Visual.NET Extensions – Tutorial

“vdxExplorerTree - Standard datamanipulation with treeview“

The extensive application development framework for the simple development of Microsoft Visual Studio.NET database applications!

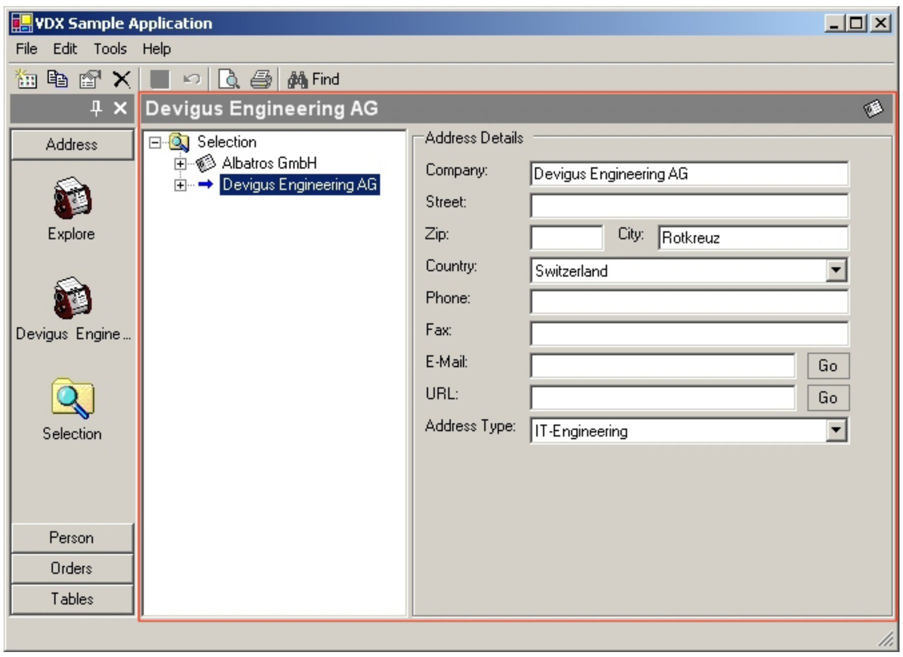
Devigus Engineering AG
Grundstrasse 3
CH-6343 Rotkreuz
Internet: http://www.devigus.com
Email: deag@devigus.com

Version: 1.1.a
Last Update: 22.11.2002

1 *vdxExplorerTree –Standard datamanipulation with treeview.*

This tutorial describes the usage of the *vdxExplorerTree*-Control showing the implementation of a standard datamanipulation with treeview. This szenario describes the treeview based management of specific data, such as an address table.

The following screen shows the *vdxExplorerTree*-Control with its child-controls *SelectionListAddress* und *DataDetailAddress*:



Address Details

Company:	Devigus Engineering AG
Street:	
Zip:	
City:	Rotkreuz
Country:	Switzerland
Phone:	
Fax:	
E-Mail:	
URL:	
Address Type:	IT-Engineering

1.1 *Implementing the ExplorerAddress class*

This section explains the construction and integration of the address container class. The database controls for searching addresses and reading addresses and address details will be embedded in this container. The data access logic of the embedded controls will also be written. A method will only be implemented in a control class where it is not possible or does not make sense to do so otherwise (more on this later).

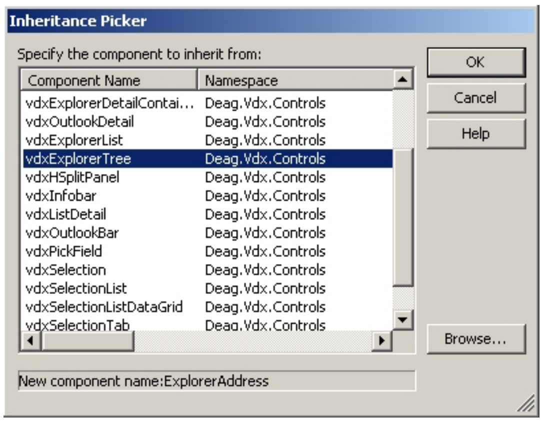
Following steps describe the creation of the *ExplorerAddress* class:

1. Add a new, derived *Control* class to the project (*Project* → *Add Inherited Control...*).

2. Name the class *ExplorerAddress* because all of the embedded controls are used to explore, i.e. view and process addresses.

3. Click *Open* to open the dialogue box for selecting the base class.

4. Click *Browse...* to supply the appropriate assembly (DLL file). Select *vdxControls\bin\release\vdxControls.dll* from the VDX installation subdirectory. Select the *vdxExplorerTree* class from the component list. The following figure shows the selection window.



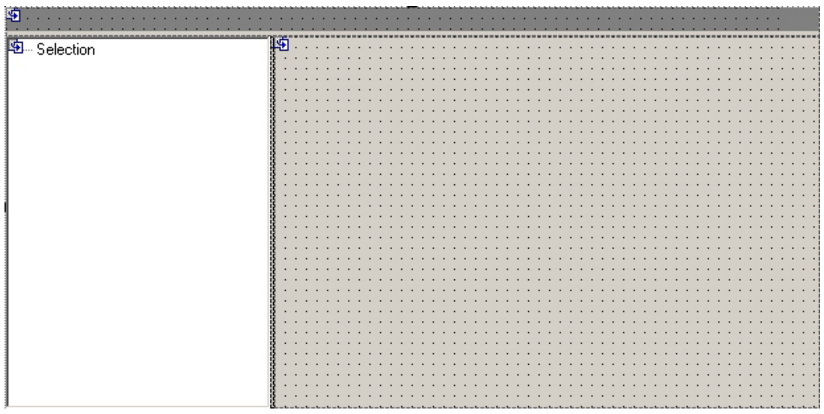
Inheritance Picker

Specify the component to inherit from:

Component Name	Namespace
vdxExplorerDetailContai...	Deag.Vdx.Controls
vdxOutlookDetail	Deag.Vdx.Controls
vdxExplorerList	Deag.Vdx.Controls
vdxExplorerTree	Deag.Vdx.Controls
vdxHSplitPanel	Deag.Vdx.Controls
vdxInfobar	Deag.Vdx.Controls
vdxListDetail	Deag.Vdx.Controls
vdxOutlookBar	Deag.Vdx.Controls
vdxPickField	Deag.Vdx.Controls
vdxSelection	Deag.Vdx.Controls
vdxSelectionList	Deag.Vdx.Controls
vdxSelectionListDataGrid	Deag.Vdx.Controls
vdxSelectionTab	Deag.Vdx.Controls

New component name:ExplorerAddress

The newly created class consists of a *Titlebar*, a *TreeView*, and a container for displaying data, all of which are inherited.



5. Add the following VDX namespace references in the code to allow easier access to the VDX classes.

```
using System.Data;
using Deag.Vdx.Common;
using Deag.Vdx.Common.Enums;
using Deag.Vdx.Controls;
```

The *ExplorerAddress* class serves as a container for individual address controls which are used for address searching and displaying the read addresses and address details. These will be created in the following sections.

1.2 Creation of the SelectionAddress-class

This section explains how to build and integrate the control class which allows a user to enter search for addresses.

1. Add a new derived *User Control* class to the project (*Project* → *Add Inherited Control...*).

2. Rename the class *SelectionAddress*. The user will be able to filter the found addresses or define a selected subset of all addresses.

3. Click *Open* to open the dialog box for selecting the base class.

4. Click *Browse...* to supply the appropriate assembly (DLL file). Select *vdxControls\bin\release\vdxControls.dll* from the VDX installation subdirectory. Select the *vdxSelectionTab* class from the component list.

5. Add the following VDX namespace references in the code to allow easier access to the VDX classes.

```
using Deag.Vdx.Common;
using Deag.Vdx.Common.Enums;
using Deag.Vdx.Controls;
```

6. The base class already contains a tab control and allows one to easily design various search forms. The derived control already contains two *Buttons*: The *Go* button starts the search and the *Clear* button deletes the contents of the text boxes on the control. Select the *VdxTabPages* property and add two *TabPages* to it.

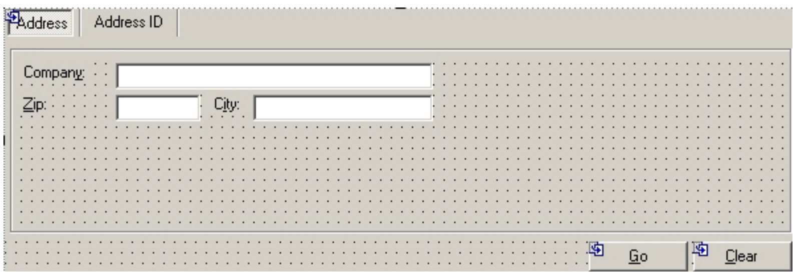
7. Rename the first *TabPage tbpAddress* and the second *tbpAddressId*. Set the *Text* and *Name* properties of the two new *TabPages* as shown in the following table.

Added TabPage	Text	Name
vdxTabPage1	Address	tbpAddress
vdxTabPage2	Address ID	tbpAddressId

8. Click the *Address TabPage* and add three *Label* and three *TextBox* controls from the *Windows Forms* toolbox. Set the *Text* und *Name* properties of the controls as shown in the following table.

Added Control	Text	Name
vdxLabel1	Company	lblCompany
vdxLabel2	ZIP	lblZip
vdxLabel3	City	lblCity
vdxTextBox1		txtCompany
vdxTextBox2		txtZip
vdxTextBox3		txtCity

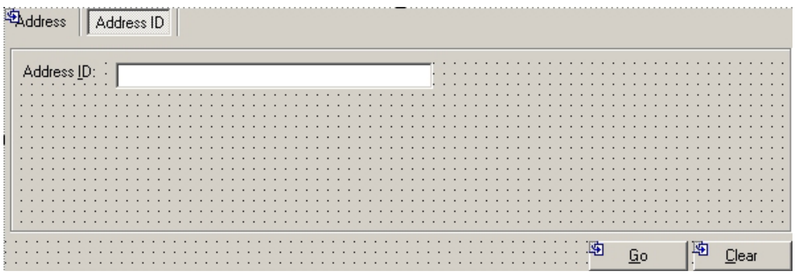
9. Arrange the new controls. The following figure offers a suggestion.



10. Click the *Address ID TabPage* and add a *Label* and a *TextBox* control. Set the *Text* and *Name* properties of the controls as shown in the following table.

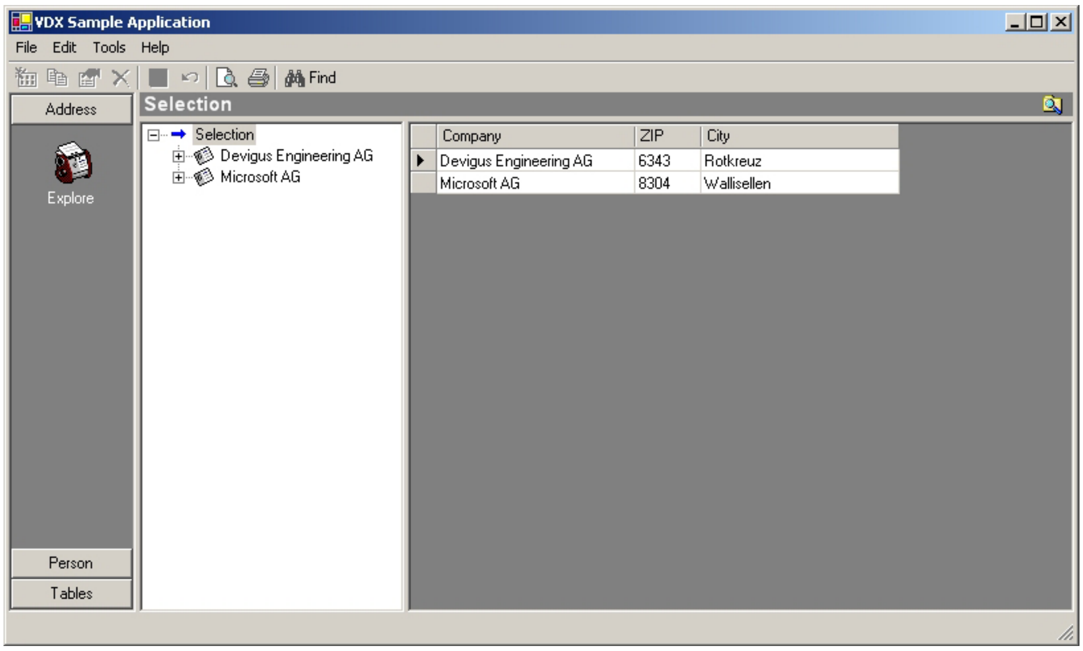
Control to be added	Text	Name
vdxLabel1	Address ID	lblAddressId
vdxTextBox1		txtAddressId

11. Arrange the controls as shown in the following figure.



1.3 *Creation of the SelectionListAddress-class*

This section describes how to build the control class which displays the found addresses to the user in a list. This list contains the same addresses as the tree view on the left side.



Company	ZIP	City
Devigus Engineering AG	6343	Rotkreuz
Microsoft AG	8304	Wallisellen

1. Add a new derived *UserControl* class to the project (*Project → Add Inherited Control...*).

2. Rename the class *SelectionListAddress*.

3. Click *Open* to open the dialogue box for selecting the base class.

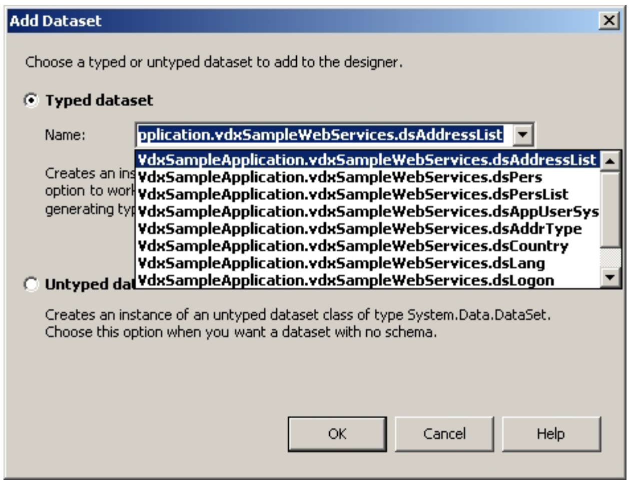
4. Click *Browse...* to supply the appropriate assembly. Select *vdxControls\bin\release\vdxControls.dll* from the VDX installation subdirectory. Select the *vdxSelectionListDataGrid* class from the component list.

1.3.1 DataSet *dsAddressList* and *vdxDataGrid* binding

A *DataSet* of type *SelectionListAddress* must be added in order to display the selected addresses in a list. The *DataSet* is part of the server application and will be integrated via a reference.

1. Drag a *DataSet* control from the *Data* tab of the toolbox onto the control.

2. Select the *MyVdxSampleApplication.vdxSampleWebServices.dsAddressList DataSet* from the drop-down list in the displayed dialogue box as shown in the following figure.



Add Dataset

Choose a typed or untyped dataset to add to the designer.

(●) Typed dataset

Name: pplication.vdxSampleWebServices.dsAddressList

VdxSampleApplication.vdxSampleWebServices.dsAddressList
VdxSampleApplication.vdxSampleWebServices.dsPers
VdxSampleApplication.vdxSampleWebServices.dsPersList
VdxSampleApplication.vdxSampleWebServices.dsAppUserSys
VdxSampleApplication.vdxSampleWebServices.dsAddrType
VdxSampleApplication.vdxSampleWebServices.dsCountry
VdxSampleApplication.vdxSampleWebServices.dsLang
VdxSampleApplication.vdxSampleWebServices.dsLogon

Creates an ins option to worl generating typ

( ) Untyped dat

Creates an instance of an untyped dataset class of type System.Data.DataSet. Choose this option when you want a dataset with no schema.

3. Rename the *DataSet dsAddressList.*

4. The *dsAddressList DataSet* must be tied to the *vdxDataGrid* control. Select the *vdxDataGrid* properties and set them as shown in the following table.

Property	Selection
DataSource	dsAddressList
DataMember	Address

NOTE: First, select the *DataSource*-Property and then select the DataSet with the table. This way, the *DataMember*-property gets automatically set.

Compile the project to check the previous code for errors.

NOTE: If the *"Property or indexer 'System.Windows.Forms.DataGrid. VisibleColumnCount' cannot be assigned to -- it is read-only"* error is displayed by the compiler, then remove the following line of code.

```
this.vdxDataGrid.VisibleColumnCount = 4;
```

Whenever a derived *DataGrid* is changed, Visual Studio .NET changes the read-only properties and these must then be manually removed. Because the Visual Studio .NET Designer also creates duplicate statements, remove the duplicates of the following statements.

```
((System.ComponentModel.ISupportInitialize)(this.vdxDataGrid)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.vdxDataGrid)).EndInit();
```

1.4 Creation of the DataDetailAddress-class

This control displays the details of a specified address. The display can be activated by either selecting an address in the tree view or by double-clicking on an address in the address list, i.e. in *SelectionListAddress*.

Create and integrate the control class:

1. Add a new, derived *UserControl* class to the project (via *Project→ Add Inherited Control...*).
2. Rename the class *DataDetailAddress*.
3. Click *Open* to open the dialogue box for selecting the base class.
4. Click *Browse...* to supply the appropriate assembly. Select *vdxControls\bin\release\vdxControls.dll* from the VDX installation subdirectory. Select the *vdxDataDetail* class from the component list.
5. Add the following VDX namespace references to the code.

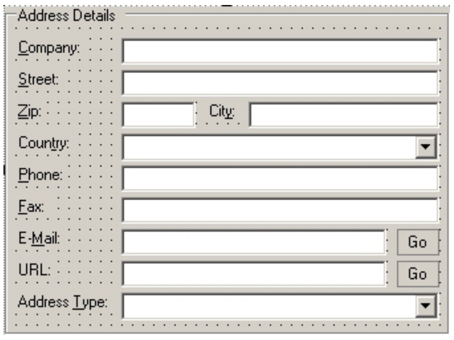
```
using System.Diagnostics;
using Deag.Vdx.Common;
using Deag.Vdx.Common.Enums;
using Deag.Vdx.Controls;
using Deag.Vdx.Controls.Delegates;
using Deag.Vdx.Controls.EventArguments;
```

6. Select the design view of *DataDetailAddress* and drag the following controls onto it. The VDX variants are on the *VDX* tab of the toolbox.

Control to be added	Text	Name
vdxLabel	Company:	lblCompany
vdxLabel	Street:	lblStreet
vdxLabel	ZIP:	lblZip
vdxLabel	City:	lblCity
vdxLabel	Country:	lblCountry
vdxLabel	Phone:	lblPhone
vdxLabel	Fax:	lblFax
vdxLabel	E-Mail:	lblEmail
vdxLabel	URL:	lblUrl
vdxLabel	Address Type:	lblAddrType
vdxButton	Go	btnGoEmail
vdxButton	Go	btnGoUrl
vdxTextBox		txtCompany
vdxTextBox		txtStreet
vdxTextBox		txtZip
vdxTextBox		txtCity
vdxTextBox		txtPhone
vdxTextBox		txtFax
vdxTextBox		txtEmail
vdxTextBox		txtUrl

vdxComboBox		cboCountry
vdxComboBox		cboAddrType

7. Arrange the controls as show in the figure below.

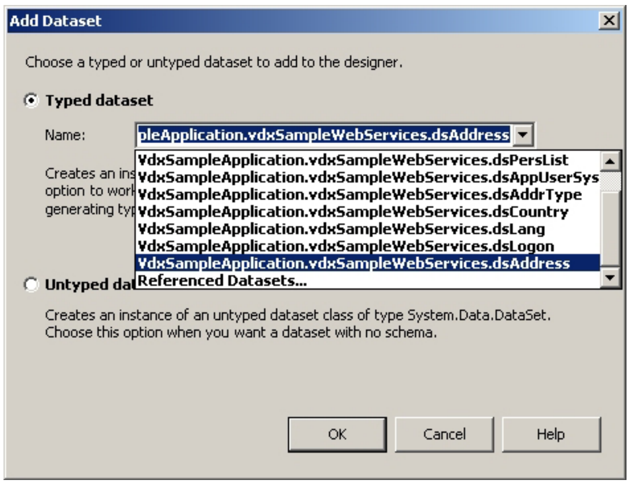


1.4.1 DataSet *dsAddress, dsAddrType* and *dsCountry* binding

Three *DataSets* derived from the *DataDetailAddress* class must be added in order to display the various address details. These *DataSets* are part of the server application and will therefore be accessed via references.

1. Drag a *DataSet* onto the control from the *Data* tab of the toolbox.

2. Select the *MyVdxSampleApplication.vdxSampleWebServices.dsAddress DataSet* from the drop-down list in the displayed dialogue box as shown in the following figure.


Add Dataset
Choose a typed or untyped dataset to add to the designer.
Typed dataset
Name:
pleApplication.vdxSampleWebServices.dsAddress
Creates an ins
option to worl
generating typ
VdxSampleApplication.vdxSampleWebServices.dsPersList
VdxSampleApplication.vdxSampleWebServices.dsAppUserSys
VdxSampleApplication.vdxSampleWebServices.dsAddrType
VdxSampleApplication.vdxSampleWebServices.dsCountry
VdxSampleApplication.vdxSampleWebServices.dsLang
VdxSampleApplication.vdxSampleWebServices.dsLogon
VdxSampleApplication.vdxSampleWebServices.dsAddress
Referenced Datasets...
Untyped dat
Creates an instance of an untyped dataset class of type System.Data.DataSet.
Choose this option when you want a dataset with no schema.
OK
Cancel
Help

3. Rename the *DataSet dsAddress*.

4. To show alternatives to the current value in the *ComboBoxes*, add additional *DataSets*. Drag another *DataSet* onto the control from the *Data* tab of the toolbox.

5. Select the *MyVdxSampleApplication.vdxSampleWebServices.dsCountry DataSet* from the drop-down list in the displayed dialogue box.

6. Repeat steps 4 and 5 for a *dsAddrType* DataSet.and rename it to *dsAddrType.*

7. The *dsAddress DataSet* must be bound to the appropriate *TextBox* controls as shown in the following table.

Control	Property DataBindings.Text
txtCompany	dsAddress - Address.Company
txtStreet	dsAddress - Address.Street
txtZip	dsAddress - Address.ZIP
txtCity	dsAddress - Address.City
txtPhone	dsAddress - Address.Phone
txtFax	dsAddress - Address.Fax
txtEmail	dsAddress - Address.Email
txtUrl	dsAddress - Address.URL

8. The *ComboBox* bindings must also be defined as shown in the following table.

Property	Control cboCountry	Control cboAddrType
SelectedValue[1)]	dsAddress - Address.Country	dsAddress - Address.AddrTypeID
DataSource	dsCountry	dsAddrType
Display Member	Country.Descr	AddrType.Descr
Value Member	Country.Country	AddrType.AddrTypeId

1) Property-Window: (DataBindings) SelectedValue

NOTE: If the selection for a specific property does not offer the desired options, recompile your solution. To force the designer to write your changes in the code, change the size of your control (or something else) and recompile.

1.4.2 Configuration of the *vdxDataStatusManager*

Because the *DataDetailAddress* control will be used to change and store data, the referenced *vdxDataStatusManager* must be configured accordingly.

1. Select the design view of the *DataDetailAddress*-class and make sure, that you select the *DataDetailAddress* object in the P*roperties*-Drop-Down-Menu.

2. Set the *vdxDataStatusManager* object's properties in the given order to the values shown in the table below:

Property	Selection
VdxBindingContainer	DataDetailAddress
VdxDataSet	dsAddress
VdxBusinessObjectClassName	[MyVdxSample…]BoAddressSoap
VdxMainDataTable	dsAddress.Address
VdxUpdateDataSetMethodName	UpdateDataSet

At this point, the design and naming tasks have been completed for the address controls. Compile the application to check the previous code for errors.

1.5 Creation of the SelectionAddress-control

1.5.1 Overwriting the *VdxFillSearchParameters*-method

In order to send the user-entered search criteria to a *Web service*, they must be "packed" into the appropriate method. The following steps implement this functionality:

1. Select the *Class View* of the project (via *View* → *Class View*).

2. Navigate to the *vdxFillSearchParameters* method in the *vdxSelection* class (*MyVdxSampleApplication* → *SelectionAddress* → *Bases and Interfaces* → *vdxSelectionTab* → *Bases and Interfaces* → *vdxSelection*).

3. Execute the *Add* → *Override* command from the context menu of the *vdxFillSearchParameters* method. This adds an empty method to the *SelectionAddress* class, overwriting the base class'.

4. Remove the *return null* statement and add the following code.

```
vdxSearchParameters sp;

if (this.VdxSelectedTabPage.Text == "Address")
{
   sp = new vdxSearchParameters("SelectionAddressByDescr");

   sp.Add(new vdxSearchParameter("Company",this.txtCompany.Text,
      vdxSearchType.StartWith));

   sp.Add(new vdxSearchParameter("Zip",this.txtZip.Text,
      vdxSearchType. StartWith));

   sp.Add(new vdxSearchParameter("City",this.txtCity.Text,
      vdxSearchType.Equal));

   sp.Add(new vdxSearchParameter("OrderBy","Company",
      vdxSearchType.Equal));
}
else
{
   sp = new vdxSearchParameters("SelectionAddressId");

   int intAddrId;
   try
   {
      intAddrId = System.Convert.ToInt32(this.txtAddressId.Text);
   }
   catch
   {
      intAddrId = 0;
   }

   sp.Add(new vdxSearchParameter("AddressId",intAddrId,
      vdxSearchType.Equal));
}

return sp;
```

This method is called when the user clicks the *Go* button on the search control. The parameters are passed by a *vdxSearchParameters* object named *SelectionAddress* or *SelectionAddressId*, depending on which search form the user used. A *vdxSearchParameters* object contains one or more *vdxSearchParameter* objects, each containing a specific search criterion. The *Add* method adds such an object to the *vdxSearchParameters* object.

The *vdxSearchParameter*'s constructor requires a name for the search criterion, the search criterion, and the search method. The following search methods are supported:

- *vdxSearchType.Equal* – Indicates that the database value must exactly match the search value (SQL: field = value).

- *vdxSearchType.StartWith* – Indicates that the database value must begin with the search value (SQL: field LIKE value%).

- *vdxSearchType.Contains* – Indicates that the database value must contain the search value (SQL: field LIKE %value%).

- *vdxSearchType.BetweenStart* – Must be combined with the *vdxSearchType.BetweenEnd* method (SQL: field BETWEEN startValue AND endValue). It indicates that database value must be greater than or equal to this search value.

- *vdxSearchType.BetweenEnd* – Must be combined with the *vdxSearchType.BetweenEnd* method (SQL: field BETWEEN startValue AND endValue). It indicates that the database value must be less than or equal to the search value.

- *vdxSearchType.SamllerAs* – Indicates that the database value must be less than the search value (SQL: field < value).

- *vdxSearchType.LargerAs* – Indicates that the database value must be greater than the search value (SQL: field > value).

1.6 Creation of the DataDetailAddress-control

In order to display the address entered in the URL textbox, the click event of the corresponding *Go* button must be programmed.

1. Open the *DataDetailAddress* control.

2. Double-click the *Click* event of the *btnGoUrl* button.

3. Add the following code to the *btnGoUrl_Click* method of the event listener method.

```
if (txtUrl.Text != "")
{
   Process process = new Process();
   try
   {
      process.StartInfo.FileName = txtUrl.Text;
      process.Start();
   }
   catch (System.Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

4. Implement the clicking of the *Go* button of the email *TextBox* (*btnGoEmail_Click*) in the same manner.

```
if (txtEmail.Text != "")
{
   Process process = new Process();
   try
   {
      process.StartInfo.FileName = "mailto:" + txtEmail.Text;
      process.Start();
   }
   catch (System.Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

5. The following code must be inserted in the constructor of the *DataDetailAddress* class in order to fill the *Country* and *Address Type* ComboBoxes with the appropriate country and address type codes. Insert the code after the call to *InitializeComponent*.

```
vdxSampleWebServices.BoCountrySoap boCountry = new
   vdxSampleWebServices.BoCountrySoap();

this.dsCountry.Merge(boCountry.GetDataSetCountry());

vdxSampleWebServices.BoAddrTypeSoap boAddrType = new
   vdxSampleWebServices.BoAddrTypeSoap();

this.dsAddrType.Merge(boAddrType.GetDataSetAddrType());
```

Compile the project to check the previous code for errors.

NOTE: If you do not get a satisfying result, please close all project files and reopen them.

1.7 Configuration of the ExplorerAddress-control

The linking of the individual address controls (*DataDetailAddress*, *SelectionAddress* and *SelectionListAddress*) is implemented centrally in the *ExplorerAddress* class.

1. Open the design view of the *ExplorerAddress* class and set the focus on the *VdxSelectionControlClassName* property.

2. Select the *SelectionAddress* control from the appropriate drop-down list.

3. Select the *SelectionListAddress* control for the *VdxSelectionListControlClassName* in the same manner.

4. Select the *DataDetailAddress* control for the *VdxMainDataDetailControlClassName* property in the same manner. The following table summarizes the settings.

Property	Selektion
VdxSelectionControlClassName	MyVdxSampleApplication.SelectionAddress
VdxSelectionListControlClassName	MyVdxSampleApplication.SelectionListAddress
VdxMainDataDetailControlClassName	MyVdxSampleApplication.DataDetailAddress

The address controls are now connected to their container, the *ExplorerAddress* class.

1.8 Programming the ExplorerAddress-control

The logic of the embedded controls will be implemented centrally in der class *ExplorerAddress*. Following steps describe the implementation of the corresponding functionality:

1.8.1 Overwriting the der *VdxLoadData*-method

When the user presses the *Go*-Button in the selection dialog, the corresponding data loading process must be invoked. The data loading will be implemented in the method *VdxLoadData*. Follow these steps, to implement the data loading:

1. Select *Class View* within the project (Menu *View* → *Class View*).

2. Navigate to the *VdxLoadData*-method; you find this method in the class *vdxOutlookDetail* (*MyVdxSampleApplication* → *ExplorerAddress* → *Bases and Interfaces* → *vdxExplorerTree* → *Bases and Interfaces* → *vdxOutlookDetail*).

3. Select *Add* → *Override* from the context menu of the *VdxLoadData*-method. This adds an empty method in the *ExplorerAddress*-class, overwriting the method from the base class.

4. Add following code:

```
DataSet dsRet = null, ds = null;
vdxSampleWebServices.BoAddressSoap boAddress;

switch(dataLoadContext)
{
   // Lade die Adress-Treedaten.
   case vdxDataLoadContext.Selection:
      boAddress = new vdxSampleWebServices.BoAddressSoap();
      ds = boAddress.GetDataSetAddressList(vdxSearchParameters.
         GetParameters());
      if (ds.Tables["Address"].Rows.Count > 0)
      {
         dsRet = ds;
      }
      else
      {
         MessageBox.Show("No Data found. Please try again!",
            "VDX Sample Application",MessageBoxButtons.OK,
            MessageBoxIcon.Information);
      }
      break;

   // Lade die Detaildaten der entsprechenden Adresse.
   case vdxDataLoadContext.Detail:
      switch (vdxSearchParameters.VdxSearchName)
      {
         case "Address":
            boAddress = new vdxSampleWebServices.BoAddressSoap();
            ds = boAddress.GetDataSetAddress(vdxSearchParameters.
               GetParameters());
            if (ds.Tables["Address"].Rows.Count > 0)
            {
               dsRet = ds;
            }
            break;
      }
      break;
}

return dsRet;
```

This method is always called when data is accessed. The parameters determine in which context the data is being accessed and with which search criteria the access is being carried out with.

Data is accessed in the following situations:

- *Selection* – The *Go* button in the search form (e.g. *SelectionAddress*) is clicked and the corresponding data is loaded.
- *SubSelection* – A tree node is clicked and the corresponding detail data are loaded. This situation has not been implemented yet, because there are no sub-nodes in the tree yet.
- *Detail* – A tree node is expanded and the corresponding sub-nodes are loaded. Currently only the address detail data is loaded.

NOTE: The detail block is implemented prematurely and enclosed with a *switch* statement. This way, as soon as further tree node levels are added, the details of those levels will be displayed.

1.8.2 Overwriting the *VdxFillData*-method

The display of previously loaded data is implemented in the *vdxFillData* method as follows:

1. Select *Class View* of the project.
2. Navigate to the *vdxFillData* method of the *vdxExplorerTree* class (*VdxSampleApplication* → *ExplorerAddress* → *Bases and Interfaces* → *vdxExplorerTree*).
3. Execute the *Add* → *Override* command from the context menu of the *vdxFillData* method. This adds an empty method to the *ExplorerAddress* class, overwriting the base class'.
4. Add the following code to the method.

```
if (dataLoadContext == vdxDataLoadContext.Selection | dataLoadContext ==
vdxDataLoadContext.SubSelection)
{
   nodeToFill.Nodes.Clear();
   nodeToFill.IsDummy = false;

   foreach(DataTable dataTable in dataSet.Tables)
   {
      switch (dataTable.TableName)
      {
         case "Address":
            foreach (DataRow rw in dataTable.Rows)
            {
               string company = "";
               if (!Convert.IsDBNull(rw["Company"]))
               {
                  company = (string)rw["Company"];
               }
               vdxSearchParameters sp = new
                  vdxSearchParameters("Address");
```

```
                    sp.Add(new vdxSearchParameter("AddressId",
                       ((int)rw["AddressId"]),vdxSearchType.Equal));

                    nodeToFill.Nodes.Add(new vdxTreeNode(company,2,0,true,sp,
                       "MyVdxSampleApplication.DataDetailAddress"));
                 }
                 break;
           }
        }
     }
```

This method is always executed after the *vdxLoadData* method, assuming it returns a *DataSet*. The method's parameters determine in which context the data is displayed, which *DataSet* is to be used, and if necessary, which tree node's sub-nodes are to be displayed.

Data is generally displayed in the following cases:

- *Selection* – Display the appropriate tree nodes, based on the data returned by a selection.
- *Sub-selection* – Display the appropriate sub-nodes of an expanded tree node.

NOTE: Use *namespace* of your project instead of *MyVdxSampleApplication*. If you have used the naming conventions of this tutorial, then nothing needs to be changed.

For each *DataRow* in the *Address* table, a *Company* is tied to the tree node as a description and a *vdxSearchParameters* object is tied to the tree node as a primary key. The primary key is needed to find the person data which belong to the address.

The following table describes the parameters of the *vdxTreeNode* constructor.

Parameter and type	Description
1. string	The description of the corresponding tree node
2. int	The *imageListTreeView* index for tree nodes which are not selected
3. int	The *imageListTreeView* index for selected tree nodes
4. bool	True, if the node should always be expandable
5. vdxSearchParameters	The *primary key* of the corresponding tree node
6. string	The *Fullqualified type* of the corresponding tree node

The tree view can now be filled.

1.8.3 Overwriting the *VdxUpdateNode*-method

If a user changes the details of an address, the corresponding tree view node must be changed. The next steps implement this.

1. Select the *Class View* of the project.
2. Navigate to the *vdxUpdateNode* method in the *vdxExplorerTree* class.
3. Execute the *Add* → *Override* command from the context menu of the *vdxUpdateNode* method. This adds an empty method to the *ExplorerAddress* class, overwriting the base class'.

4. Add the following code to the method.

```
switch (nodeToUpdate.TreeNodeType)
{
   case "vdxSampleApplication.DataDetailAddress":
      string company = "";
      if (!Convert.IsDBNull(dataRowFromDataDetailControl["Company"]))
      {
         company = (string)dataRowFromDataDetailControl["Company"];
      }
      nodeToUpdate.Text = company;

      // Falls die Drag'n Drop-Funktionalität von Tree-Knoten auf die
      // Outlookbar aktiviert ist, dann sollten jeweils die Suchparameter
      // nach einem Knoten-Update neu gesetzt werden. Damit können auch
      // neu eingefügte Tree-Knoten per Drag'n Drop auf die Outlookbar
      // gezogen werden.
      nodeToUpdate.VdxSearchParameters = new
         vdxSearchParameters("Address");
      nodeToUpdate.VdxSearchParameters.Add(new vdxSearchParameter(
         "AddressId",((int)dataRowFromDataDetailControl["AddressId"]),
         vdxSearchType.Equal));
      break;

   default:
      break;
}
```

This method is always called when an address' detail data is changed and stored and the detail data is being held in the context of the tree view.

The *nodeToUpdate* parameter determines which tree node is affected by the change. The current data for the corresponding tree node are read as parameters. The data in the *DataRow* are used to change the corresponding tree node.

1.8.4 Overwriting the *VdxUpdateSelectionList*-method

The following steps describe the implementation of a method which updates the changed address data to the list.

1. Select the *Class View* of the project.

2. Navigate to the *vdxUpdateSelectionList* methode of the *vdxExplorerTree* class.

3. Execute the *Add* → *Override* command from the context menu of the *vdxUpdateSelectionList* method. This adds an empty method to the *ExplorerAddress* class, overwriting the base class'.

4. Add the following code to the method.

```
if (sourceDataRow.Table.TableName == "Address")
{
   targetValues = base.VdxDataRowToSelectionListValueArray(sourceDataRow);
   base.VdxUpdateSelectionList(targetValues,sourceDataRow);
}
```

This method receives the list values of the corresponding *DataRow* and passes them to the *DataGrid* in the *SelectionListAddress* class.

NOTE: Because the *targetValues* parameter is normally passed by reference, all local changes have an effect on the original *targetValues* variable.

Compile the project to check the previous code for errors.

1.9 Programming the vdxActionItems

In order to load the address maintenance form by clicking on the address icon on the Outlookbar, either the corresponding click event must be implemented or the icon must be linked to the *vdxDataActionManager* object. The following steps describe the latter approach:

1. Select the properties of the *vdxDataActionManager* object on the *MainForm*.

2. Open the collection editor of the *VdxActionItem* properties.

3. Click *Add* and rename the new *vdxActionItem* to *aiAddressExplore*.

4. To link the action event with the corresponding control, select the *iconAddressExplore* entry in the *VdxInvokerControl1* property.

5. Select the properties of the *aiAddressExplore* object.

6. Select the events view (button with the yellow lightning bolt) and double-click on the *vdxActionInvoked* event of the *aiAddressExplore* object. This adds the *aiAddressExplore_vdxActionInvoked* event listener method to the *MainForm* class.

7. **NOTE:** Because one cannot switch to the various events directly from the collection editor, one must close the collection editor and select the *ActionItem* object by selecting it from the ComboBox of the *Properties* window.

8. Add the following code to the method:

```
this.vdxMultiContainer.VdxCurrentControlType = typeof(ExplorerAddress);
((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl).Show();
```

   When the address icon on the Outlookbar is clicked, this code will load the appropriate search form using the *SelectionAddress* class. If a selection already exists, then it will be displayed.

To enable the user to start another address search, the *Find* button will be linked to the *ActionManager* An alternative would be to link the *Find* button to an event handler. The following steps describe the first variation.

1. Select the properties of the *vdxDataActionManager* object.

2. Open the collection editor of the *VdxActionItemProperties*.

3. Click on *Add* and rename the *vdxActionItem* to *aiFind*.

4. Select the *mnuEditFind* item for the *VdxInvokerMenu1* property in order to link the action event to the menu item.

5. Select the *tbbFind* entry for the *VdxInvokerToolBarButton1* property in order to link the action event with the corresponding toolbar *Button*.

6. Select the *aiFind* object properties.

7. Double-click the *vdxActionInvoked* event of the *aiFind* object. This adds the *aiFind_vdxActionInvoked* event listener method to the *MainForm* class.

8. Add the following code to the method.

```
if ((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl != null)
{
   ((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl).
      Show(vdxExplorerMode.Selection);
}
```

The address part of the sample application is now complete. Compile and run the project to execute the previous code. Click on the *Explore* button under the *Address* category on the Outlookbar to start the address maintenance. You can enter search criteria and retrieve the corresponding addresses. These can be edited and the changes saved to the database. You can also add new addresses or copy existing addresses.

The maintenance of persons will be implemented next. Since this is very similar to the maintenance of addresses, only the differences will be highlighted.