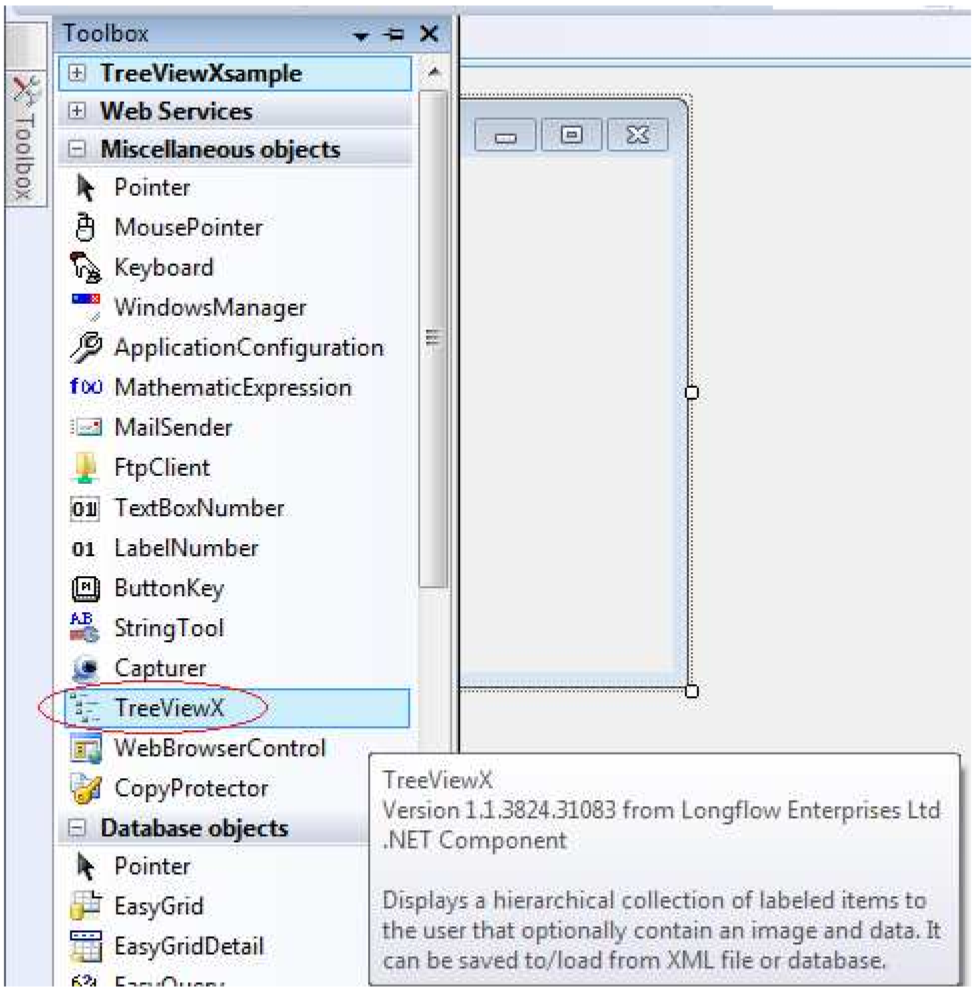
- `void DeleteSelectedCategoryNode(bool confirm)` – Delete selected node. If there is not a selected node or the selected node is for value then this method does nothing. If parameter 'confirm' is true then a message box appears to ask the user to confirm the deletion.
- `void DeleteSelectedValue(bool confirm)` – Delete selected value node. If there is not a selected node or the selected node is not for value then this method does nothing. If parameter 'confirm' is true then a message box appears to ask the user to confirm the deletion.
- `bool EditNodes()` – Launch Tree Nodes Editor to edit the nodes and values. It returns false if the editing is canceled.
- `void RemoveAllNodes()` – Remove all nodes.
- `void LoadRootNodesFromDatabase() – Load root nodes from database. Deeper level tree nodes will be automatically loaded from databases when a parent node is expanded.`

- New events
  - CategoryNodeSelected – it occurs when a node is selected, and the node is not is not for a piece of attached value
  - ShortcutNodeSelected – it occurs when a shortcut node is selected. CategoryNodeSelected event will follow this event.
  - ValueNodeSelected – it occurs when a node for a piece of attached data is selected
- Load Tree Nodes from databases

Prepare images for nodes

We use a Windows Form application project to demonstrate the use of the TreeViewX.

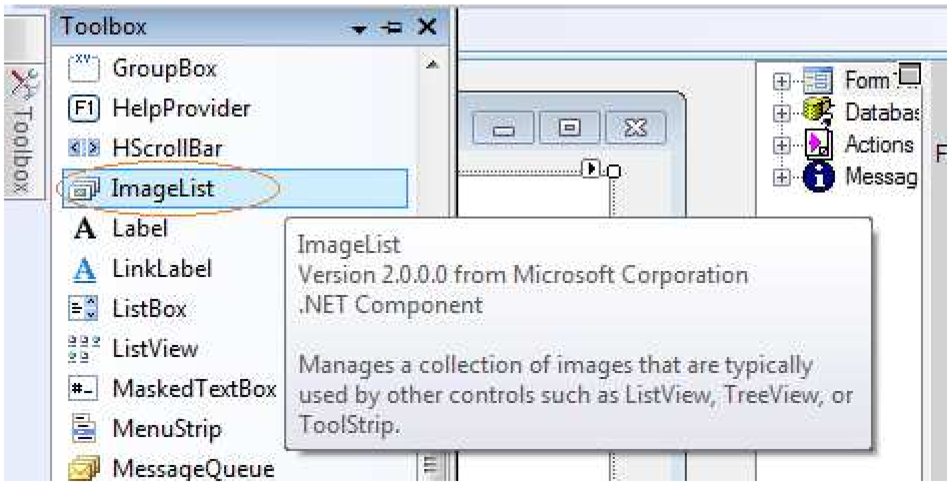
Drop TreeViewX to the form:



TreeViewX
Version 1.1.3824.31083 from Longflow Enterprises Ltd
.NET Component

Displays a hierarchical collection of labeled items to the user that optionally contain an image and data. It can be saved to/load from XML file or database.

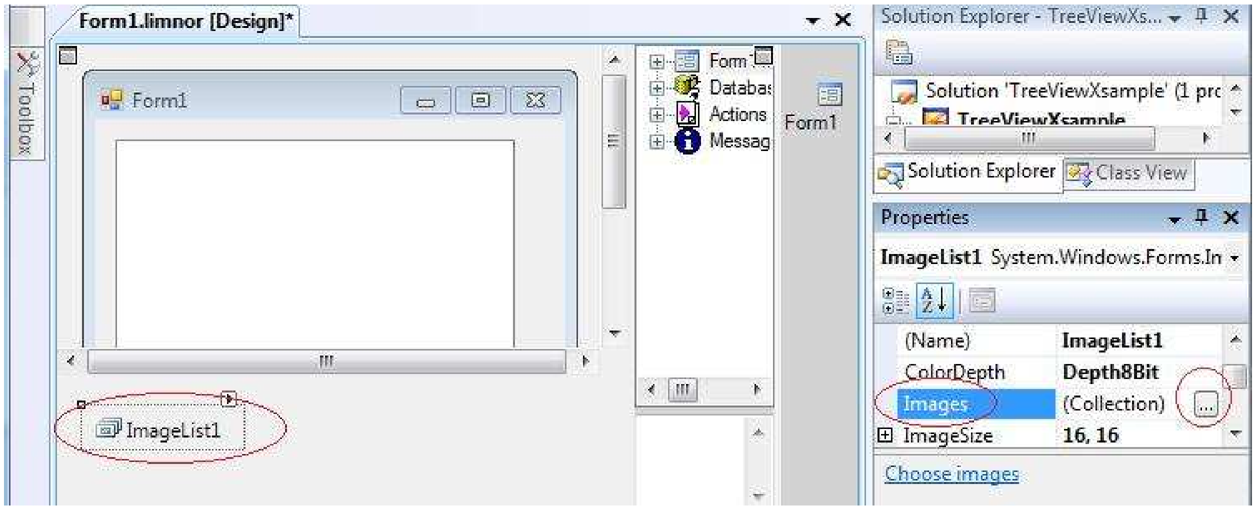
We may use an ImageList to provide icons for tree nodes. Drop ImageList to the form:



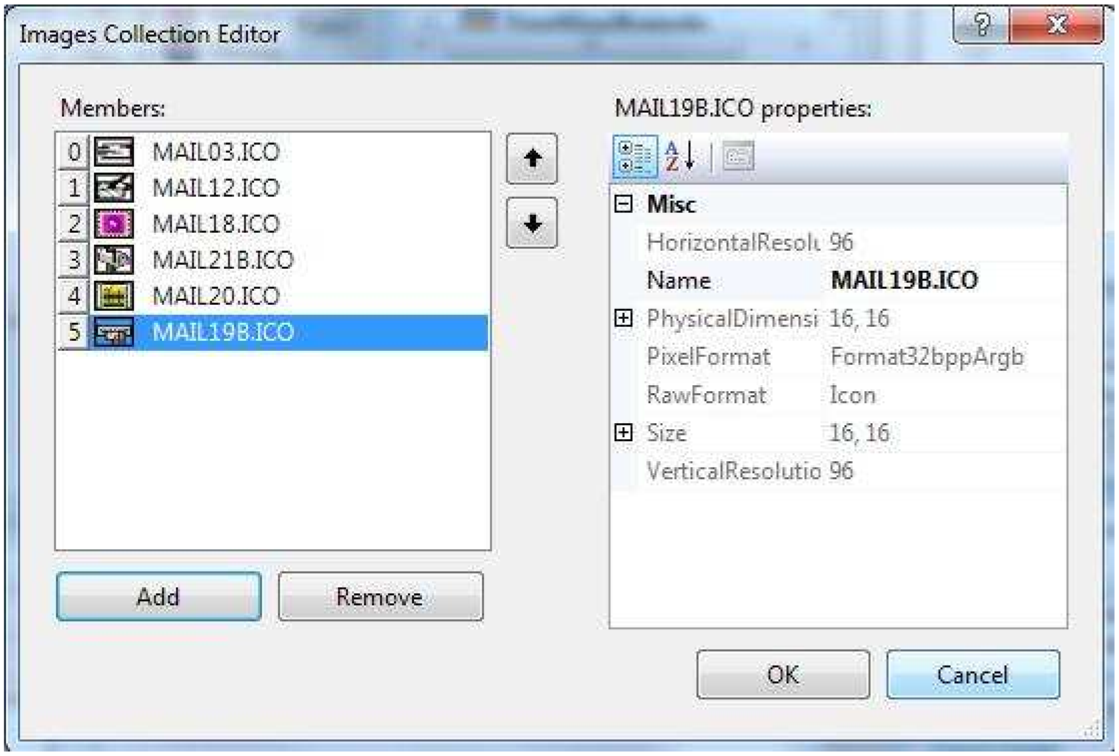
ImageList
Version 2.0.0.0 from Microsoft Corporation
.NET Component

Manages a collection of images that are typically used by other controls such as ListView, TreeView, or ToolStrip.

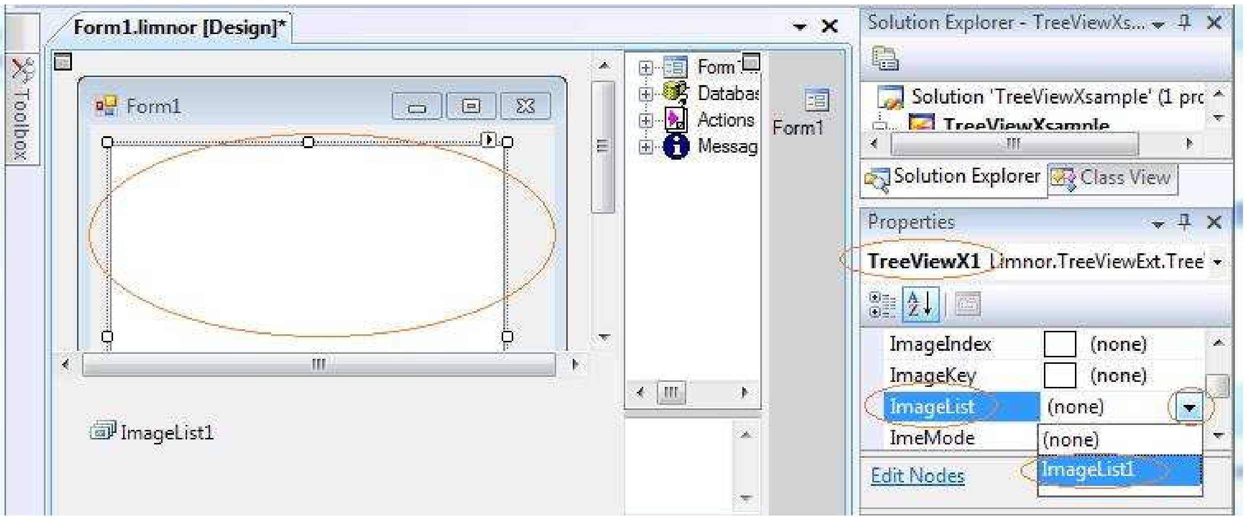
Click [...] to add images to the ImageList:



Click Add button to add images to the list:



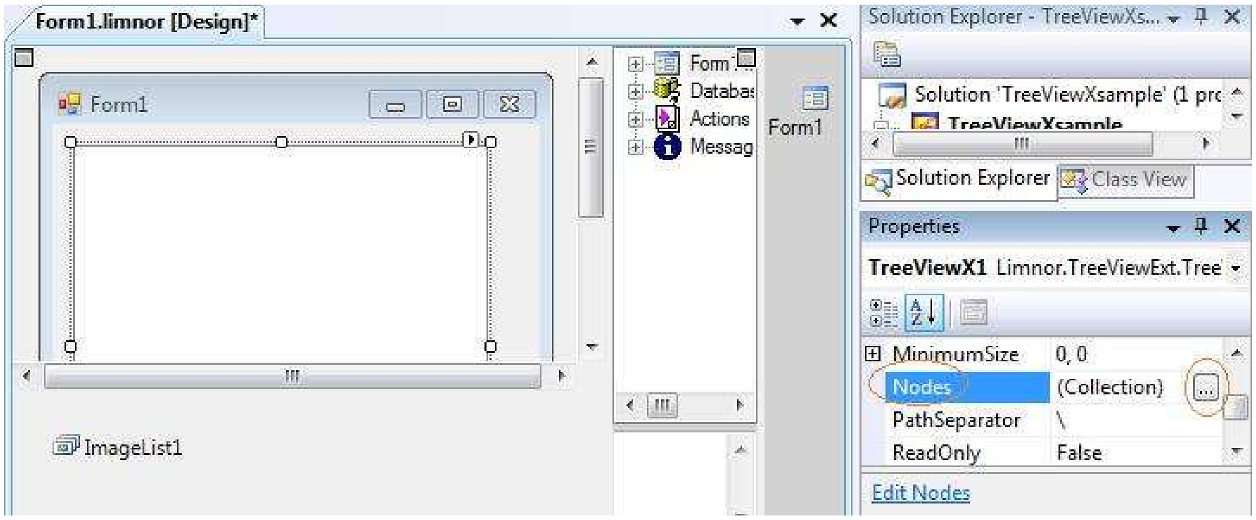
To tell the tree view to use the image list, select ImageList1 as the ImageList property of the tree view:



TreeViewX1 Limnor.TreeViewExt.Tree

ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
ImeMode	

(none)
ImageList1



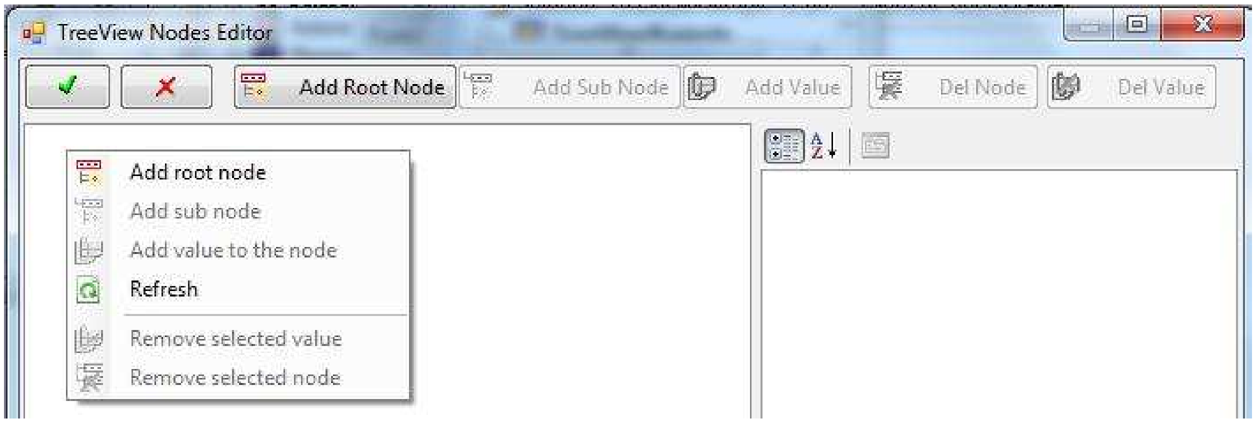
TreeViewX1 Limnor.TreeViewExt.Tree

MinimumSize	0, 0
Nodes	(Collection)
PathSeparator	\
ReadOnly	False

Edit nodes at design time

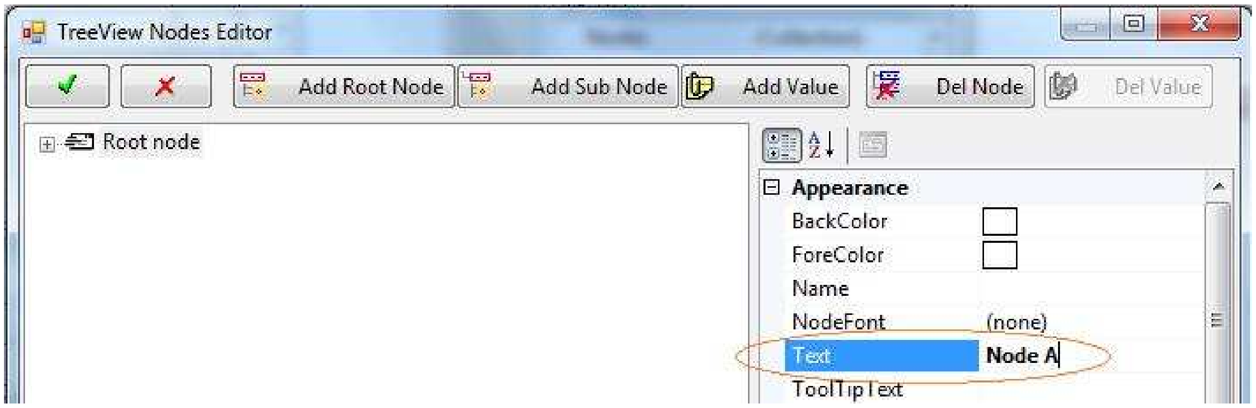
Select Nodes property; click [...] to edit nodes for the tree view. A "TreeView Nodes Editor" appears.

You may use the buttons or the context menu to edit the nodes.



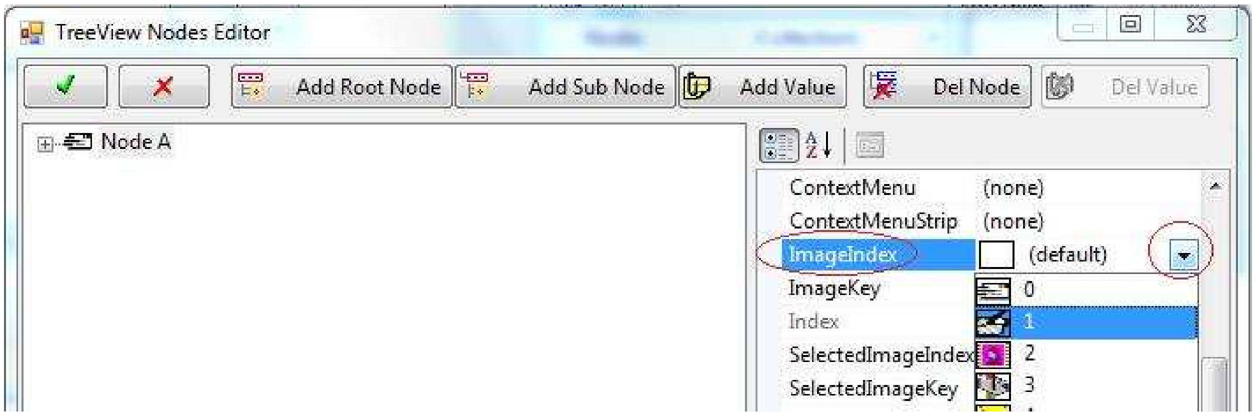
Create root node

Click “Add Root Node” to create the first node. We may set its properties. For example, set its Text property:



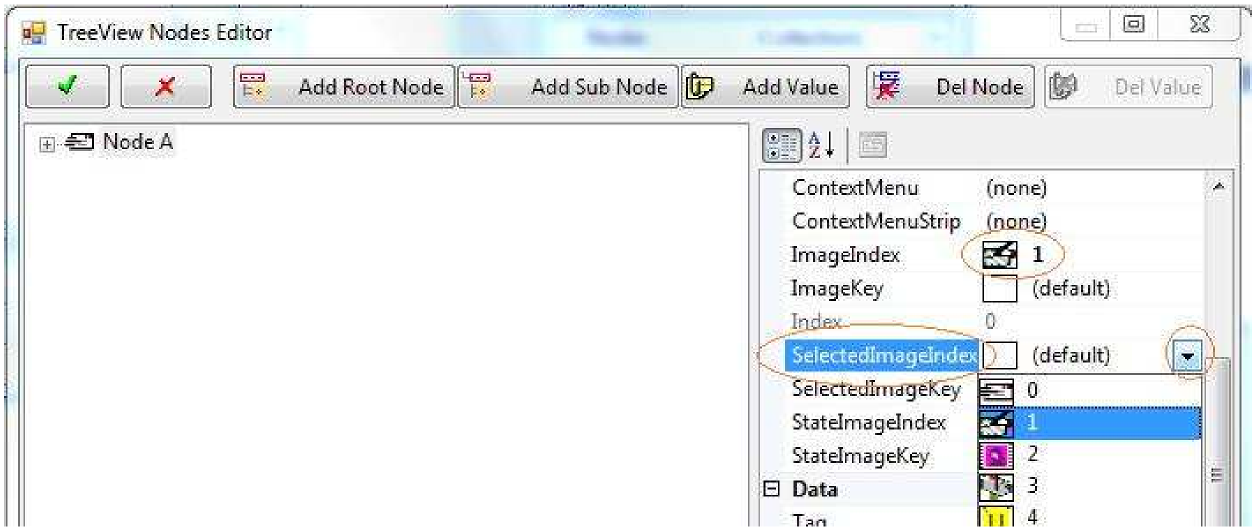
NodeFont (none)
Text Node A

Setting its ImageIndex to specify image for the node:



ContextMenu (none)
ContextMenuStrip (none)
ImageIndex (default)

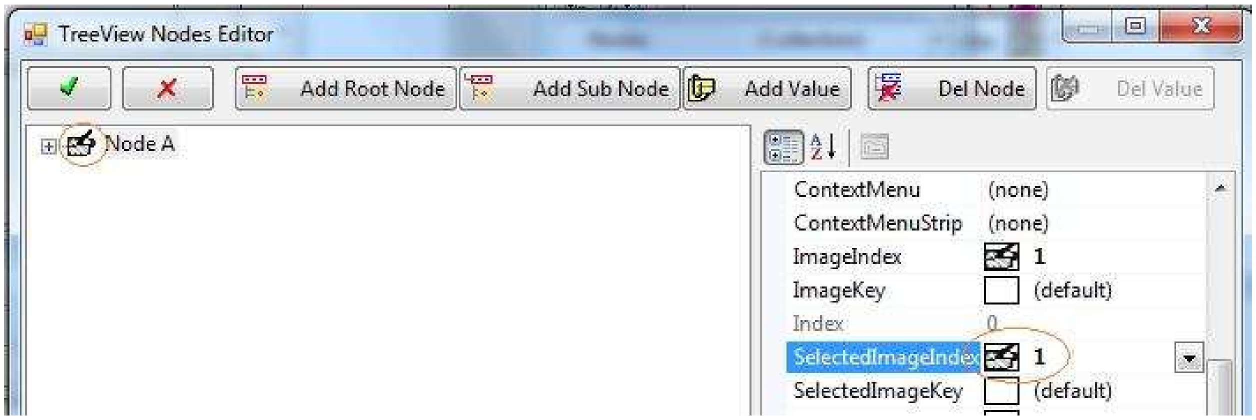
Note that after choosing ImageIndex the image for the Node does not change in the tree view. This is because ImageIndex is for the node when the node is not selected. Currently the node is selected; its image is determined by property SelectedImageIndex. Let’s modify SelectedImageIndex:



ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	1
ImageKey	(default)
Index	0
SelectedImageIndex	(default)
SelectedImageKey	
StateImageIndex	
StateImageKey	
Data	
Tag	

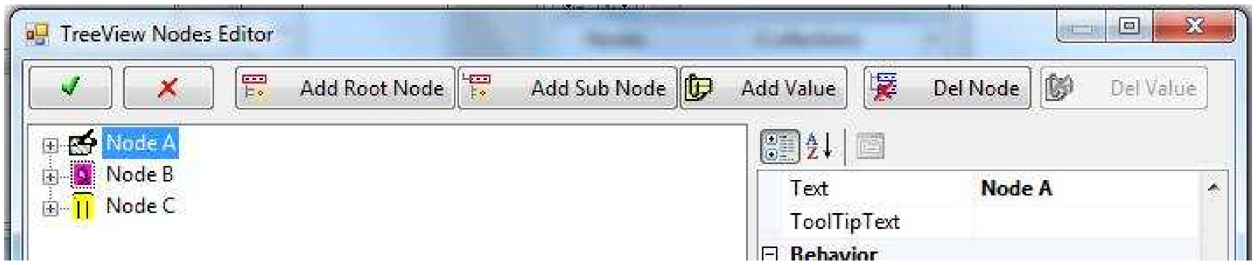
0
1
2
3
4

Now the image in the tree view is changed:



ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	1
ImageKey	(default)
Index	0
SelectedImageIndex	1
SelectedImageKey	(default)

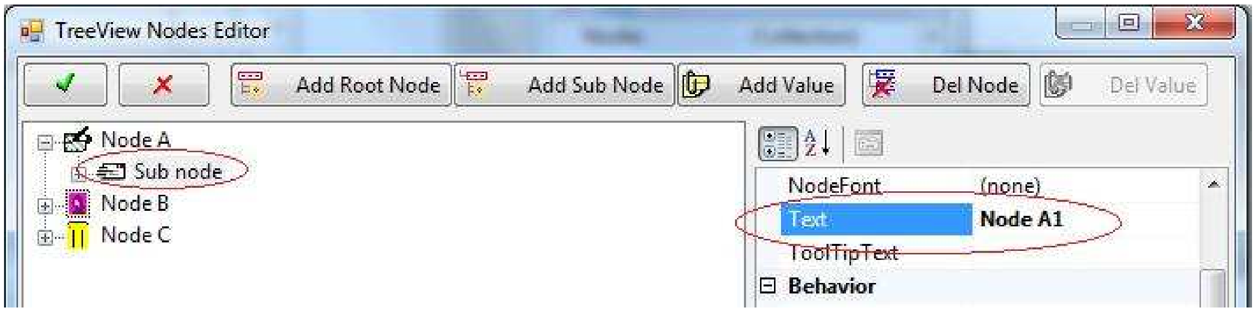
We may create more root nodes in this way.



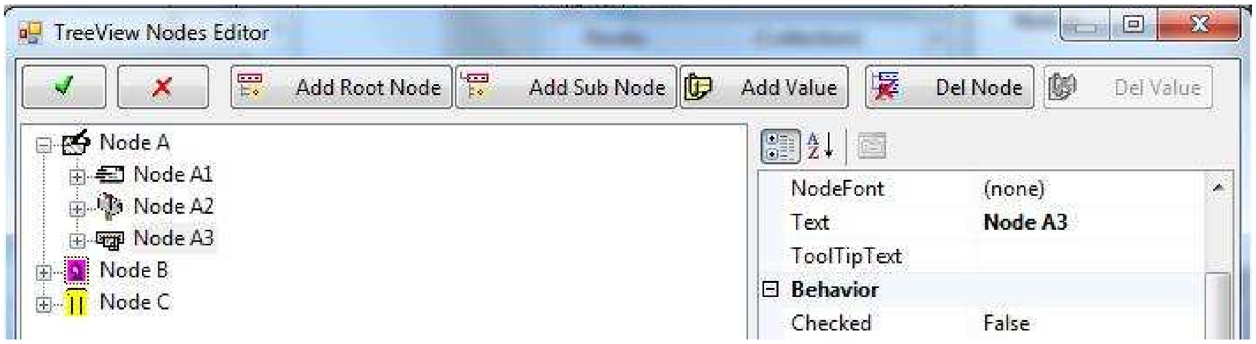
Text	Node A
ToolTipText	
Behavior	

Create sub-nodes

Click button “Add Sub Node”. A new sub-node appears under the selected node. We may change its properties. For example, change its Text:

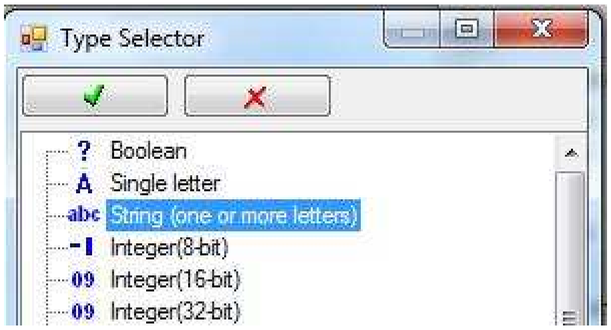


We may add more sub nodes:

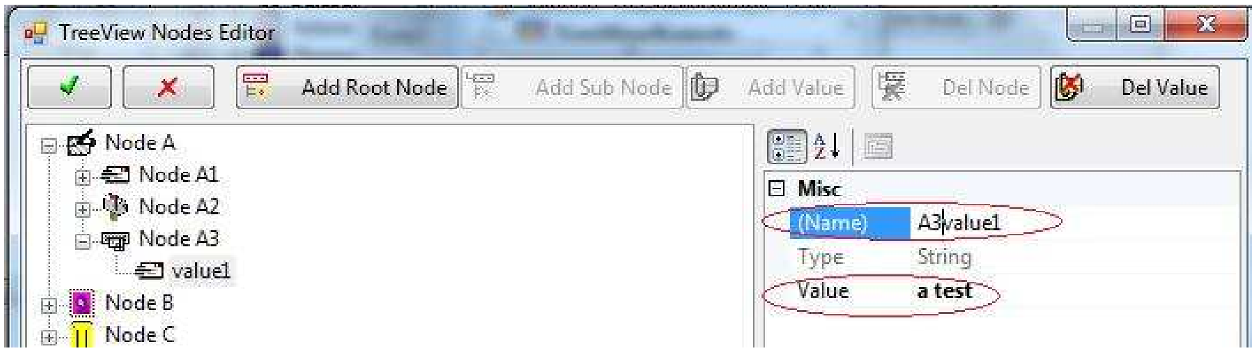


Attach values to nodes

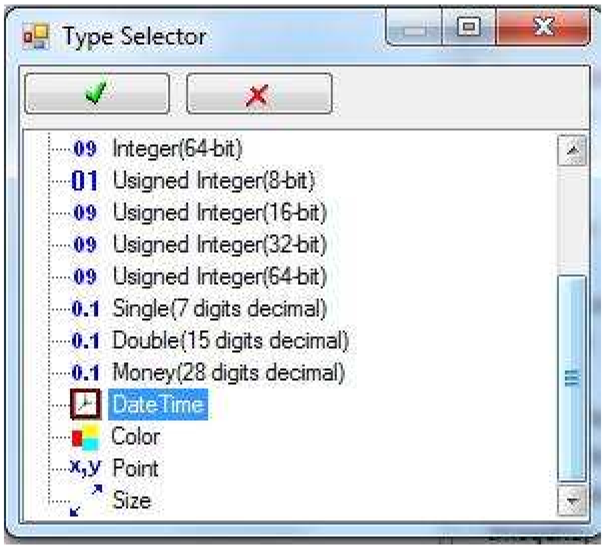
Click button "Add Value". A dialogue box appears asking for the type of data to be attached. If we want to attach text to this node then we may choose String:



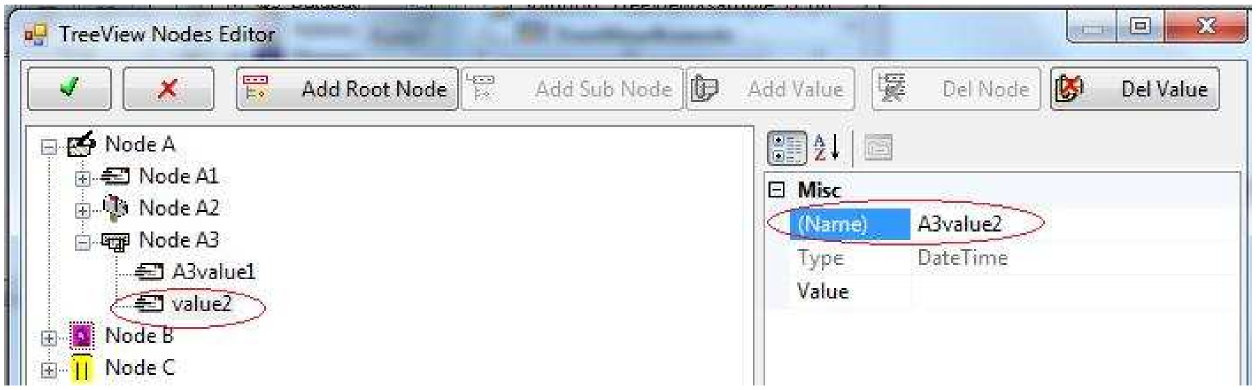
A new value appears under the selected node. We may change the value and the name:



As an example, let’s attach a date-time value to the same node:

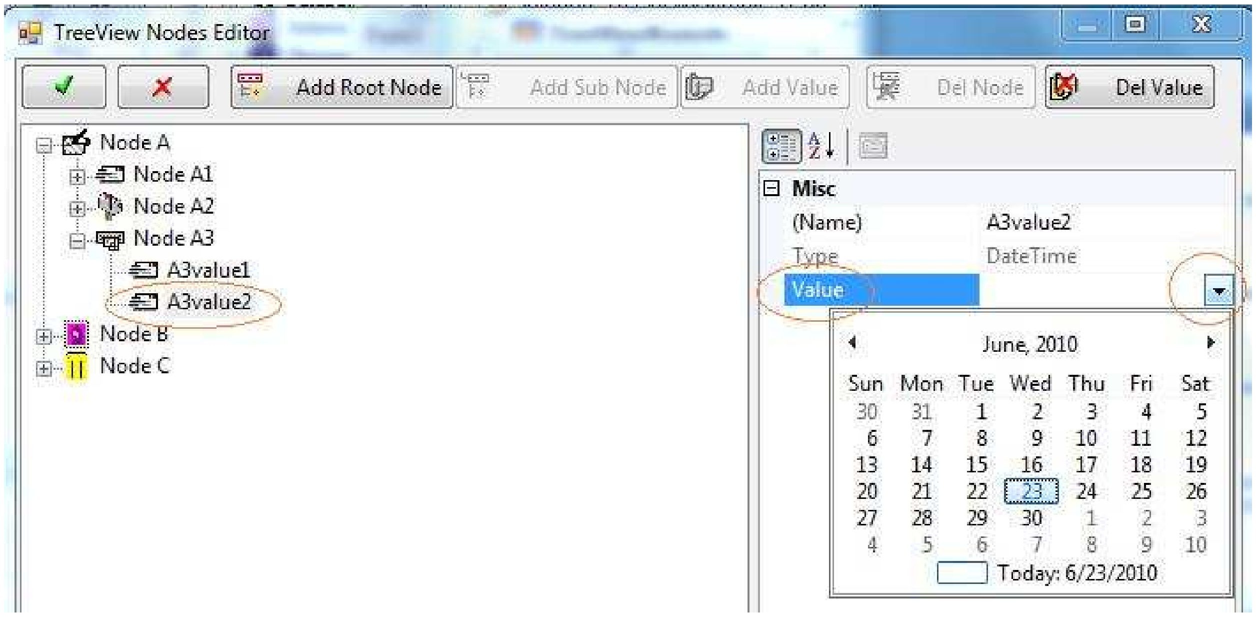


The new value appears. We may change its name:



(Name)	A3value2
Type	DateTime
Value	

We may change its value:

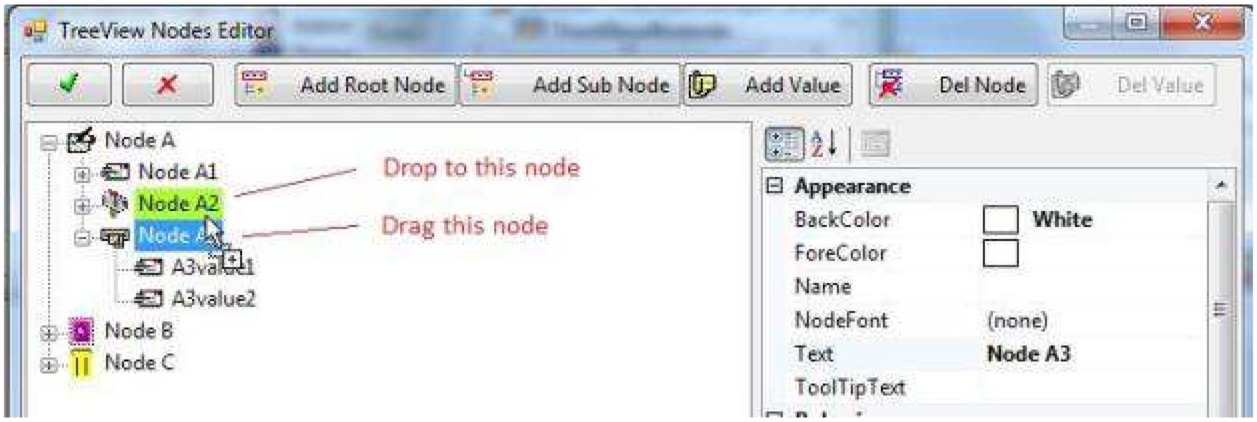


Change node category structure

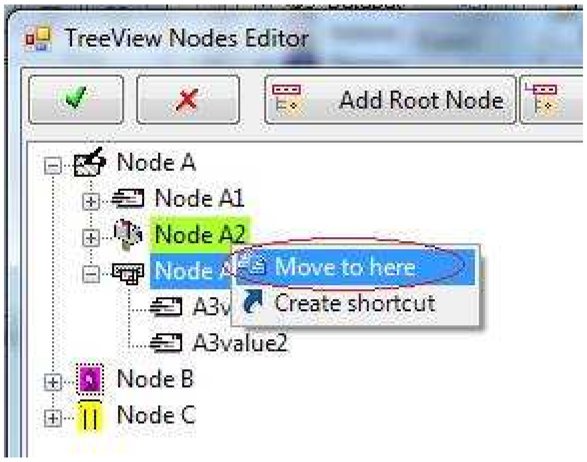
A tree structure represents a category system. So, we also call a tree node a category node, if the node is not for an attached value. A value node cannot have sub nodes. A category node may have sub-nodes.

The category structure can be changed by drag and drop.

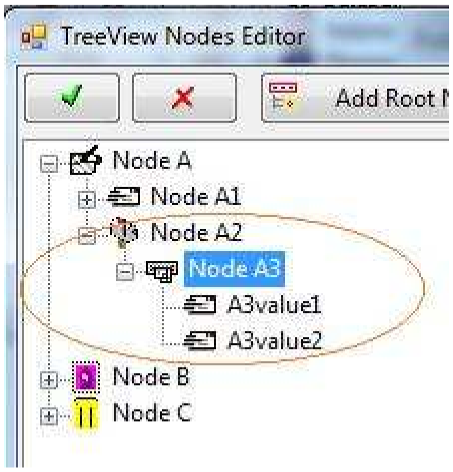
Drag node "Node A3" and drop it to node "Node A2":



When a category node is dropped to another category node, a context menu appears. Choose "Move here" to move the node:



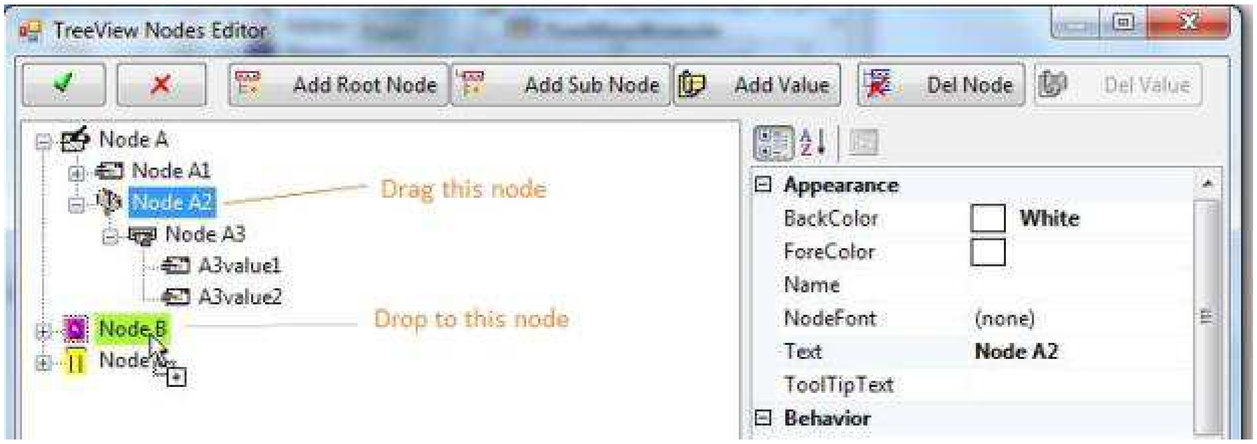
The node “Node A3” becomes a sub node of “Node A2”:



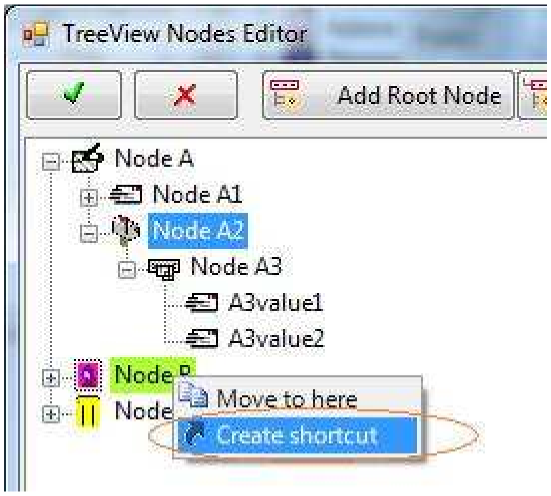
Create Shortcuts

In this sample, “Node A2” is a sub node of “Node A”. So, we need to access “Node A2” through “Node A”. Normally we cannot access “Node A2” through other category nodes. But we create shortcut nodes in other category nodes to access “Node A2”.

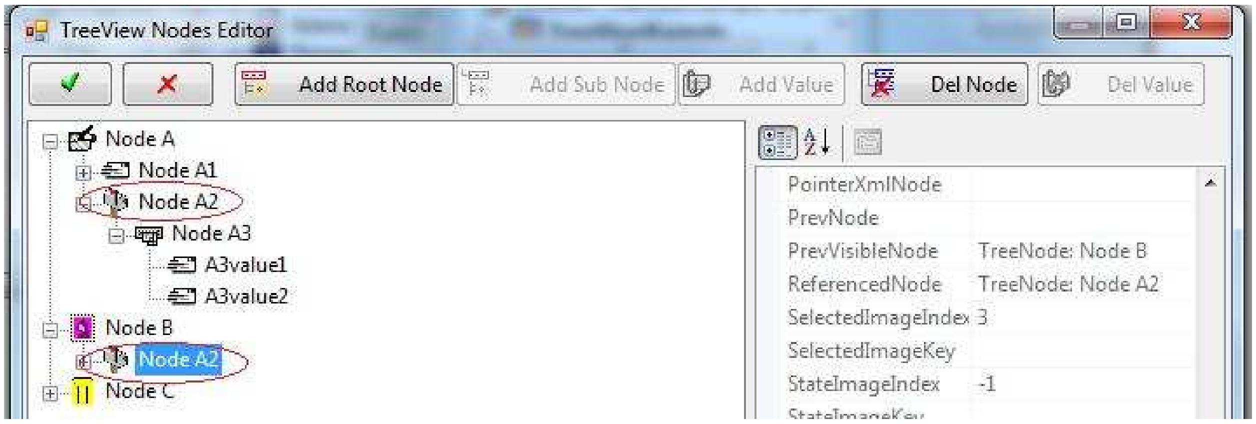
Suppose we want to be able to access “Node A2” through “Node B”. Drag “Node A2” and drop it to “Node B”:



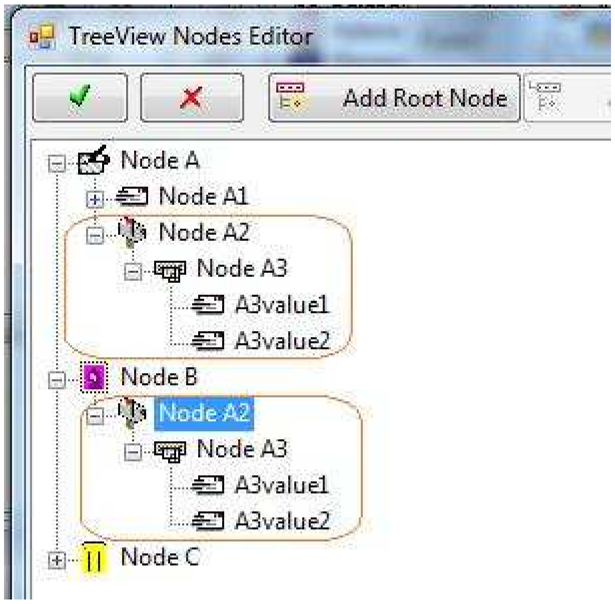
Choose context menu “Create shortcut”:



A shortcut to node “Node A2” appears under “Node B”:



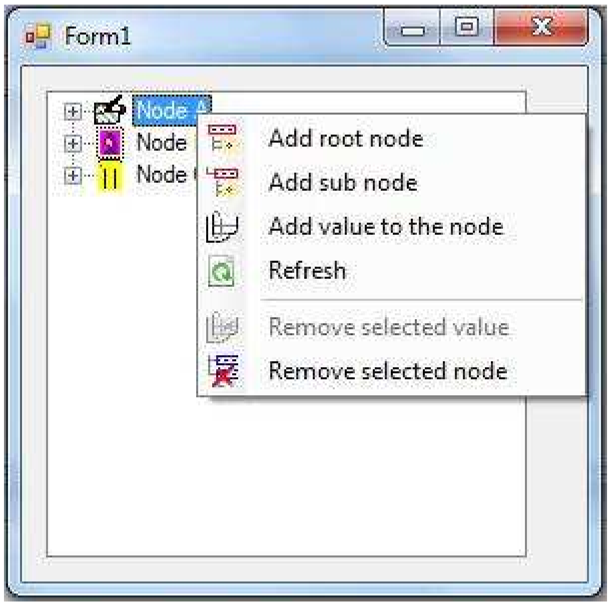
Through the shortcut we may access all the contents of the original node it points to:



Editing at runtime

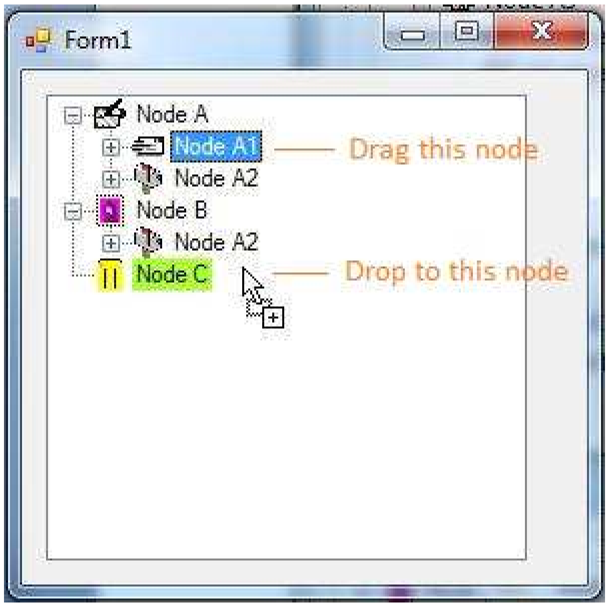
Create and delete nodes

At runtime, right-click the tree view, a context menu appears to let the user create/delete category nodes and value nodes:

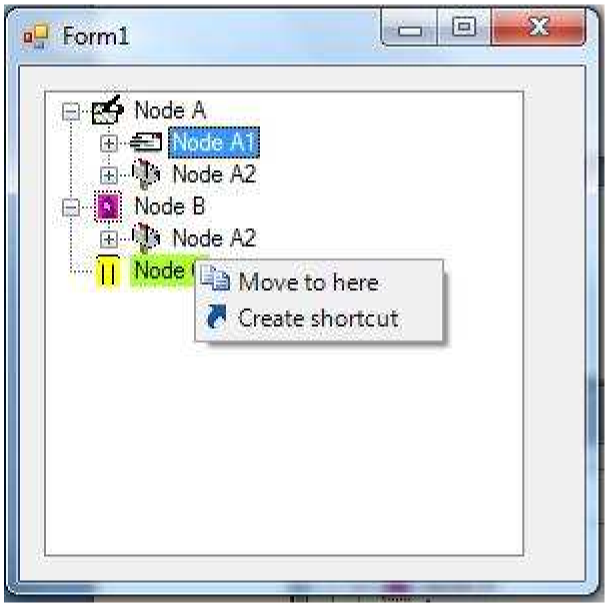


Change structure and create shortcuts

Drag a node and drop it to another node

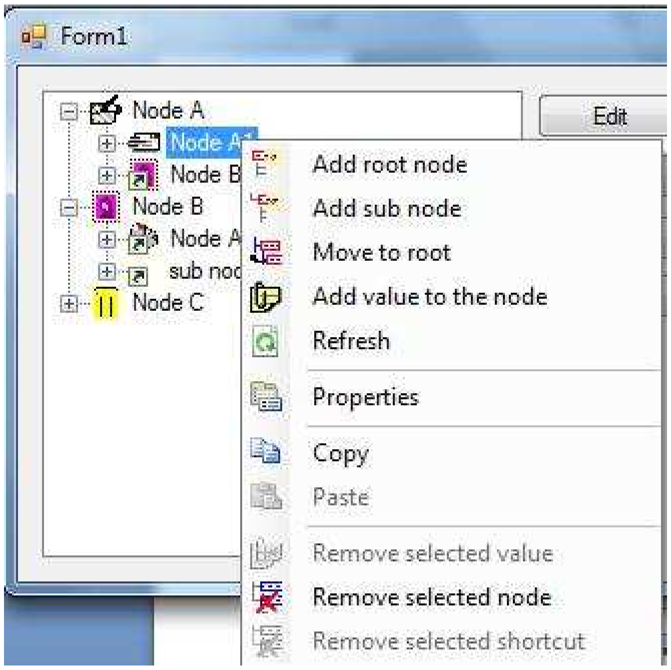


A context menu appears to allow moving the node or to create a shortcut:

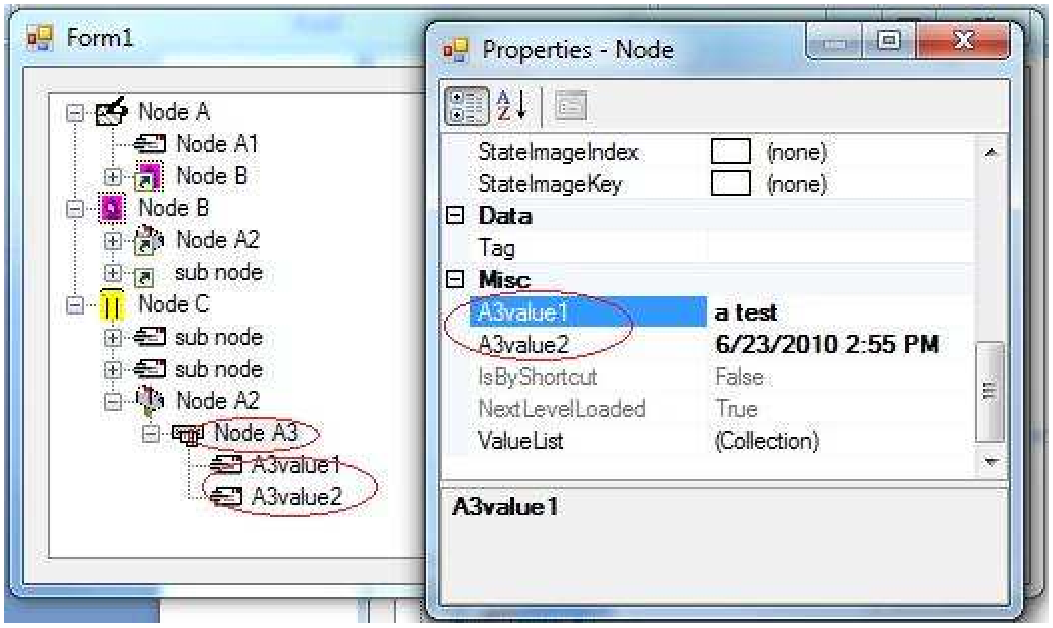


Nodes and values editing

At runtime, right-click the tree view, a context menu allows editing nodes and attached values:



For a node, its attached values are displayed at node properties. In the following example, Node A3 has two attached values A3value1 and A3value2. These two values shown in the Properties window for the node as two properties:



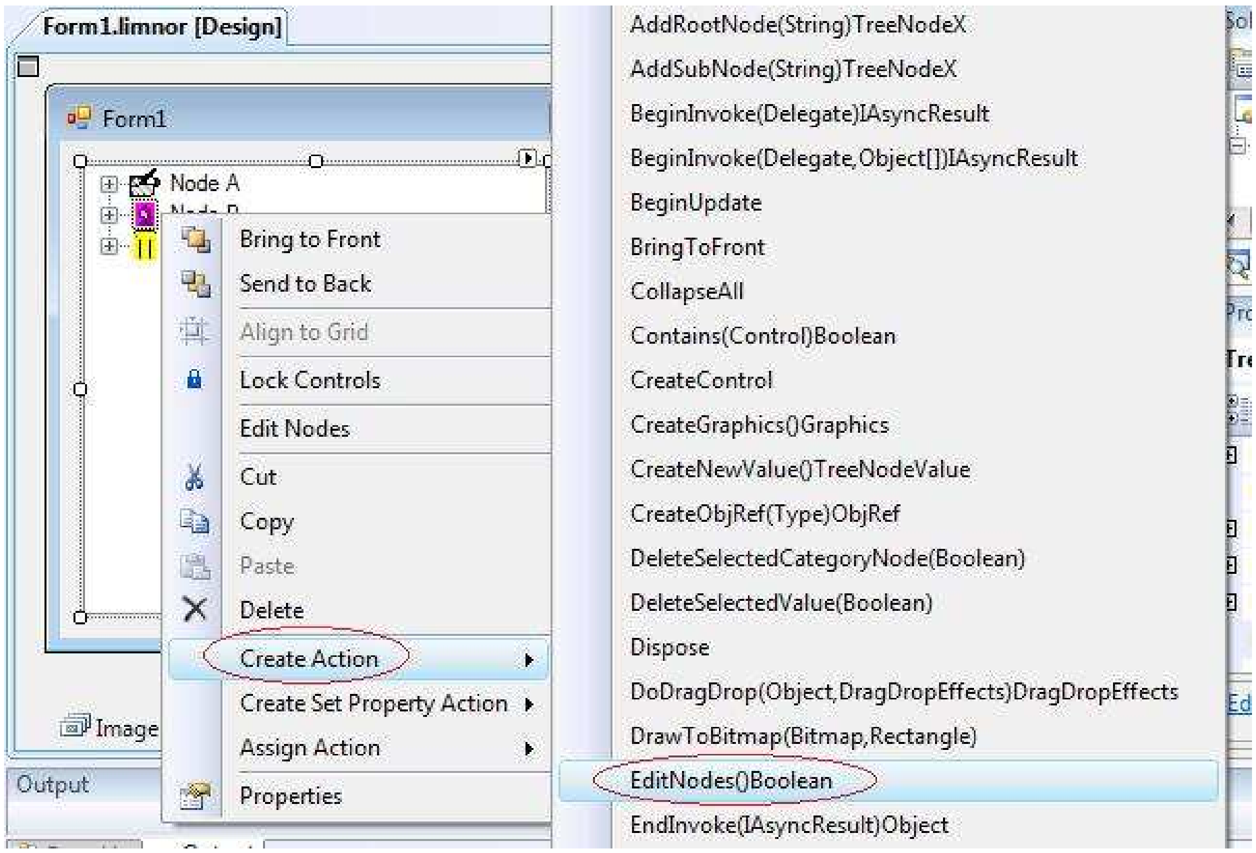
StateImageIndex	(none)
StateImageKey	(none)
Tag	
A3value1	a test
A3value2	6/23/2010 2:55 PM
IsByShortcut	False
NextLevelLoaded	True
ValueList	(Collection)

A3value1

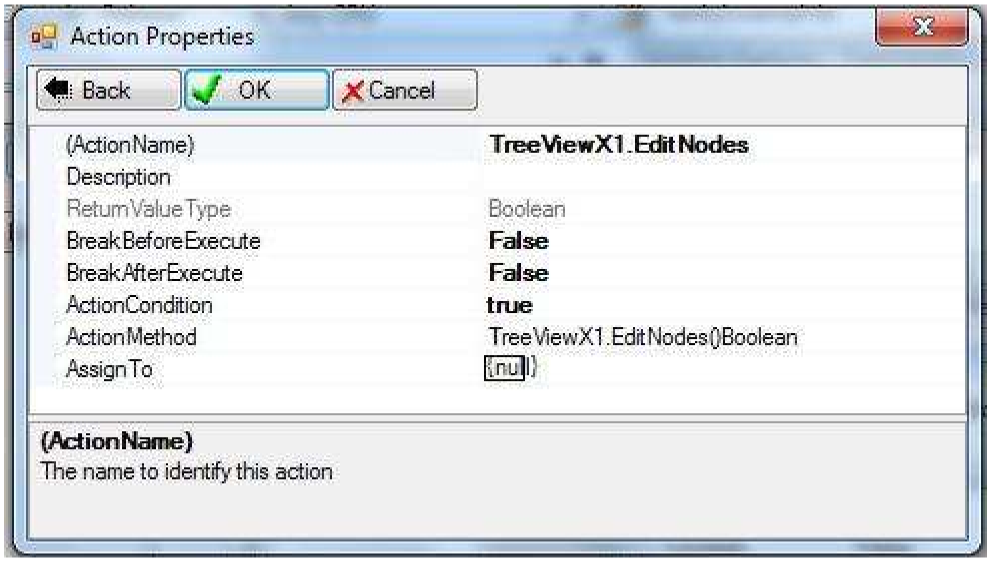
Calling EditNode method

The design time editing can be launched at runtime via an action of "EditNodes".

Right-click the tree view, choose "Create Action". Choose "EditNodes":


Form1.limnor [Design]
Form1
Node A
Bring to Front
Send to Back
Align to Grid
Lock Controls
Edit Nodes
Cut
Copy
Paste
Delete
Create Action
Create Set Property Action
Assign Action
Properties
Image
Output
AddRootNode(String)TreeNodeX
AddSubNode(String)TreeNodeX
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult
BeginUpdate
BringToFront
CollapseAll
Contains(Control)Boolean
CreateControl
CreateGraphics()Graphics
CreateNewValue()TreeNodeValue
CreateObjRef(Type)ObjRef
DeleteSelectedCategoryNode(Boolean)
DeleteSelectedValue(Boolean)
Dispose
DoDragDrop(Object,DragDropEffects)DragDropEffects
DrawToBitmap(Bitmap,Rectangle)
EditNodes()Boolean
EndInvoke(IAsyncResult)Object

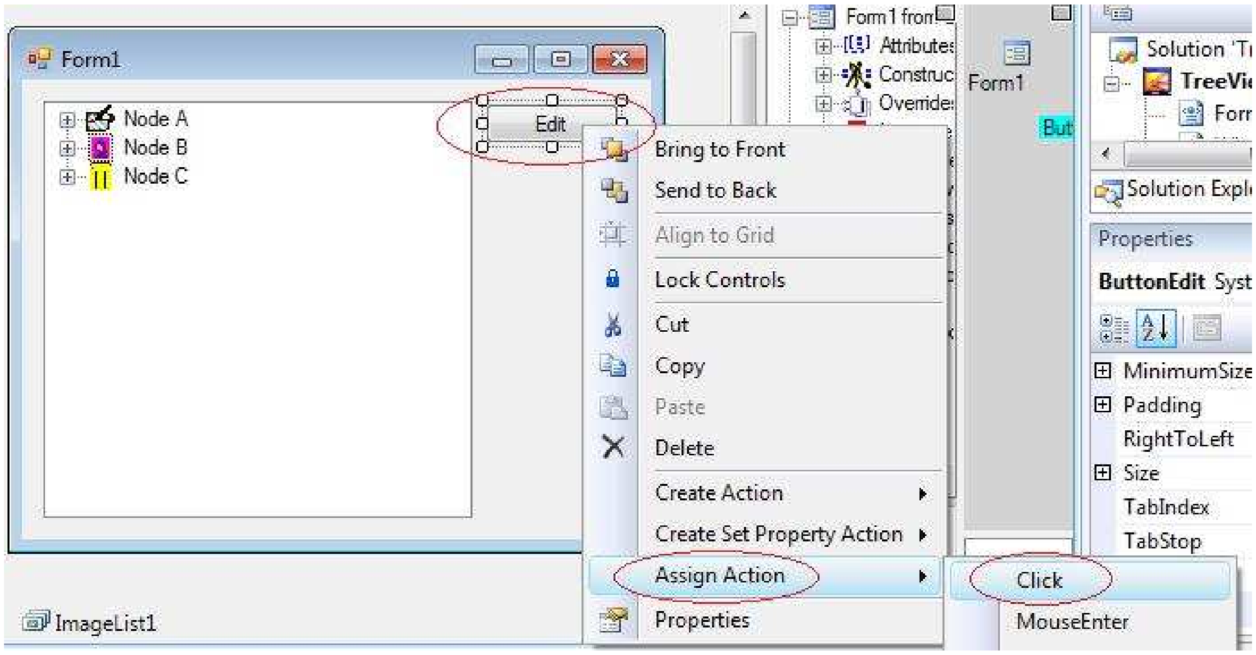
Click OK:



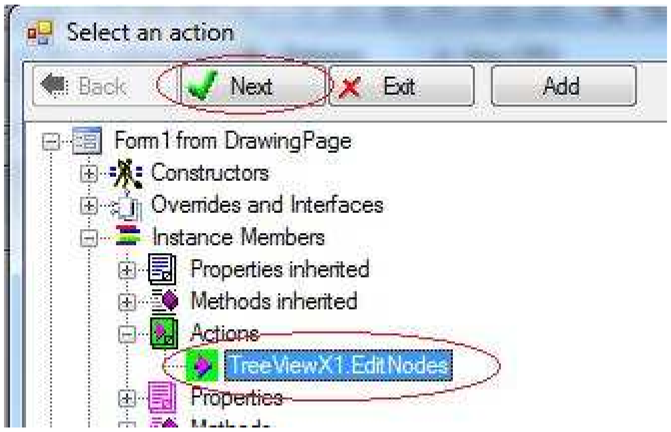
(ActionName)	TreeViewX1.EditNodes
Description	
ReturnValueType	Boolean
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	TreeViewX1.EditNodes()Boolean
AssignTo	(null)

(ActionName)
The name to identify this action

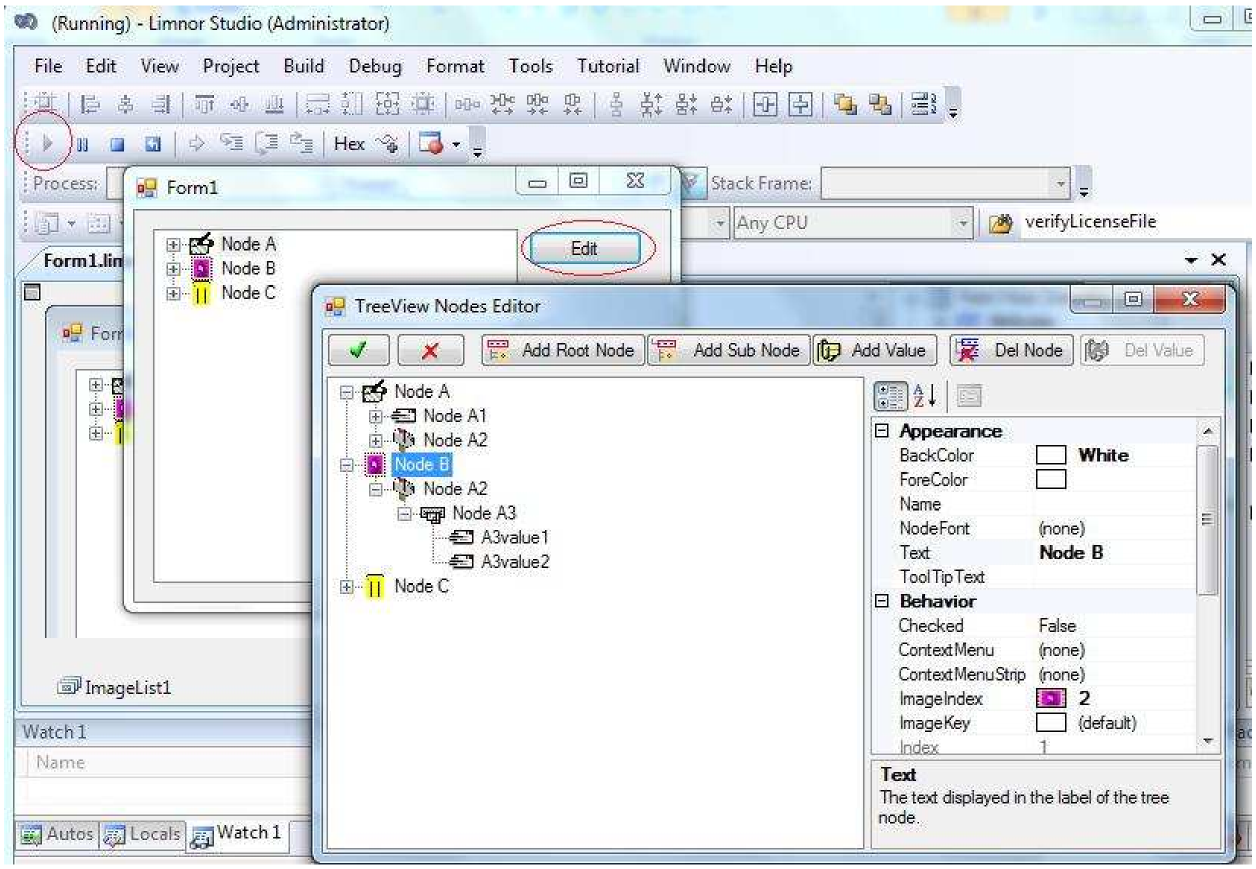
To assign this action, TreeViewX1.EditNodes, to a button, right-click the button, choose “Assign Action”. Choose “Click”:



Choose the action. Click Next:



Click ▶ to test it at runtime. The application runs. Click button Edit. The Nodes Editor appears:



Appearance	
BackColor	White
ForeColor	
Name	
NodeFont	(none)
Text	**Node B**
ToolTipText	
Behavior	
Checked	False
ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	2
ImageKey	(default)
Index	1

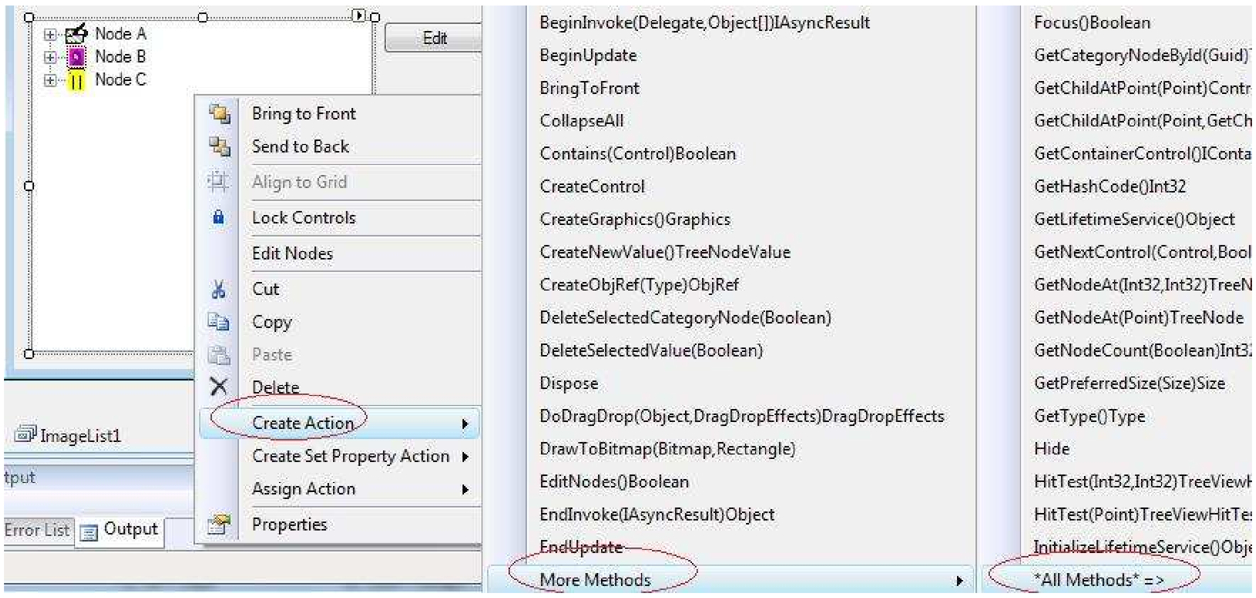
Text
The text displayed in the label of the tree node.

Use XML Files

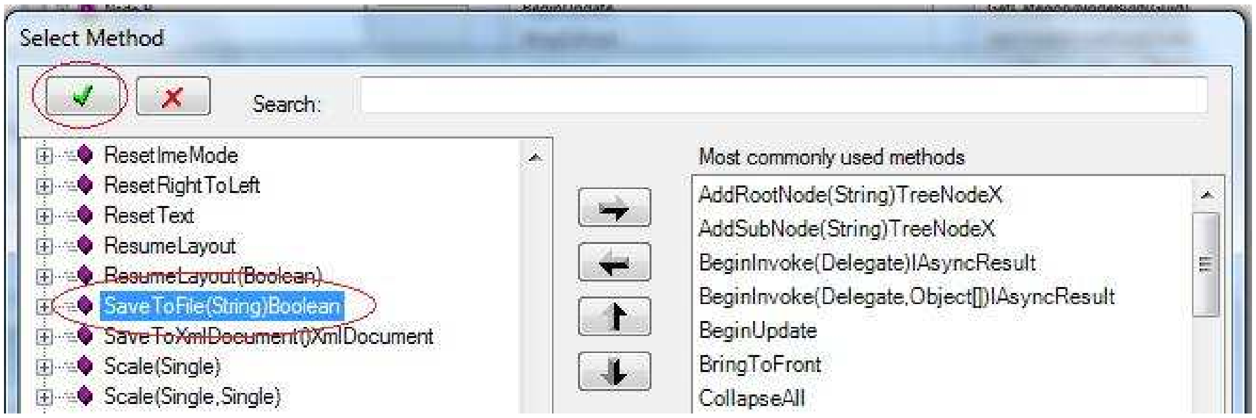
Save nodes to XML file

We may create an action to save nodes to an XML file.

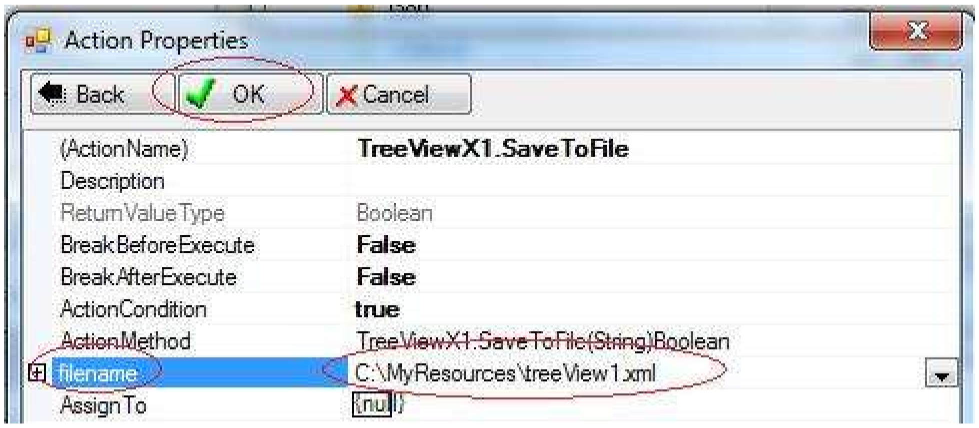
Right-click the tree view, choose “Create Action”. Choose “More methods”. Choose “*All methods *=>”


Node A
Node B
Node C
Edit
Bring to Front
Send to Back
Align to Grid
Lock Controls
Edit Nodes
Cut
Copy
Paste
Delete
Create Action
Create Set Property Action
Assign Action
Properties
ImageList1
Error List
Output
BeginInvoke(Delegate,Object[])IAsyncResult
BeginUpdate
BringToFront
CollapseAll
Contains(Control)Boolean
CreateControl
CreateGraphics()Graphics
CreateNewValue()TreeNodeValue
CreateObjRef(Type)ObjRef
DeleteSelectedCategoryNode(Boolean)
DeleteSelectedValue(Boolean)
Dispose
DoDragDrop(Object,DragDropEffects)DragDropEffects
DrawToBitmap(Bitmap,Rectangle)
EditNodes()Boolean
EndInvoke(IAsyncResult)Object
EndUpdate
More Methods
Focus()Boolean
GetCategoryNodeById(Guid)
GetChildAtPoint(Point)Contr
GetChildAtPoint(Point,GetCh
GetContainerControl()ICont
GetHashCode()Int32
GetLifetimeService()Object
GetNextControl(Control,Bool
GetNodeAt(Int32,Int32)TreeN
GetNodeAt(Point)TreeNode
GetNodeCount(Boolean)Int3
GetPreferredSize(Size)Size
GetType()Type
Hide
HitTest(Int32,Int32)TreeView
HitTest(Point)TreeViewHitTe
InitializeLifetimeService()Obje
All Methods =>

Select “SaveToFile”:

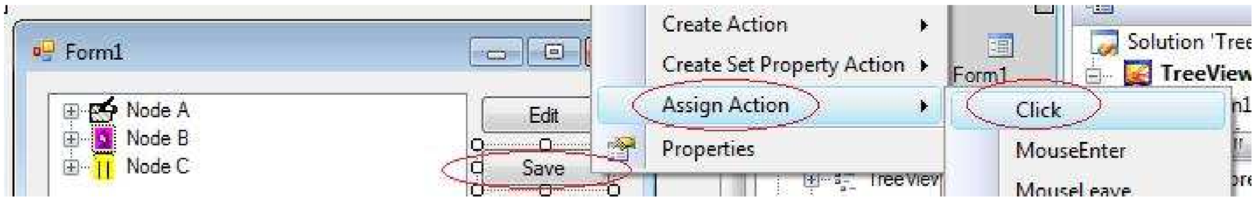

Select Method
Search:
ResetImeMode
ResetRightToLeft
ResetText
ResumeLayout
ResumeLayout(Boolean)
SaveToFile(String)Boolean
SaveToXmlDocument()XmlDocument
Scale(Single)
Scale(Single,Single)
Most commonly used methods
AddRootNode(String)TreeNodeX
AddSubNode(String)TreeNodeX
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult
BeginUpdate
BringToFront
CollapseAll

This action needs a “filename” parameter indicating the name of file to be created. For this sample, we use a constant “C:\MyResources\treeView1.xml”:

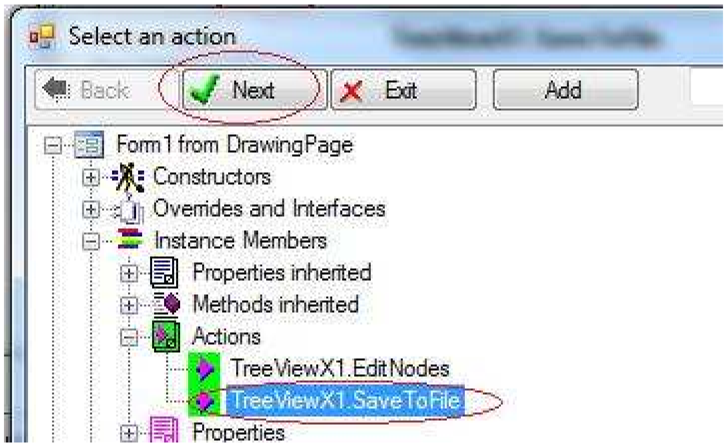


(ActionName)	TreeViewX1.SaveToFile
Description	
ReturnValueType	Boolean
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	TreeViewX1.SaveToFile(String)Boolean
filename	C:\MyResources\treeView1.xml
AssignTo	(null)

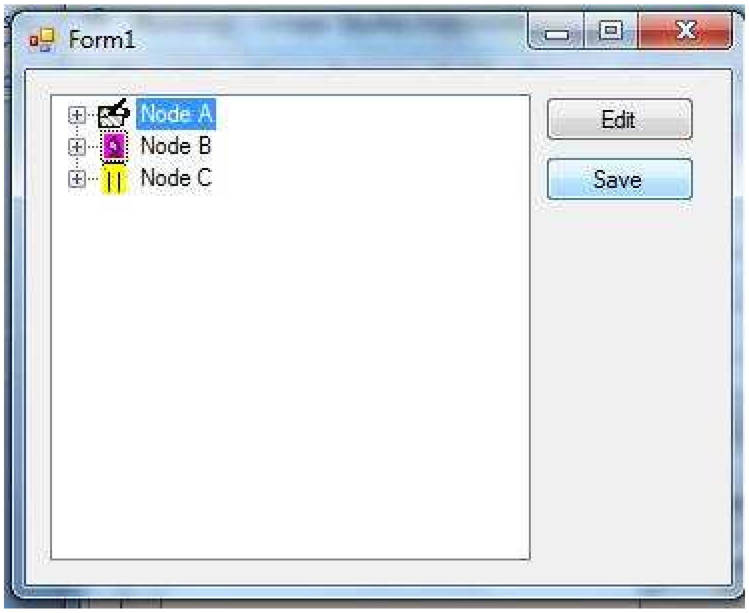
Assign this action to a button:



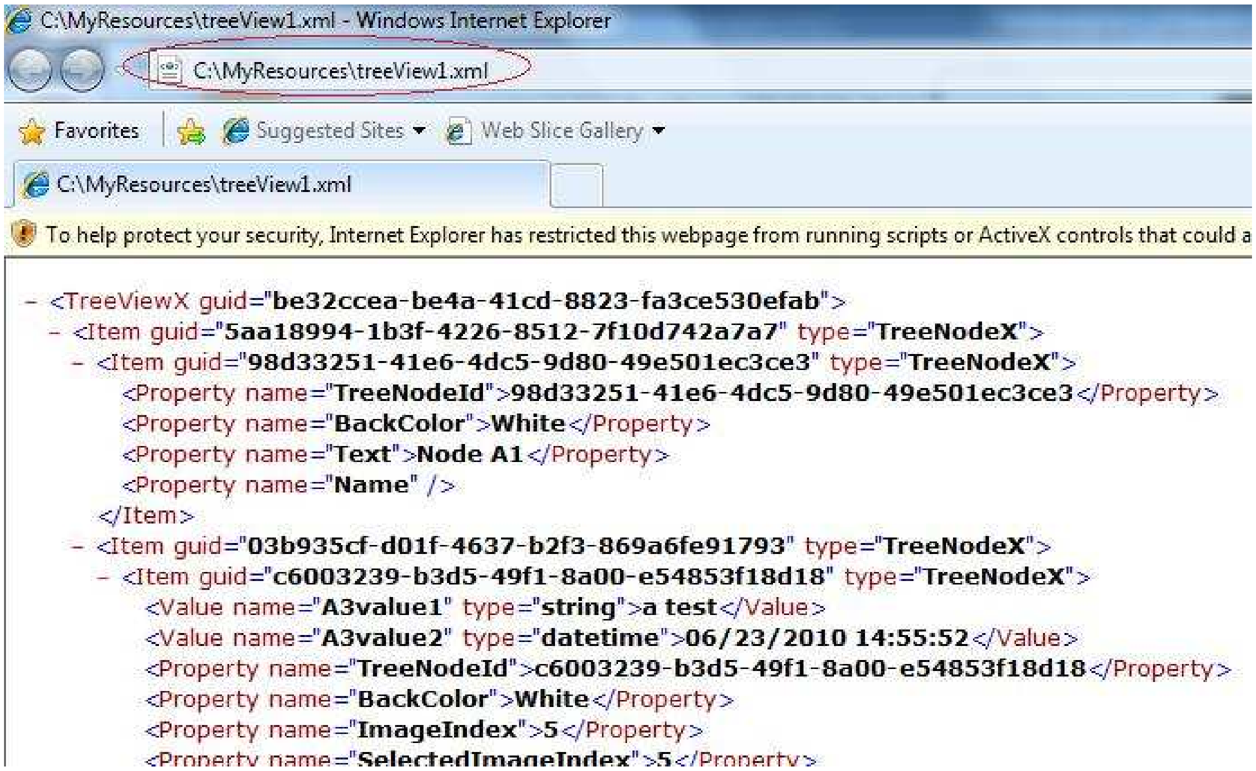
Select the action. Click Next:



Click ▶ to test it at runtime. The application runs. Click “Save” button:



The XML file is created. We may use IE to view it:


C:\MyResources\treeView1.xml - Windows Internet Explorer
C:\MyResources\treeView1.xml
Favorites
Suggested Sites
Web Slice Gallery
C:\MyResources\treeView1.xml
To help protect your security, Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could a

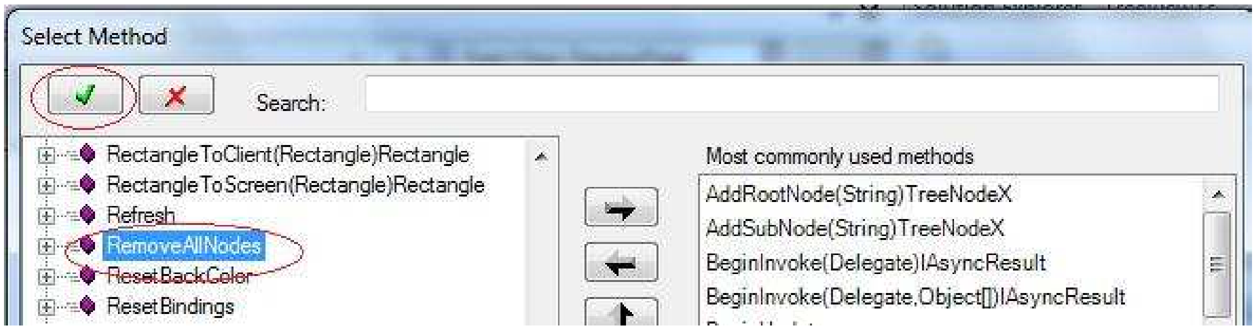
```
- <TreeViewX guid="be32ccea-be4a-41cd-8823-fa3ce530efab">
  - <Item guid="5aa18994-1b3f-4226-8512-7f10d742a7a7" type="TreeNodeX">
    - <Item guid="98d33251-41e6-4dc5-9d80-49e501ec3ce3" type="TreeNodeX">
        <Property name="TreeNodeId">98d33251-41e6-4dc5-9d80-49e501ec3ce3</Property>
        <Property name="BackColor">White</Property>
        <Property name="Text">Node A1</Property>
        <Property name="Name" />
      </Item>
    - <Item guid="03b935cf-d01f-4637-b2f3-869a6fe91793" type="TreeNodeX">
      - <Item guid="c6003239-b3d5-49f1-8a00-e54853f18d18" type="TreeNodeX">
          <Value name="A3value1" type="string">a test</Value>
          <Value name="A3value2" type="datetime">06/23/2010 14:55:52</Value>
          <Property name="TreeNodeId">c6003239-b3d5-49f1-8a00-e54853f18d18</Property>
          <Property name="BackColor">White</Property>
          <Property name="ImageIndex">5</Property>
          <Property name="SelectedImageIndex">5</Property>
```

Load nodes from XML file

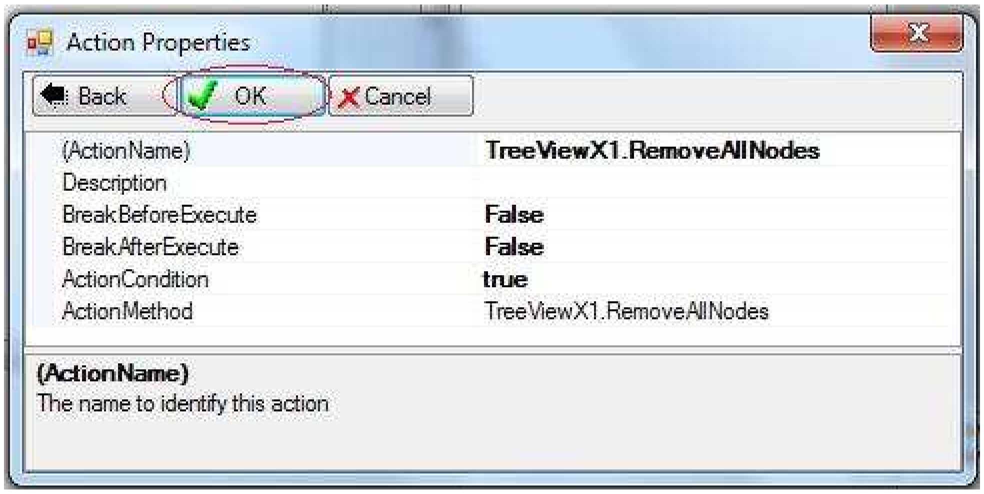
We may use a LoadFromFile action to load tree nodes from a XML file. The file must be created by executing a SaveToFile action.

We have a file, “C:\MyResources\treeView1.xml”. We may create a new application to load this file. For simplicity, we use the same sample application but add a “Clear” button to remove all the tree nodes to prepare for the loading of nodes from the XML file.

Right-click the tree view, choose “Create Action”. Choose “More methods”. Choose “*All methods *=>”. Select “RemoveAllNodes”. Click ✓:


Select Method
Search:
RectangleToClient(Rectangle)Rectangle
RectangleToScreen(Rectangle)Rectangle
Refresh
RemoveAllNodes
ResetBackColor
ResetBindings
Most commonly used methods
AddRootNode(String)TreeNodeX
AddSubNode(String)TreeNodeX
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult

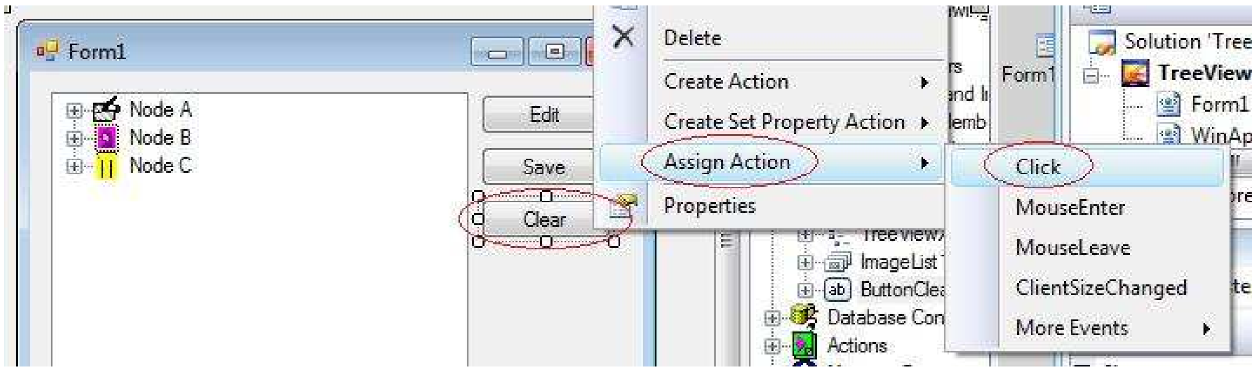
Click OK:



(ActionName)	TreeViewX1.RemoveAllNodes
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	TreeViewX1.RemoveAllNodes

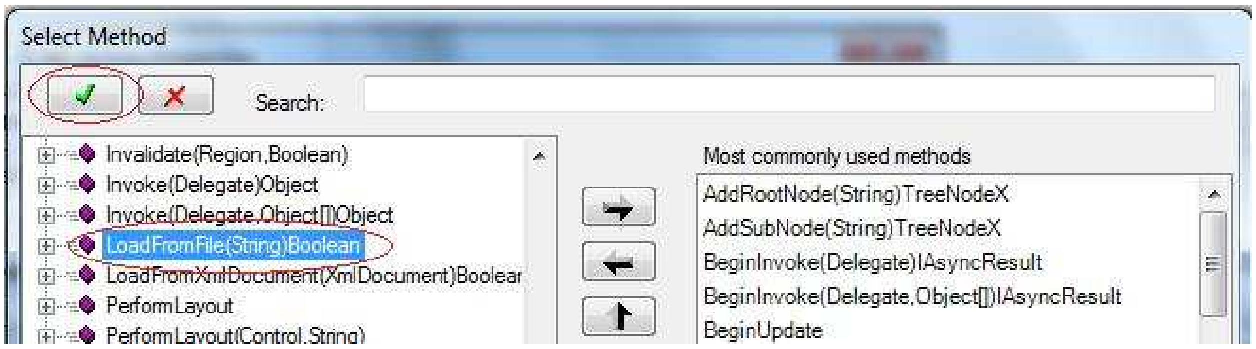
(ActionName)
The name to identify this action

Assign this action a "Clear" button:

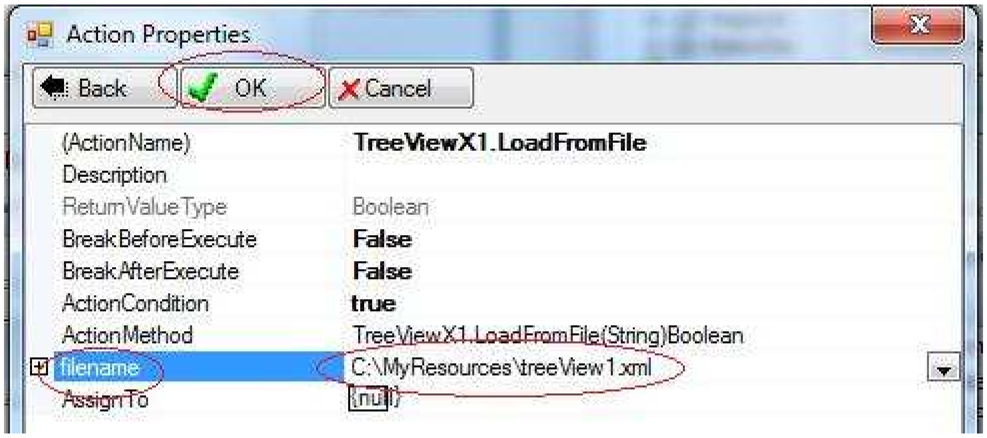


Select action TreeViewX1.RemoveAllNodes to assign to this event.

Now, let's create LoadFromFile action. Right-click the tree view, choose "Create Action". Choose "More methods". Choose "*All methods *=>". Select "LoadFromFile". Click ✓:

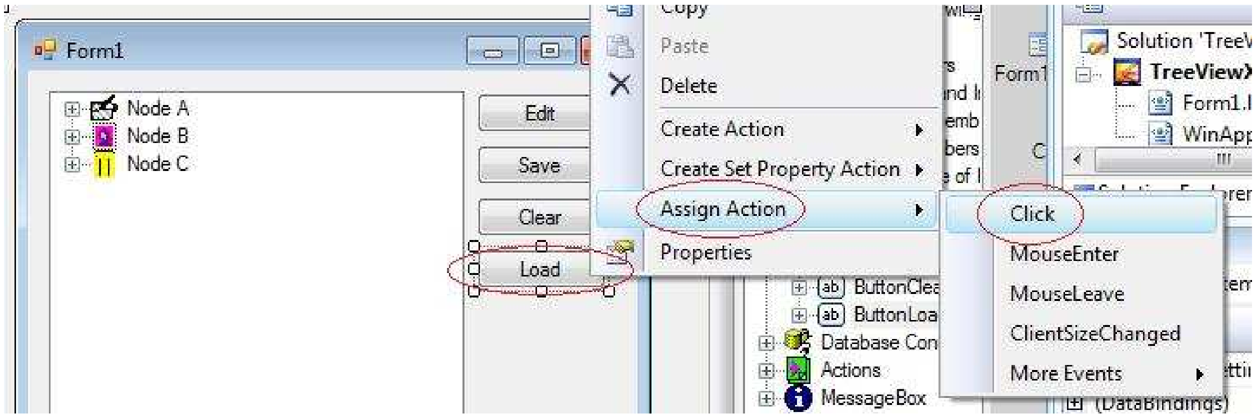

Select Method
Search:
Invalidate(Region,Boolean)
Invoke(Delegate)Object
Invoke(Delegate,Object[])Object
LoadFromFile(String)Boolean
LoadFromXmlDocument(XmlDocument)Boolean
PerformLayout
PerformLayout(Control,String)
Most commonly used methods
AddRootNode(String)TreeNodeX
AddSubNode(String)TreeNodeX
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult
BeginUpdate

This action needs a "filename" parameter indicating the name of file to be created. For this sample, we use a constant "C:\MyResources\treeView1.xml":

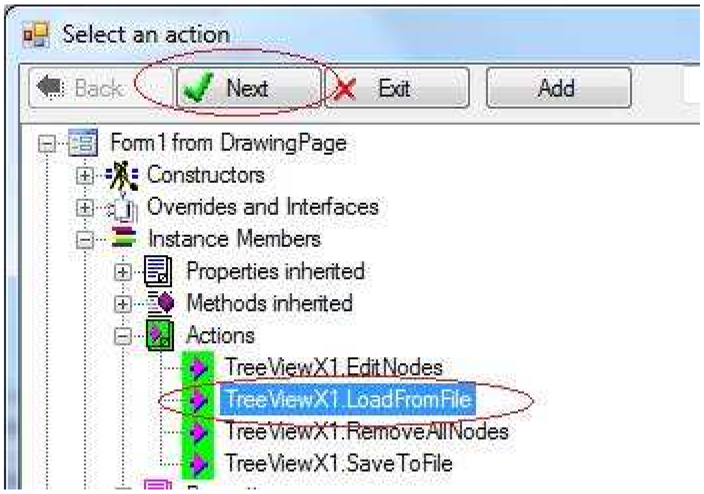


(ActionName)	TreeViewX1.LoadFromFile
Description	
ReturnValueType	Boolean
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	TreeViewX1.LoadFromFile(String)Boolean
filename	C:\MyResources\treeView1.xml
AssignTo	(null)

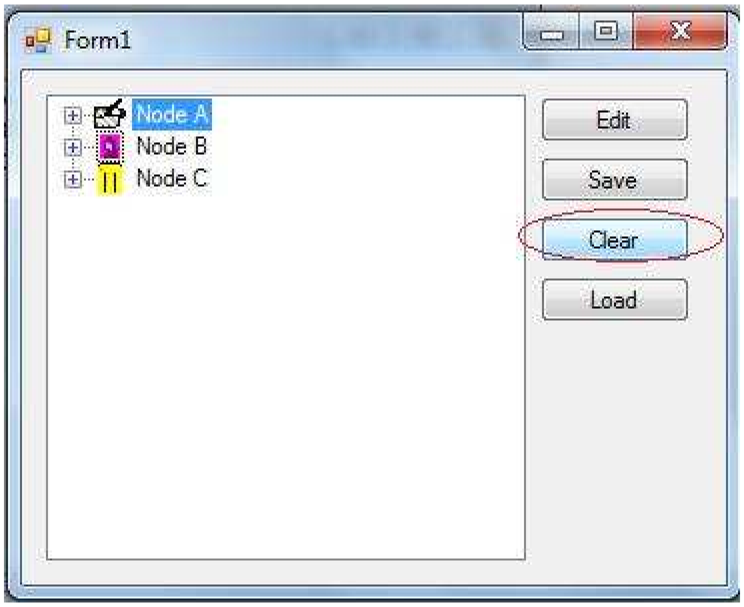
Assign this action to a button:



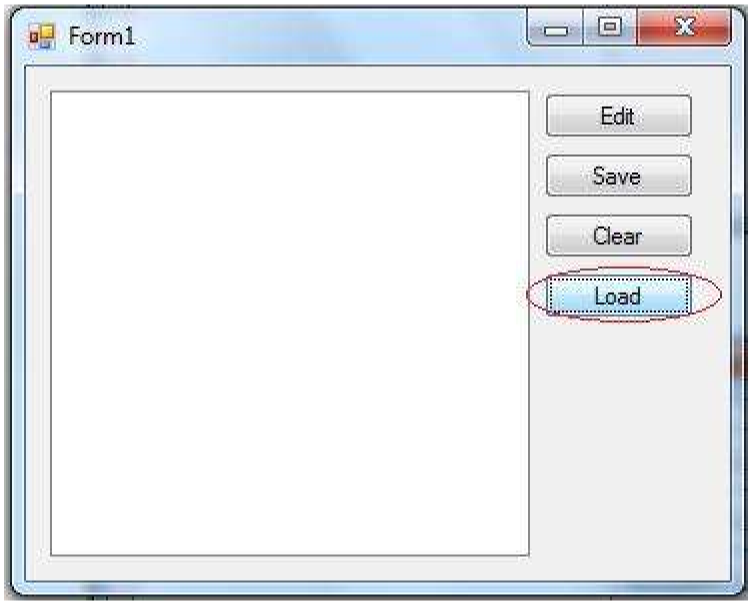
Select the action, click Next:



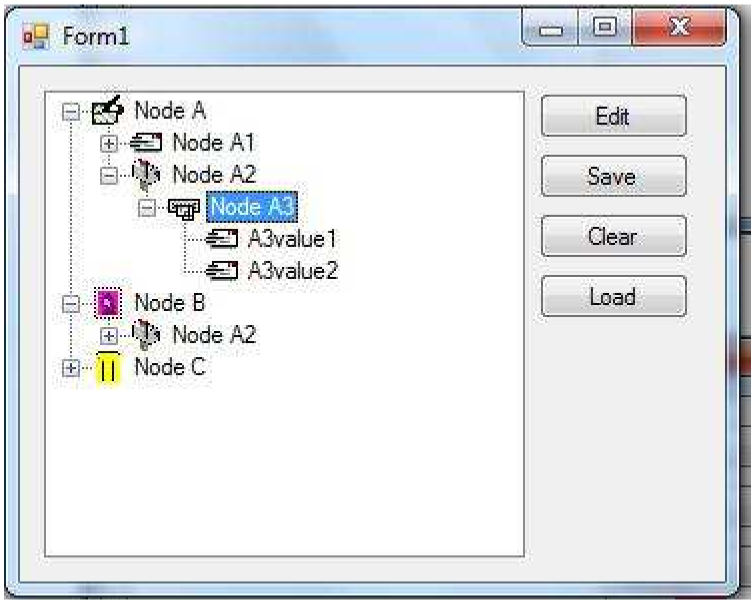
Click ▶ to test it at runtime. The application runs. Click “Clear” button:



All nodes are removed. Click “Load” button:



The nodes are loaded from the XML file:



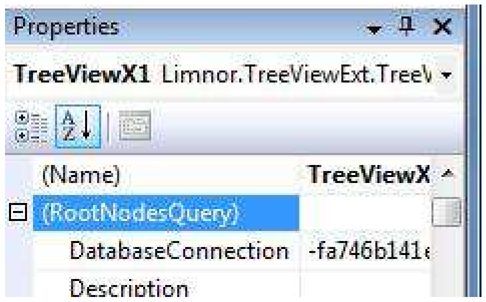
Use Databases

Tree Nodes may be created using data from databases. This chapter describes how to create tree nodes from database query and pass data from database to tree nodes.

Load root nodes

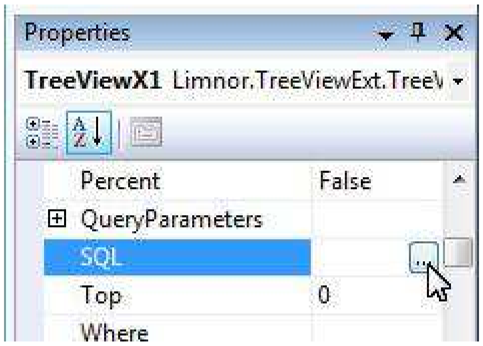
Data Query

A TreeViewX has a RootNodesQuery property for specifying database query to load root nodes:

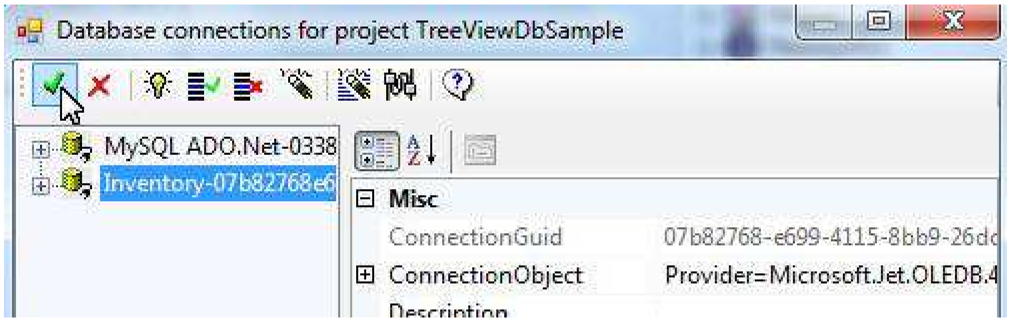


(Name)	TreeViewX
(RootNodesQuery)	
DatabaseConnection	-fa746b141e
Description	

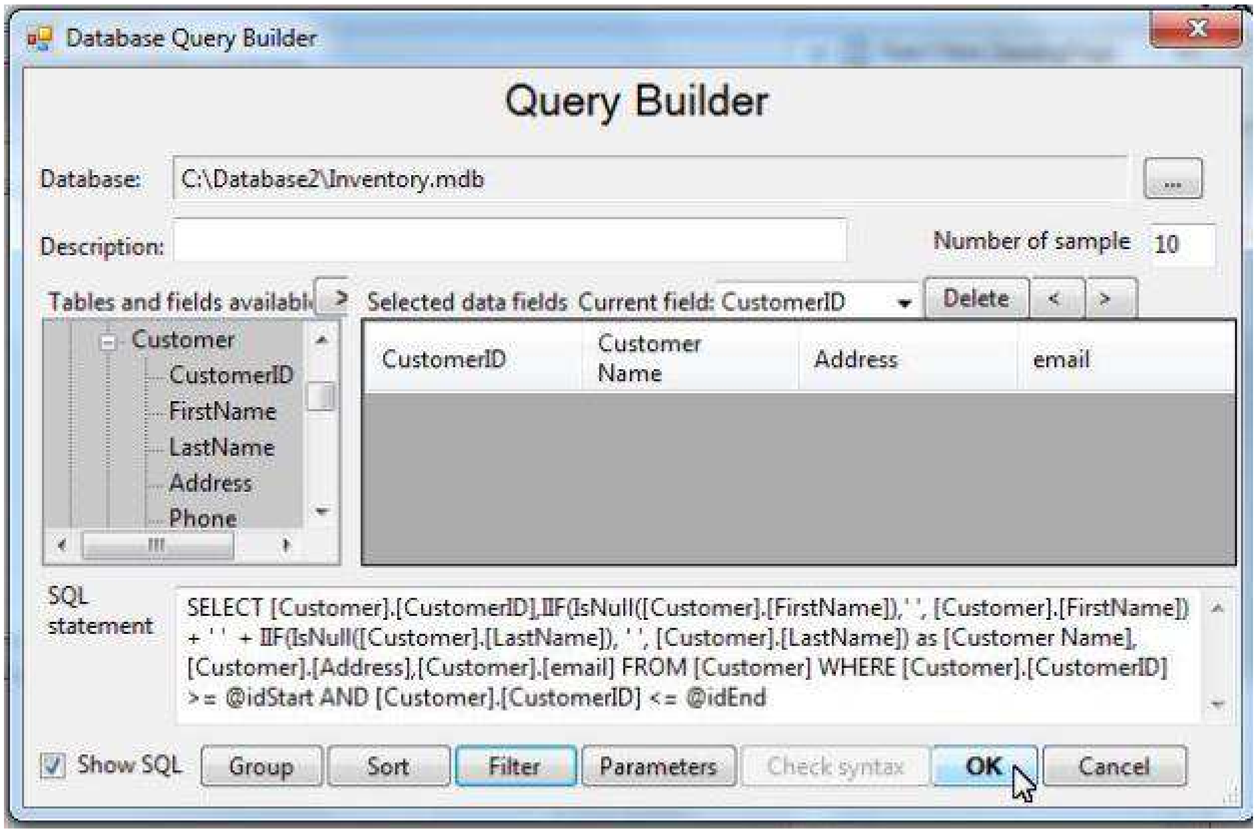
Set the SQL property of the RootNodesQuery to launch the Query Builder:



Select a desired database connection:


Database connections for project TreeViewDbSample
MySQL ADO.Net-0338
Inventory-07b82768e6
Misc
ConnectionGuid 07b82768-e699-4115-8bb9-26dc
ConnectionObject Provider=Microsoft.Jet.OLEDB.4
Description

Suppose the database has a Customer table; suppose we want to use a Tree Node for each Customer record. We may build a query to include desired customer information:



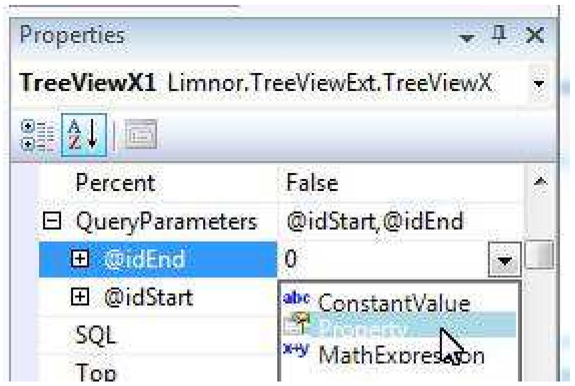
```
SELECT [Customer].[CustomerID],IIF(IsNull([Customer].[FirstName]),' ', [Customer].[FirstName])
+ ' ' + IIF(IsNull([Customer].[LastName]), ' ', [Customer].[LastName]) as [Customer Name],
[Customer].[Address],[Customer].[email] FROM [Customer] WHERE [Customer].[CustomerID]
>= @idStart AND [Customer].[CustomerID] <= @idEnd
```

Note that we use two parameters in this sample query.

For more information on how to use database and Query Builder, see http://www.limnor.com/support/Limnor%20Studio%20-%20User%20Guide%20-%20Part%20VI.pdf

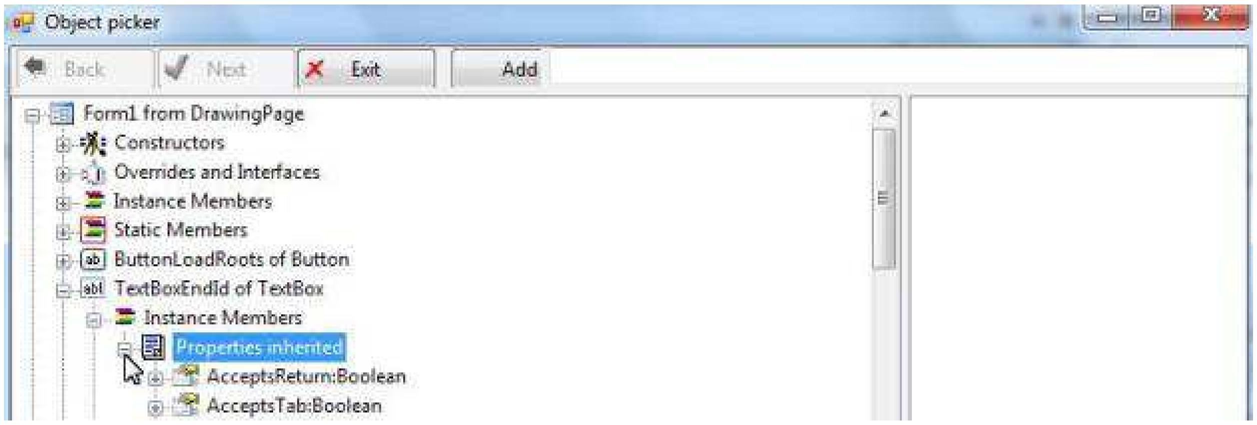
All parameters defined in the query are listed under QueryParameters. We may create actions to set parameter values at runtime. For this sample, we simply link each parameter to a Text Box at design time.

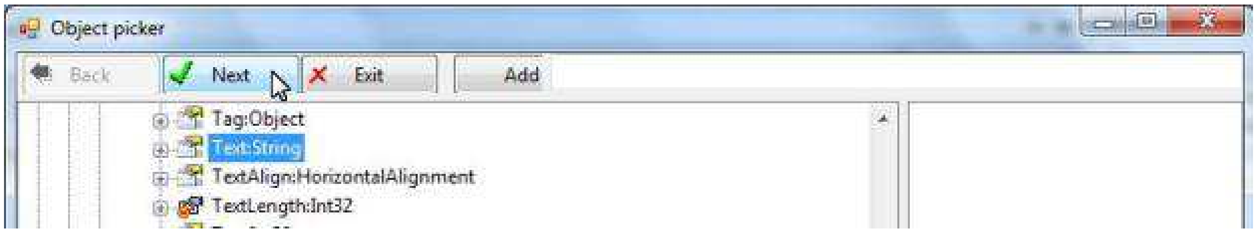
Select Property for parameter @idEnd:



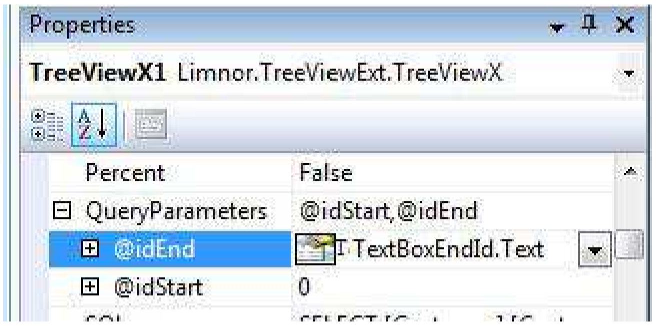
Percent	False
QueryParameters	@idStart,@idEnd
@idEnd	0
@idStart	
SQL	
Top	

Select Text property of the Text Box named TextBoxEndId:



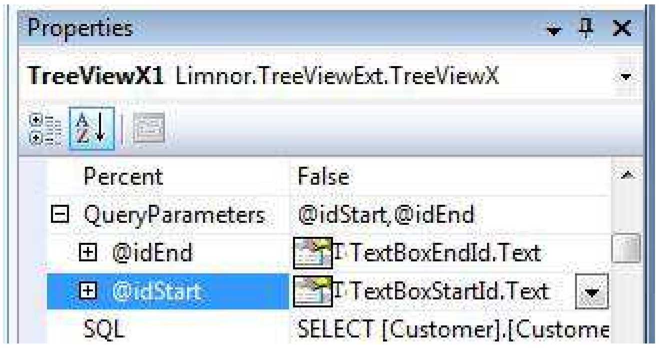


The Text property of the text box TextBoxEndId is assigned to the @idEnd:



Percent	False
⊟ QueryParameters	@idStart,@idEnd
⊞ @idEnd	TextBoxEndId.Text
⊞ @idStart	0

We also assign the Text property of TextBoxStartId to @idStart:



Percent	False
⊟ QueryParameters	@idStart,@idEnd
⊞ @idEnd	TextBoxEndId.Text
⊞ @idStart	TextBoxStartId.Text
SQL	SELECT [Customer].[Custome

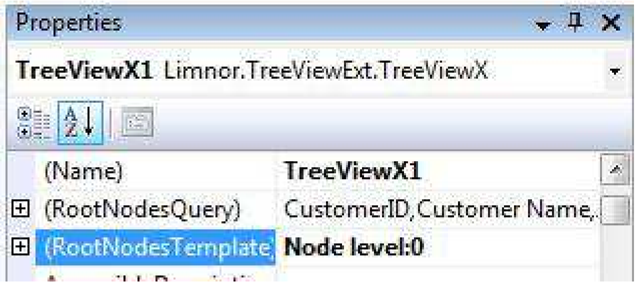
We are done setting up data query for loading root nodes.

Every tree node has a Fields property. The query results will be available through the Fields property.

To apply the query results to each tree node, RootNodesTemplate property can be used.

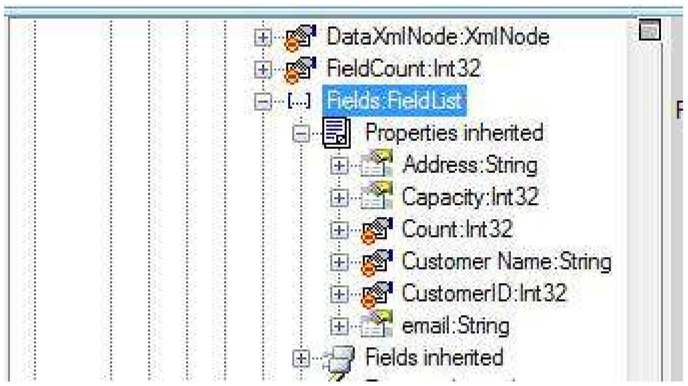
Node template

A TreeViewX has a RootNodesTemplate property. We may use this property to specify how each node should be created.



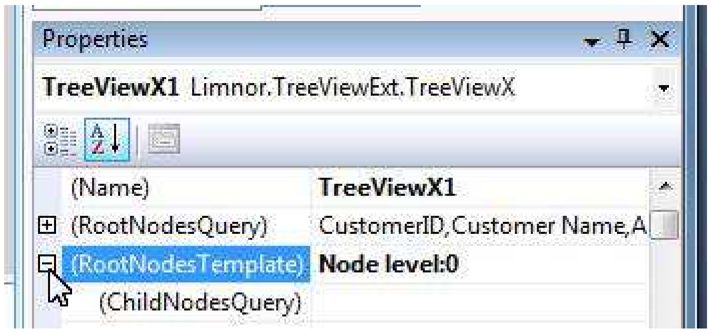
(Name)	TreeViewX1
⊞ (RootNodesQuery)	CustomerID,Customer Name,.
⊞ (RootNodesTemplate)	Node level:0

RootNodesTemplate has a Fields property, which shows fields defined by the data query

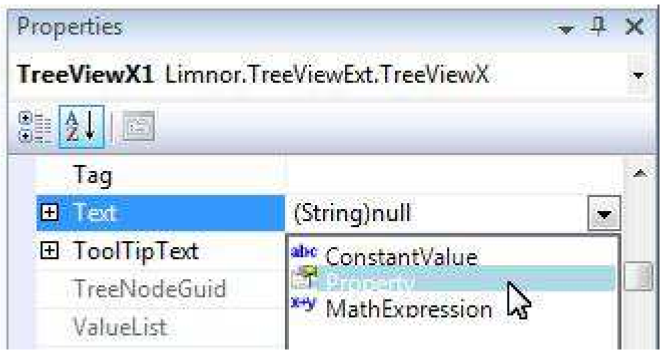


Suppose we want to display Customer Name field on each node. To do that, we may assign the Customer Name field to the Text property of the node template. It is done as described below.

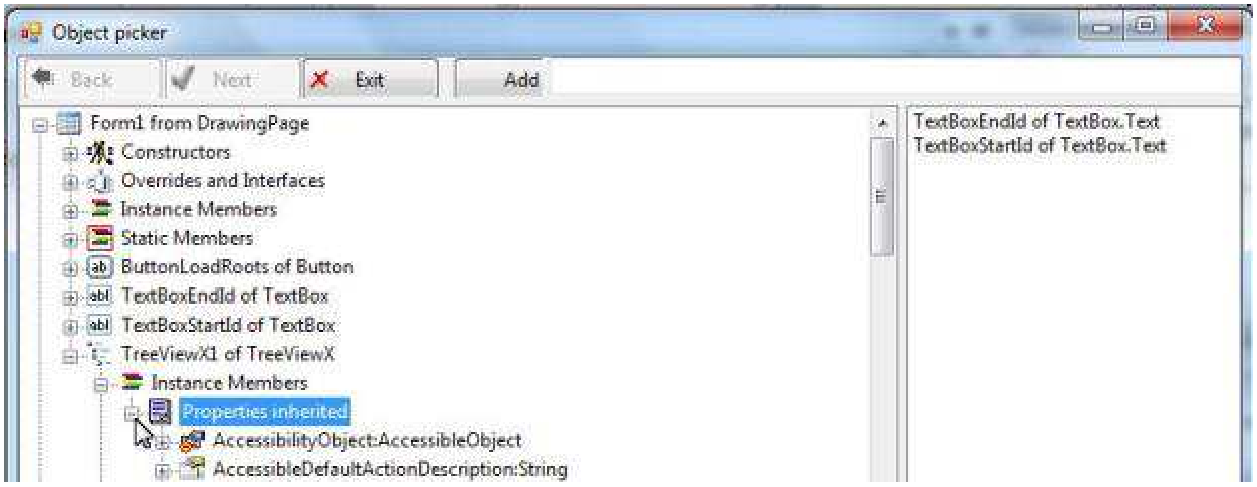
Expand RootNodesTemplate:



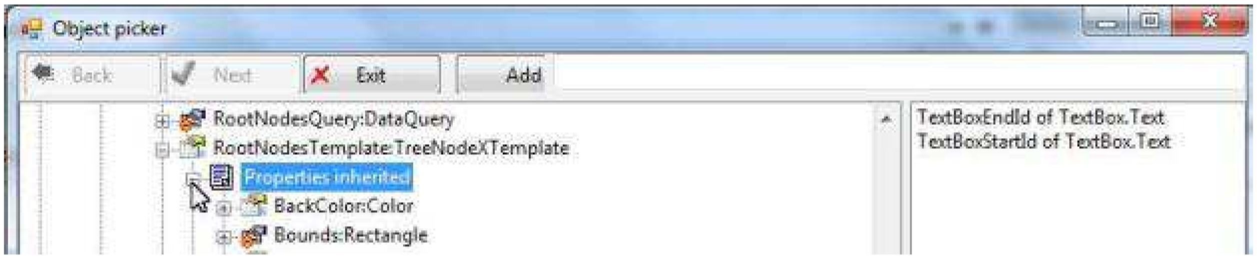
Select Text property and select Property:



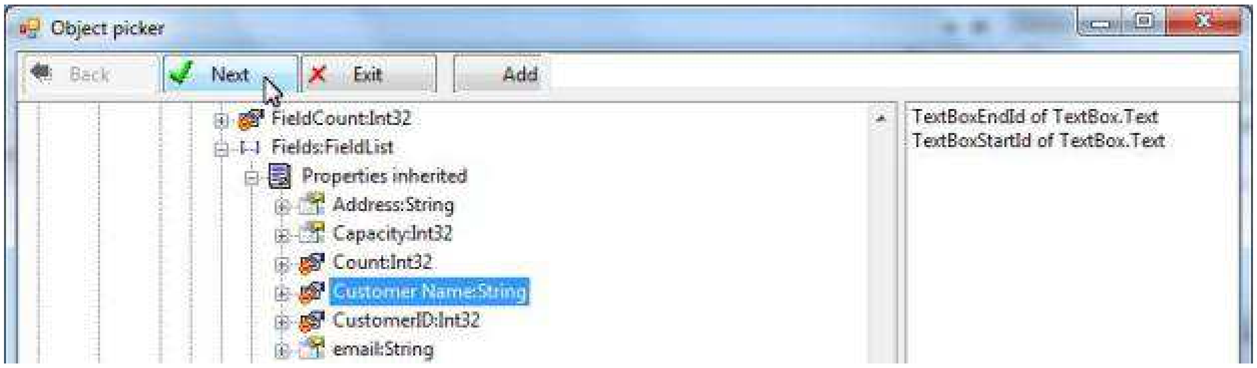
Expand the properties of TreeViewX:



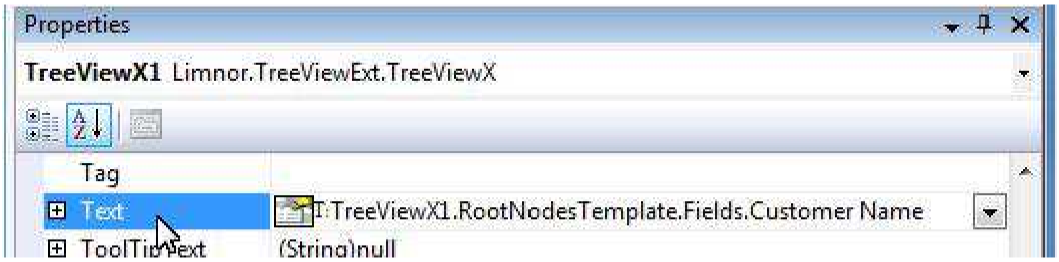
Expand the properties of RootNodesTemplate:



Select Customer Name under Fields:



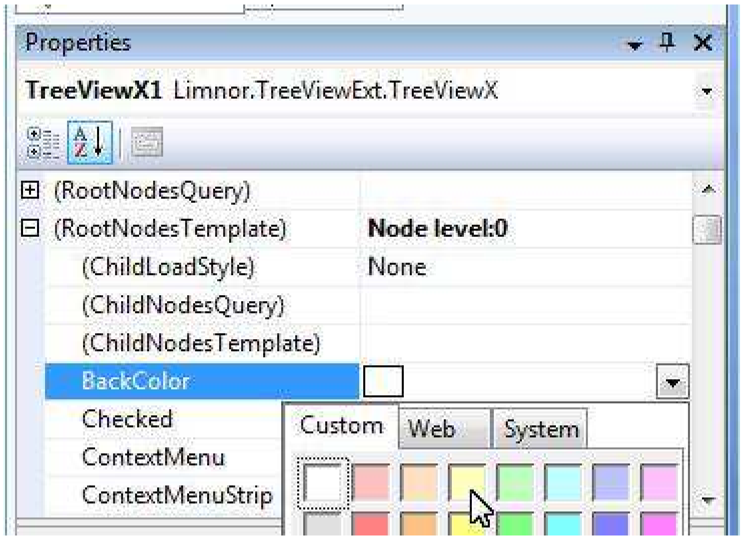
The Customer Name field is assigned to the Text property of the root node template:



Tag	
Text	T:TreeViewX1.RootNodesTemplate.Fields.Customer Name
ToolTipText	(String)null

We may set other properties of the template node. For example, set the ImageIndex and SelectedImagedIndex for showing node icons. It does not have to be linked to database fields. For example, ImageIndex may be linked to the Level property of the node if you want to use one icon for one level.

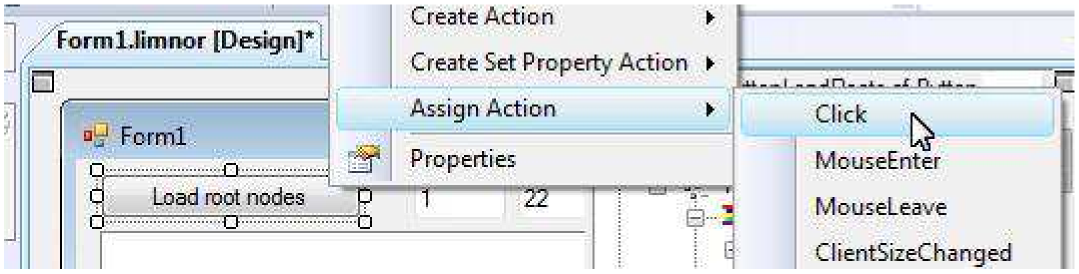
As another example of setting template, we set the BackColor to light yellow:



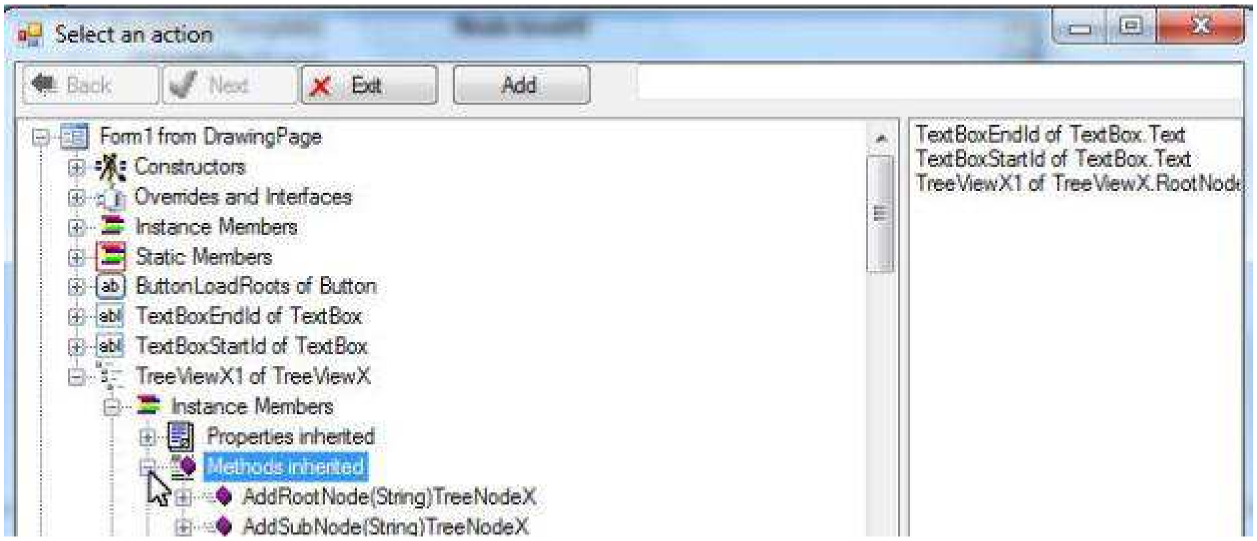
(RootNodesQuery)	
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplate)	
BackColor	
Checked	
ContextMenu	
ContextMenuStrip	

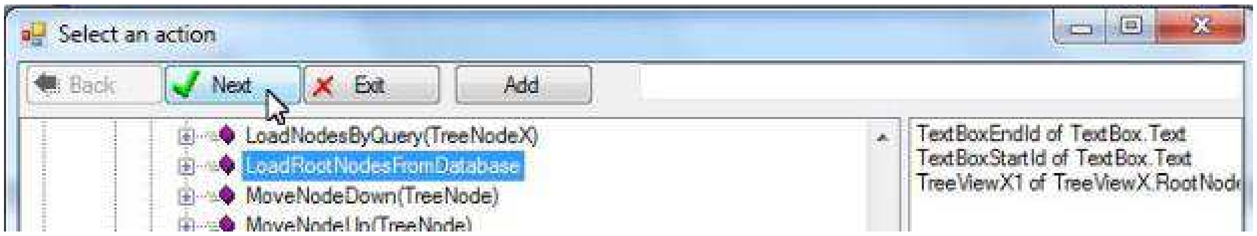
Load root nodes

A TreeViewX has a `LoadRootNodesFromDatabase` method. In this sample it is executed by clicking a button. Right-click the button; choose “Assign Action”; choose “Click”:

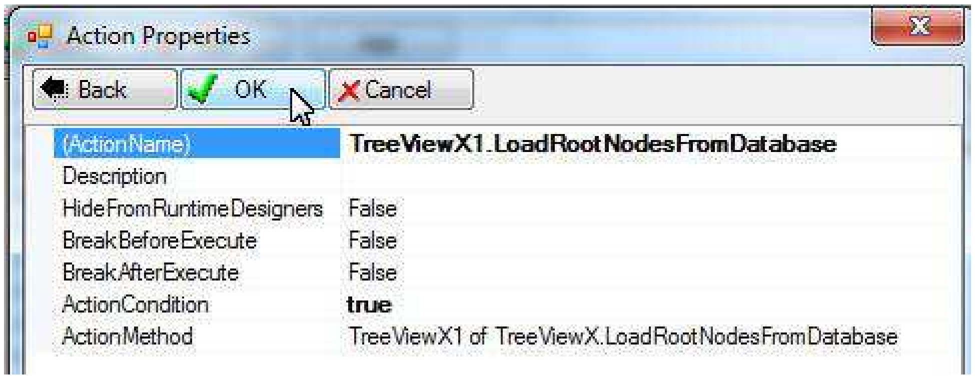


Select `LoadRootNodesFromDatabase` of the TreeViewX:

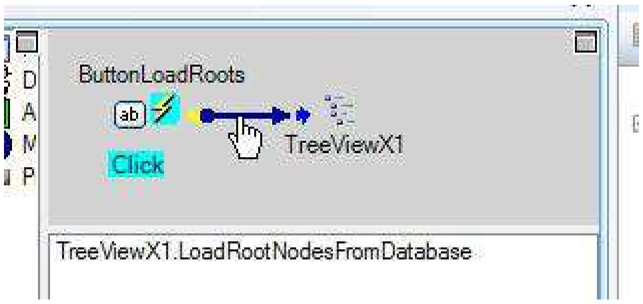



Select an action
Back
Next
Exit
Add
LoadNodesByQuery(TreeNodeX)
LoadRootNodesFromDatabase
MoveNodeDown(TreeNode)
MoveNodeUp(TreeNode)
TextBoxEndId of TextBox.Text
TextBoxStartId of TextBox.Text
TreeViewX1 of TreeViewX.RootNode

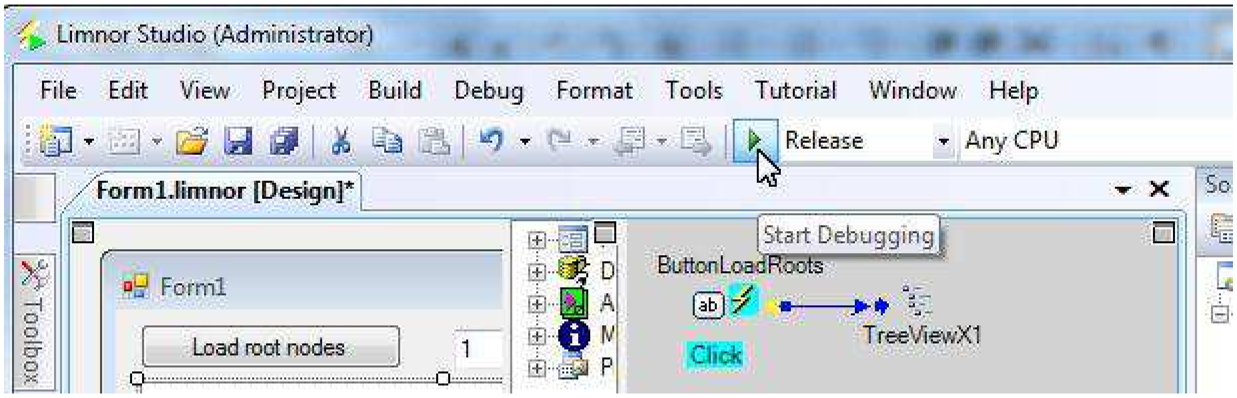
Click OK to create an action and assign it to the button:



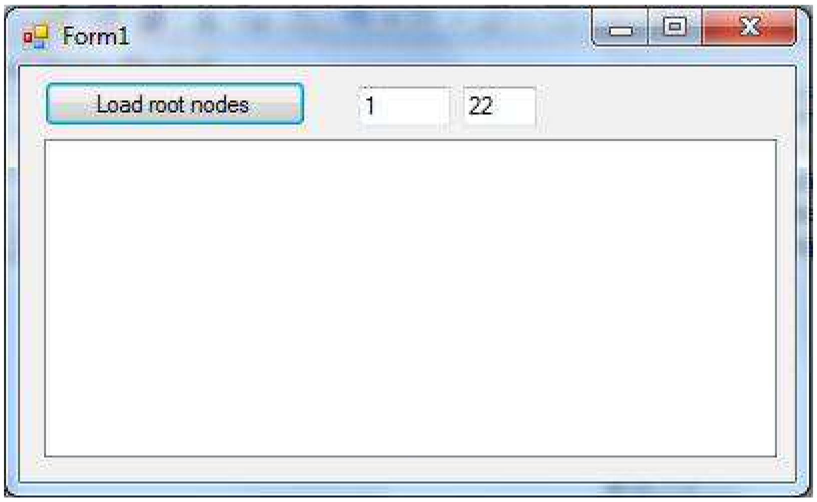
(ActionName)	TreeViewX1.LoadRootNodesFromDatabase
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	TreeViewX1 of TreeViewX.LoadRootNodesFromDatabase



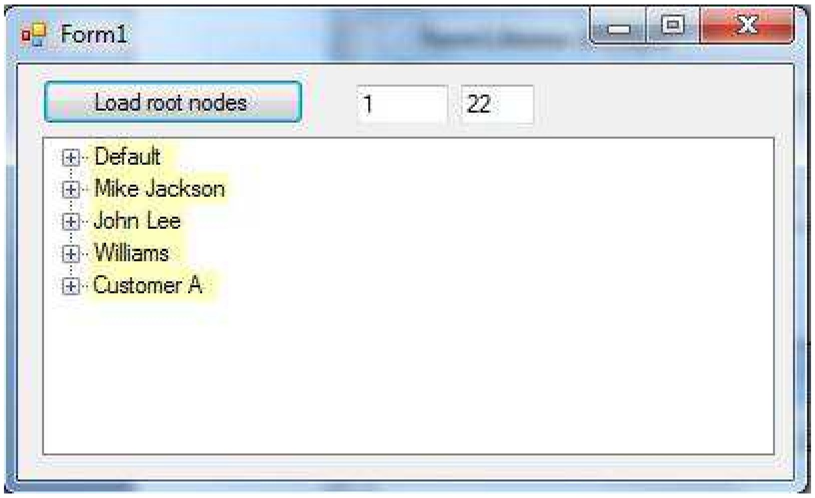
We may test it. Click the Run button:



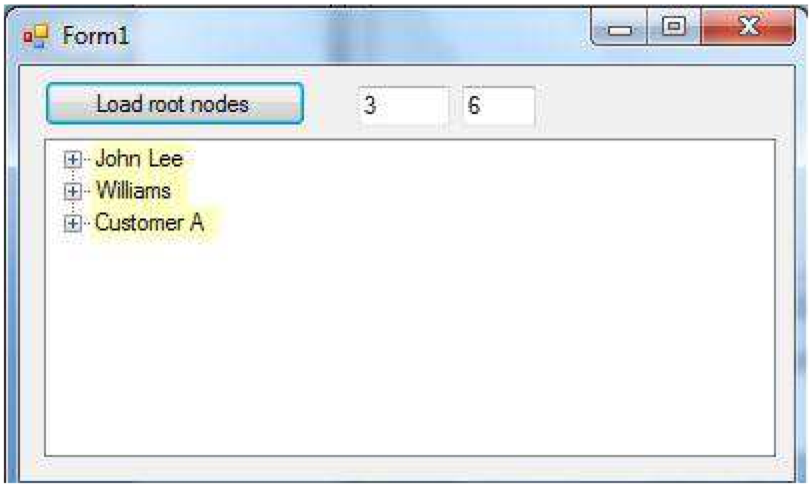
The form appears:



Click the button. A root node is created for each customer record. Background color for the nodes is light yellow:



If we change the values in the text boxes and click the button again then we will see that the range of customers is changed accordingly.

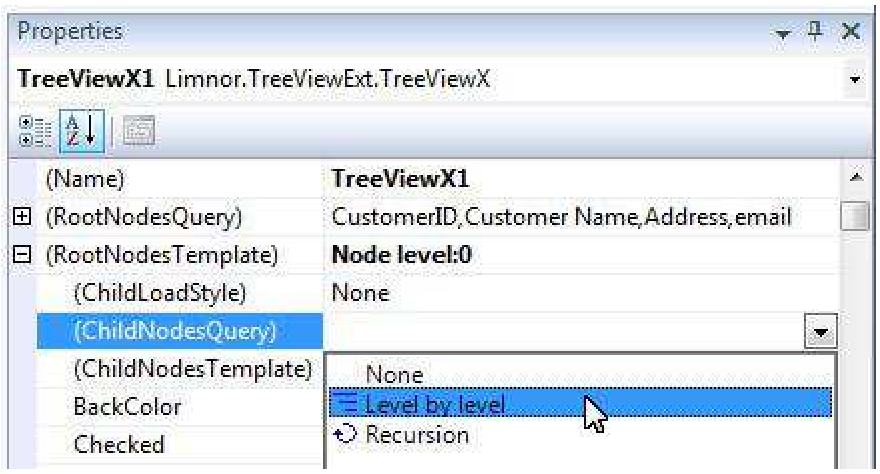


Load child nodes level by level

Suppose our database has an Order table which is linked to the Customer table, that is, each customer may have many orders. We want to display orders for each customer. That is, we want the child nodes of the root nodes to be created by Order records.

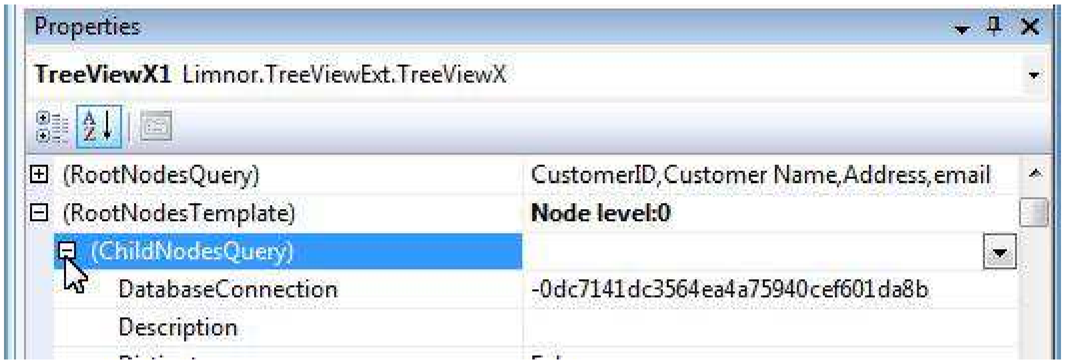
Data query for child nodes

The RootNodesTemplate has a ChildNodesQuery. By default, it is disabled. Turn it on by selecting “Level by level”


Properties
TreeViewX1 Limnor.TreeViewExt.TreeViewX

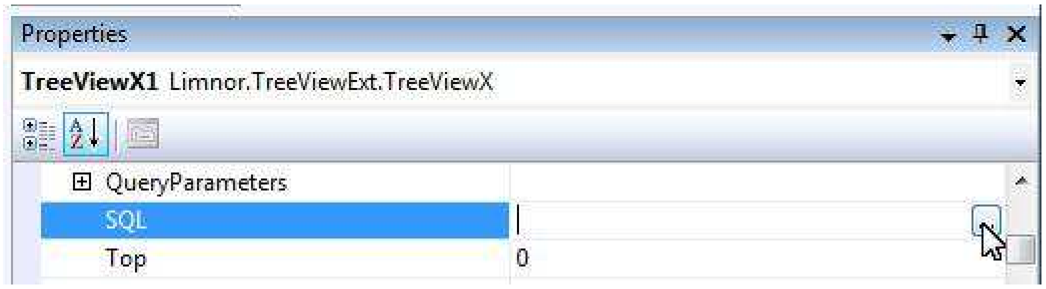
(Name)	TreeViewX1
⊞ (RootNodesQuery)	CustomerID,Customer Name,Address,email
⊟ (RootNodesTemplate)	Node level:0
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplate)	
BackColor	
Checked	

Expand ChildNodesQuery:



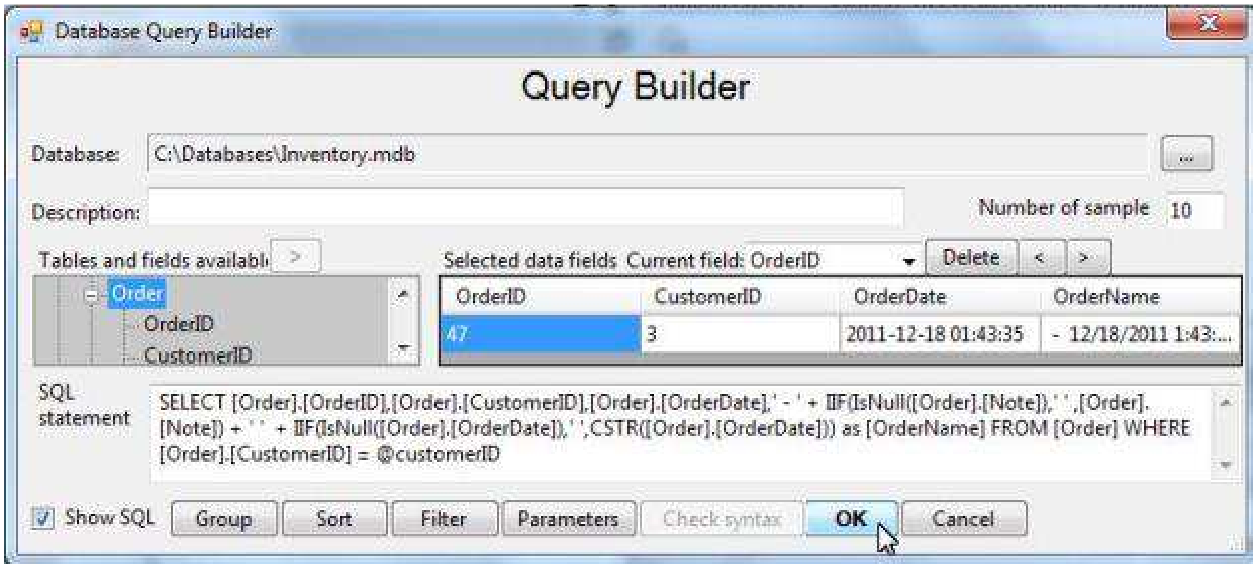
(RootNodesQuery)	CustomerID,Customer Name,Address,email
(RootNodesTemplate)	**Node level:0**
(ChildNodesQuery)	
DatabaseConnection	-0dc7141dc3564ea4a75940cef601da8b
Description	

Set SQL property of the ChildNodesQuery:



QueryParameters	
SQL	
Top	0

Build a query to include desired Order information:



Database: C:\Databases\Inventory.mdb

Description:

Number of sample 10

Selected data fields Current field: OrderID

OrderID	CustomerID	OrderDate	OrderName
47	3	2011-12-18 01:43:35	- 12/18/2011 1:43:...

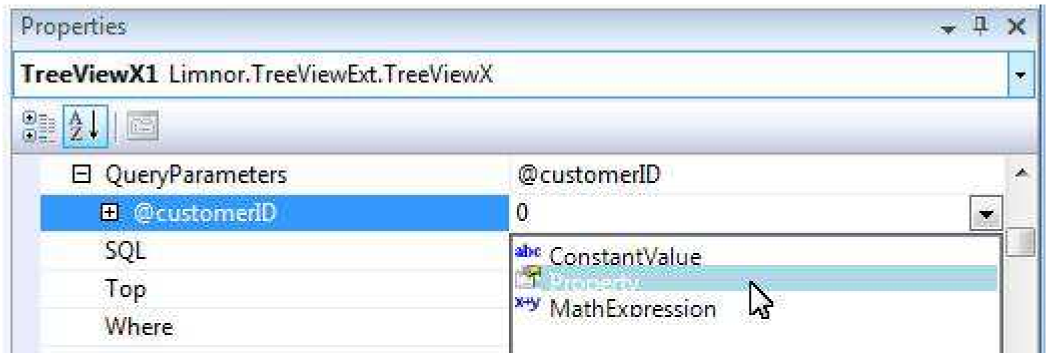
SQL statement:

```
SELECT [Order].[OrderID],[Order].[CustomerID],[Order].[OrderDate],' - ' + IIF(IsNull([Order].[Note]),' ',[Order].
[Note]) + ' ' + IIF(IsNull([Order].[OrderDate]),' ',CSTR([Order].[OrderDate])) as [OrderName] FROM [Order] WHERE
[Order].[CustomerID] = @customerID
```

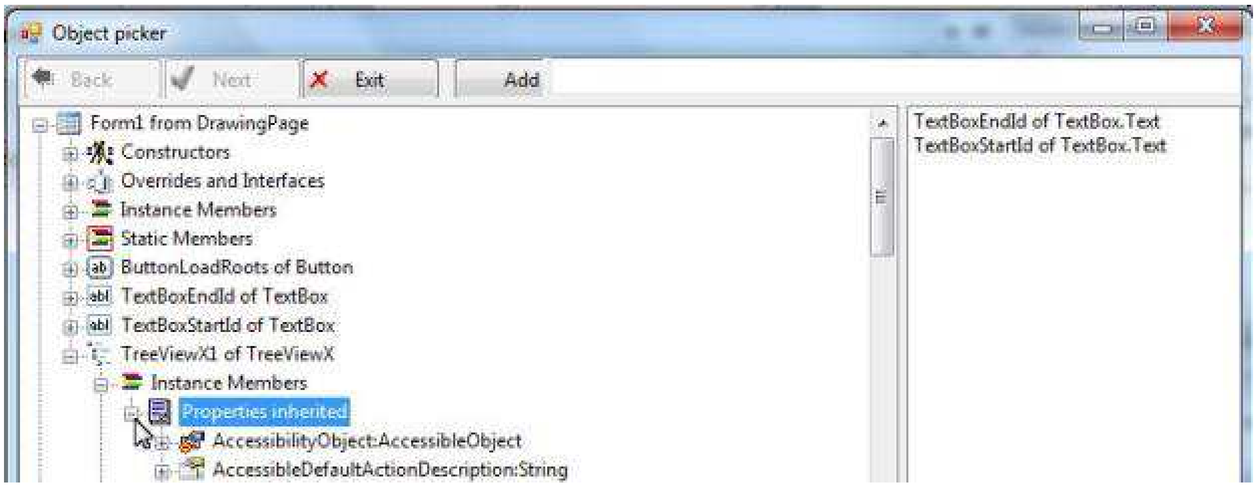
The query forms a “OrderName” field for tree node text.

It uses a parameter, @customerID, to associate Order records to Customer records. We need to link this parameter to the OrderID field of the parent tree node.

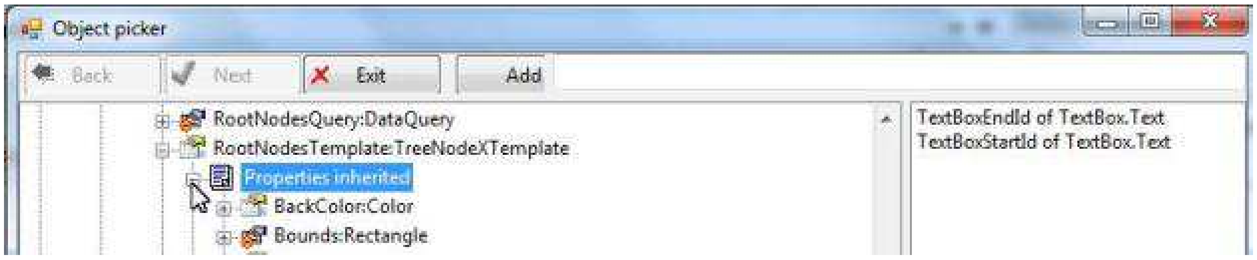
Select Property for the parameter @customerID:



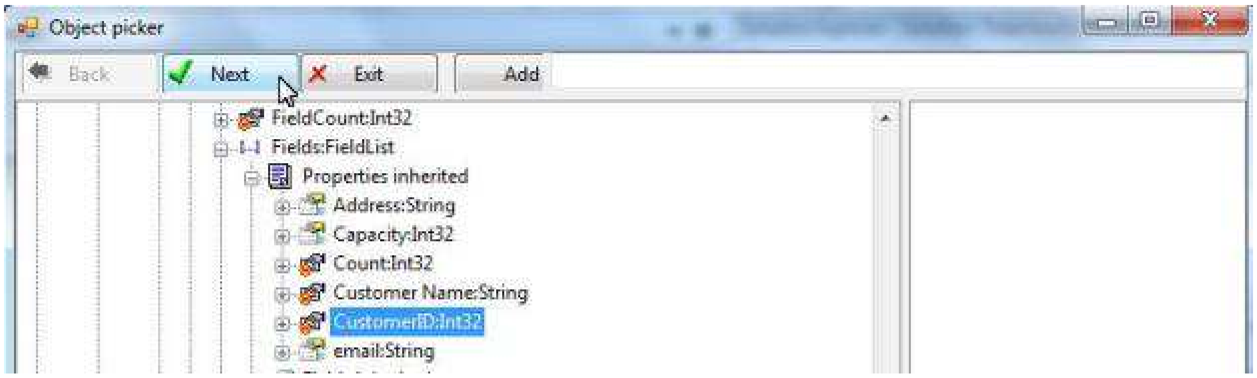
Expand the properties of TreeViewX:



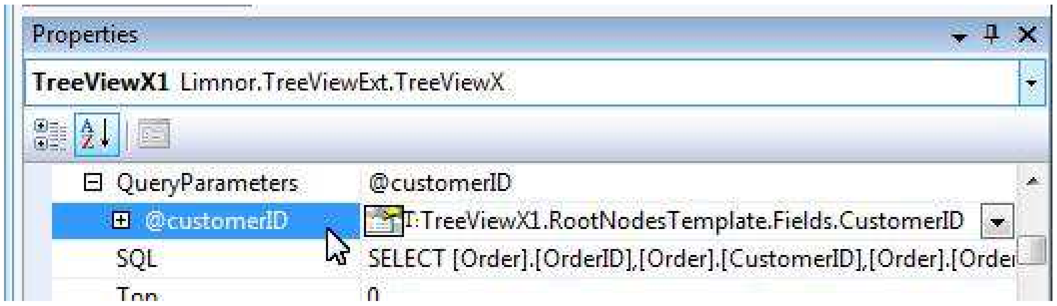
Expand the properties of RootNodesTemplate:



Select CustomerID under Fields:



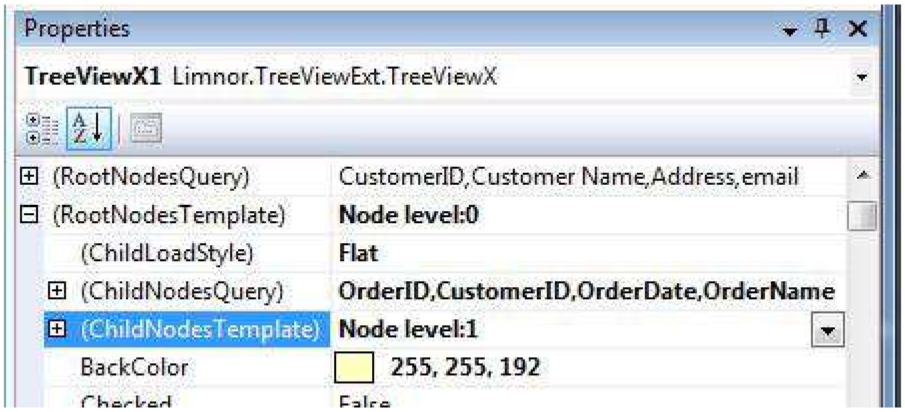
The CustomerID of the RootNodesTemplate is assigned to parameter @customerID:



QueryParameters	@customerID
@customerID	I:TreeViewX1.RootNodesTemplate.Fields.CustomerID
SQL	SELECT [Order].[OrderID],[Order].[CustomerID],[Order].[Orde
Top	0

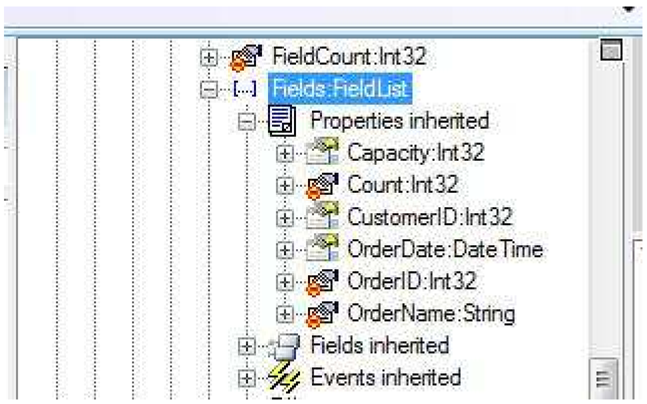
Child node template

The RootNodesTemplate has a ChildNodesTemplate property. We may use this property to specify how each child node should be created.



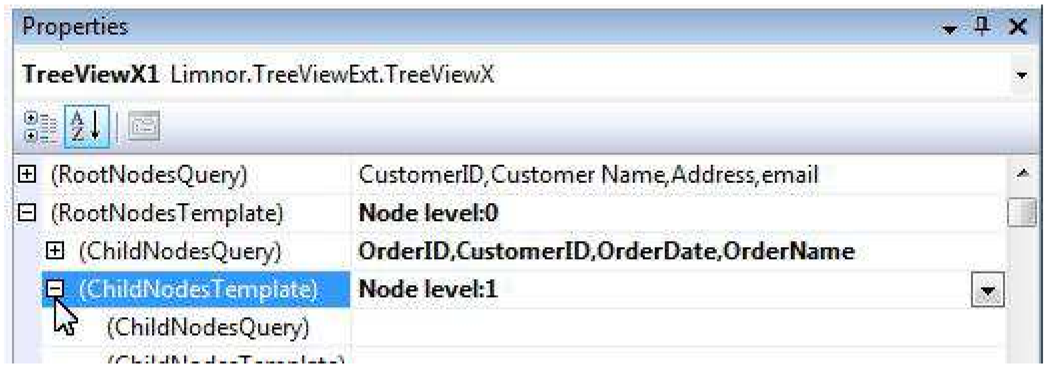
(RootNodesQuery)	CustomerID,Customer Name,Address,email
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	Flat
(ChildNodesQuery)	OrderID,CustomerID,OrderDate,OrderName
(ChildNodesTemplate)	Node level:1
BackColor	255, 255, 192
Checked	False

The child node template has a Fields property, which shows fields defined by the data query:



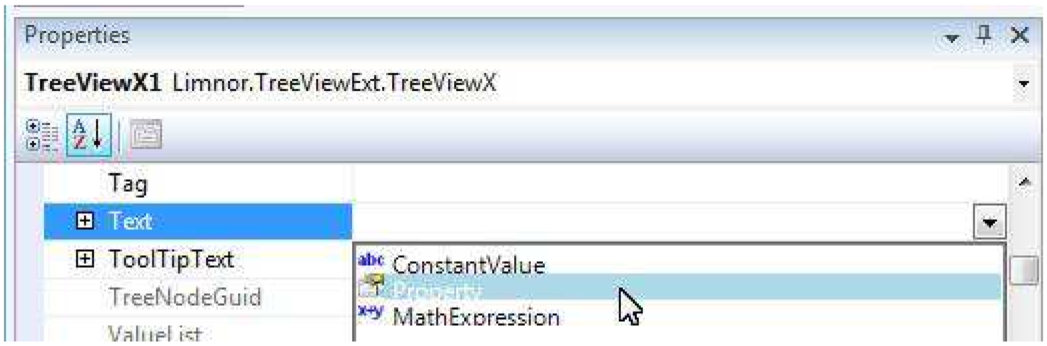
Suppose we want to display OrderName field on each node. To do that, we may assign the OrderName field to the Text property of the node template. It is done as described below.

Expand ChildNodesTemplate:

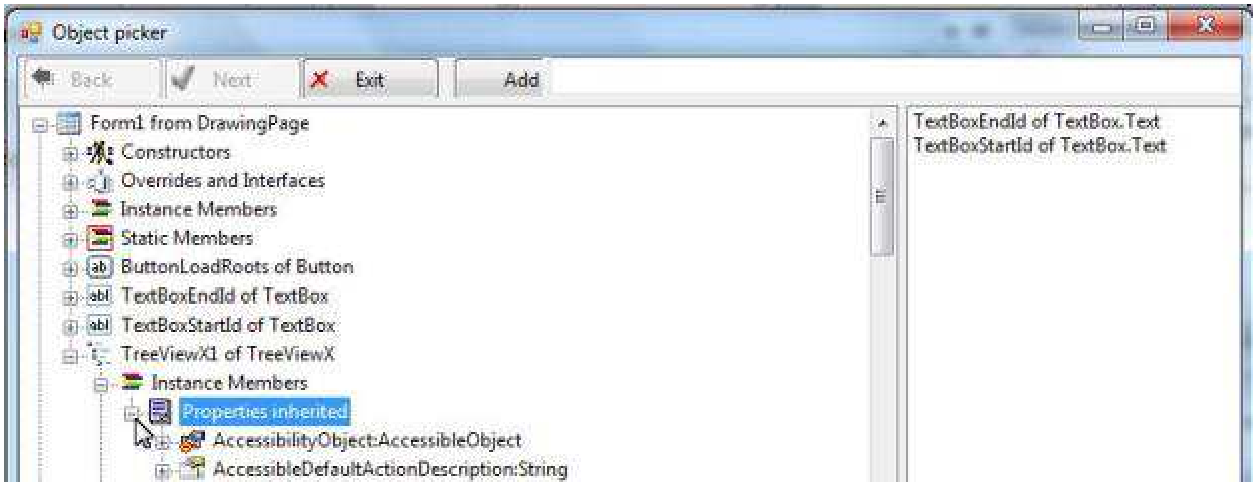


(RootNodesQuery) CustomerID,Customer Name,Address,email
(RootNodesTemplate) Node level:0
(ChildNodesQuery) OrderID,CustomerID,OrderDate,OrderName
(ChildNodesTemplate) Node level:1
(ChildNodesQuery)

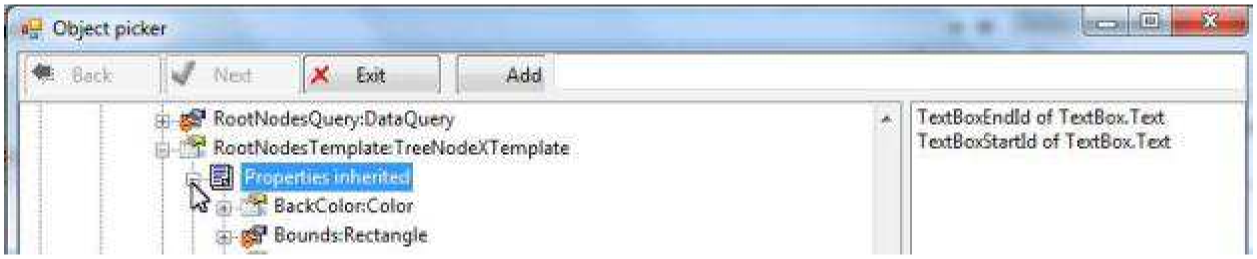
Select Text property and select Property:



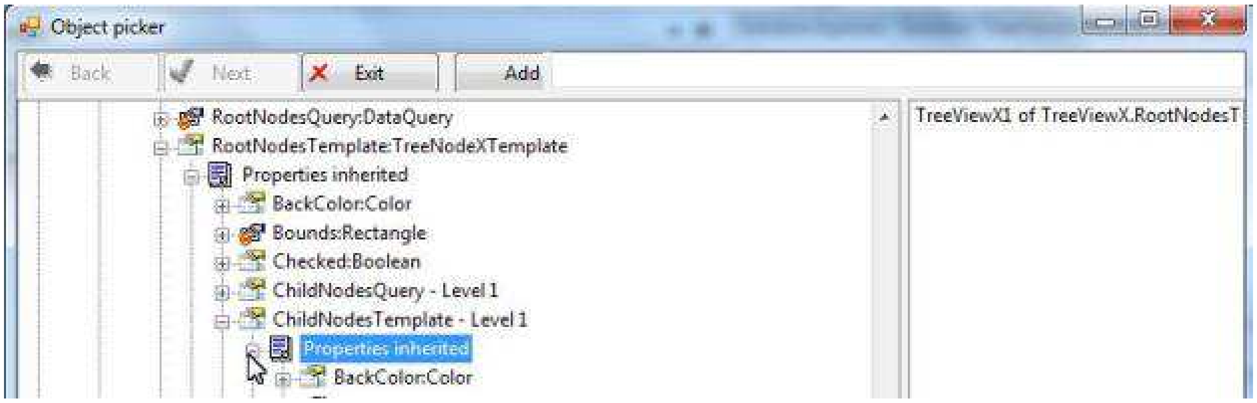
Expand the properties of TreeViewX:



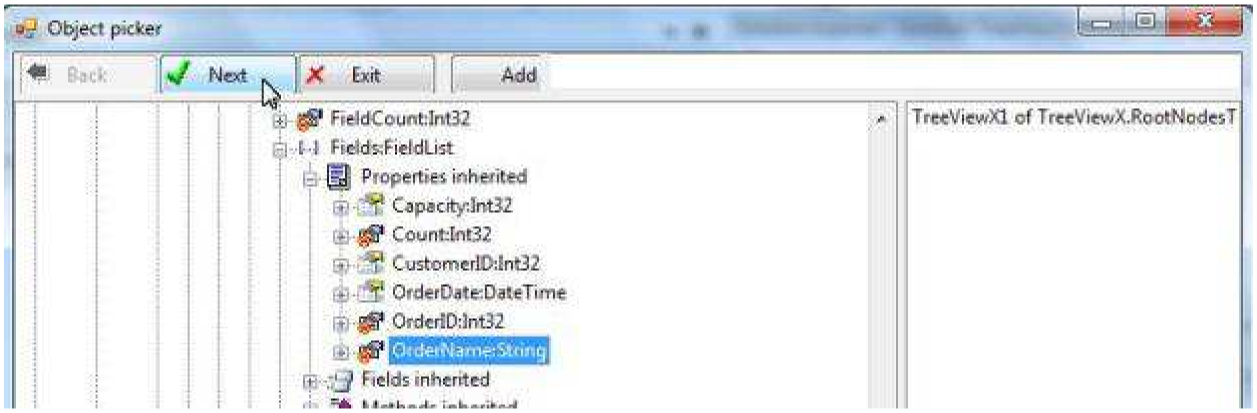
Expand the properties of RootNodesTemplate:



Expand the properties of ChildNodesTemplate:



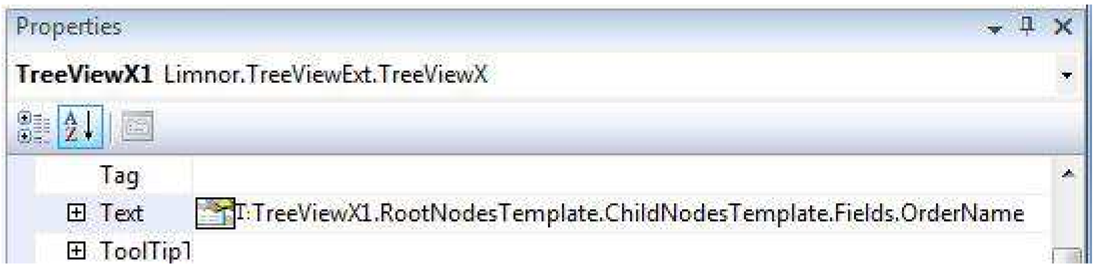
Select OrderName field under Fields:



FieldCount:Int32
Fields:FieldList
Properties inherited
Capacity:Int32
Count:Int32
CustomerID:Int32
OrderDate:DateTime
OrderID:Int32
OrderName:String
Fields inherited

TreeViewX1 of TreeViewX.RootNodesT

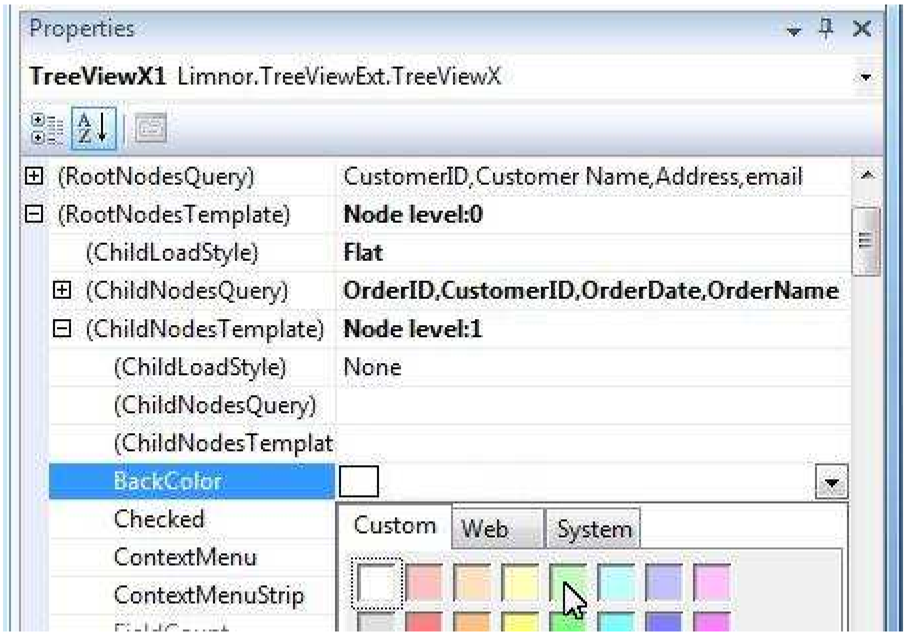
The OrderName field is assigned to the Text property of the child node template:



TreeViewX1 Limnor.TreeViewExt.TreeViewX

Tag	
Text	T:TreeViewX1.RootNodesTemplate.ChildNodesTemplate.Fields.OrderName
ToolTip1	

We may set other properties of the template node. For example, we set the BackColor to light green:

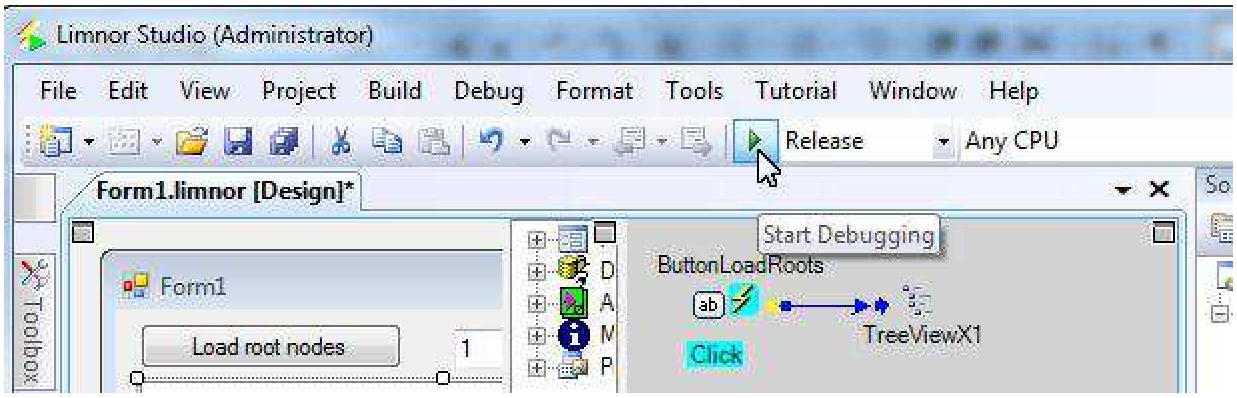


TreeViewX1 Limnor.TreeViewExt.TreeViewX

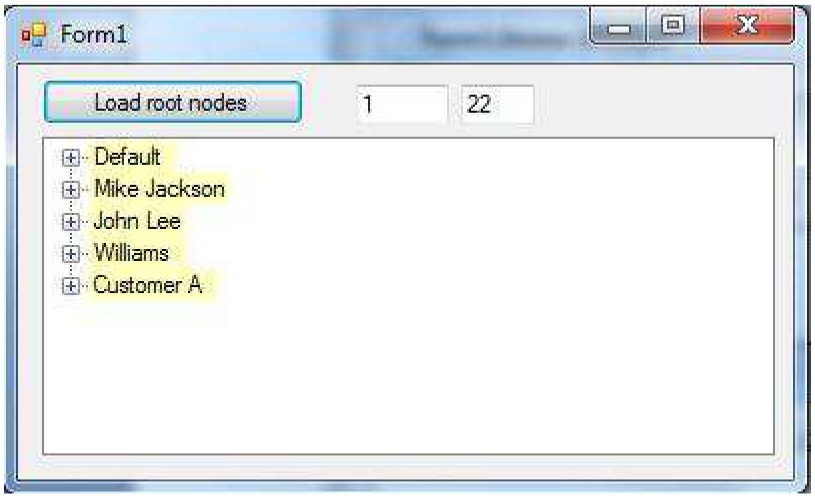
(RootNodesQuery)	CustomerID,Customer Name,Address,email
(RootNodesTemplate)	**Node level:0**
(ChildLoadStyle)	**Flat**
(ChildNodesQuery)	**OrderID,CustomerID,OrderDate,OrderName**
(ChildNodesTemplate)	**Node level:1**
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplat	
BackColor	⬜
Checked	
ContextMenu	
ContextMenuStrip	

Test

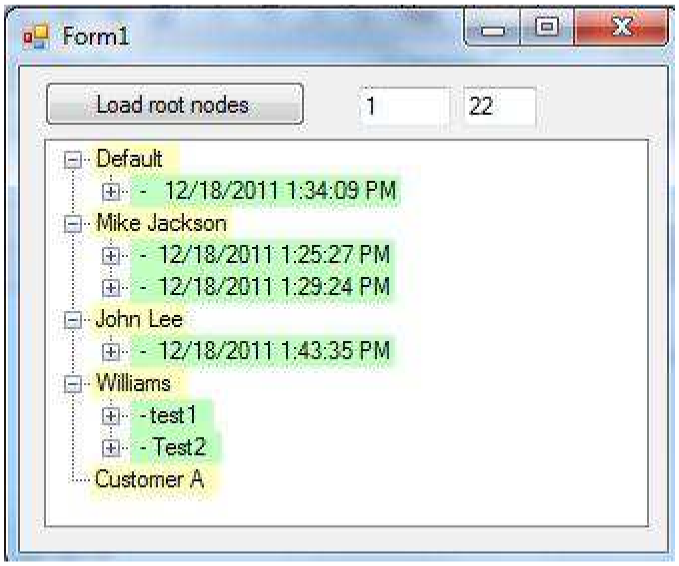
The above settings will display Order nodes under Customer nodes. Let’s test it by clicking the Run button.



Click the button to load the root nodes:



Expand each root node, order nodes appear under each root node:

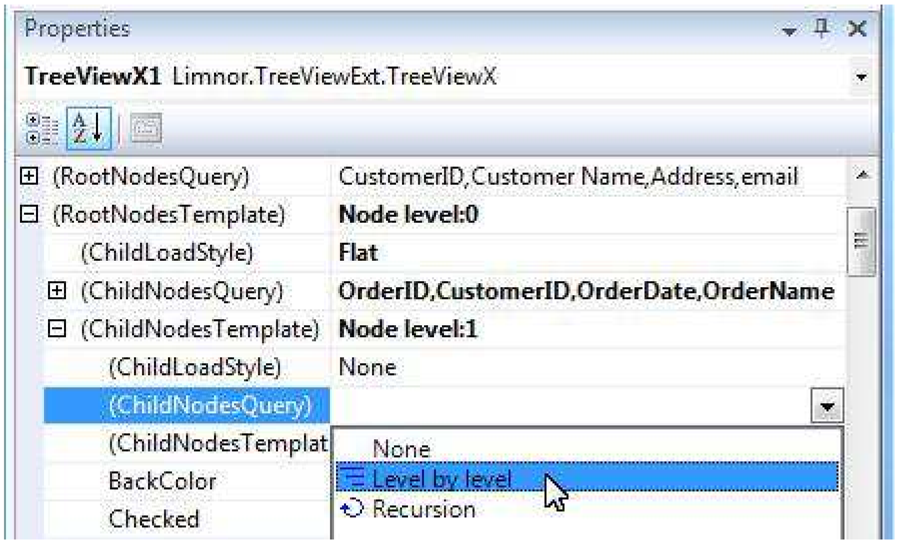


Deeper level child nodes

Every ChildNodesTemplate in turn has its own ChildNodesQuery and ChildNodesTemplate properties for loading its own children. Their usage is exactly the same. Let's show one more example of loading tree nodes level by level.

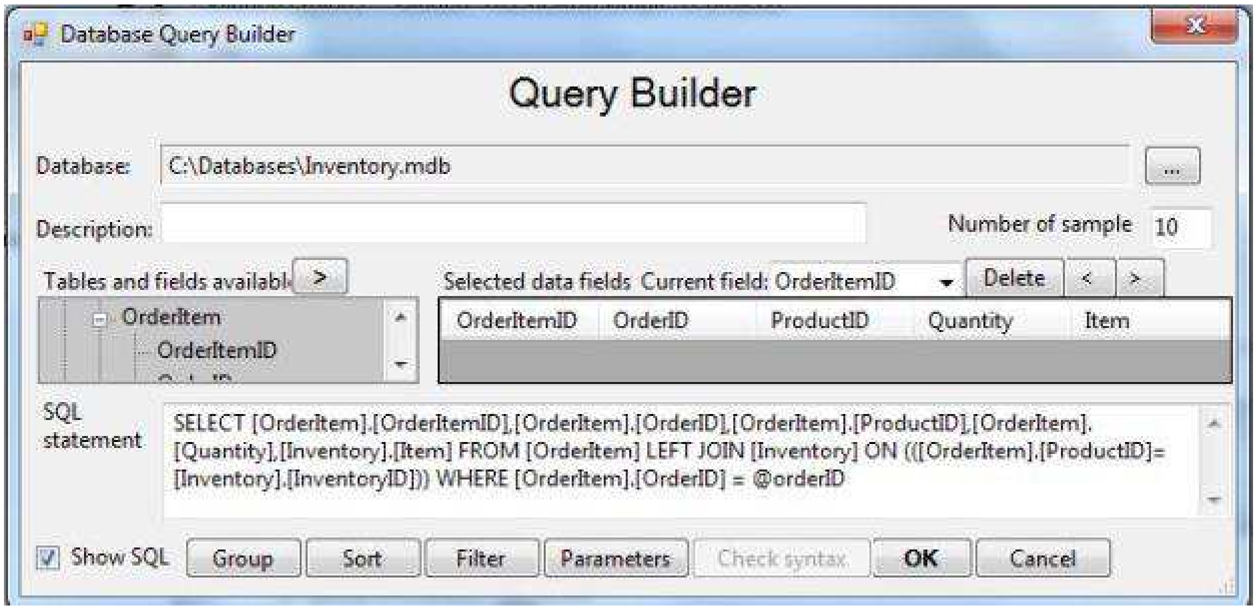
Suppose we want to display order items under each order node. It is done in the same steps for loading order nodes:

1. Select "Level by level":



(RootNodesQuery)	CustomerID,Customer Name,Address,email
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	Flat
(ChildNodesQuery)	OrderID,CustomerID,OrderDate,OrderName
(ChildNodesTemplate)	Node level:1
(ChildLoadStyle)	None
(ChildNodesQuery)	

2. Setup data query



Database: C:\Databases\Inventory.mdb

Description:

Number of sample 10

Tables and fields availabl

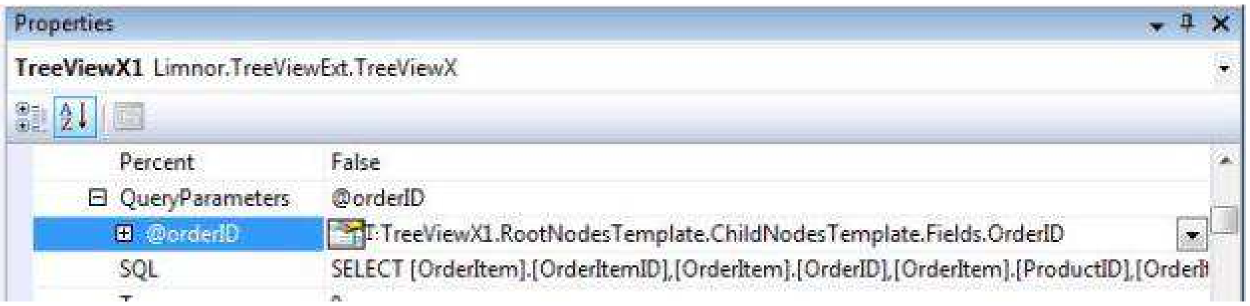
Selected data fields Current field: OrderItemID

OrderItemID	OrderID	ProductID	Quantity	Item

SQL statement:

```
SELECT [OrderItem].[OrderItemID],[OrderItem].[OrderID],[OrderItem].[ProductID],[OrderItem].
[Quantity],[Inventory].[Item] FROM [OrderItem] LEFT JOIN [Inventory] ON (([OrderItem].[ProductID]=
[Inventory].[InventoryID])) WHERE [OrderItem].[OrderID] = @orderID
```

3. Link query parameters to the values in the parent templates

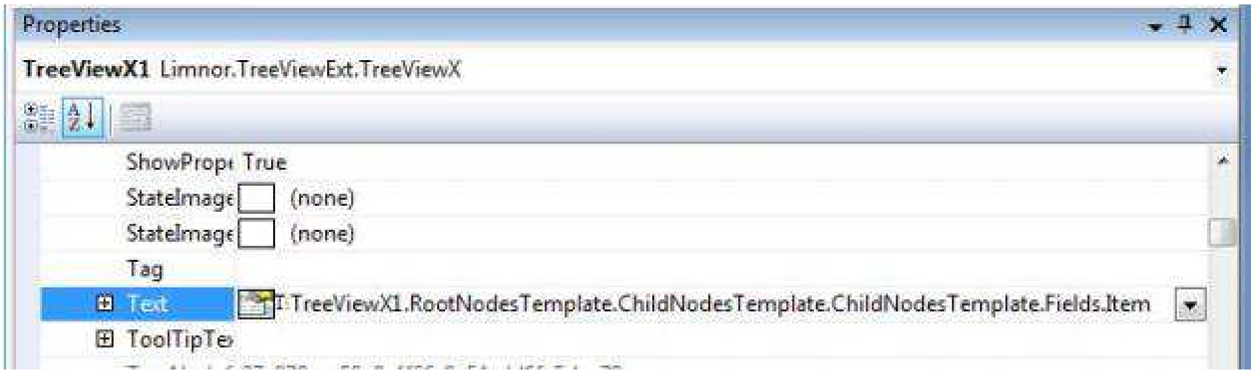


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

Percent	False
⊟ QueryParameters	@orderID
⊞ @orderID	I:TreeViewX1.RootNodesTemplate.ChildNodesTemplate.Fields.OrderID
SQL	SELECT [OrderItem].[OrderItemID],[OrderItem].[OrderID],[OrderItem].[ProductID],[OrderI

4. Setup template. Usually the Text property of the template is linked to values from Fields property of the template.

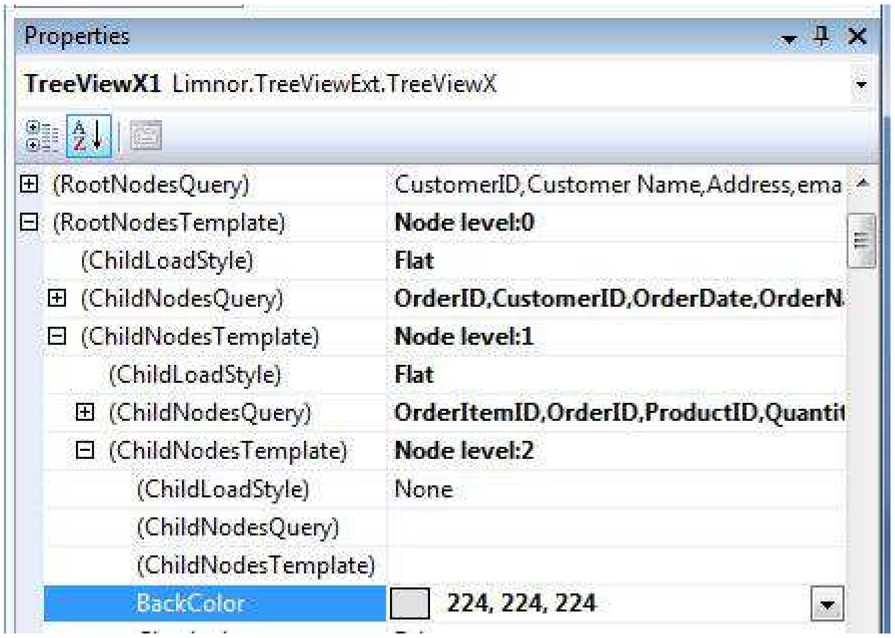


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

ShowPrope	True
StateImage	(none)
StateImage	(none)
Tag	
⊞ Text	I:TreeViewX1.RootNodesTemplate.ChildNodesTemplate.ChildNodesTemplate.Fields.Item
⊞ ToolTipTe	

Other properties of the template can also be used. For example, BackColor property:

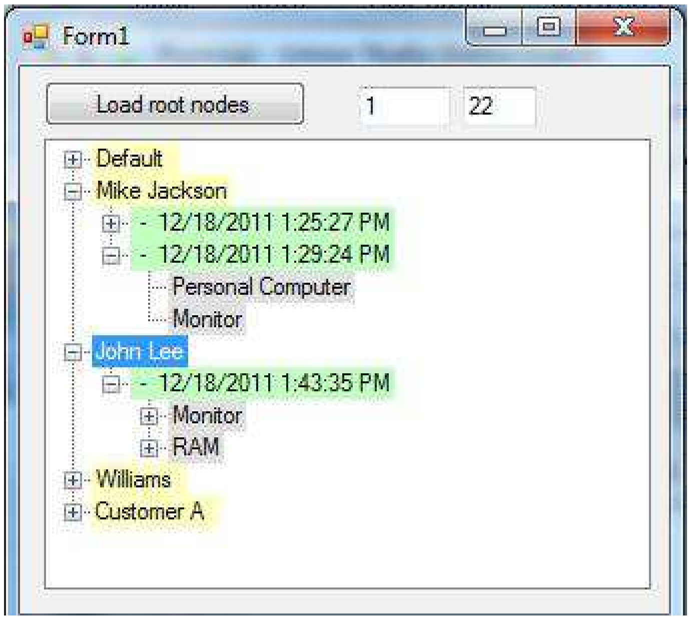


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

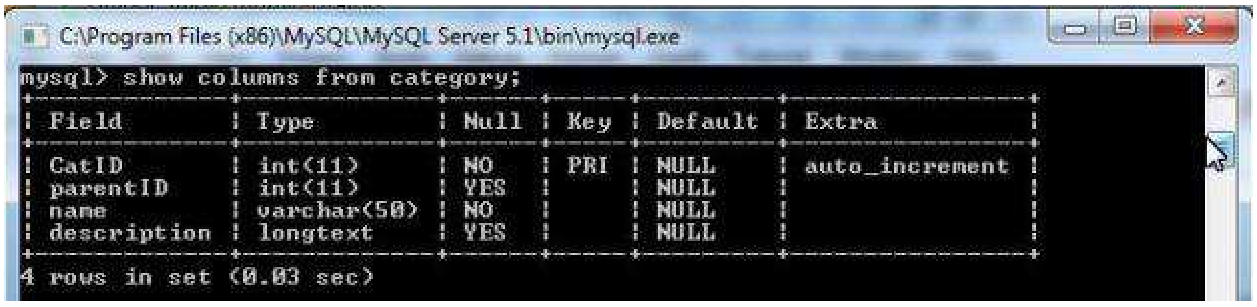
⊞ (RootNodesQuery)	CustomerID,Customer Name,Address,ema
⊟ (RootNodesTemplate)	**Node level:0**
(ChildLoadStyle)	**Flat**
⊞ (ChildNodesQuery)	**OrderID,CustomerID,OrderDate,OrderN**
⊟ (ChildNodesTemplate)	**Node level:1**
(ChildLoadStyle)	**Flat**
⊞ (ChildNodesQuery)	**OrderItemID,OrderID,ProductID,Quantit**
⊟ (ChildNodesTemplate)	**Node level:2**
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplate)	
BackColor	**224, 224, 224**

Sample results:



Load child nodes recursively

A table may be linked to itself to form a tree of unlimited levels. The following “Category” table in a MYSQL database is such an example:


C:\Program Files (x86)\MySQL\MySQL Server 5.1\bin\mysql.exe

```
mysql> show columns from category;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| CatID       | int(11)     | NO   | PRI | NULL    | auto_increment |
| parentID    | int(11)     | YES  |     | NULL    |                |
| name        | varchar(50) | NO   |     | NULL    |                |
| description | longtext    | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
4 rows in set (0.03 sec)
```

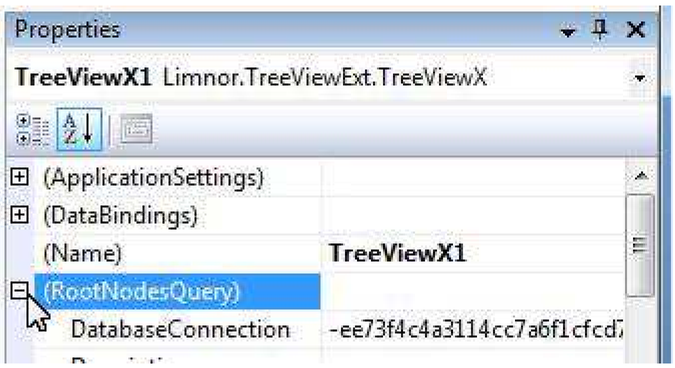
In this example, CatID is the primary key; parentID is a foreign key. If parentID is null then the record is a top level category; if parentID is not null then it must be an existing value of CatID to indicate the parent category.

We may load this Category table into a tree view.

Load root nodes

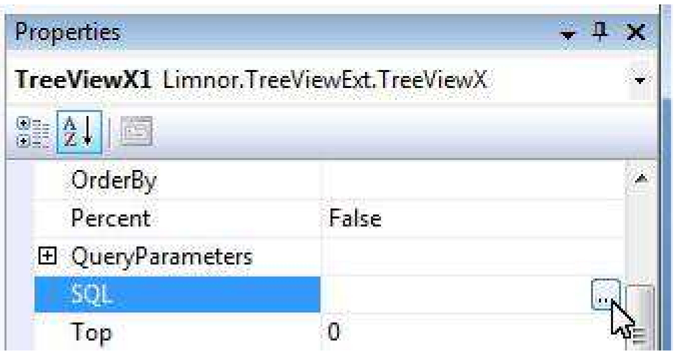
This is the same as we did before.

1. Set RootNodesQuery


Properties

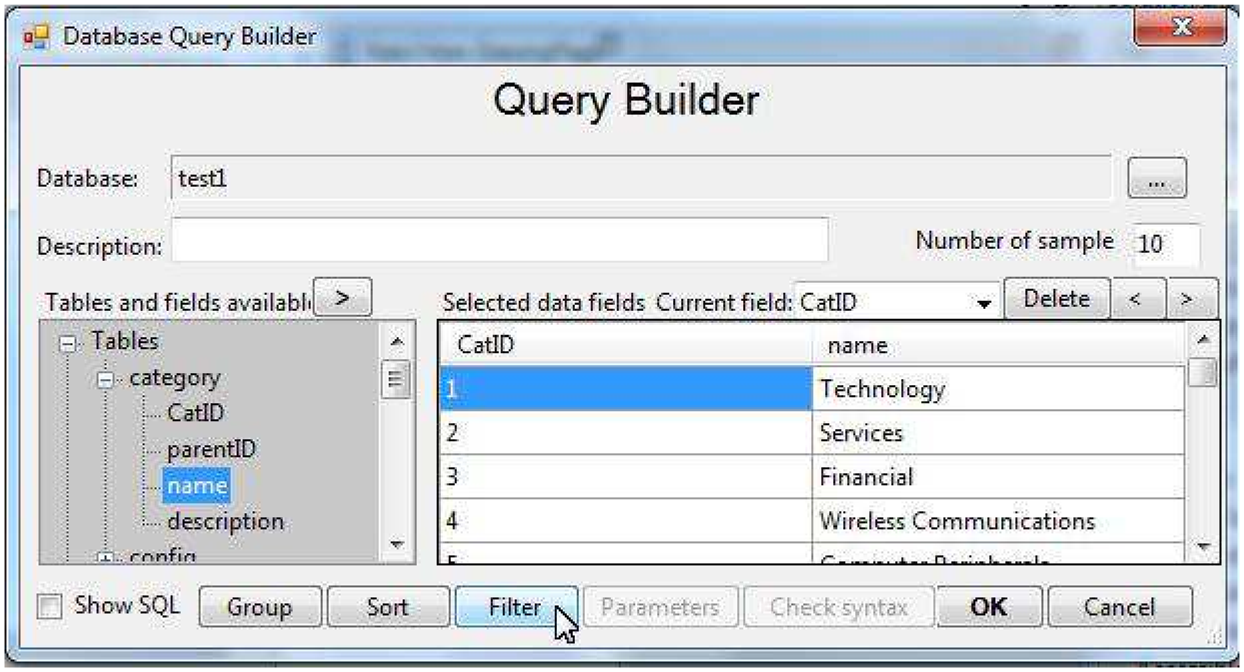
TreeViewX1 Limnor.TreeViewExt.TreeViewX

(ApplicationSettings)	
(DataBindings)	
(Name)	**TreeViewX1**
(RootNodesQuery)	
DatabaseConnection	-ee73f4c4a3114cc7a6f1cfcd7


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

OrderBy	
Percent	False
QueryParameters	
SQL	...
Top	0


Database Query Builder

Query Builder

Database: test1

Description:

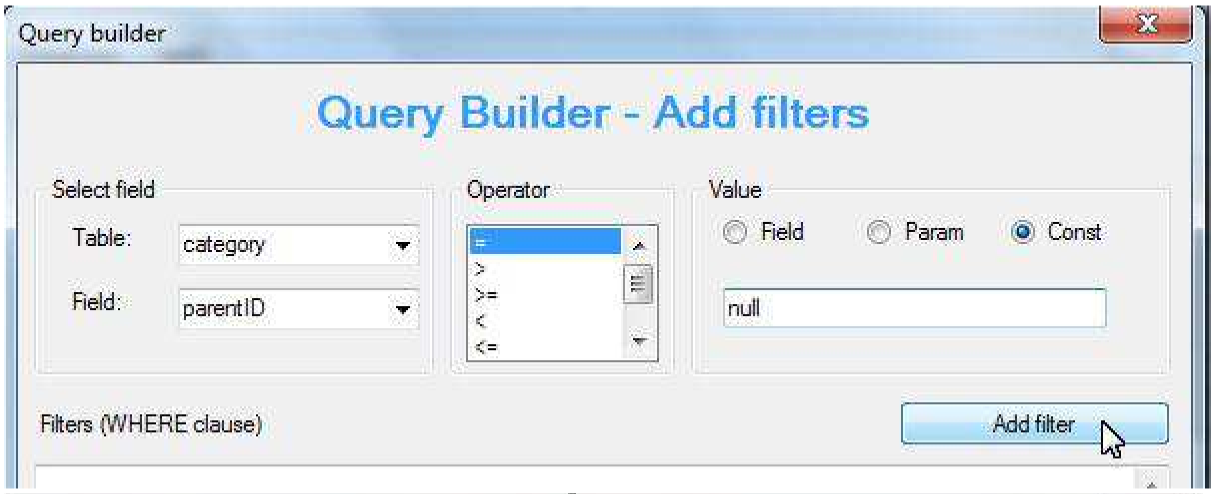
Number of sample 10

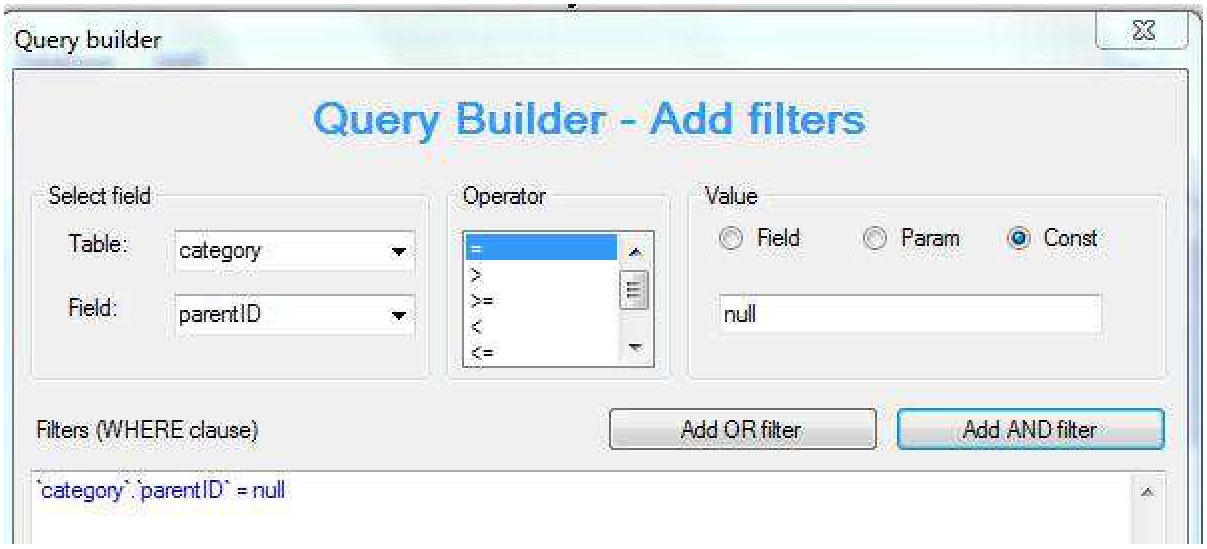
Tables and fields available

Selected data fields Current field: CatID

CatID	name
1	Technology
2	Services
3	Financial
4	Wireless Communications

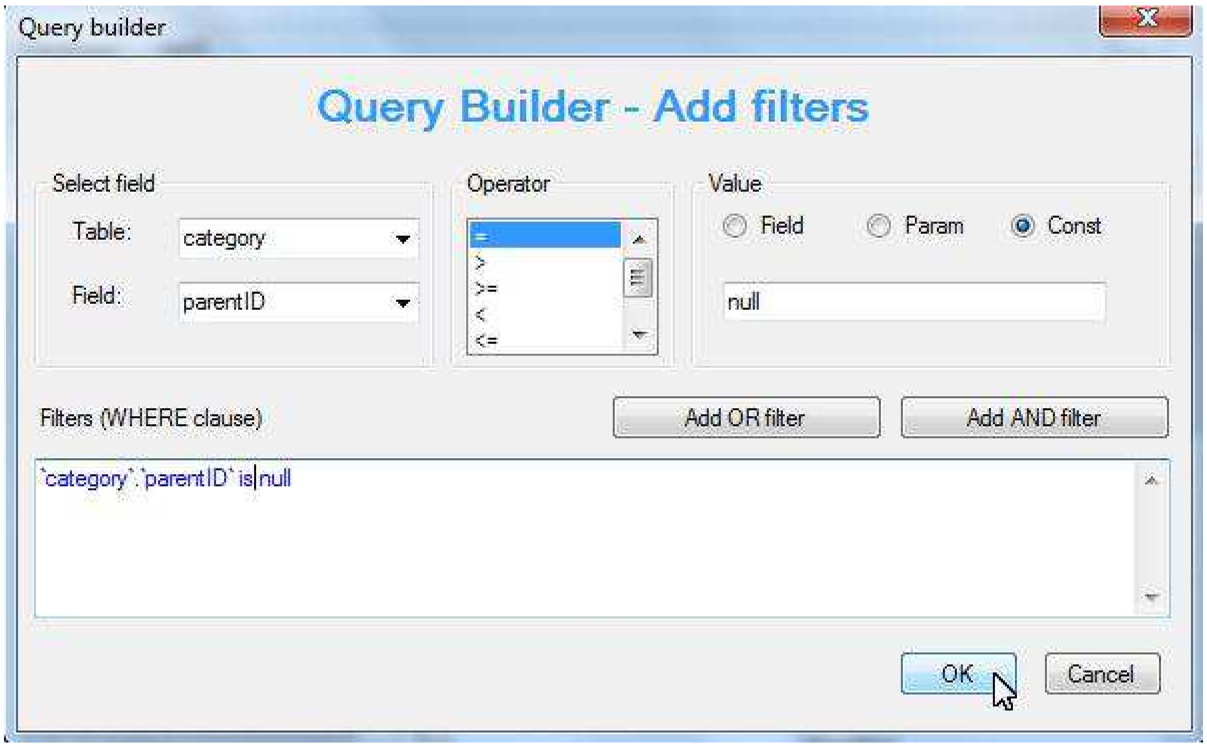
In this sample, we want to set the filter to be “parentID is null”:





```
`category`.`parentID` = null
```

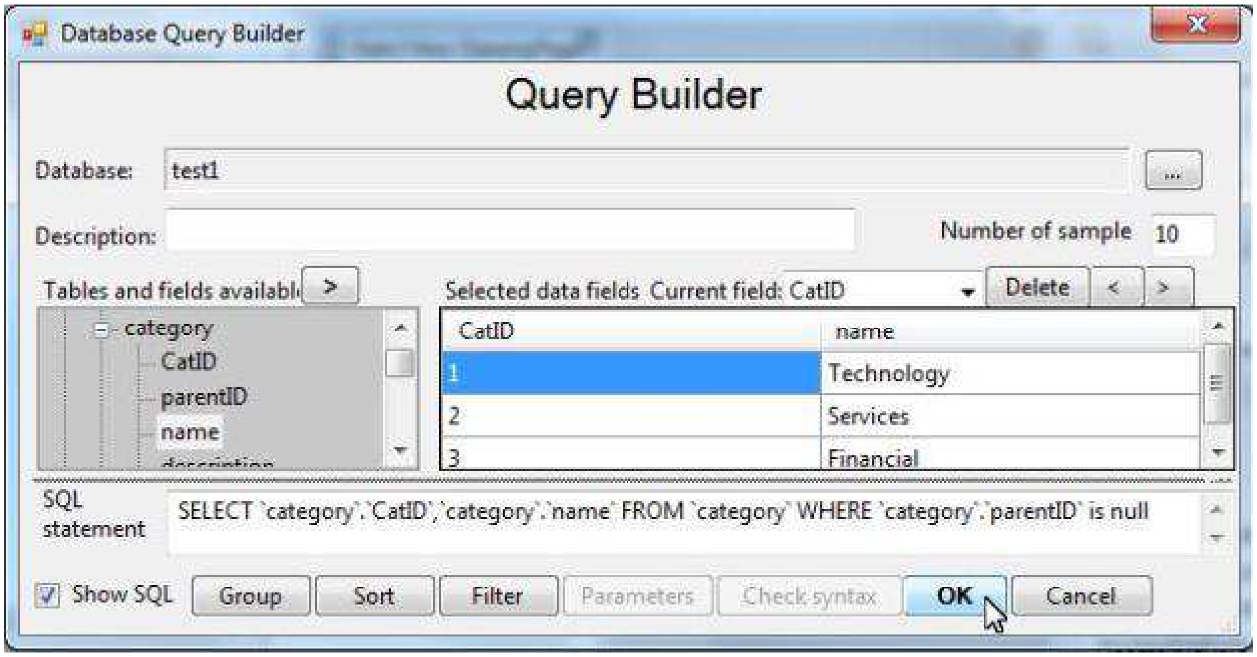
But “null” value is a special case. It cannot use “=”. We need to modify it to be “is”:



Value: Field [ ] Param [ ] [✓ Const]

```
`category`.`parentID` is null
```

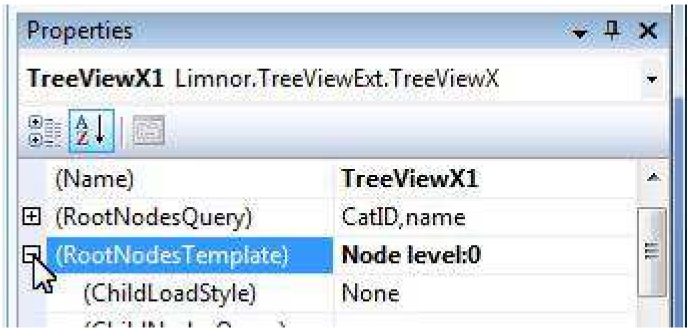
This is our query for loading top level nodes:



CatID	name
1	Technology
2	Services
3	Financial

```
SELECT `category`.`CatID`,`category`.`name` FROM `category` WHERE `category`.`parentID` is null
```

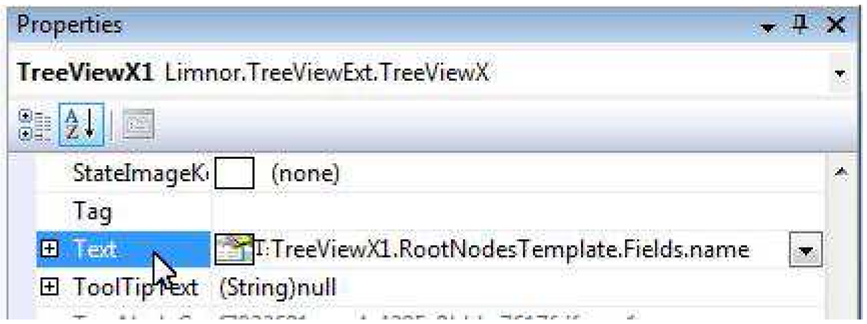
2. Setup node template through RootNodesTemplate


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

(Name)	TreeViewX1
⊞ (RootNodesQuery)	CatID,name
⊟ (RootNodesTemplate)	Node level:0
(ChildLoadStyle)	None

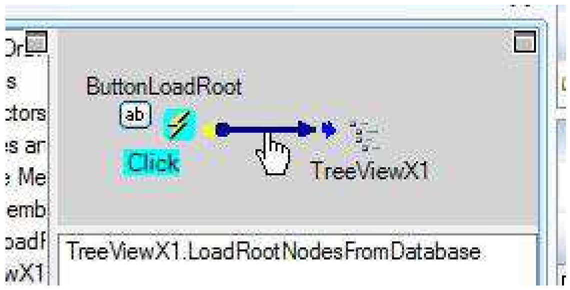
Usually we want to set Text of the template to information from database. In this example, we set it to be the “name” field:


Properties

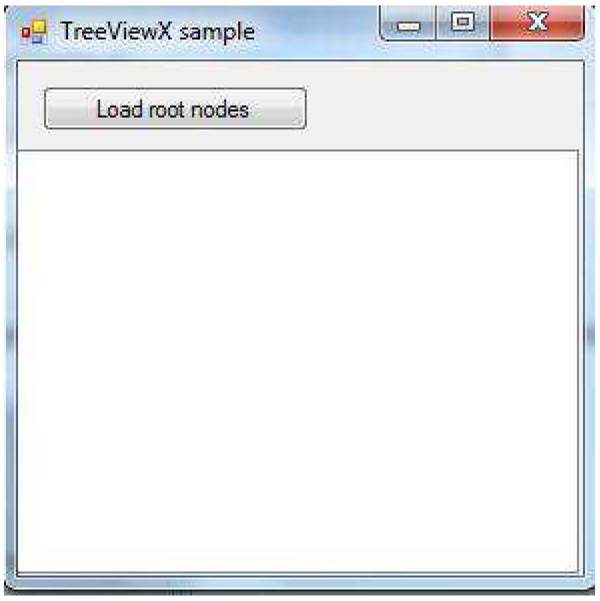
TreeViewX1 Limnor.TreeViewExt.TreeViewX

StateImageK	(none)
Tag	
⊞ Text	I:TreeViewX1.RootNodesTemplate.Fields.name
⊞ ToolTipText	(String)null

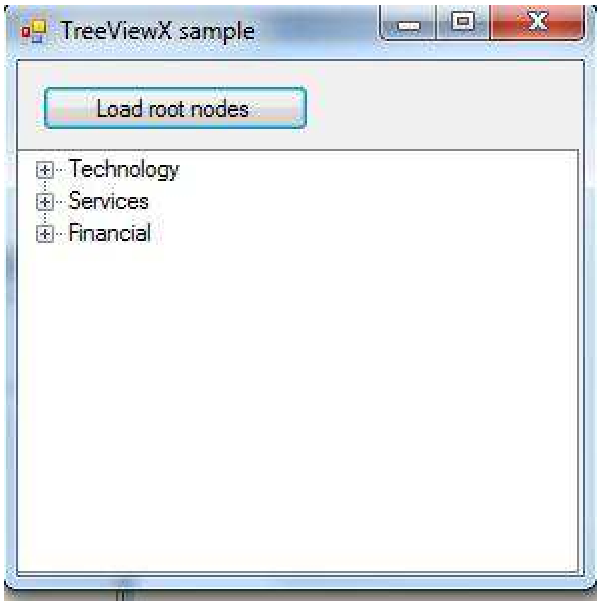
3. A LoadRootNodesFromDatabase action is used to load root nodes. In this example, we use a button to execute this action:



Run the sample. The form appears:



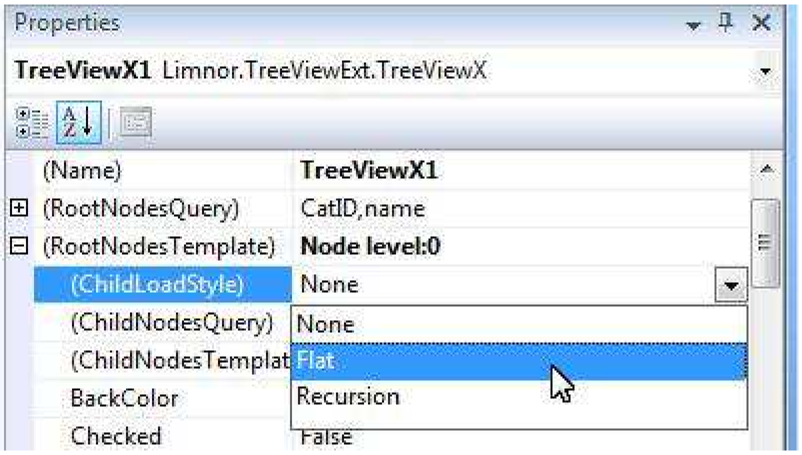
Click the button. The top level categories appear as root nodes:



Load first level child nodes

It is also done just as we did before.

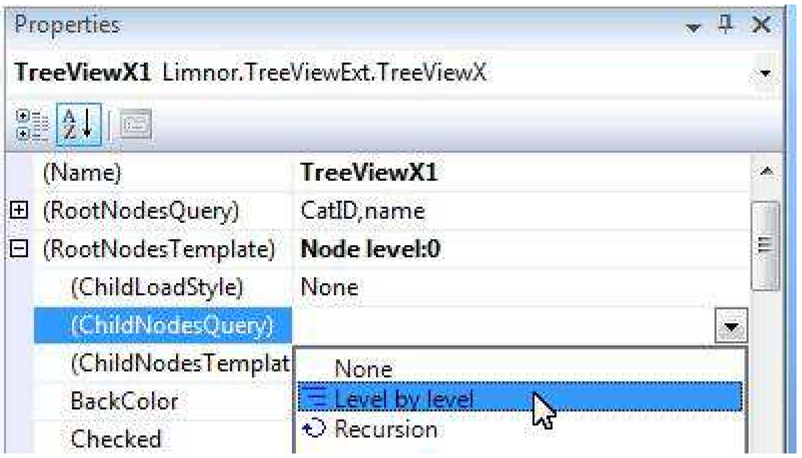
1. Turn on the next level
   It can be done by selecting “Flat” for ChildLoadStyle property:


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

(Name)	TreeViewX1
(RootNodesQuery)	CatID,name
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplat	
BackColor	
Checked	False

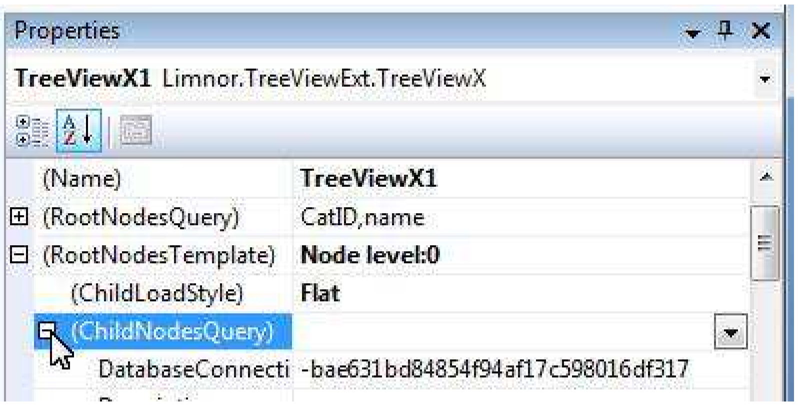
Or choose "Level by level" for ChildNodesQuery or ChildNodesTemplate:


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

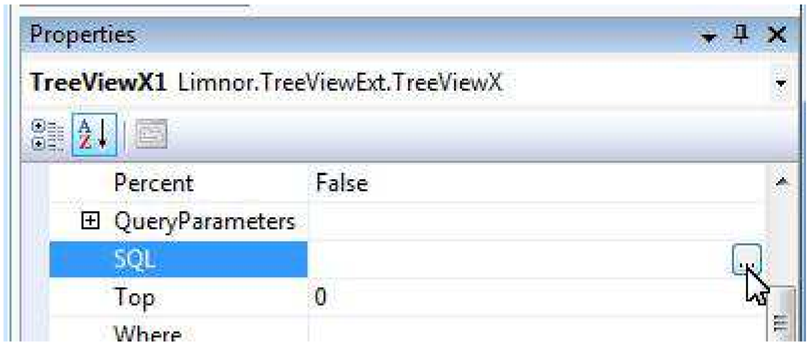
(Name)	TreeViewX1
(RootNodesQuery)	CatID,name
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplat	
BackColor	
Checked	

2. Setup ChildNodesQuery


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

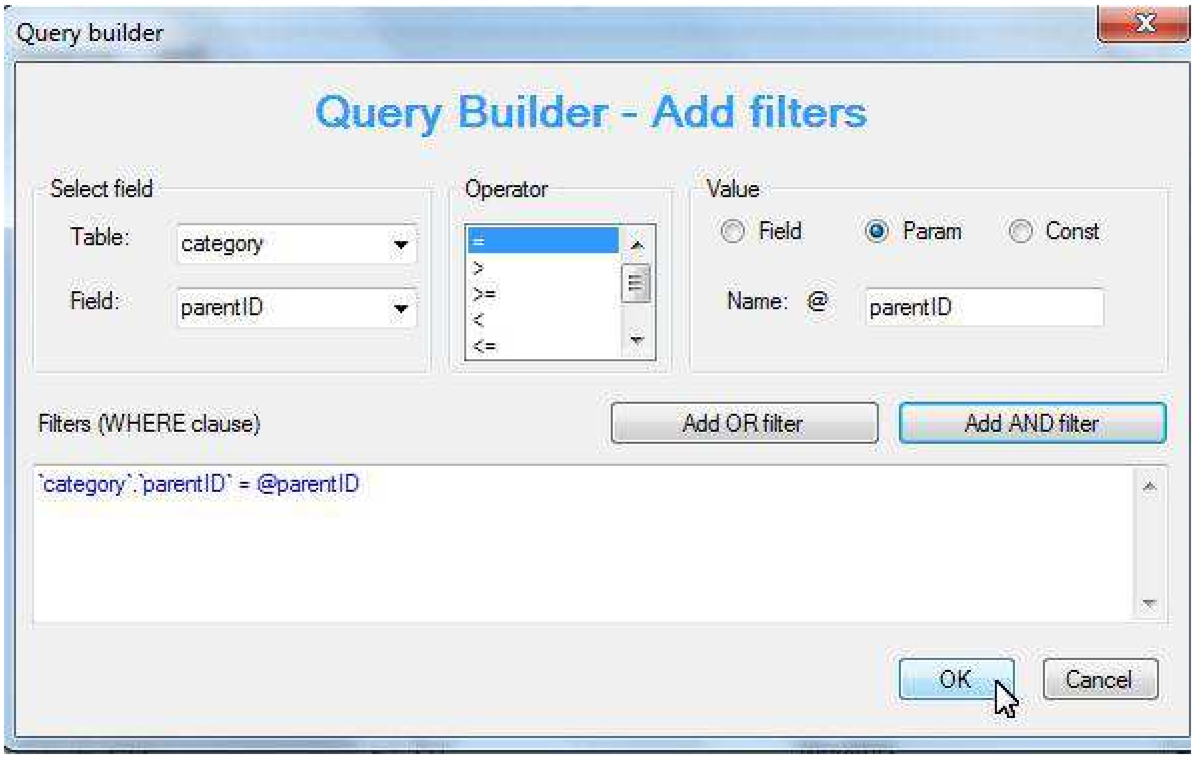
(Name)	TreeViewX1
(RootNodesQuery)	CatID,name
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	Flat
(ChildNodesQuery)	
DatabaseConnecti	-bae631bd84854f94af17c598016df317


Properties

TreeViewX1 Limnor.TreeViewExt.TreeViewX

Percent	False
QueryParameters	
SQL	
Top	0
Where	

We use a parameter in the query for parentID field:


Query builder

Query Builder - Add filters

Select field

Table: category

Field: parentID

Operator: =

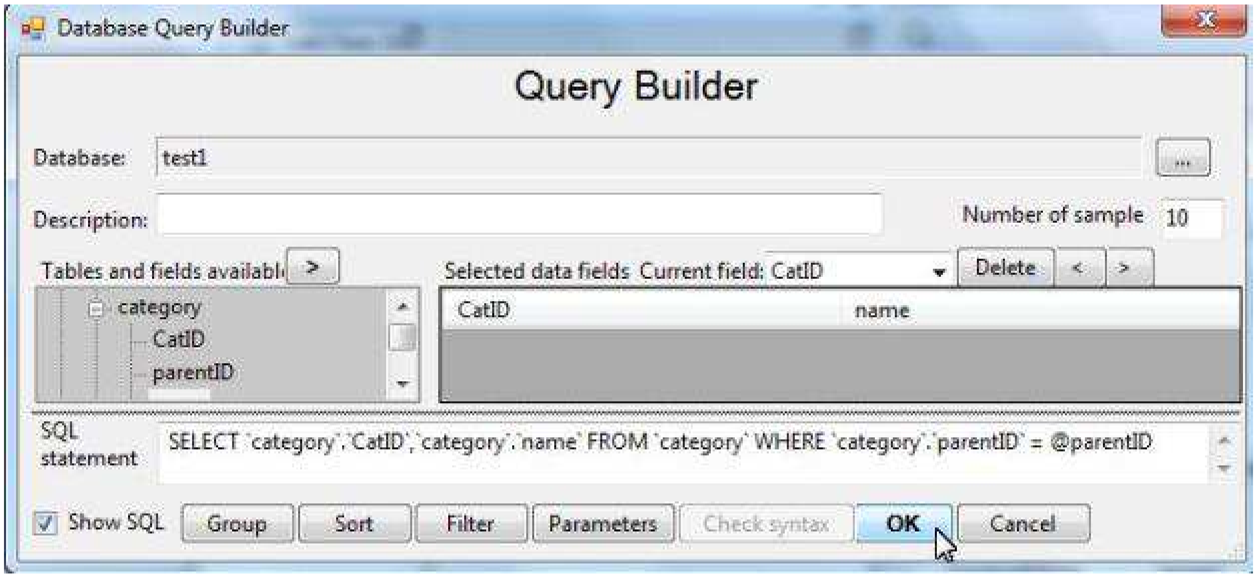
Value: Field [✓ Param] Const

Name: @ parentID

Filters (WHERE clause)

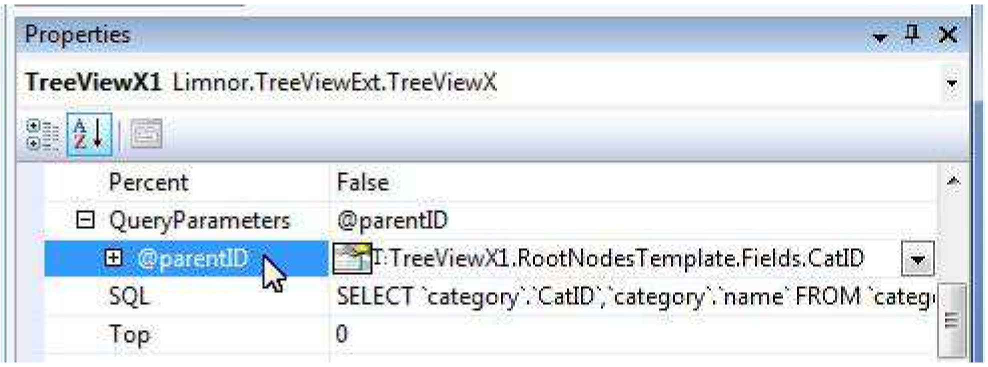
```
`category`.`parentID` = @parentID
```

This is the query we use to load child nodes:



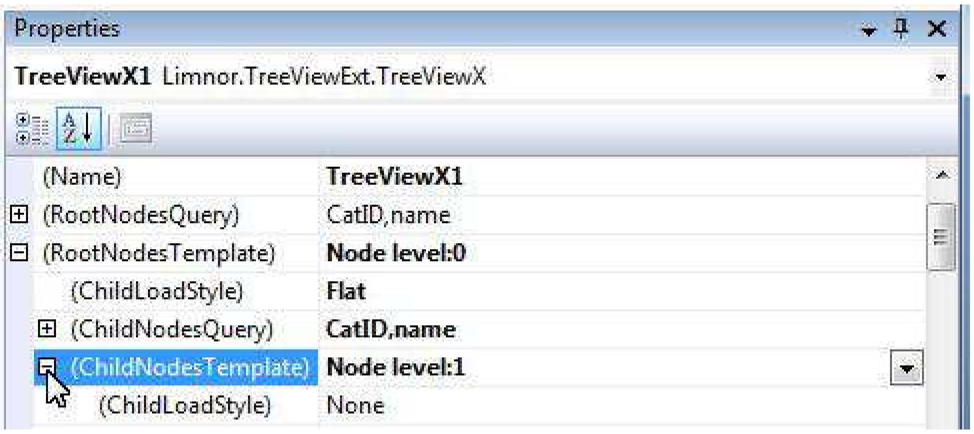
SQL statement: SELECT `category`.`CatID`,`category`.`name` FROM `category` WHERE `category`.`parentID` = @parentID

3. Link query parameters to parent level templates. In this sample, we link the parameter @parentID to the CatID of the root node template:



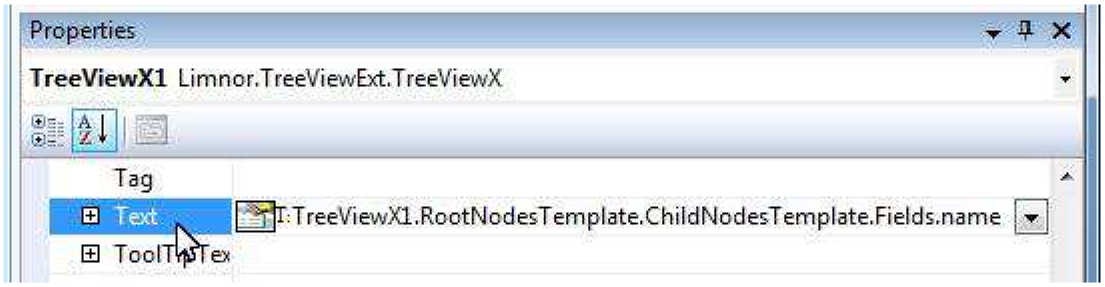
Percent	False
QueryParameters	@parentID
@parentID	T:TreeViewX1.RootNodesTemplate.Fields.CatID
SQL	SELECT `category`.`CatID`,`category`.`name` FROM `categ
Top	0

4. Setup node template through ChildNodesTemplate



(Name)	TreeViewX1
(RootNodesQuery)	CatID,name
(RootNodesTemplate)	Node level:0
(ChildLoadStyle)	Flat
(ChildNodesQuery)	CatID,name
(ChildNodesTemplate)	Node level:1
(ChildLoadStyle)	None

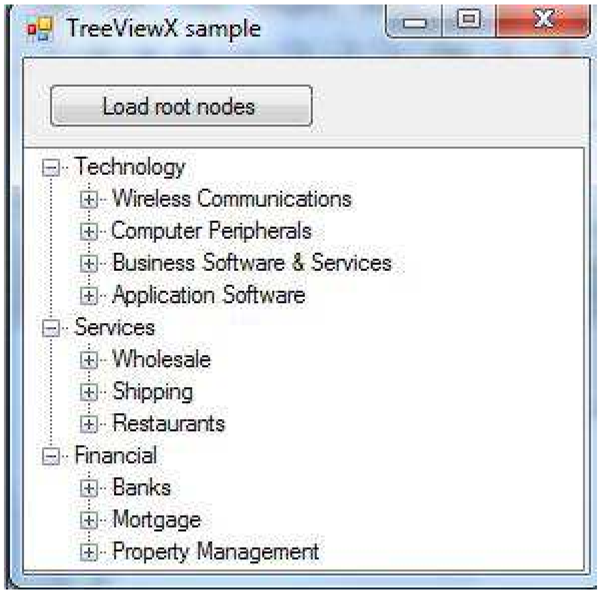
Usually we want to set Text of the template to information from database. In this example, we set it to be the “name” field:



TreeViewX1 Limnor.TreeViewExt.TreeViewX

Tag	
⊞ Text	T:TreeViewX1.RootNodesTemplate.ChildNodesTemplate.Fields.name
⊞ ToolTipTex	

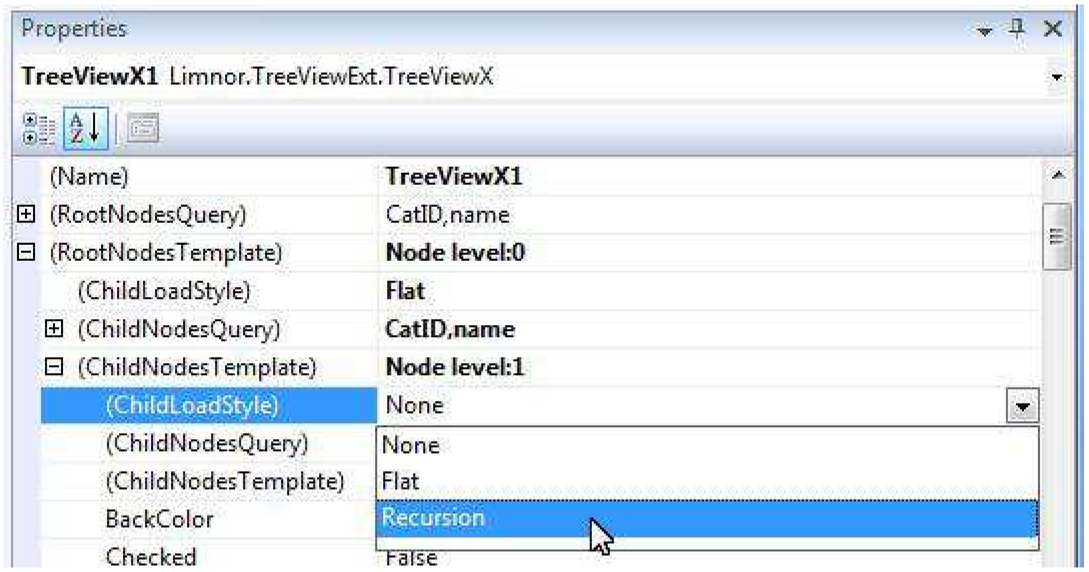
Run this sample, we may expand the root nodes to see the second level categories:



```
⊟ Technology
    ⊞ Wireless Communications
    ⊞ Computer Peripherals
    ⊞ Business Software & Services
    ⊞ Application Software
⊟ Services
    ⊞ Wholesale
    ⊞ Shipping
    ⊞ Restaurants
⊟ Financial
    ⊞ Banks
    ⊞ Mortgage
    ⊞ Property Management
```

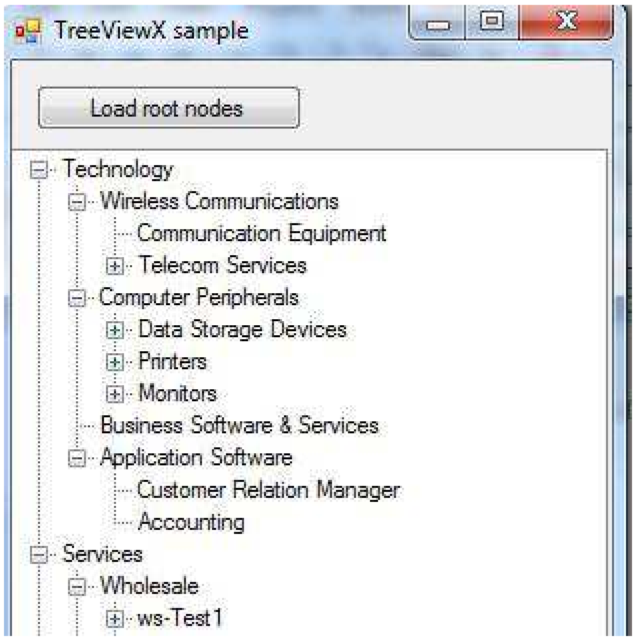
Load child nodes recursively

Because the ChildNodesQuery and the ChildNodesTemplate we setup can be used for loading any levels of child nodes, we do not have to setup each child level loading level by level. We may simply select "Recursion" for ChildLoadStyle:



(Name)	**TreeViewX1**
(RootNodesQuery)	CatID,name
(RootNodesTemplate)	**Node level:0**
(ChildLoadStyle)	**Flat**
(ChildNodesQuery)	**CatID,name**
(ChildNodesTemplate)	**Node level:1**
(ChildLoadStyle)	None
(ChildNodesQuery)	
(ChildNodesTemplate)	
BackColor	
Checked	False

Run the sample, we can keep expanding to as deep as there are data in the database:



Feedback

Please send your feedback to support@limnor.com