Visual.NET Extensions – Tutorial

"Setup of a new Client Application using the VDX Framework"

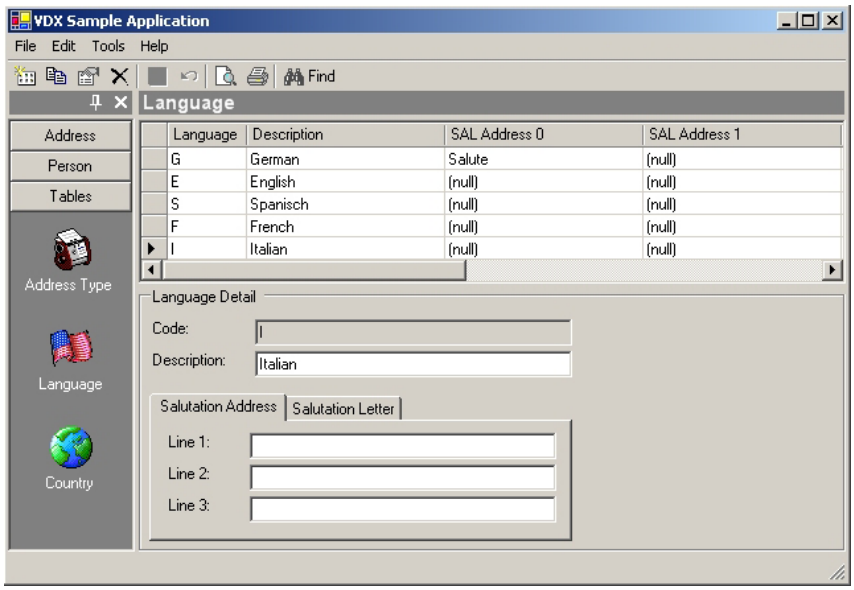
The extensive application development framework for the simple development of Microsoft Visual Studio.NET database applications!

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

Version: 1.1
Last Update: 22.11.2002

1 Setup of a new Client Application using the VDX Framework

This section shows how to build a sample database application using the VDX class libraries. The following figure shows the completed application, which is also supplied with VDX (see also product delivery in the VDX Technical Reference). In the following tutorials, this application gets completed step-by-step.



The following concepts apply to the design of the application:

Server:

- Distributed server application based on XML *Webservices.*
- Clients and servers communicate via SOAP so that even clients behind a firewall can access the Web services.
- 3-tier architecture
- The services supplied by the server, i.e. the Web services, can be easily moved to another computer at any time.
- The server database is based on MS SQL Server 2000.
- The services are implemented with C# .NET. They could be implemented with any SOAP-compatible language, but it is recommended to use a .NET language in order to take advantage of various server classes offered by the VDX Framework, which could not be used otherwise.

Client:

- There is a locally installed .NET application on each client which is built using the VDX classes.
- The clients access the server via SOAP proxy objects.

- The sample application is a single-document interface (SDI) application without additional dialog windows, i.e. the entire application uses one window, with the exception of message boxes.

Requirements for the development of the address application:

- The .NET framework class libraries, which can be downloaded from Microsoft's web site, must be installed. These contain the CLR (run-time component for .NET programs) and a command-line compiler.
- Visual Studio .NET is recommended for a more comfortable development environment.
- Basic OO knowledge

1.1 Set Up Client Project

Set up an empty VDX client project.

1. Start Visual Studio .NET.
2. Create a new C# project. Select the *Empty Project* template (not *Windows Application*) and rename it *MyVdxSampleApplication*.
3. Add the following assembly references to the project (via *Project→ Add Reference...*):

Assembly	Path
vdxControls	%VDX_Installationspath%\vdxControls.dll
vdxCommon	%VDX_Installationspath%\vdxCommon.dll

4. Add the following Web Reference (via *Project→ Add Web Reference...*) in order to access the supplied server application.

```
http://%Computername%/vdxSampleWebServices/vdxSampleWebServices.vsdisco
```

NOTE: The Installation of the *WebServices* is described in the file readme.doc (under Program Files\VDX\VDX 1.1 SampleClient\).

5. Click on *Add Reference*.
6. Rename the added reference from *localhost* to *vdxSampleWebServices*.
7. Add a new class to the project (via *Project→ Add Class...*) and rename it to *SampleApp.cs*.
8. Insert the following VDX namespace references at the top of the code in the *SampleApp* class to allow easier access the VDX classes.

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

9. In order to use the VDX functions in the sample application, inherit it from the *vdxApp* class by modifying the code as follows.

```
public class SampleApp : vdxApp
```

10. Insert the following method into the *SampleApp* class:

```
[STAThread]
static void Main()
{
   new SampleApp();
   VdxStartApp();
}
```

This is the entry point for the application. For the time being it will only start the application, execute its own constructor to initialize its properties with appropriate values, but not display a window yet.

NOTE: Because the variables and methods of the *vdxApp* class are declared as s*tatic*, you do not need to create a local instance of *SampleApp*.

11. To define this method as the entry point for the application, open the project properties (select the project and then *Project* → *Properties*) and set the *Output Type* and *Startup Object* to the specified values:

Property	Selection
Output Type	Windows Application
Startup Object	MyVdxSampleApplication.SampleApp

NOTE: The *Startup Object* must contain the fully-qualified name of the *SampleApp* class, where the *Namespace* part depends on the name of your project (*MyVdxSampleApplication* for the sample application).

The initial client-side setup of the VDX application is concluded for now. The following sections systematically describe how to build the client application.

1.2 Create and Integrate the Main Window

This section shows how to build the actual application – the SDI window in which the Address database application will run. This window class will contain the GUI part as well as some of the logic. The following steps create and integrate an empty window.

1. Add a new *Windows Form* class to the project (via *Project* → *Add Windows Form...*) and name it *MainForm*.

2. Add the following VDX namespace references to the *MainForm* code.

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

3. This windows class must implement the *IvdxMainForm* interface so that the application can be controlled by the *SampleApp* class. odify the class declaration as follows:

```
public class MainForm : System.Windows.Forms.Form, IvdxMainForm
```

4. The *IvdxMainForm* interface defines a *Show* method which will be called by VDX. Because it is an interface, the method must be implemented. Add the following code to the *MainForm* class.

```
public new void Show()
{
   this.ShowDialog();
}
```

5. So that VDX can show the *MainForm* window at the desired time, the *SampleApp* class must define a reference to it. Add the following code between the *new SampleApp* and the *VdxStartApp* statements in the *SampleApp* class.

```
new SampleApp();
VdxMainForm = new MainForm();
VdxStartApp();
```

NOTE: The *VdxMainForm* variable is defined in the *vdxApp* base class and is used to access the *MainForm* class.

6. Compile and run the project. An empty window should be displayed.

1.3 Add the MainMenu

Add the main menu to the sample application:

1. Select the design view of the *MainForm* class and drag & drop a *MainMenu* control from the *Toolbox* mto the *MainForm*.

2. Rename the menu *mainMenu*.

3. Add the following *MenuItem*s to the *mainMenu*. The *Name* property of the respective *MenuItem* is enclosed in square brackets:

File [mnuFile]	Edit [mnuEdit]
New Address [mnuFileNewAddress]	Undo (Ctrl+Z) [mnuEditUndo]
New Person [mnuFileNewPerson]	Save (Ctrl+S) [mnuEditSave]
(Separator) [mnuFileSeparator1]	*(Separator)* [mnuEditSeparator1]
Exit (Alt+F4) [mnuFileExit]	Edit Record (Ctrl+E) [mnuEditEditRecord]
	Add Record (Ctrl+N) [mnuEditAddRecord]
	Copy Record [mnuEditCopyRecord]
	Delete Record (Ctrl+D) [mnuEditDeleteRecord]
	(Separator) [mnuEditSeparator2]
	Find (Ctrl+F) [mnuEditFind]

NOTE: Insert separators with the *Insert Separator* command on the *mainMenu* context menu. To define shortcuts for menu items, set the *Show Shortcut* property to *true* and select the desired keystroke sequence for the *Shortcut* property. The shortcuts are only visible at run time.

4. Compile and run the project. The windows forms and its menu should be displayed.

The *MenuItem*s will be linked to actions after all of the controls have been placed on the form.

1.4 Add the Toolbar ImageList

Define the *Images Collection* for the toolbar.

1. Select the design view of the *MainForm* class and drag an *ImageList* control onto it from the *Windows Forms* tab of the toolbox.

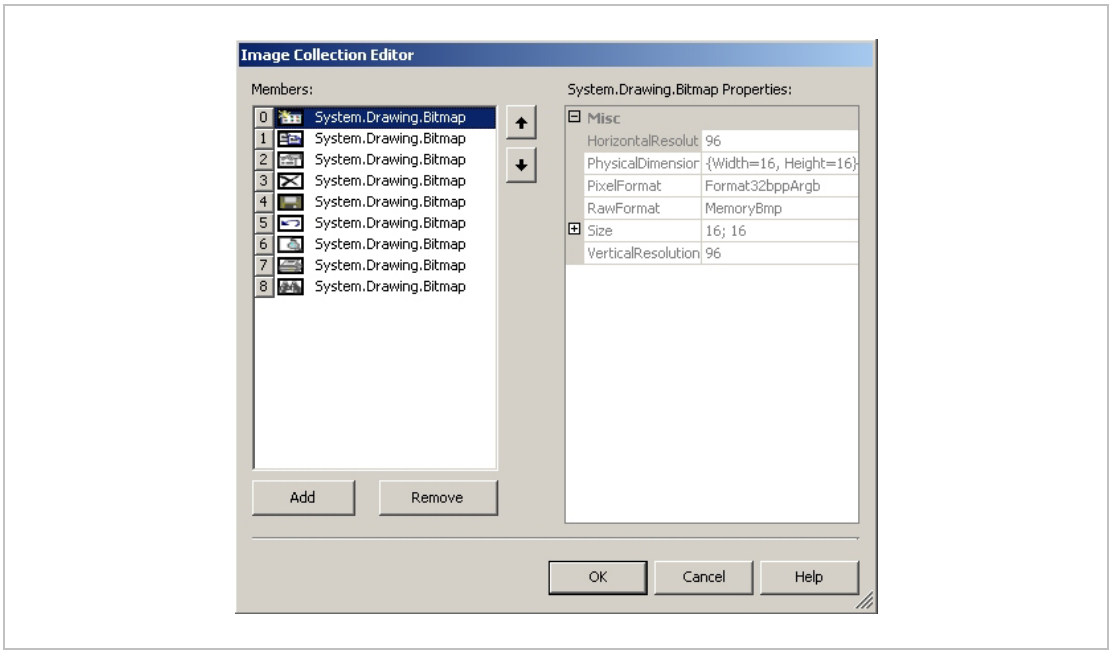
2. Rename the *ImageList imageListToolBar*.

3. Set the *ImageSize* property to 16x16 pixels, the default.

4. Add the following *Images* to the list, assigning them to the specified index. To open the collection editor, click the selection button of the *Images* property. The icons are in the *icons\toolbar* VDX installation subdirectory.

Collection Index	Image
0	Add.ico
1	Copy.ico
2	Edit.ico
3	Delete.ico
4	Save.ico
5	Undo.ico
6	Find.ico

The following figure shows the collection editor and the added icons.



The *ImageList* will be used by the buttons of the toolbar which will be added in the next section.

1.5 Add the Toolbar

The next steps define the toolbar for the sample application. The toolbar is used for quick access to the most important menu commands.

1. Select the design view of the *MainForm* class and drop a *Toolbar* control onto it from the *Windows Forms* tab of the toolbox.

2. Rename the *Toolbar toolbar*.

3. Associate the previously created *ImageList* with the toolbar by setting the *ImageList* property to *imageListToolbar*.

4. Select the *Button* property and add the following *Buttons* and *Separators* to the *Collection*.

Button Name	Icon Index	ToolTip Text	Text
tbbAddRecord	0	Add	
tbbCopyRecord	1	Copy	
tbbEditRecord	2	Edit	
tbbDeleteRecord	3	Delete	
tbbSeparator1			
tbbSave	4	Save	
tbbUndo	5	Undo	
tbbSeparator2			
tbbFind	6	Find	Find

The *Toolbar Buttons* will be linked to actions later.

1.6 Add the Outlookbar ImageList

Similar to the toolbar, you must define the *ImageList* for the Outlookbar.

1. Select the design view of the *MainForm* class and drag an *ImageList* control onto it from the *Windows Forms* tab of the toolbox.

2. Rename the *ImageList imageListOutlookBar*.

3. Set the *ImageSize* to 32x32 pixels.

4. Add the following *Images* to the list, assigning them to the specified index. The icons are in the *icons\outlookbar\large* VDX installation subdirectory.

Collection Index	Image
0	Address.ico
1	Person.ico
2	Language.ico
3	Country.ico

The *ImageList* will be used by the buttons of the Outlookbar which will be added in the next section.

1.7 Add the vdxOutlookbar

The next steps define the Outlookbar for the sample application. The Outlookbar is used to access various subunits of the application, e.g. addresses, country codes, etc.

1. Select the design view of the *MainForm* class and drag a *vdxOutlookBar* control onto it from the *VDX* tab of the toolbox.

2. Rename the *Outlookbar vdxOutlookBar*.

3. Dock the Outlookbar to the left side of the application window by setting the *Dock* property to *Left*.

1.7.1 Configuration of the vdxOutlookBar

To fill the Outlookbar with groups and icons within the groups, follow these steps:

1. Select the Outlookbar's *ImageListLarge* property and select the previously created *imageListOutlookBar ImageList*.

2. Repeat the previous step for the *ImageListSmall* property.

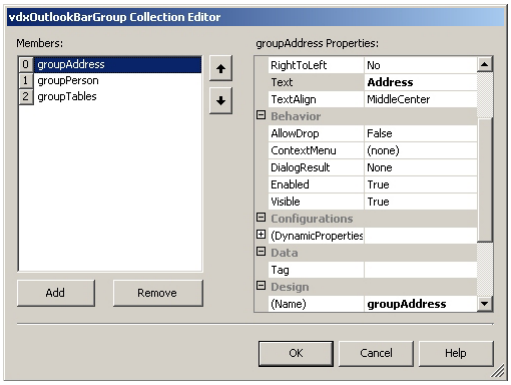
   NOTE: In case alternate or optimized icons are to be used for the small icon view of the Outlookbar, define an additional *ImageList* with those icons, assigning them the same collection index as in the *imageListOutlookBar* main image list.

3. Click the *GroupList* property of the Outlookbar to open the *vdxOutlookBarGroup* collection editor.

4. Click *Add* three times and set the properties of the new groups as follows.

Group	Text	Name
vdxOutlookBarGroup1	Address	groupAddress
vdxOutlookBarGroup2	Person	groupPerson
vdxOutlookBarGroup3	Tables	groupTables

   The *Collection*-Editor looks like this:



5. Insert the icons into their respective groups. Select the *groupAddress* group and click the *IconList* property. Click *Add* and set the properties of the icon as follows.

Text	Name	ImageIndex
Explore	iconAddressExplore	0

   Select the *groupPerson* group and repeat the previous step:

Text	Name	ImageIndex
Explore	iconPersonExplore	1

   Do the same for the *groupTables* group: The icons of the Outlookbar will be linked to actions later.

Text	Name	ImageIndex
Address Type	iconTablesAddrType	0
Language	iconTablesLang	2

Country	iconTablesCountry	3

1.8 Add the TreeView ImageList

The sample application supplies a *TreeView* to navigate the selected data. The following steps define the icons for the various nodes of the *TreeView*.

1. Select the design view of the *MainForm* class and drag an *ImageList* control onto it from the *Windows Forms* tab of the toolbox.

2. Rename the control *imageListTreeView*.

3. Confirm that the *ImageSize* is set to 16x16 pixels.

4. Add the following *Images* to the list, assigning them to the specified index. The *icons\treeview* VDX installation subdirectory contains a selection of icons which can be used.

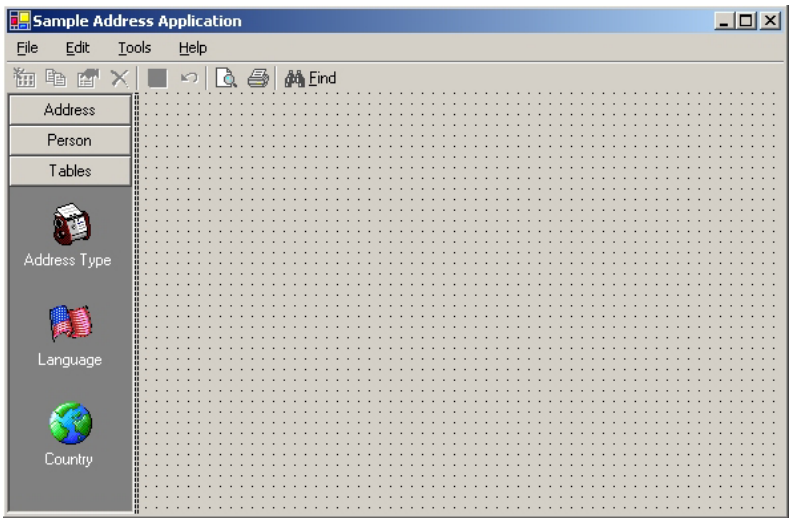
Collection Index	Image
0	Arrow.ico
1	Selection.ico
2	Address.ico
3	Person.ico

1.9 Add the Splitter -Control

A *Splitter* control will be added to allow the user to change the size of the Outlookbar at run time.

1. Select the design view of the *MainForm* class and drag a *Splitter* control onto it from the *Windows Forms* tab of the toolbox. The *Splitter* should be positioned against the left edge of the Outlookbar. It does not require any further configuration.

The following figure shows the current state of the *MainForm*.



To verify the work up to this point, build the application (via *Build* → *Rebuild Solution*) and run it.

1.10 Add the vdxDataActionManager

A *vdxActionManager* will be added to associate the click of a button with an action. For demonstration purposes, only the standard buttons and the menu items *Add*, *Copy*, *Delete*, *Save, Undo* and *Find* will be implemented. The application-specific *ActionItems* will be implemented in the appropriate places as needed.

1. Select the design view of the *MainForm* class and drag a *vdxDataActionManager* control onto it from the *VDX* tab of the toolbox.

2. Rename the control *vdxDataActionManager*.

3. To link action events with the appropriate controls, the appropriate properties of the *vdxDataActionManager* must be set as shown in the following table.

Property	Selection
ToolBar1	toolBar
VdxDataActionItemAdd.VdxInvokerMenu1	mnuEditAddRecord
VdxDataActionItemAdd.VdxInvokerToolBarButton1	tbbAddRecord
VdxDataActionItemCopy.VdxInvokerMenu1	mnuEditCopyRecord
VdxDataActionItemCopy.VdxInvokerToolBarButton1	tbbCopyRecord
VdxDataActionItemDelete.VdxInvokerMenu1	mnuEditDeleteRecord
VdxDataActionItemDelete.VdxInvokerToolBarButton1	tbbDeleteRecord
VdxDataActionItemEdit.VdxInvokerMenu1	mnuEditEditRecord
VdxDataActionItemEdit.VdxInvokerToolBarButton1	tbbEditRecord
VdxDataActionItemSave.VdxInvokerMenu1	mnuEditSave
VdxDataActionItemSave.VdxInvokerToolBarButton1	tbbSave
VdxDataActionItemUndo.VdxInvokerMenu1	mnuEditUndo
VdxDataActionItemUndo.VdxInvokerToolBarButton1	tbbUndo

4. Double-click the *VdxDataActionItemAdd.VdxActionInvoked* event of the *vdxDataActionManager* object. The *vdxDataActionManager_VdxDataActionItemAdd_vdxActionInvoked* event listener method will be added to the *MainForm* class.

   NOTE: If the name of the event listener method is inserted incorrectly, repeat this step, but first delete the event listener method and the corresponding registration of the event (‚+=' statements) in the *InitializeComponent* method.

5. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   this.vdxDataActionManager.VdxDataStatusManager.Add();
}
```

   Clicking the *Add* button or selecting the *Add* menu item will execute the *Add* method of the *DataStatusManager*, which inserts a new record.

6. Double-click the *VdxDataActionItemCopy.VdxActionInvoked* event of the *vdxDataActionManager* object. The *vdxDataActionManager_VdxDataActionItemCopy_vdxActionInvoked* event listener method will be added to the *MainForm* class.

7. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   try
   {
      this.vdxDataActionManager.VdxDataStatusManager.Copy();
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

NOTE: If the copy method fails, the thrown exception will be caught in the *catch* block where you can implement the error handling of your choice.

8. Double-click the *VdxDataActionItemDelete.VdxActionInvoked* event of the *vdxDataActionManager* object. The *vdxDataActionManager_VdxDataActionItemDelete_vdxActionInvoked* event listener method will be added to the *MainForm* class.

9. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   try
   {
      this.vdxDataActionManager.VdxDataStatusManager.Delete();
   }
   catch (System.Exception ex)
   {
      this.vdxDataActionManager.VdxDataStatusManager.Undo();
      MessageBox.Show(ex.Message);
   }
}
```

Clicking the *Delete* button or selecting the *Delete* menu item will execute the *Delete* method of the *DataStatusManager*, which deletes a record. If the deletion throws an exception, the *Undo* method will be executed to reverse the deletion.

10. Double-click the *VdxDataActionItemEdit.VdxActionInvoked* event of the *vdxDataActionManager* object. The *vdxDataActionManager_VdxDataActionItemEdit_vdxActionInvoked* event listener method will be added to the *MainForm* class.

11. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   this.vdxDataActionManager.VdxDataStatusManager.Edit();
}
```

12. Double-click the *VdxDataActionItemSave.VdxActionInvoked* event of the *vdxDataActionManager* object. The

vdxDataActionManager_VdxDataActionItemSave_vdxActionInvoked event listener method will be added to the *MainForm* class.

13. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   try
   {
      this.vdxDataActionManager.VdxDataStatusManager.Save();
   }
   catch (System.Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

14. Double-click the *VdxDataActionItemUndo.VdxActionInvoked* event of the *vdxDataActionManager*-Objektes. The *vdxDataActionManager_VdxDataActionItemUndo_vdxActionInvoked* event listener method will be added to the *MainForm* class.

15. Add the following code to the new event listener method.

```
if (this.vdxDataActionManager.VdxDataStatusManager != null)
{
   this.vdxDataActionManager.VdxDataStatusManager.Undo();
}
```

The *ToolBarButtons* and *MenuItems* will now trigger the appropriate actions but will not carry out these actions because the necessary controls do not yet exist. Compile and run the application to the previous code for errors.

The next sections explain the implementation and integration of the database controls for the currently empty right section of the *MainForm*. These controls will all be placed into a *Container* control which manages all of them.

1.11 Add the vdxMultiContainer Control

Select the design view of the *MainForm* class and drag a *vdxMultiContainer* control from the *VDX* tab of the toolbox onto the right, empty area of the window. Rename the control *vdxMultiContainer*. Set the *Dock* property to *Fill*.

The *vdxMultiContainer* control manages the controls embedded on it. For example, a most recently loaded control which displays addresses is stored in the *vdxMultiContainer* control so that the addresses can be shown again instantly at any time without delay. The *vdxMultiContainer* control also provides four events which are always executed when an embedded control is created, activated, deactivated or reactivated. These events will not be linked to actions yet.

1.11.1 Programming the *vdxMultiContainer*-Controls

To link the embedded objects with the v*dxMultiContainer*-Objekt, follow these steps:

1. Select the *vdxMultiContainer*-Object in the designer area of the *MainForm*-class.

2. Double-click the *VdxControlActivated* event.

3. Repeat these steps for the v*dxControlCreated*- and v*dxControlDeactivated*-Events. Following table lists these events with their corresponding Event-Listener-Methods entered in the MainForm class.

Event	Event-Listener
VdxControlActivated	vdxMultiContainer_vdxControlActivated
VdxControlCreated	vdxMultiContainer_vdxControlCreated
VdxControlDeactivated	vdxMultiContainer_vdxControlDeactivated

4. Add the following code in the *vdxMultiContainer_vdxControlCreated* event listener method. You find this method in the *MainForm* class.

```
((vdxOutlookDetail)e.Control).Show();

((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl).
   VdxImageList = this.imageListTreeView;
```

   The first statement loads the Address-Selection, if the ExplorerAddress class gets loaded for the first time. In the second statement the corresponding *ImageList*, needed for the visualization of the Tree-Node-Icon, gets set.

5. Add the following code in the *vdxMultiContainer_vdxControlActivated* event listener method:

```
((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl).
   VdxDataActionManager = this.vdxDataActionManager;
```

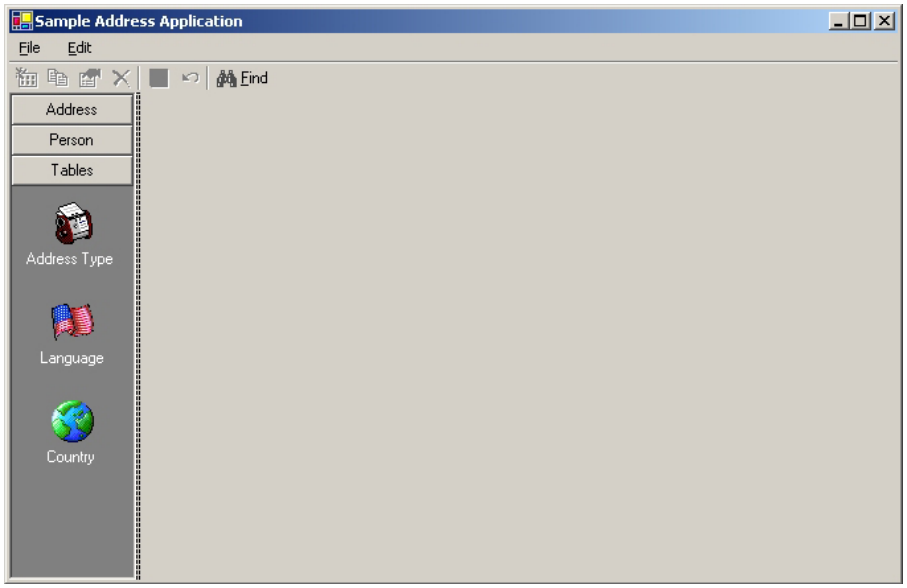
   In order to link the *vdxDataActionManager* with the visible controls, we set the VdxDataActionManager every time, an Address-Control gets activated.

6. Add the following code in the *vdxMultiContainer_vdxControlDeactivated* event listener method:

```
((vdxOutlookDetail)this.vdxMultiContainer.VdxCurrentControl).
   VdxDataActionManager = null;
```

   Contrasting with the above, the *vdxDataActionManager* gets removed, as soon as an Address-Control gets deactivated.

Compile and run the project. Following window should be displayed:



In the following tutorials, the VDX Sample Application gets completed with specific application scenarios.