Getting Started - Code Samples - vdFramedControl

This guide will help you add the new VectorDraw Developers Framework (VDF) version 6 vdFramedControl in your Visual Studio 2005 projects and also show you some basic functionality of the vdFramedControl.

Click on the links below to see a "Getting started" guide, that is adding the vdFramedControl in a project and creating a very simple project for the platform that you use :

- Visual C# 2005
- Visual C++ 2005
- Visual Basic 2005
- Authorizing the VDF Libraries

The topics above contain information on how to add the VDF vdFramedControl to an empty Visual Studio 2005 project and also some steps to add to this project some basic functionality like open/save/new and zoom.

You can also see some more complex examples on specific issues like (alphabetic order):

- ActionUtility
- Blocks
- Colors & Palette
- CommandAction
- Dimensions
- DimStyles
- Document Properties & Events
- Drag & Drop
- Globalization of Resources
- Grips
- Handle
- Hatches
- Images
- Layers
- Layouts & Viewports
- Linetypes
- MText & Text
- Osnaps
- Section Clipping
- Selections
- SupportPath
- TextStyles
- Tooltips & URL
- Undo & Redo
- Units (Linear - Angle)

- XProperties
- External Reference (Xrefs)

See Also:

You can check the **CADBasics.chm** help file that is installed in your disk with our VDF setup (in a folder like "*C:\Program Files\VectorDraw\VectorDraw Developer Framework*") to get a general idea regarding CAD programs, and basic CAD terminology (like Line, Layer, Block, Dimension etc)

You can also check our samples that are using the VectorDraw Developer Framework version 6 libraries that are compressed in a cab file (named vdfexamples.cab) and is installed with our VDF setup in a folder like "*C:\Program Files\VectorDraw\VectorDraw Developer Framework\vdFramedControl Samples*". These samples are available for C# and VB.NET 2005;

- **AddEntities** : Is a simple project on how to add all graphical entities (like lines, texts, polylines etc) in a drawing
- **Commands** : This sample demonstrates the use of predefined commands that can be used to draw graphic objects like lines, circles etc (cmdLine, cmdCircle etc) and also transform these objects like Copy, Move, Rotate etc.
- **Collections** : This sample demonstrates the use of collections like XProperties, Layers, Blocks, Images, TextStyles etc
- **MDICAD** : Is a simple CAD program (only in C#) containing a properties list to check the properties of VectorDraw objects and a command line that the use can use to draw/select/edit CAD objects and collections.
- **UserActions** & **Utilities** : A sample that demonstrates user actions like GetPoint, GetRect that can be used for example in selecting objects
- **ActionDraw Event** : A sample using ActionDraw event that can be used by the user to draw an object (like a circle) while the user is inside a user-command like cmdLine
- **CustomObjects** : Is a advanced project on how to create your own graphical objects that have special properties, like a "blinking" object

Below you can find some useful information and How-To regarding some basic objects and functionality of the vdFramedControl

NOTE: In Windows Vista/Seven/Server2008 oper. systems and due to UAC make sure that you ALWAYS run the VS2005/VS2008/VS2010 IDE with right-click and "Run as Administrator" option.

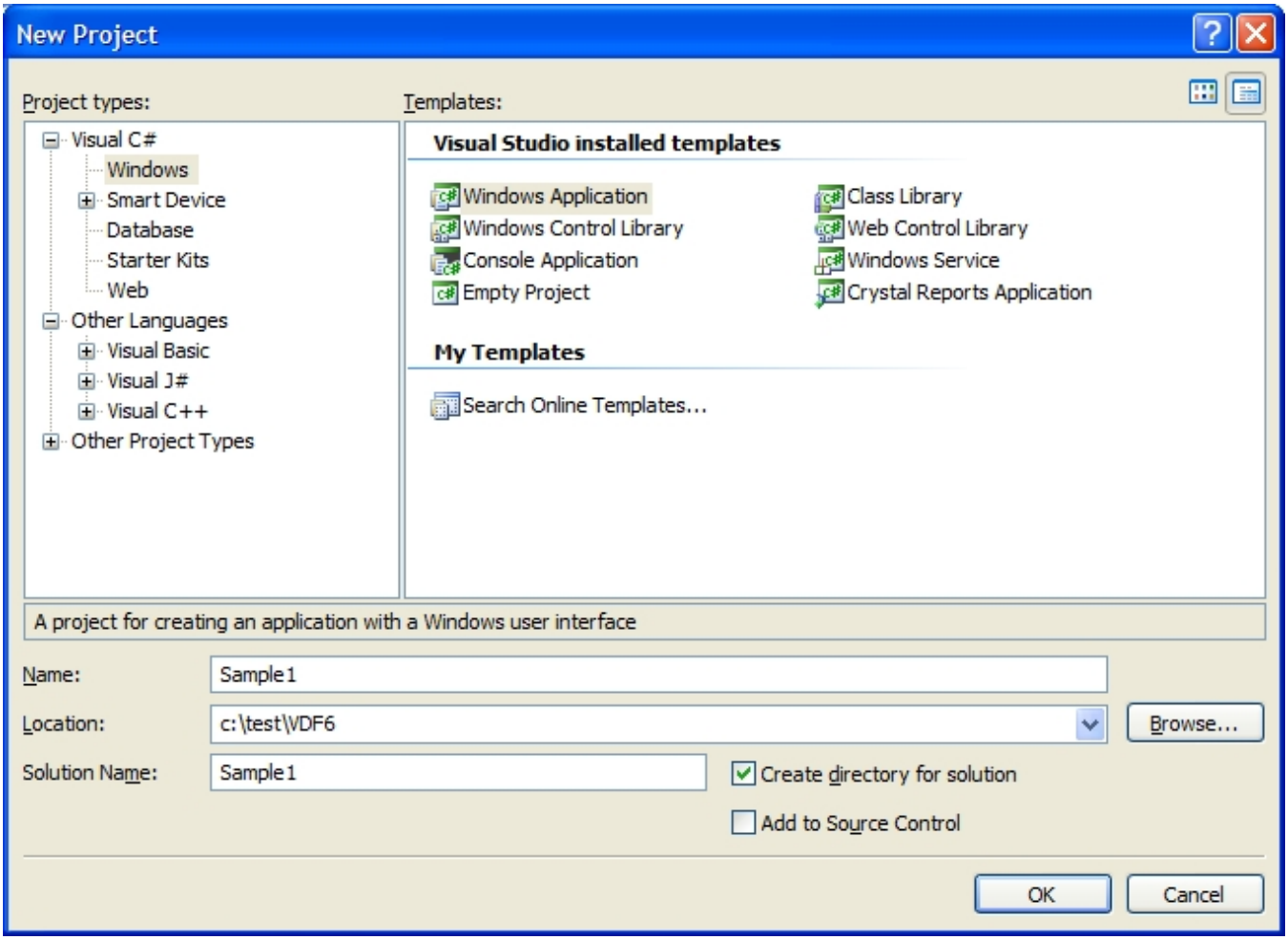
You can also use the VisualStudio 2008 or 2010 instead of VS2005 that is used in this guide. The samples below without any major change can be made in VS2008 or VS2010 development IDEs.

If you use VS2008 or VS2010: When you create a new project then in VS2008 C# do not check "Client-only Framework subset" option. In VS2010 C# choose ".NET Framework 4" and not the ".NET Framework 4 Client Profile".

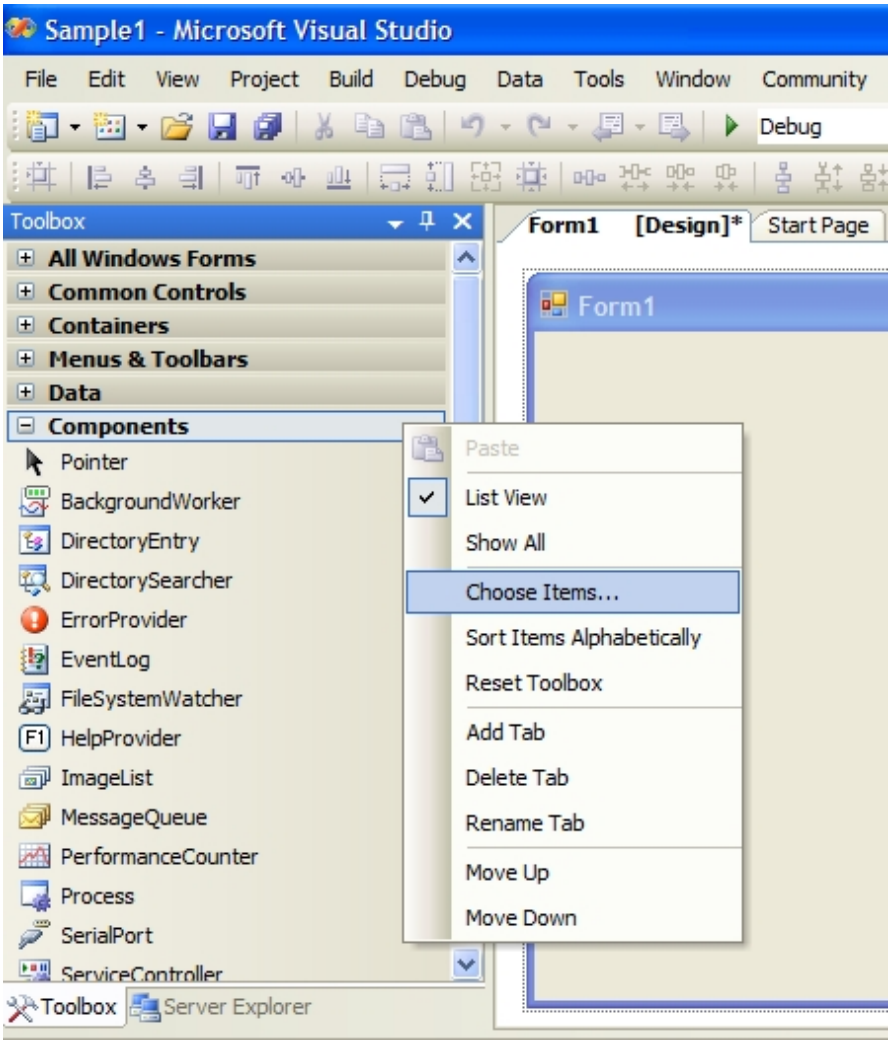
Getting Started - C# 2005 - vdFramedControl

In order to add the VDF vdFramedControl in Visual C# 2005 the steps 1 to 5 must be followed:

1) Open the Visual C# 2005 IDE (see NOTE in 2nd page) and choose to create a new Windows project as "Windows Application"

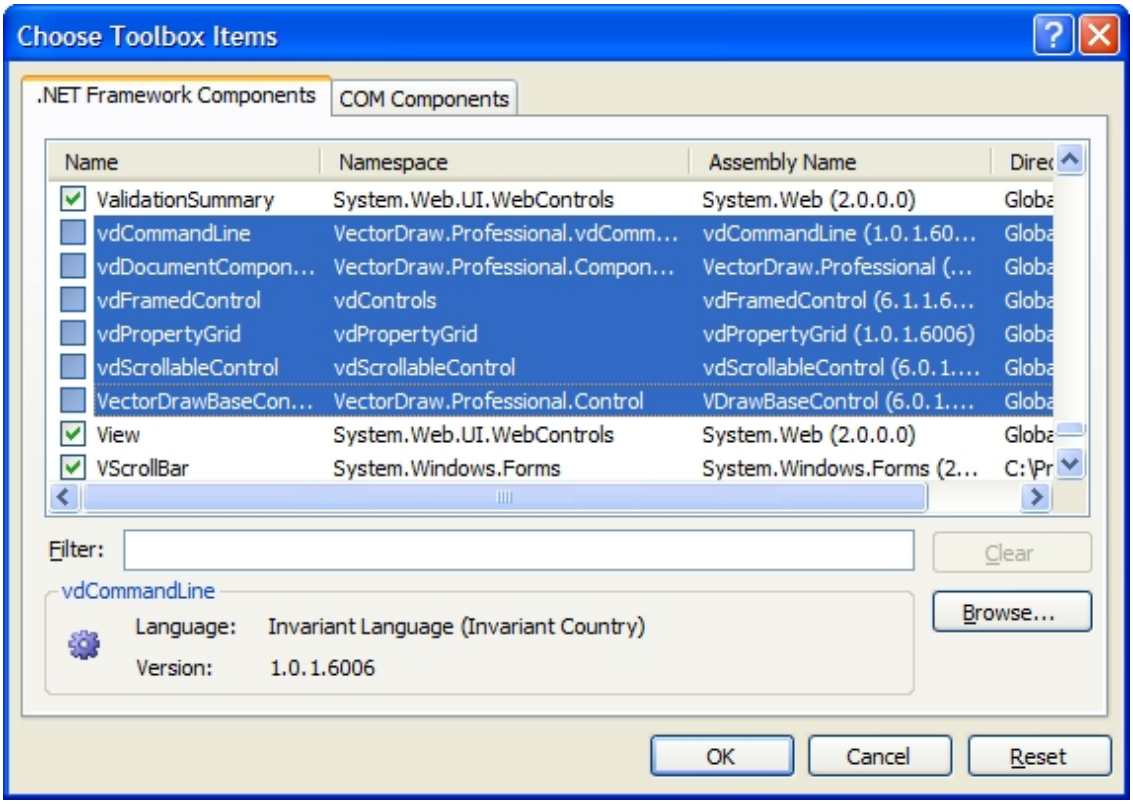


2) Right click on the toolbox and in the menu that appears choose "Choose Items..."



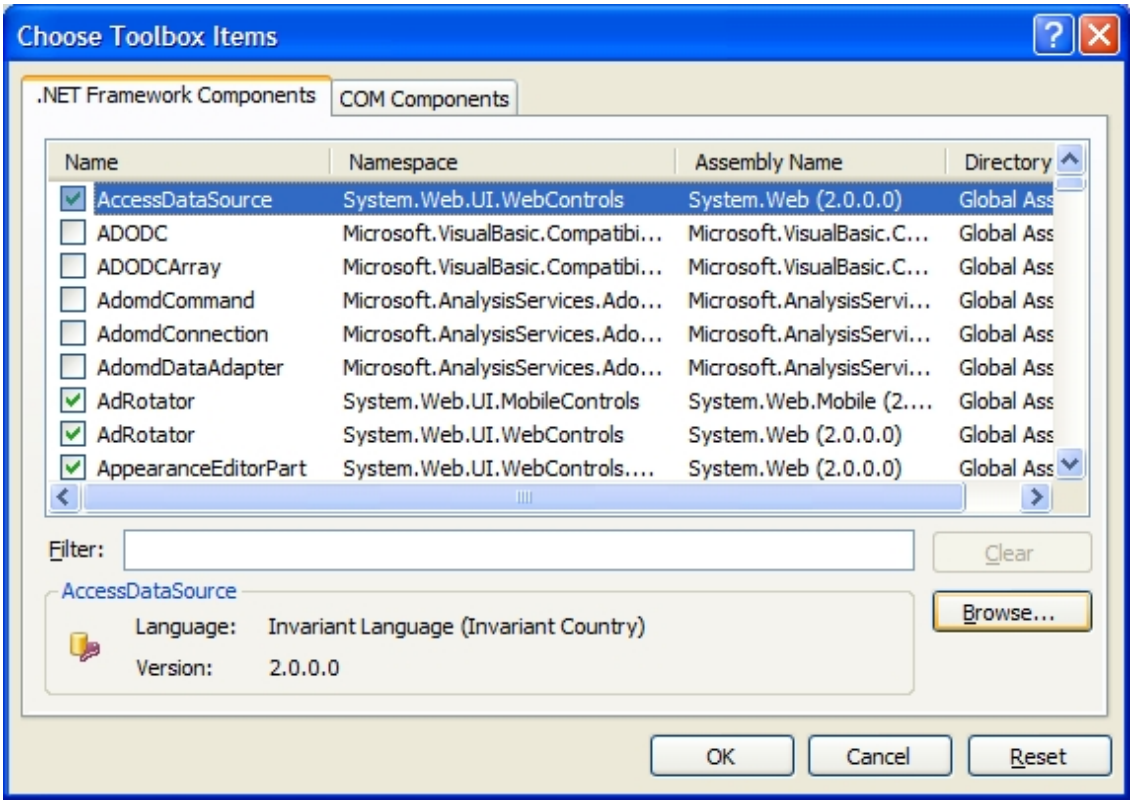
3) You can add the vdFramedControl with 2 different ways:

A) In the "Choose Toolbox Items" scroll down to the "vdFramedControl" and check it.

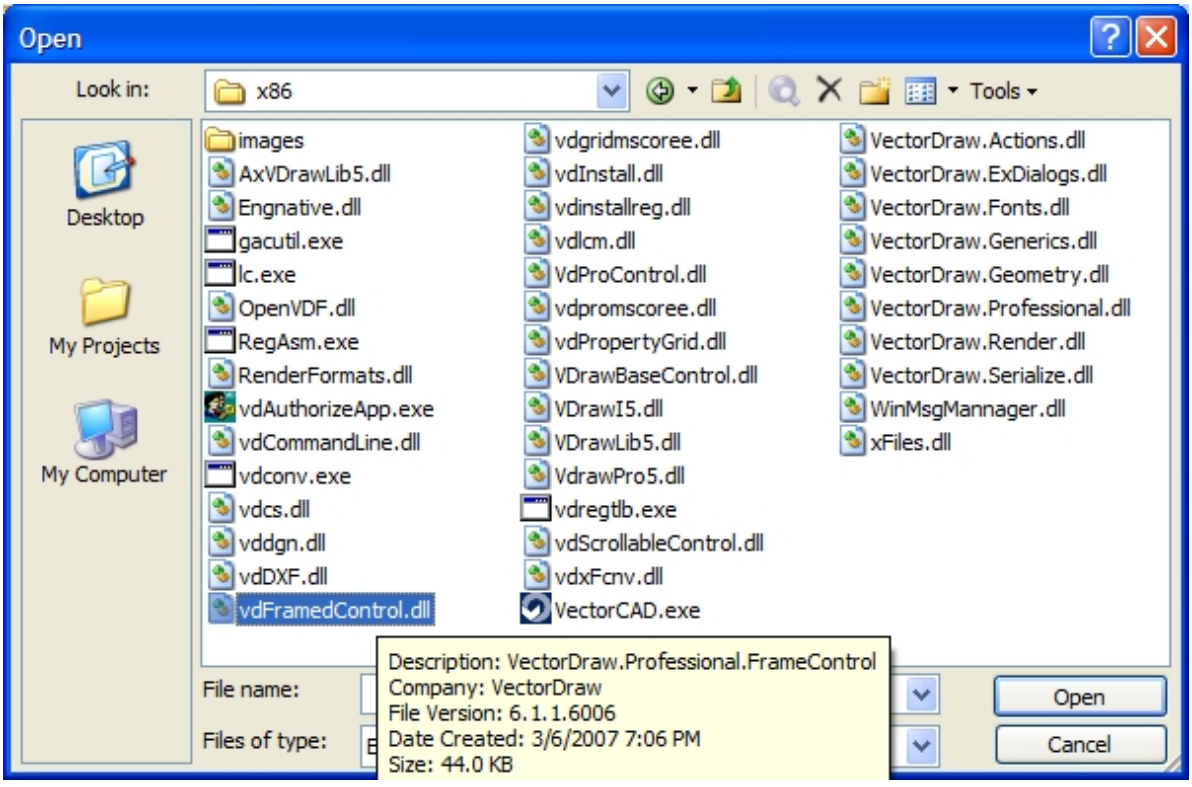


- OR -

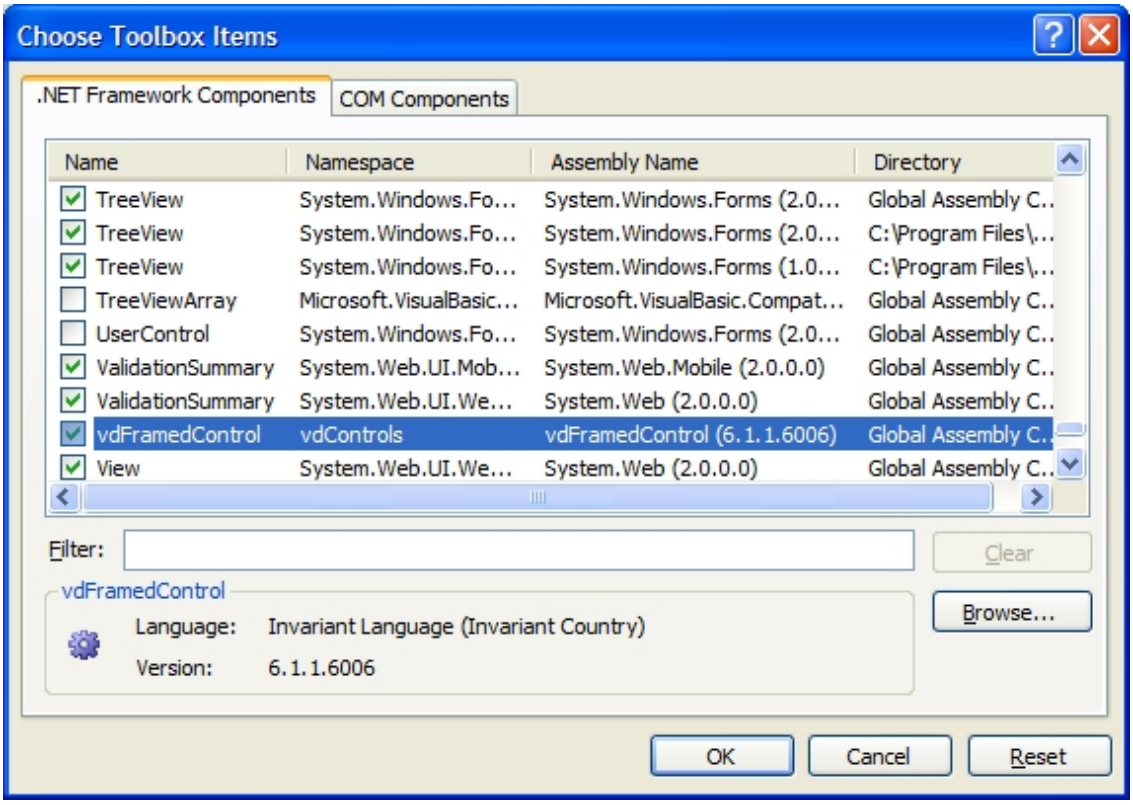
B 1)In the "Choose Toolbox Items" window and click "Browse..." button.



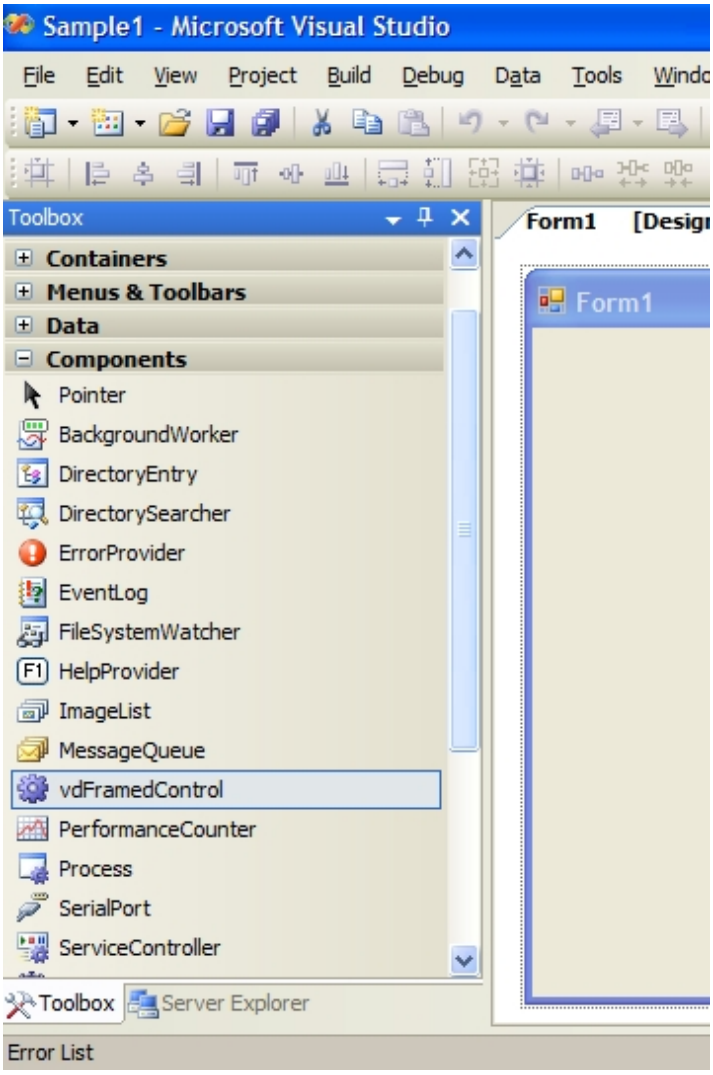
B 2) Go to the folder where the VectorDraw Developer Framework is installed (must be like "*C:\Program Files\VectorDraw\Common*") and click on (select) the vdFramedControl.dll and then click "Open" button.



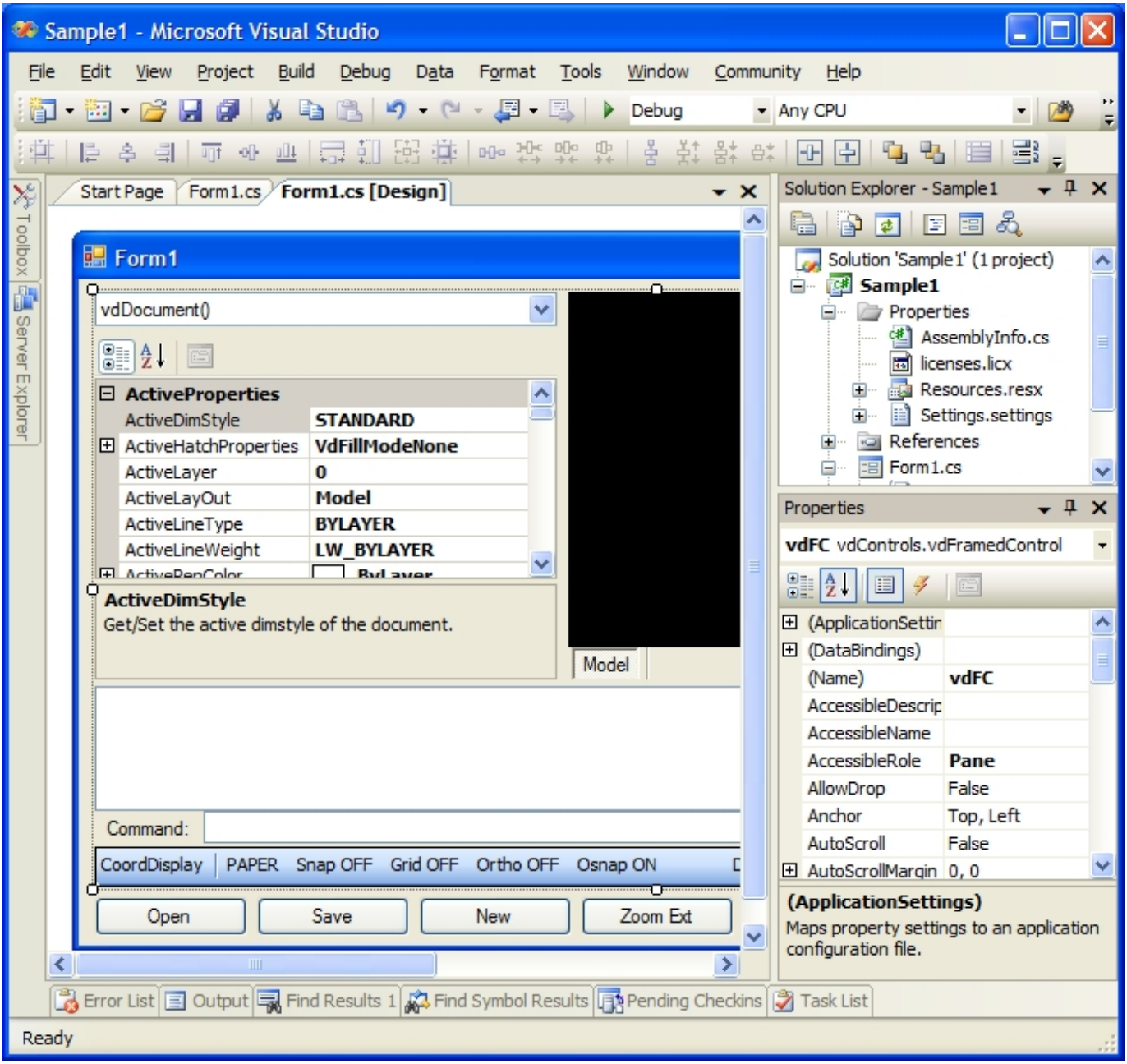
4) You should see it checked in the "Choose Toolbox Items". Click the "OK" button.



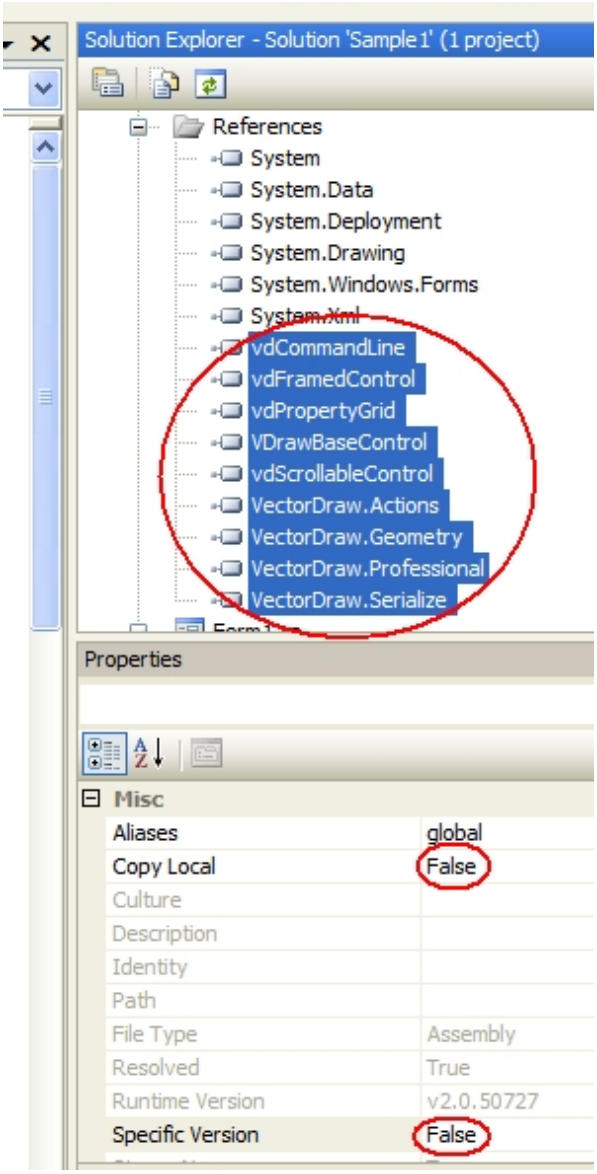
5) The vdFramedControl component should be in the Toolbox.



6) Add it to the form and also 4 buttons that will do some simple functions like Open/Save a drawing, Create a new drawing and Zoom Extends.



7) Select the VDF references and in their properties set Copy Local to False and Specific Version to False too.



9) The code that is added to the buttons click is like below :.

```
        private void btOpen_Click(object sender, EventArgs e)
        {
            object ret = vdFC.BaseControl.ActiveDocument.GetOpenFileNameDlg(0, "", 0);
            if (ret == null) return;
            string fname = (string)ret;
            bool success = vdFC.BaseControl.ActiveDocument.Open(fname);
            if (success) vdFC.BaseControl.ActiveDocument.Redraw(true);
        }

        private void btSave_Click(object sender, EventArgs e)
        {
            vdFC.BaseControl.ActiveDocument.SaveAs("c:\\temp\\test.vdcl");
        }

        private void btNew_Click(object sender, EventArgs e)
        {
            vdFC.BaseControl.ActiveDocument.New();
        }

        private void btZoom_Click(object sender, EventArgs e)
        {
            vdFC.BaseControl.ActiveDocument.CommandAction.Zoom("E", 0, 0);
        }
```

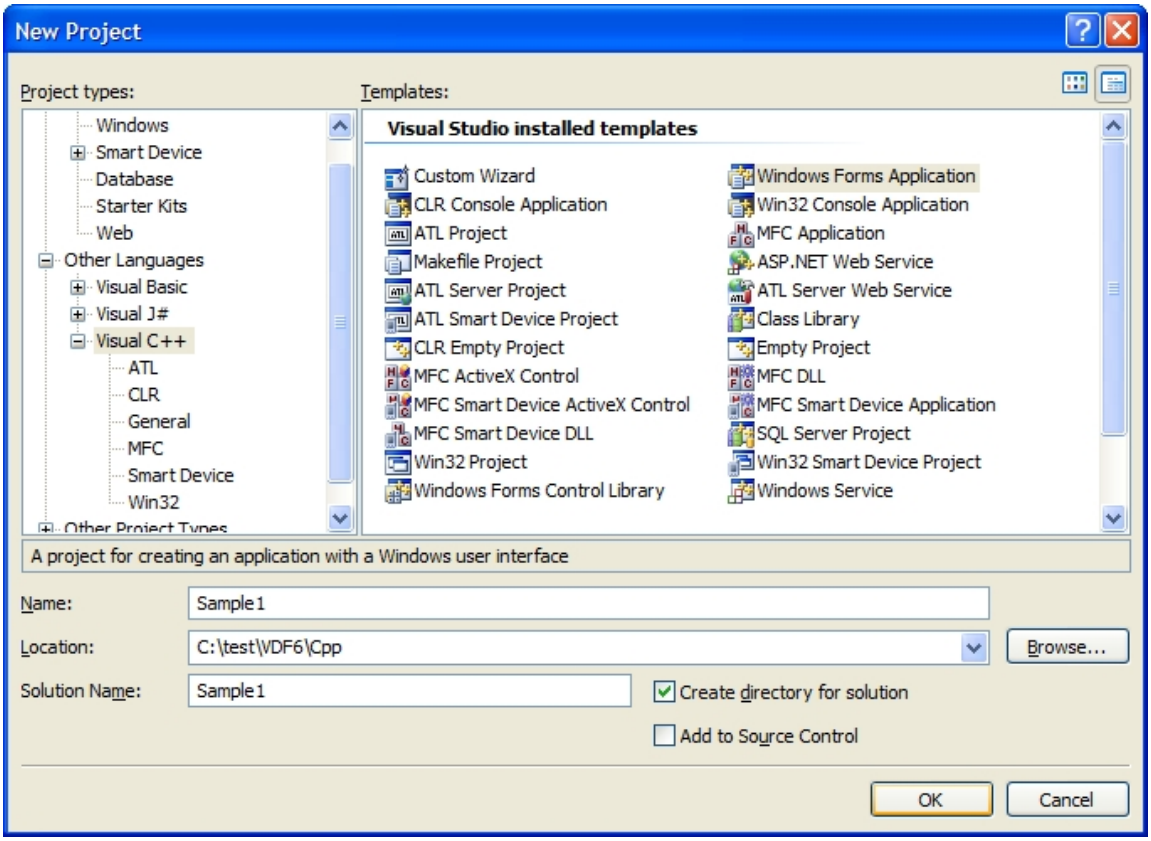
```
private void btOpen_Click(object sender, EventArgs e)
```

```
{
    object ret = vdFC.BaseControl.ActiveDocument.GetOpenFileNameDlg(0, "", 0);
    if (ret == null) return;
    string fname = (string)ret;
    bool success = vdFC.BaseControl.ActiveDocument.Open(fname);
    if (success) vdFC.BaseControl.ActiveDocument.Redraw(true);
}

private void btSave_Click(object sender, EventArgs e)
{
    vdFC.BaseControl.ActiveDocument.SaveAs("c:\\temp\\test.vdcl");
}

private void btNew_Click(object sender, EventArgs e)
{
    vdFC.BaseControl.ActiveDocument.New();
}

private void btZoom_Click(object sender, EventArgs e)
{
    vdFC.BaseControl.ActiveDocument.CommandAction.Zoom("E", 0, 0);
}
```

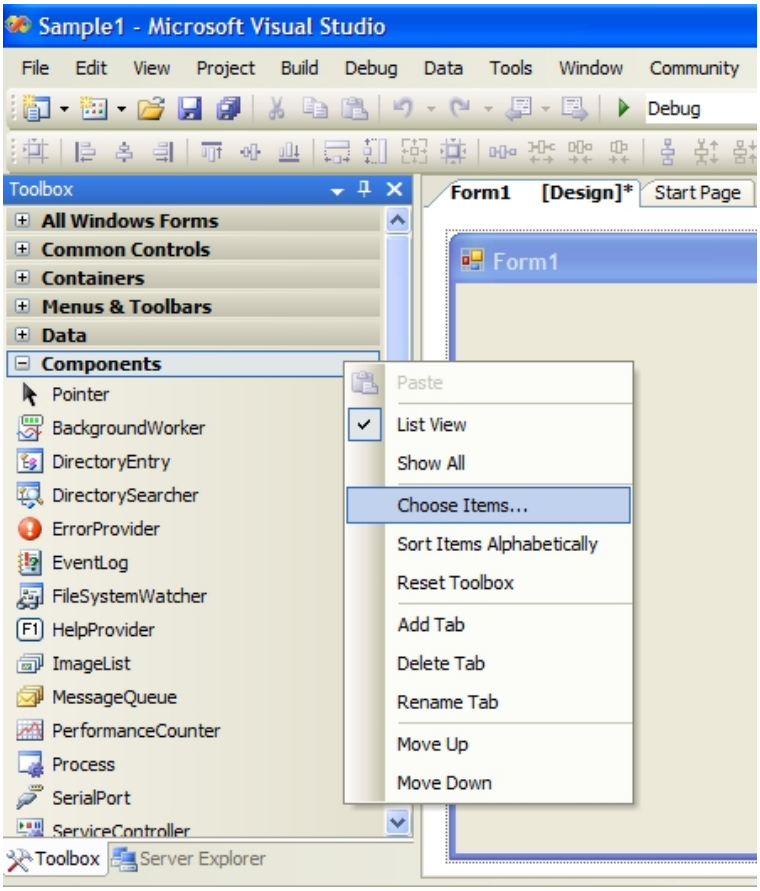
Getting Started - C++ 2005 - vdFramedControl

In order to add the VDF vdFramedControl in Visual C++ 2005 the steps 1 to 5 must be followed:

1) Open the Visual C++ 2005 IDE (See NOTE in 2nd page) and choose to create a new Windows project as "Windows Forms Application"

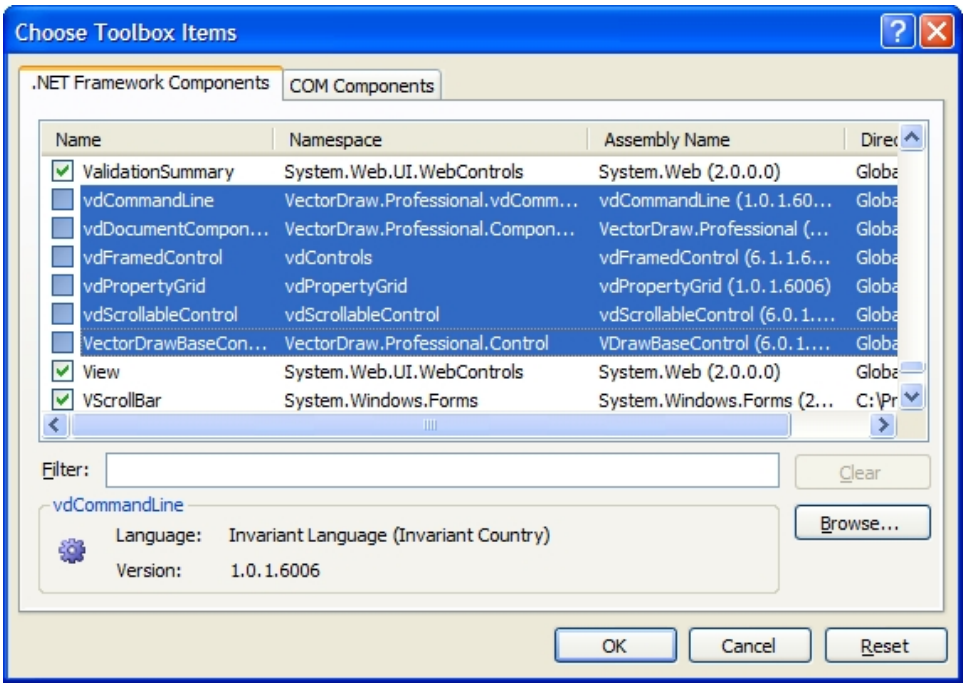


2) Right click on the toolbox and in the menu that appears choose "Choose Items..."



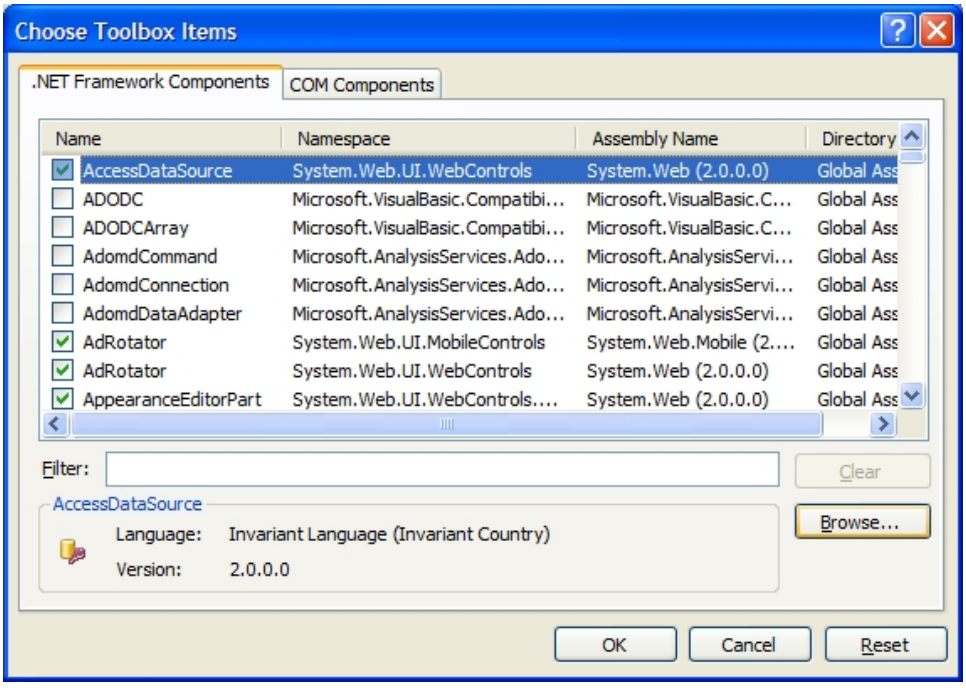
3) You can add the vdFramedControl with 2 different ways:

A) In the "C hoose Toolbox Items" scroll down to the "vdFramedControl" and check it.

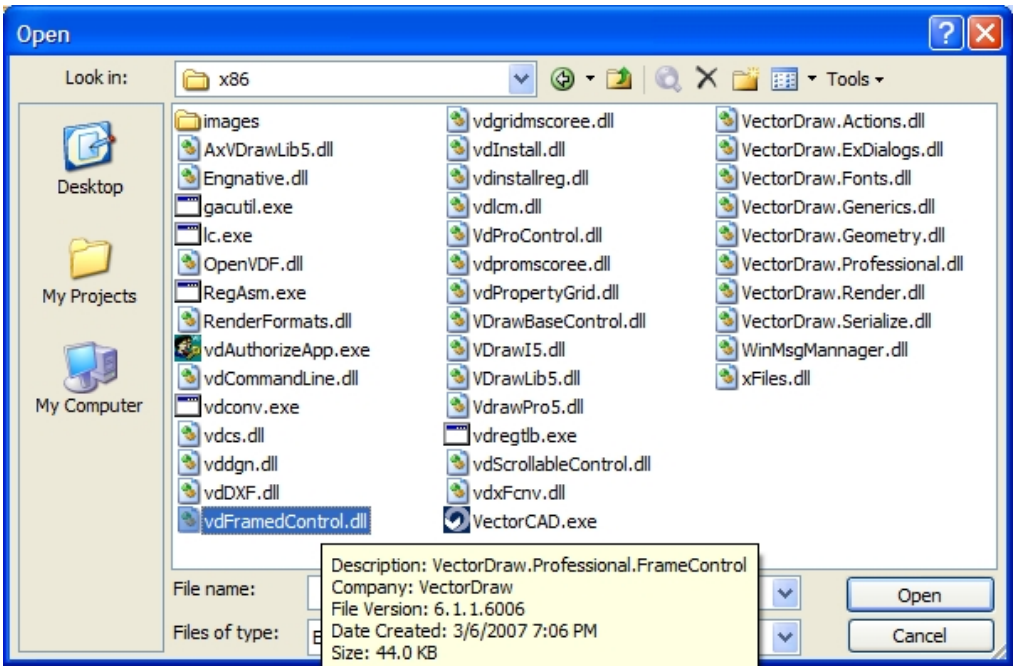


- OR -

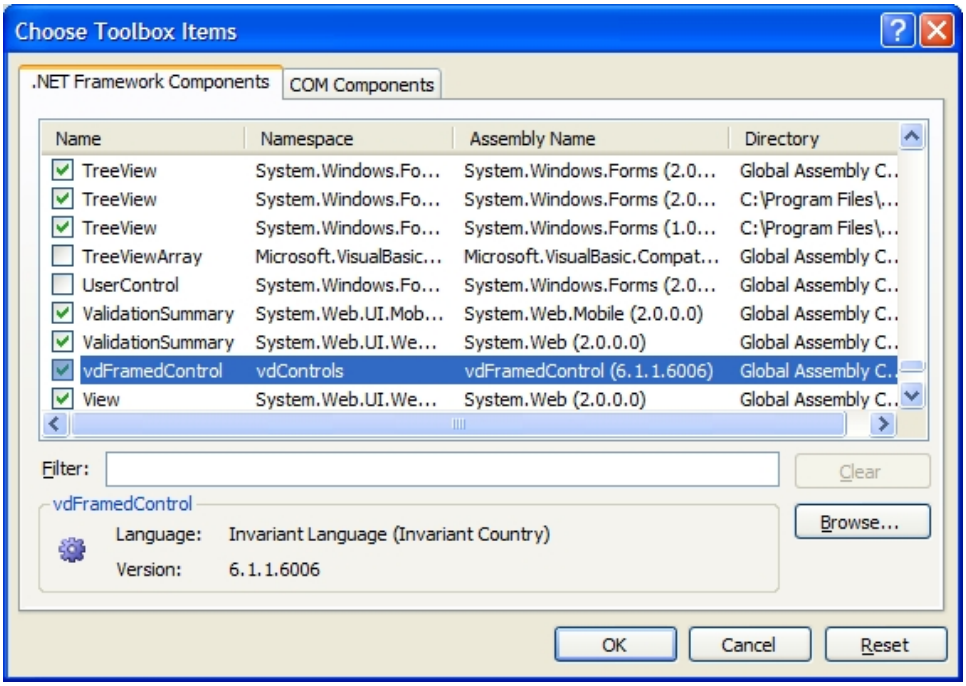
B 1)In the "Choose Toolbox Items" window and click "Browse..." button.



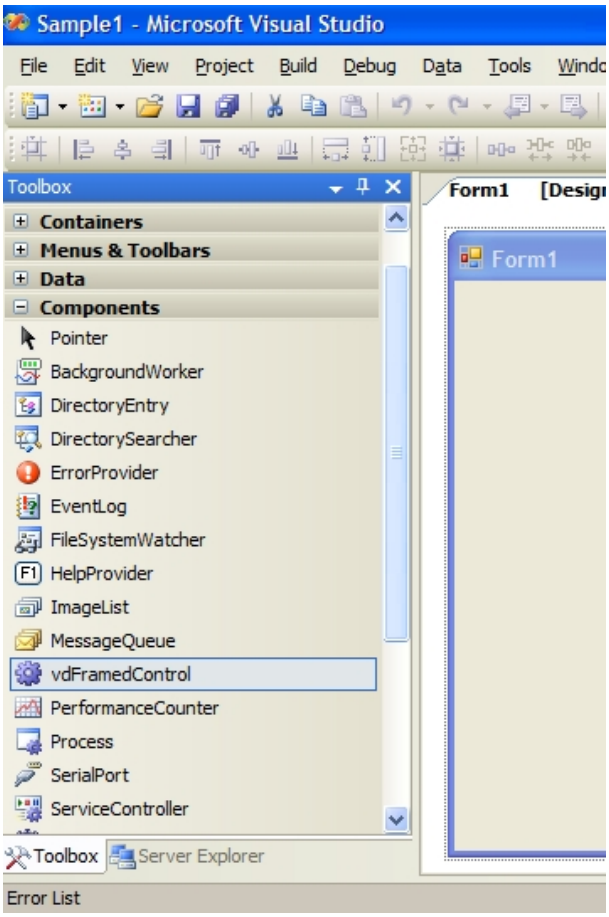
B 2) Go to the folder where the VectorDraw Developer Framework is installed (must be like "*C:\Program Files\VectorDraw\Common\6xxx*") and click on (select) the vdFramedControl.dll and then click "Open" button.



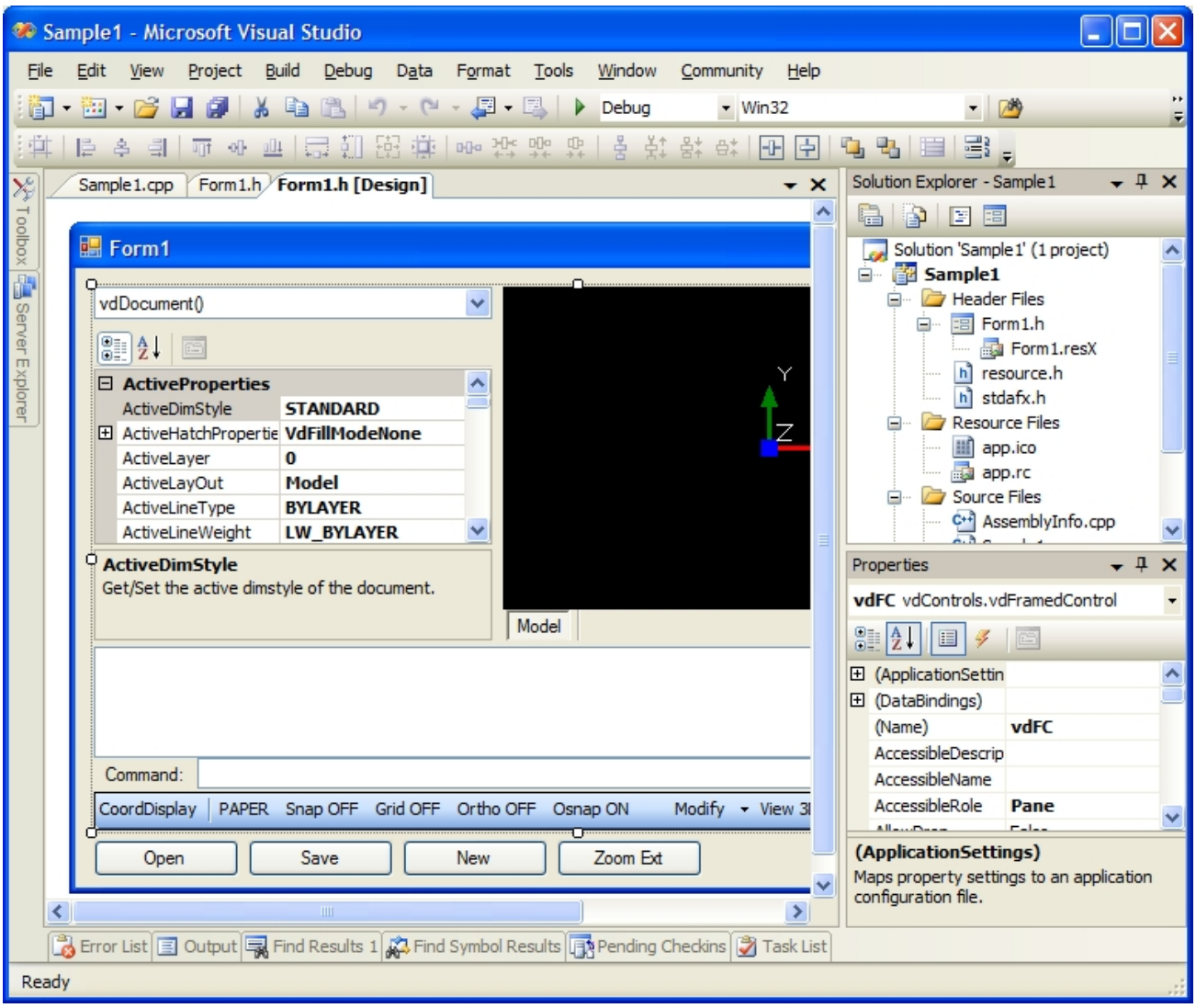
4) You should see it checked in the "Choose Toolbox Items". Clikc the "OK" button.



5) The vdFramedControl component should be in the Toolbox.



6) Add it to the form and also 4 buttons that will do some simple functions like Open/Save a drawing, Create a new drawing and Zoom Extends.



7) The code that is added to the buttons click is like below :.

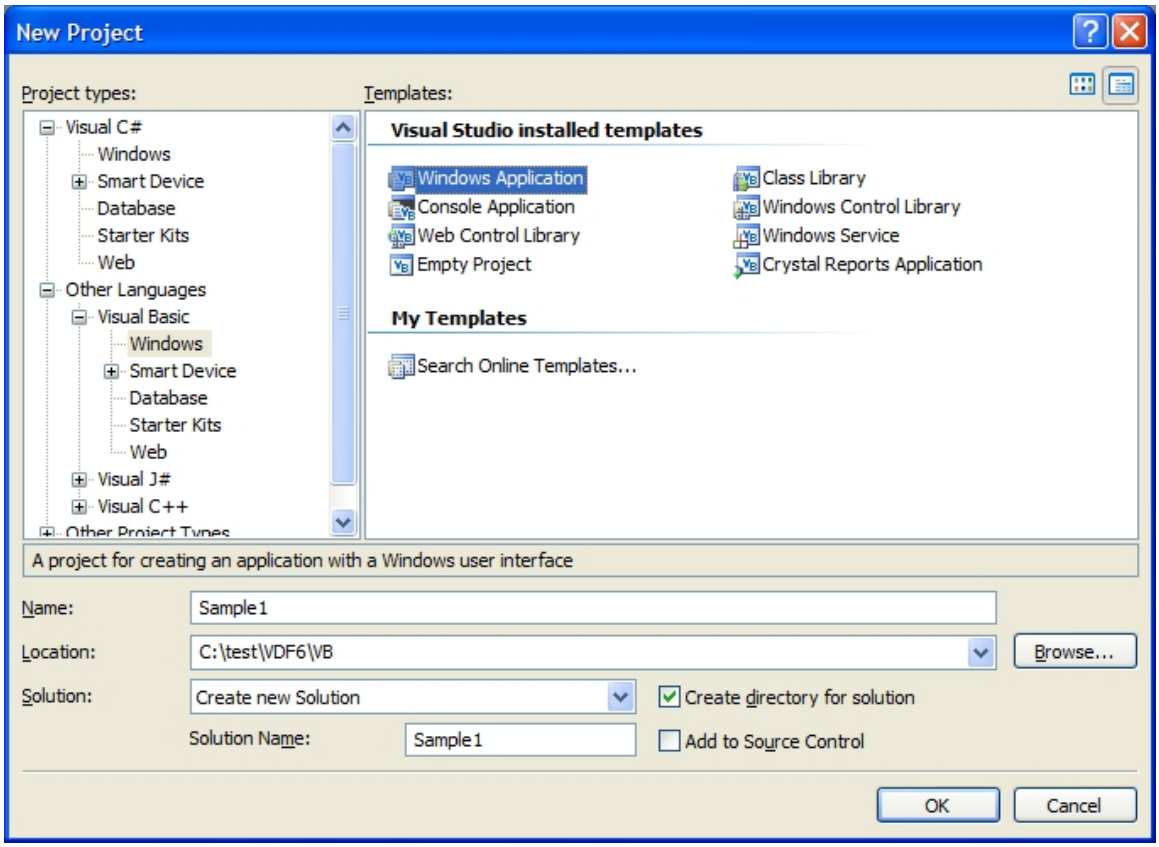

```
private: System::Void btOpen_Click(System::Object^ sender, System::EventArgs^ e) {
    VectorDraw::Professional::Control::VectorDrawBaseControl ^ctl = vdFC->BaseControl;
    System::Object^ ret = ctl->ActiveDocument->GetOpenFileNameDlg(0,"",0);
    if (ret == NULL) return;
    System::String ^fname = (System::String^)ret;
    bool success = ctl->ActiveDocument->Open(fname);
    if (success) ctl->ActiveDocument->Redraw(true);
}
private: System::Void btSave_Click(System::Object^ sender, System::EventArgs^ e) {
    VectorDraw::Professional::Control::VectorDrawBaseControl ^ctl = vdFC->BaseControl;
    ctl->ActiveDocument->SaveAs("c:\\temp\\test.vdcl");
}
private: System::Void btNew_Click(System::Object^ sender, System::EventArgs^ e) {
    VectorDraw::Professional::Control::VectorDrawBaseControl ^ctl = vdFC->BaseControl;
    ctl->ActiveDocument->New();
}
private: System::Void btZoom_Click(System::Object^ sender, System::EventArgs^ e) {
    VectorDraw::Professional::Control::VectorDrawBaseControl ^ctl = vdFC->BaseControl;
    ctl->ActiveDocument->CommandAction->Zoom("E",0,0);
}
```

Note: you may also need to add this #include directive : `#include <stdlib.h>`

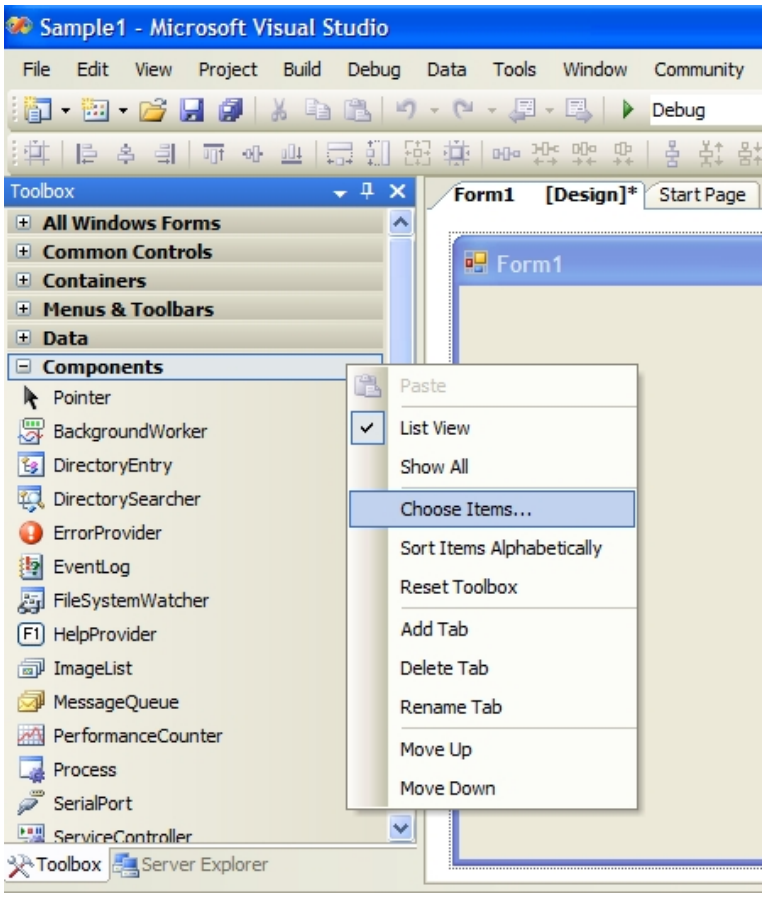
Getting Started - VisualBasic 2005 - vdFramedControl

In order to add the VDF vdFramedControl in Visual Basic 2005 the steps 1 to 5 must be followed:

1) Open the Visual Basic 2005 IDE (See NOTE **in** 2nd page) and choose to create a new Windows project as "Windows Application"

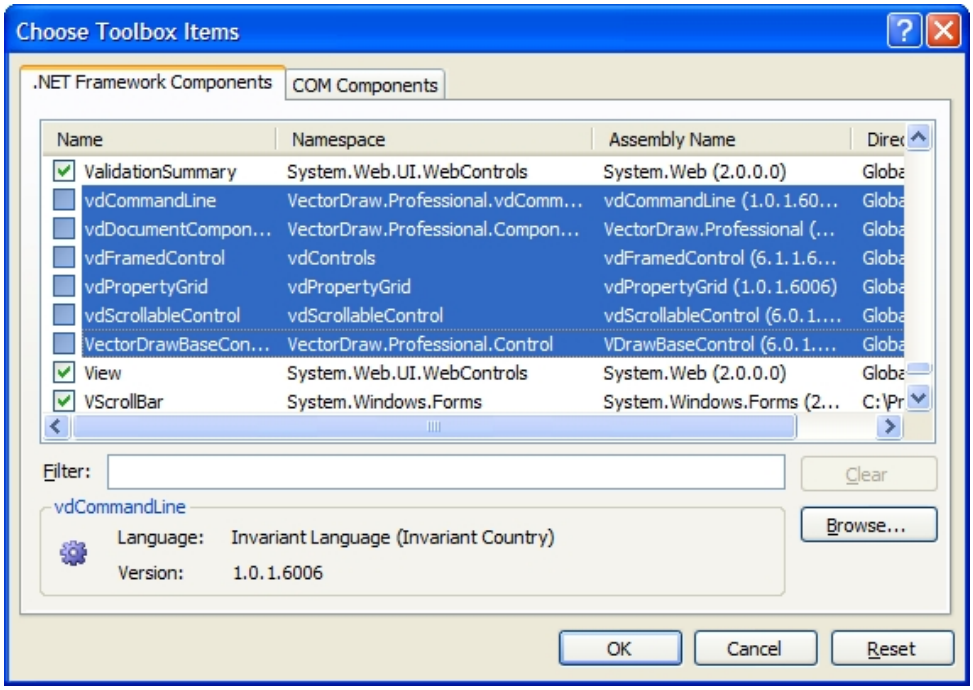


2) Right click on the toolbox and in the menu that appears choose "Choose Items..."



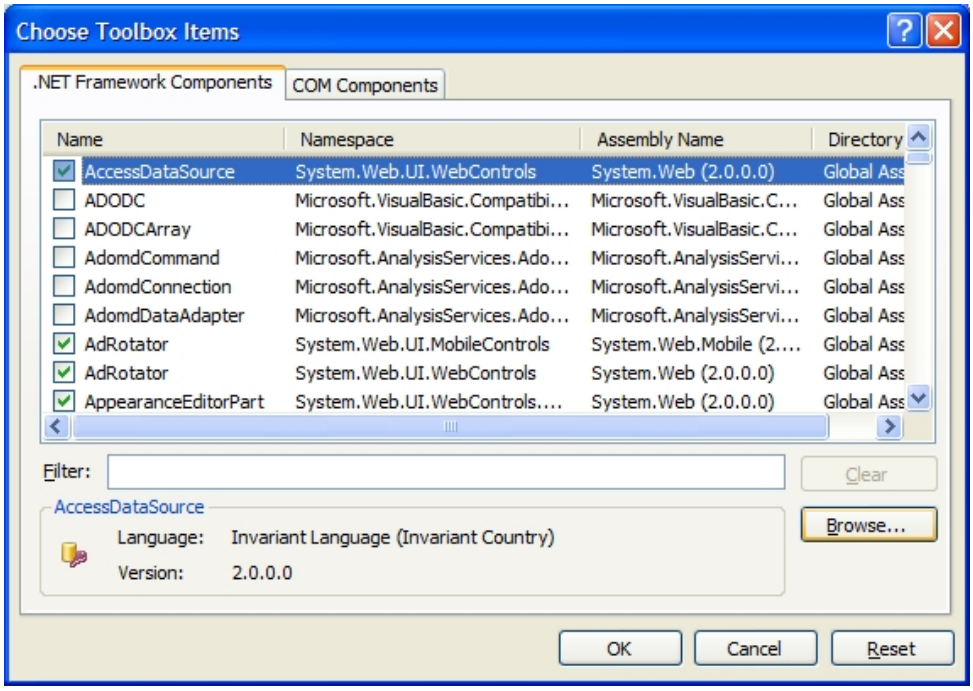
3) You can add the vdFramedControl with 2 different ways:

A) In the "C hoose Toolbox Items" scroll down to the "vdFramedControl" and check it.

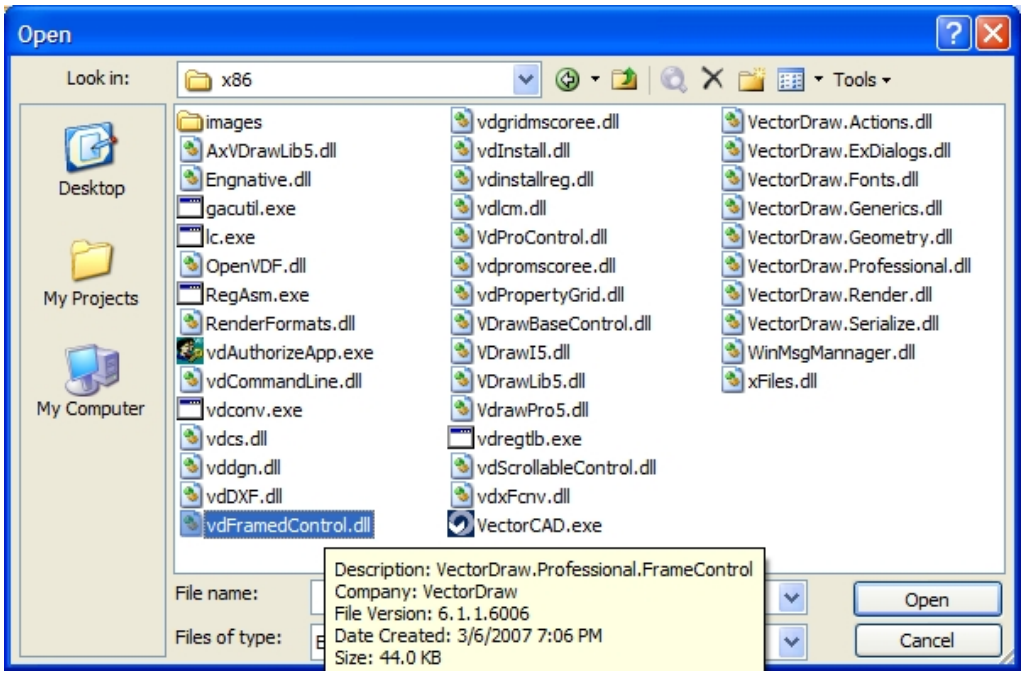


- OR -

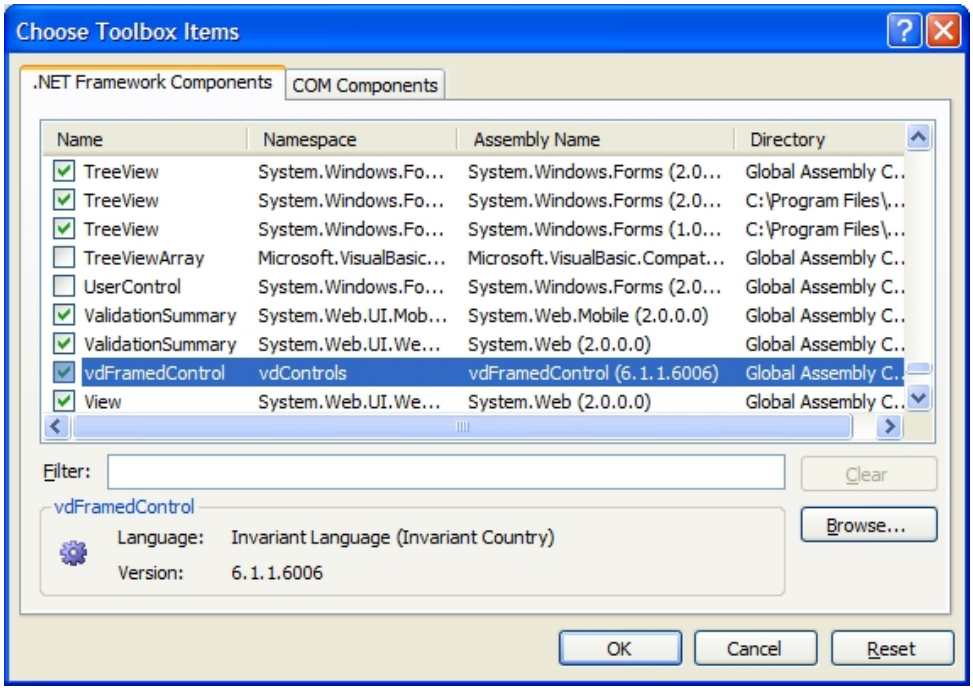
B 1)In the "Choose Toolbox Items" window and click "Browse..." button.



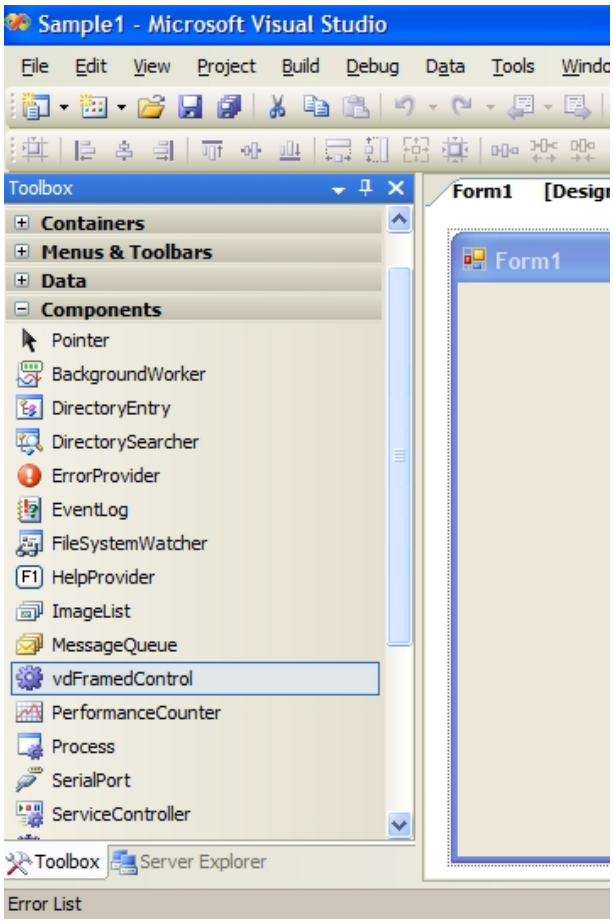
B 2) Go to the folder where the VectorDraw Developer Framework is installed (must be like "*C:\Program Files\VectorDraw\Common\6xxx*") and click on (select) the vdFramedControl.dll and then click "Open" button.



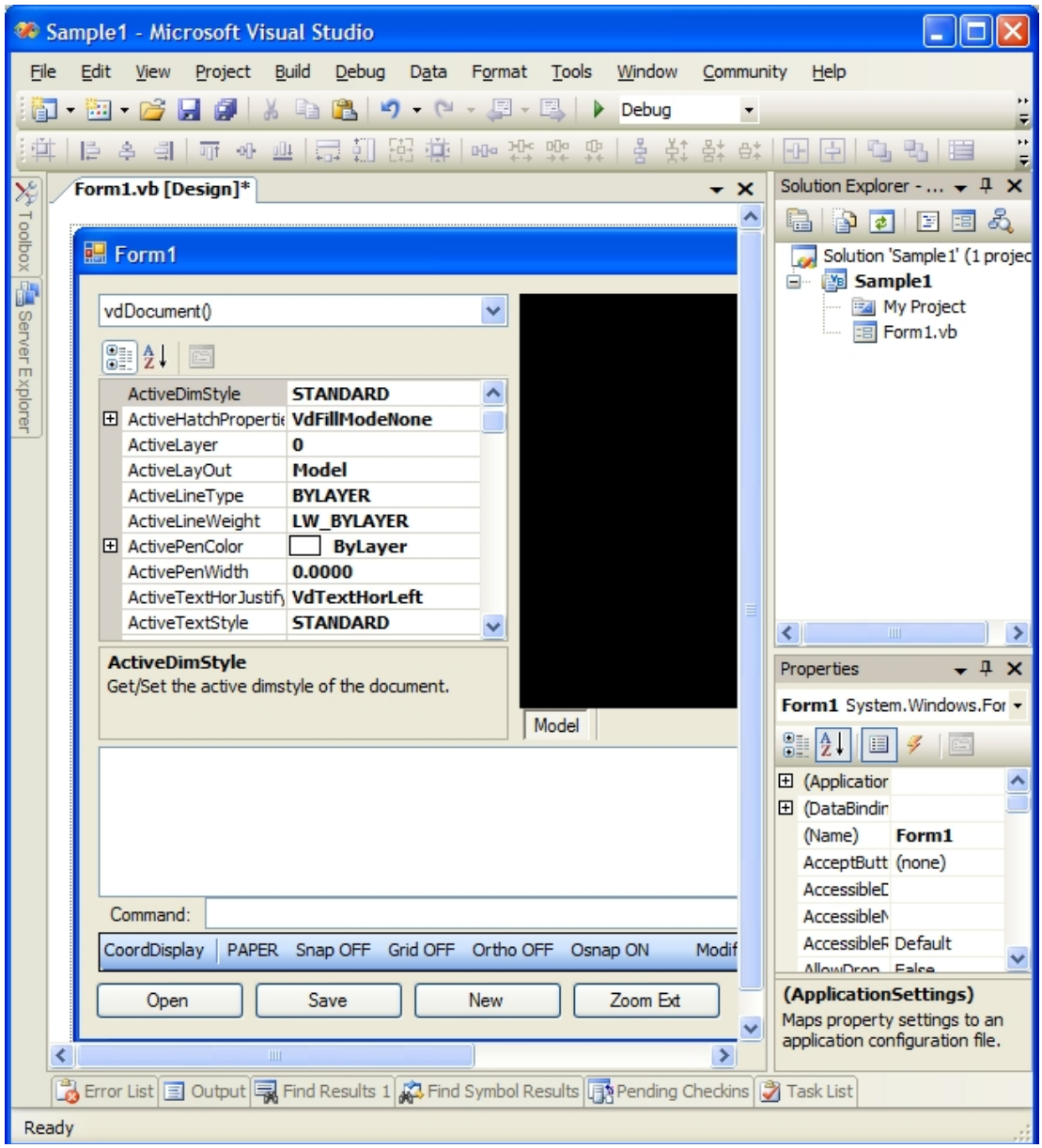
4) You should see it checked in the "Choose Toolbox Items". Clikc the "OK" button.



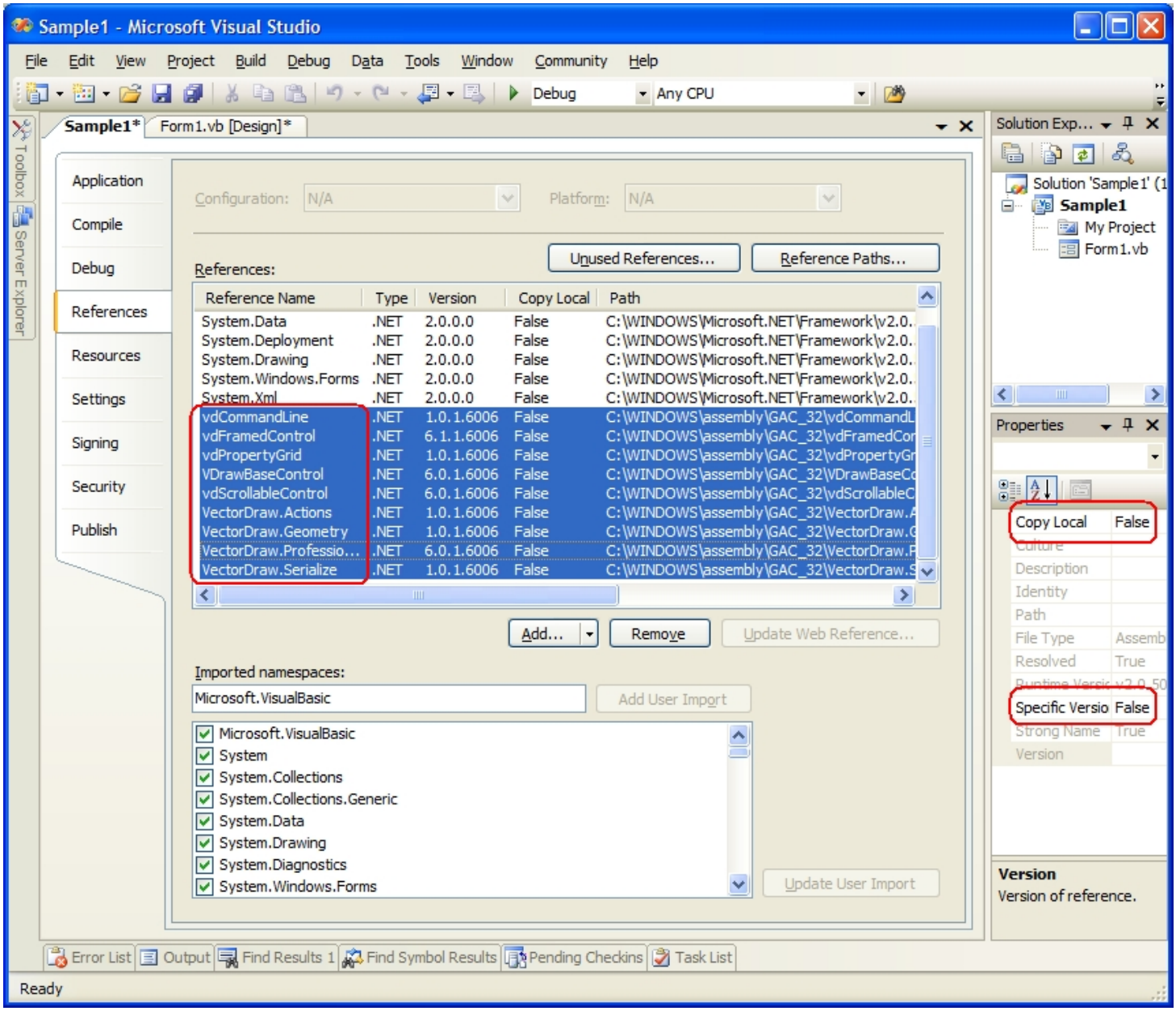
5) The vdFramedControl component should be in the Toolbox.



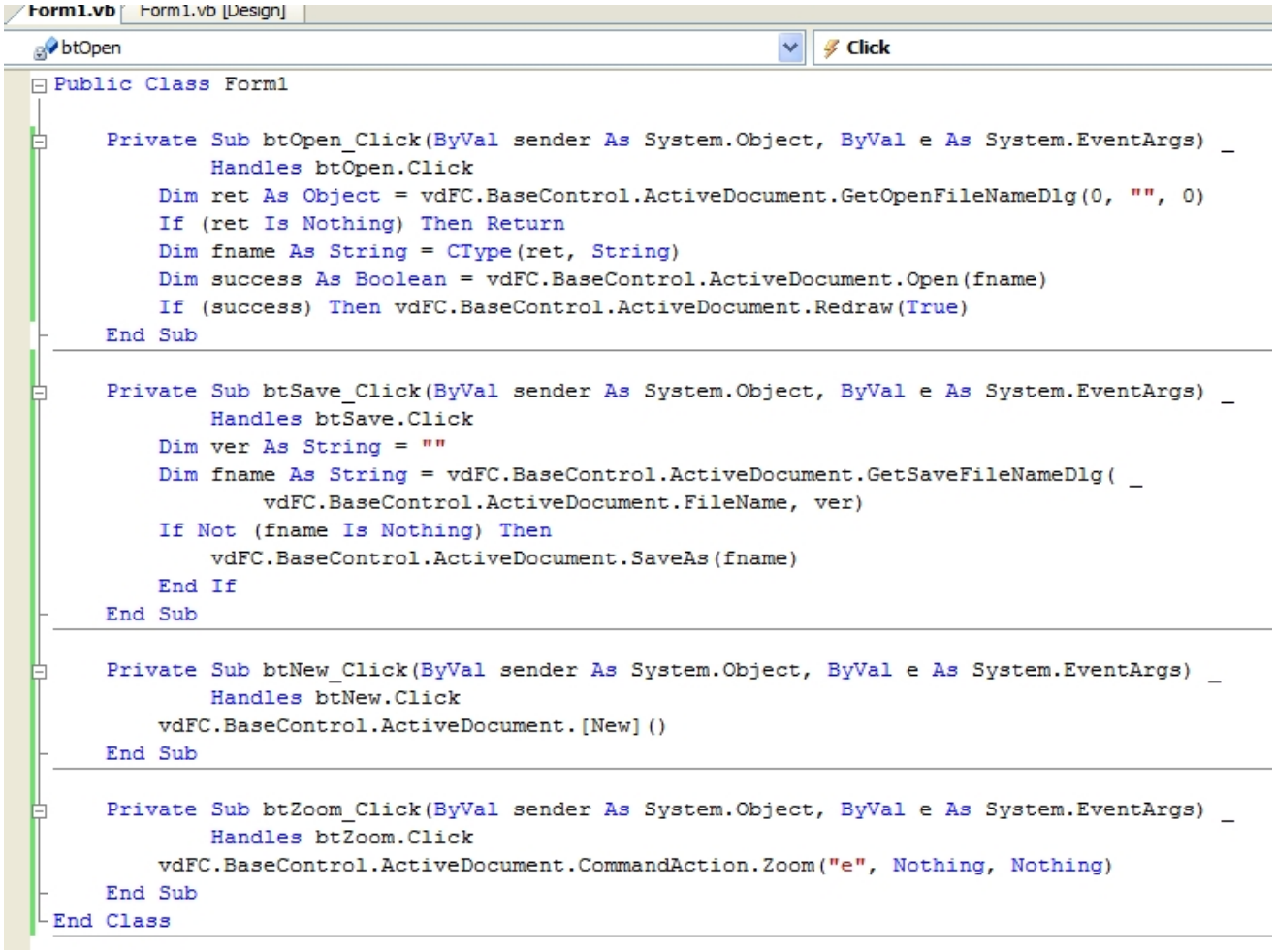
6) Add it to the form and also 4 buttons that will do some simple functions like Open/Save a drawing, Create a new drawing and Zoom Extends.



7) Go to the Project properties, select the VDF references and in their properties set Copy Local to False and Specific Version to False too.



8) The code that is added to the buttons click is like below :




```
Private Sub btOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
          Handles btOpen.Click

    Dim ret As Object = vdFC.BaseControl.ActiveDocument.GetOpenFileNameDlg(0, "", 0)
   If (ret Is Nothing) Then Return
   Dim fname As String = CType(ret, String)
   Dim success As Boolean = vdFC.BaseControl.ActiveDocument.Open(fname)
   If (success) Then vdFC.BaseControl.ActiveDocument.Redraw(True)
End Sub

Private Sub btSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
          Handles btSave.Click

    Dim ver As String = ""
   Dim fname As String = vdFC.BaseControl.ActiveDocument.GetSaveFileNameDlg( _
    vdFC.BaseControl.ActiveDocument.FileName, ver)
   If Not (fname Is Nothing) Then
   vdFC.BaseControl.ActiveDocument.SaveAs(fname)
End If
End Sub

Private Sub btNew_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
          Handles btNew.Click
    vdFC.BaseControl.ActiveDocument.[New]()
End Sub

Private Sub btZoom_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
          Handles btZoom.Click

    vdFC.BaseControl.ActiveDocument.CommandAction.Zoom("e", Nothing, Nothing)
End Sub
```

Register VDF in you Development machine

Licensing the VDF version 6 in your machine

If you are a customer and not an evaluator then in order to be able to use the VDF libraries (like vdFramedControl) and VectorDraw Professional 6.x Wrapper (vdraw.ocx) in your development platform you should license/register it. This is done by using your Serial number of the version6 that is provided with the purchase of the libraries and the vdAuthorizeApp.exe utility. You machine should be also connected to the internet.

Run the vdAuthorizeApp.exe utility as administrator or as a user with administrative rights in your Development machine and type your VDF serial (case sensitive) and click the "Authorize" button. Check the "Result" window.

If you like to install and authorize the libraries in another machine like a presentation laptop PC then you should Unauthorize your development machine and Authorize it again on the laptop. You cannot have multiple machines authorized simultaneously with the same license. If you don't have access to the Internet or you having troubles with this procedure then contact our sales department including the result you get.

NOTE: In Windows Vista/Seven/Server2008 oper.systems, due to UAC, you should ALWAYS run the vdAuthorizeApp.exe with right-click and "Run as Administrator" option.

Getting Started - ActionUtility - vdFramedControl

The ActionUtility Object can be used in order to ask the user to input data in the application, like a point, a rectangle, a distance etc.

For example if you like the user to input a point in the drawing the you can call a code like :

```
gPoint userpoint;
vdFC.BaseControl.ActiveDocument.Prompt("Select a Point:");
//The user can either click a point or type at the command line a point like
5,5,2
StatusCode ret =
vdFC.BaseControl.ActiveDocument.ActionUtility.getUserPoint(out userpoint);
vdFC.BaseControl.ActiveDocument.Prompt(null);
if (ret == StatusCode.Success){
   MessageBox.Show("The user selected: x:" + userpoint.x.ToString() + " y:" +
userpoint.y.ToString() + " z:" + userpoint.z.ToString() + " In UCS(user
coordinate system)");
}
```

The above sample code can be changed so the user is asked for a second point and then create a rectangle from these two points. Like :

```
vdFC.BaseControl.ActiveDocument.New();
gPoint userpoint;
vdFC.BaseControl.ActiveDocument.Prompt("Select a Point:");
//The user can either click a point or type at the command line a point like
5,5,2
StatusCode ret =
vdFC.BaseControl.ActiveDocument.ActionUtility.getUserPoint(out userpoint);
vdFC.BaseControl.ActiveDocument.Prompt(null);
if (ret == StatusCode.Success){
    MessageBox.Show("The user selected: x:" + userpoint.x.ToString() + " y:"
+ userpoint.y.ToString() + " z:" + userpoint.z.ToString() + " In UCS(user
coordinate system)");
}
vdFC.BaseControl.ActiveDocument.Prompt("Other corner:");
//The user can either click a point or type at the command line a point like
5,5,2
object ret2 =
vdFC.BaseControl.ActiveDocument.ActionUtility.getUserRect(userpoint);
vdFC.BaseControl.ActiveDocument.Prompt(null);
Vector v = ret2 as Vector;
if (v != null) {
   double angle = v.x;
   double width = v.y;
   double height = v.z;
   //Calculate the point the user clicked.
    //Use polar command to find the bottom right point moving width distance
from the initial point.
   gPoint userpoint2 = userpoint.Polar(0.0, width);
   //Use the polar again to go up height distance to find the upper right
point.
    userpoint2 = userpoint2.Polar(VectorDraw.Geometry.Globals.HALF_PI,
height);
   MessageBox.Show("The user selected 2nd point : x:" +
userpoint2.x.ToString() + " y:" + userpoint2.y.ToString() + " z:" +
userpoint2.z.ToString() + " In UCS(user coordinate system)");
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdRect(userpoint,
userpoint2);
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "Utility Class" and "ActionUtility Property".

Getting Started - Blocks - vdFramedControl

A block is a collection of objects you can associate together to form a single object, or block definition. You can insert, scale, and rotate a block in a drawing (this is called vdInsert). You can explode a block into its component objects, modify them, and redefine the block definition. The use of Blocks streamline the drawing process. For example, you can use blocks to build a standard library of frequently used symbols, components, or standard parts.

You can insert the same block numerous times instead of re-creating the drawing elements each time. Revise drawings efficiently by inserting, relocating, and copying blocks as components rather than individual geometric objects. Save disk space by storing all references to the same block as one block definition in the drawing database. When you insert a block in your drawing, you are creating a block instance. Each time you insert a block instance, you assign a scale factor and rotation angle to the inserted block. You can also scale a block instance using different values in any coordinate (X, Y, Z) direction. Blocks make it possible for you to organize your drawing tasks in a systematic way, so that you can set up, redesign, and sort the objects in your drawings and the information associated with them.

Below is a sample code that will add an new Block (vdBlock) object in the drawing named "CustomBlock" and also an existing drawing (drawing is named vdblk.vdml) as "CustomBlock2" :

```
private void AddBlockItems()
{
    //We create a block object and initialize it's default properties.
    VectorDraw.Professional.vdPrimaries.vdBlock blk = new
VectorDraw.Professional.vdPrimaries.vdBlock();
    blk.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    blk.setDocumentDefaults();
    //We add some entities to the block.
    VectorDraw.Professional.vdFigures.vdPolyline poly = new
VectorDraw.Professional.vdFigures.vdPolyline();
    poly.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    poly.setDocumentDefaults();
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint());
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(1.0,0.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(2.0,1.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(4.0,-1.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(6.0,1.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(8.0, -1.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(10.0, 1.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(11.0, 0.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(12.0, 0.0));
    blk.Entities.AddItem(poly);
    blk.Origin = new VectorDraw.Geometry.gPoint(6.0, 0.0);
    blk.Name = "CustomBlock";
    VectorDraw.Professional.vdFigures.vdAttribDef attribdef = new
VectorDraw.Professional.vdFigures.vdAttribDef();
    attribdef.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    attribdef.setDocumentDefaults();
    attribdef.InsertionPoint = new VectorDraw.Geometry.gPoint(5.0,1.2);
    //Name of the attribute used to be found when using the block.
    attribdef.TagString = "resistance";
    //Default value used when inserted the block from the block's dialog.
    attribdef.ValueString = "1W";
    blk.Entities.AddItem(attribdef);
    //And then we add this block to the document's blocks collection
    vdFC.BaseControl.ActiveDocument.Blocks.AddItem(blk);
    //We will also add a block from a precreated file.
    string path = Application.ExecutablePath.Substring(0,
Application.ExecutablePath.LastIndexOf('\\')) + "\\..\\..\\vdblk.vdml";
    string path1 = "";
    VectorDraw.Professional.vdPrimaries.vdBlock blk2;
    if (vdFC.BaseControl.ActiveDocument.FindFile(path, out path1))
    {
        blk2 = vdFC.BaseControl.ActiveDocument.Blocks.AddFromFile(path,
```

```
false);
        //We check if a block with name CustomBlock2 already exists and if not
we change the name of the block to CustomBlock2.
        if (vdFC.BaseControl.ActiveDocument.Blocks.FindName ("CustomBlock2")
== null) blk2.Name = "CustomBlock2";
    }
}
```

You can see the existing block in a drawing and also insert them using the InsertBlockDIalog:

```
private void OpenBlocksForm()
{
    VectorDraw.Professional.Dialogs.InsertBlockDialog form =
VectorDraw.Professional.Dialogs.InsertBlockDialog.Show(vdFC.BaseControl.ActiveDocument,
vdFC.BaseControl.ActiveControl, false, "");
    if (form.DialogResult == DialogResult.OK)
    {
         if (form.insertionPoint is double[])
        {
            double[] pt = form.insertionPoint as double[];
            form.insertionPoint = new VectorDraw.Geometry.gPoint
(pt[0],pt[1],pt[2]);
        }

        if(form.scales is string)

vdFC.BaseControl.ActiveDocument.CommandAction.CmdInsert(form.blockname,
form.insertionPoint, form.scales, form.scales, form.rotationAngle);
        else
        {
            double[] scales = form.scales as double[];

vdFC.BaseControl.ActiveDocument.CommandAction.CmdInsert(form.blockname,
form.insertionPoint, scales[0], scales[1], form.rotationAngle);
        }
    }
    //The dialog can also be called using the CmdInsertBlockDialog command.
    //vdFC.BaseControl.ActiveDocument.CommandAction.CmdInsertBlockDialog();
}
```

You can also insert the blocks created in AddBlockItems() using a code like :

```
private void AddInsertObjects()
{
    //We will add 5 instances(inserts) of each block to the
Document with different properties.
    //First we check if the blocks have been already added to the
blocks collection.
    if (vdFC.BaseControl.ActiveDocument.Blocks.FindName("CustomBlock") !=
null)
    {
        VectorDraw.Professional.vdFigures.vdInsert ins;
        for (short i = 0; i < 5; i++)
        {
            ins = new VectorDraw.Professional.vdFigures.vdInsert();
            ins.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
            ins.setDocumentDefaults();
            ins.Block =
vdFC.BaseControl.ActiveDocument.Blocks.FindName("CustomBlock");
            ins.InsertionPoint = new VectorDraw.Geometry.gPoint(i * 10, 10);
            ins.PenColor.ColorIndex = i;
            ins.Rotation = VectorDraw.Geometry.Globals.HALF_PI/2.0 * i;
            //This will create the necessary vdAtrib objects to the
```

```
insert.
            ins.CreateDefaultAttributes();
            //Now we will change the value of the attribute
            if (ins.Attributes.Count == 1) ins.Attributes[0].ValueString =
i.ToString() + "W";
            //And we add the entities to the Model Layout.
            vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(ins);
            ins = new VectorDraw.Professional.vdFigures.vdInsert();
            ins.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
            ins.setDocumentDefaults();
            ins.Block =
vdFC.BaseControl.ActiveDocument.Blocks.FindName("CustomBlock2");
            ins.InsertionPoint = new VectorDraw.Geometry.gPoint(i * 10, -
10);
            ins.PenColor.ColorIndex = (short)(i*40+10);
            ins.Xscale = 2.0;
            ins.Yscale = 2.0;
            ins.Rotation = -VectorDraw.Geometry.Globals.HALF_PI * i;
            //Since the active Layout is the model adding this
insert to the Model or the ActiveLayout is the same thing.

vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(ins);
        }
         //Zoom extends and redraw to see the changes.
        vdFC.BaseControl.ActiveDocument.ActiveLayOut.ZoomExtents();
        vdFC.BaseControl.ActiveDocument.Redraw(true);
    }
    else        MessageBox.Show("The custom block was not found");
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdBlock Class" and "vdBlocks Class".

Getting Started - Colors & Palette - vdFramedControl

Graphical objects in VectorDraw can have a specific pencolor. This color can be an RGB (Red Green Blue) value or a specific color index from the VectorDraw pallete. The color of an object can also take some specific values that can be "Color by Layer" or "Color by Block" where the object takes the color specified by the layer that is or the block. Some object may be filled (like circles or polylines) and the fill color of these objects might be transparent (alpha blending).

See the following code:

```
private void AddSomeEntities()
{
    vdCircle circ = new vdCircle();
    vd.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(circ);
    circ.setUnRegisterDocument(vd.BaseControl.ActiveDocument); circ.setDocumentDefaults();
    circ.PenColor.SystemColor = Color.Red; // Using system color
    circ.Center = new gPoint(5, 5);
    circ.Radius = 2;

    circ = new vdCircle();
    vd.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(circ);
    circ.setUnRegisterDocument(vd.BaseControl.ActiveDocument); circ.setDocumentDefaults();
    circ.PenColor.ColorIndex = 1; // Using pallete's color
    circ.Center = new gPoint(10, 5);
    circ.Radius = 2;

    circ = new vdCircle();
    vd.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(circ);
    circ.setUnRegisterDocument(vd.BaseControl.ActiveDocument); circ.setDocumentDefaults();
    byte red = 10; byte green = 30; byte blue = 100;
    circ.PenColor.TrueColor = (uint)(red + 256 * green + 256 * 256 * blue); //Using
RGB color
    circ.Center = new gPoint(15, 5);
    circ.Radius = 2;

    circ = new vdCircle();
    vd.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(circ);
    circ.setDocumentDefaults();
    circ.PenColor.Red = 50; circ.PenColor.Green = 150; circ.PenColor.Blue = 250;
//Using specific Red, Green and Blue values
    circ.Center = new gPoint(20, 5);
    circ.Radius = 2;
    vd.BaseControl.ActiveDocument.ActiveLayOut.ZoomWindow(new gPoint(), new
gPoint(20, 15));
    vd.BaseControl.ActiveDocument.Redraw(true);
}

private void AddTransparent()
{
    vdCircle circ = new vdCircle();
    vd.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(circ);
    vd.BaseControl.ActiveDocument.ActiveLayer.PenColor.ColorIndex = 4;
    //Set the current, active layers color to blue. All antities in this laye with
pencolor bylayer will be blue.
    circ.setUnRegisterDocument(vd.BaseControl.ActiveDocument); circ.setDocumentDefaults();
    circ.PenColor.ByLayer = true; // Color BYLAYER
    circ.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    circ.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModeSolid;
    circ.HatchProperties.FillColor.SystemColor = Color.Green;
    circ.HatchProperties.FillColor.AlphaBlending = 100; // Transparent Fill.
    circ.Center = new gPoint(12.5, 5);
    circ.Radius = 6;
    vd.BaseControl.ActiveDocument.ActiveLayOut.ZoomWindow(new gPoint(), new
gPoint(25, 15));
    vd.BaseControl.ActiveDocument.Redraw(true);
}
```

You can also change one color calling the frmColor dialog, like:

```
private void ChangeActiveLayersColor()
{
    VectorDraw.Professional.Dialogs.frmColor dialog = new
VectorDraw.Professional.Dialogs.frmColor(vd.BaseControl.ActiveDocument.ActiveLayer.PenColor,
false);
    dialog.ShowDialog(this);
    vd.BaseControl.ActiveDocument.Redraw(true);
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdColor Class".

Getting Started - CommandAction - vdFramedControl

The CommandAction object offer to the developer a variety of tools in order to do some user specific actions. For example developer may wants to offer to the end user the ability to add lines with points that the user selects. In this case the only code that is required by the developer is to call cmdLine with "USER" parameters, like:

```
vdFramedControl1.BaseControl.ActiveDocument.CommandAction.CmdLine(null);
//parameter null means that the user will pick the required point- distance
or everything that is required for the command.
```

There are also these commands that allow the user to add entities like *CmdXLine, CmdRay, CmdPoint, CmdArc, CmdCircle, CmdEllipse, CmdPolyLine, CmdText, CmdRect, CmdImage, CmdDim, CmdBox3d, CmdSphere, CmdCone, Cmd3DMesh, Cmd3dFace and CmdPlineToMesh*.

CommandAction has functionality to manipulate entities like copying, moving, deleting them etc. Like :

```
vdFramedControl1.BaseControl.ActiveDocument.CommandAction.CmdCopy(null, null,
null);
//This will prompt user to select entities and copy them from one point to
another
```

Developer can use : *CmdRotate, CmdCopy, CmdErase, CmdMove, CmdExplode, CmdMirror, CmdBreak, CmdOffset, CmdExtend, CmdTrim, CmdFillet, CmdStretch, CmdArrayPolar and CmdArrayRectangular, CmdClipCopy, CmdClipPaste, CmdClipCut* for editing/manipulating entities using CommandAction.

Besides the above functions CommandAction has viewing functionality like zooming to the whole document or to a small window, change the 3D view etc. You can zoom to a user selected window like :

```
vdFramedControl1.BaseControl.ActiveDocument.CommandAction.Zoom("w", null,
null);
//This will prompt user to select entities and copy them from one point to
another
```

Developer can use : *Zoom, Pan, View3D, UCS* to change the view of the document.

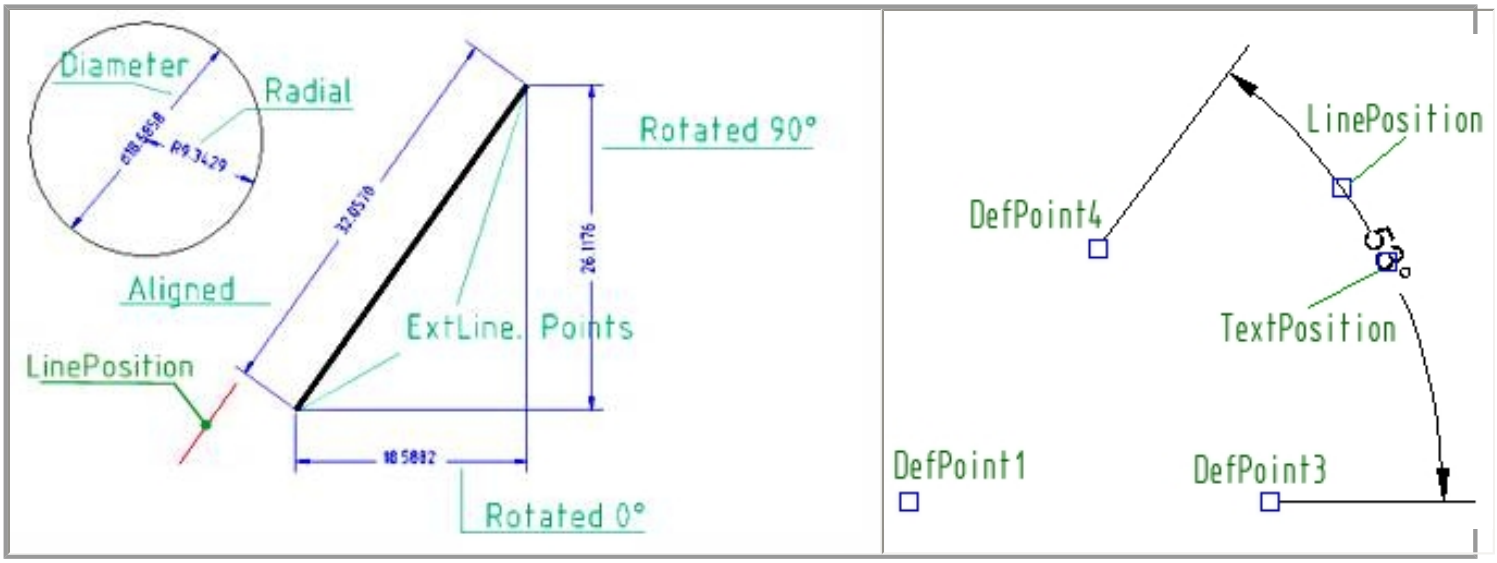
You can see the "Commands" sample of the vdFrameWork samples collection. This sample demonstrates the use of predefined commands that can be used to draw graphic objects like lines, circles etc (cmdLine, cmdCircle etc) and also transform these objects like Copy, Move, Rotate etc.

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdCommandAction Class".

Getting Started - Dimensions - vdFramedControl

Dimensioning is the process of adding measurement annotation to a drawing. User has many ways to dimension objects and many ways to format dimensions. You can create dimensions for a wide variety of object shapes in many different orientations. You can create dimension styles to format dimensions quickly and ensure that dimensions in your drawing conform to industry or project standards.

Dimensions show the measurements of objects, the distances or angles between objects, or the distance of a feature from an origin you specify. User has three basic types of dimensioning: linear, radial, and angular. Dimensions can be horizontal, vertical, aligned, rotated or angular. A linear dimension measuring the distance between two points which is displayed parallel to the points being measured. In aligned dimensions, the dimension line is parallel to the extension line origins. The extension line origins are specified using the DefPoint1 and DefPoint2 properties.



Angular dimensions measure the angle between two lines or three points.

Dimensions use an existing DimStyle in order to display the lines and the texts/numbers of the dimension, see in this article.

You can also add dimensions with code like below (prior to this you should run the code in this article) :

```
private void AddDimensions()
{
    string dimname = "DimStyle1";
    VectorDraw.Professional.vdPrimaries.vdDimstyle style =
vdFC.BaseControl.ActiveDocument.DimStyles.FindName(dimname);
    if (style == null)
    style = vdFC.BaseControl.ActiveDocument.DimStyles.Standard;
    VectorDraw.Professional.vdFigures.vdDimension dim1 = new
VectorDraw.Professional.vdFigures.vdDimension();
    dim1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    dim1.setDocumentDefaults();
    dim1.dimType = VectorDraw.Professional.Constants.VdConstDimType.dim_Rotated;
    dim1.DefPoint1 = new VectorDraw.Geometry.gPoint(5.0, 5 * i);
    dim1.DefPoint2 = new VectorDraw.Geometry.gPoint(10.5, 5 * i+2.0);
    dim1.LinePosition = new VectorDraw.Geometry.gPoint(7.0, 5 * i + 4.0);
    dim1.Rotation = 0.0;
    //We set the dim style that we found before.
    dim1.Style = style;
    //We add the object to the model entities collection
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(dim1);
}
vdFC.BaseControl.ActiveDocument.Model.ZoomExtents();
vdFC.BaseControl.ActiveDocument.Redraw(true);
}
```

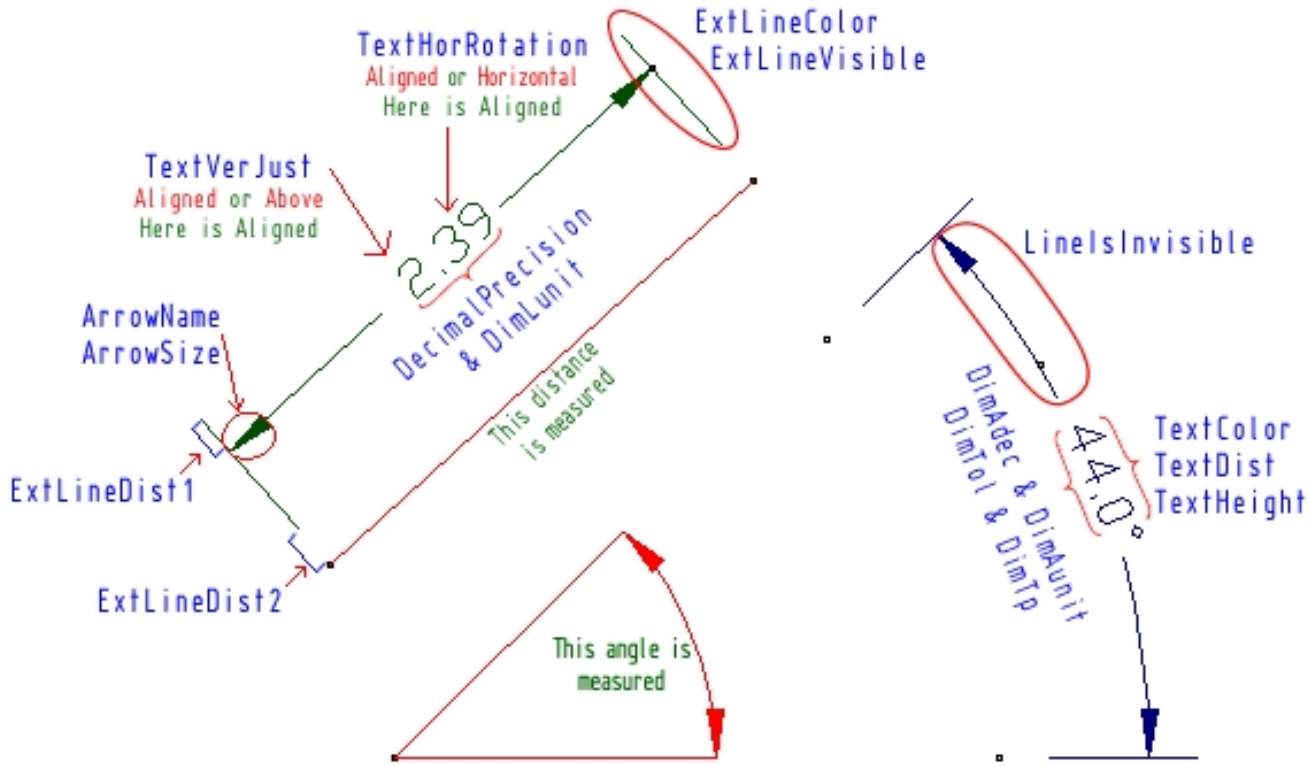
For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdDimension Class".

Getting Started - Document Events - vdFramedControl

Dimension style (vdDimStyle object) is a group of dimension settings that determines the appearance of a dimension. The active dimension style determines the appearance of new dimensions created in the drawing. To change the style of an existing dimension, use the Style property found on the dimension.

When you create a dimension, the current dimension style is associated with that dimension. The dimension retains this dimension style unless you apply a new dimension style to it or set up dimension style overrides.

The picture below shows where the vdDimstyle's properties apply to vdDimension objects.



In the DimStyle you can use an existing TextStyle in order to display the texts/numbers of the dimension, see in this article.

You can see them using the DimStyle's dialog :

```
private void OpenDimStylesDialog()
{
    VectorDraw.Professional.Dialogs.frmDimStyle.Show (vdFC.BaseControl.ActiveDocument);
}
```

You can also create your own DimStyles with code like below:

```
private void AddDimstylesItems()
{
//We create a vdDimstyle object.
   VectorDraw.Professional.vdPrimaries.vdDimstyle style1 = new
VectorDraw.Professional.vdPrimaries.vdDimstyle();
    style1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    style1.setDocumentDefaults();
   //And change some properties.
    style1.Name = "DimStyle1";
    style1.TextColor.ColorIndex = 1;
    style1.TextHeight *= 4.0;
//And add this object to the Dimstyles collection of the document.
    vdFC.BaseControl.ActiveDocument.DimStyles.AddItem(style1);
//We can also add dimstyles using the Add function of the dimstyles collection which is much
easier.
```

```
   VectorDraw.Professional.vdPrimaries.vdDimstyle style2 =
vdFC.BaseControl.ActiveDocument.DimStyles.Add("DimStyle2");
    style2.ExtLineVisible = false;
    VectorDraw.Professional.vdPrimaries.vdDimstyle style3 =
vdFC.BaseControl.ActiveDocument.DimStyles.Add("DimStyle3");
    style3.ExtLineColor.ColorIndex = 1;
    style3.DimTol = true;
    style3.DimTp = 0.3;
    style3.DimTm = 0.3;
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdDimstyle Class".

Getting Started - TextStyles - vdFramedControl

TextStyle is a named, saved collection of settings that determines the appearance of text strings. You can create your own text styles which can have specific fonts and text height. You can also specify if the texts (vdText/vdMText) will be underlined,bold etc. There is no limit to the number of text styles you can create in your drawing. When you enter text, it uses the current text style, which sets the font, size, and other text characteristics. If you want to create text using a different text style, you can make another text style active.

In the TextStyle you can use Windows TTF fonts or CAD fonts named SHX fonts. More information regarding these you can see in this article.

You can see them using the TextStyle's dialog :

```
private void OpenTextstylesDialog()
{
    VectorDraw.Professional.Dialogs.frmTextStyle.Show
(vdFramedControl1.BaseControl.ActiveDocument);
}
```

You can also create your own TextStyles with code like below:

```
private void AddTextstylesItems()
{
//We add a textstyle with font name Verdana.
   VectorDraw.Professional.vdPrimaries.vdTextstyle style1 = new
VectorDraw.Professional.vdPrimaries.vdTextstyle();
    style1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    style1.setDocumentDefaults();
    style1.Name = "TextStyle1";
    style1.FontFile = "Verdana";
    vdFramedControl1.BaseControl.ActiveDocument.TextStyles.AddItem(style1);
//You can also add Textstyles using the Add function of the collection which
is much easier.
   VectorDraw.Professional.vdPrimaries.vdTextstyle style2 =
vdFC.BaseControl.ActiveDocument.TextStyles.Add("Textstyle2");
    style2.Extra.IsUnderLine = true;
    style2.Extra.IsStrikeOut = true;
    style2.FontFile = "isocr.shx"; //This TextStyle will use a CAD SHX font.
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdTextstyle Class".

Getting Started - Document Properties & Events - vdFramedControl

The VectorDraw Document object (named vdDocument Class; check also the "vdDocument Class" topic in our VDF Help file) is the top-most object in the drawing and of course the basic object. It has many properties and methods. Some properties of Document object (and ActiveLayout) are used in order to change the **view/appearance** of the document, like the "RenderMode" property, the "ViewCenter" & "ViewSize" properties, the "Palette" etc. It also contains information regarding the **interface**, like "OsnapMode" and "OrthoMode", "SnapAngle, SnapBase, SnapIso" and similar properties. It is also containing the **table collections** of the drawing like the "TextStyles, DimStyles, Layers, Blocks etc" that are used in the drawing. And of course it contains the **basic I/O** functions like Open and Save.

You can also see topics: "vdDocument Class", "vdDocument Class, Methods", "vdDocument Class, Properties" and "vdDocument Class, Overview" in our VDF help (VDF.CHM).

The VectorDraw Document object has events that the developer can use to customize his/her application. The complete list of these events and also a brief description can be found in our VDF Help file under "vdDocument Class Events " topic, check also the "vdDocument Class" topic.

Lets see a simple example. We will add a form and a button. In the button click a cmdLine with user input will be run :

```
private void btLine_Click(object sender, EventArgs e)
{//the code bellow will start a cmdLine command that waits the user to input the points
   this.vdSC.BaseControl.ActiveDocument.CommandAction.CmdLine("USER");
}
```

While this command is running and the user inputs points a "rubber" line is displayed. This example will also draw another "rubber" entity that will be a circle besides the rubber line. The Event that fires while the command is active is the OnActionDraw (the action is the cmdLine). So we have to "define" the event in the Form Load, like

```
private void Form1_Load(object sender, EventArgs e)
{//The Form1_Load runs when the program starts. Here we will define the OnActionDraw event
    vdFC.BaseControl.ActiveDocument.OnActionDraw += new
VectorDraw.Professional.vdObjects.vdDocument.ActionDrawEventHandler(ActiveDocument_OnActionDraw);
}
```

After this we have to add the code to the OnActionDraw event that will draw the rubber circle along with the rubber line that is drawn automatically by the cmdLine. The code :

```
void ActiveDocument_OnActionDraw(object sender, object action, bool isHideMode, ref bool
cancel)
{//This code will run on every Action draw
   if (!(action is VectorDraw.Actions.ActionGetRefPoint)) return; //if the action is not an
"input point" then exit
    VectorDraw.Actions.BaseAction act = action as VectorDraw.Actions.BaseAction;
    VectorDraw.Geometry.gPoint refpoint = act.ReferencePoint; //This is the base point
    VectorDraw.Geometry.gPoint currentpoint = act.OrthoPoint; //This is the cursor position
    VectorDraw.Professional.vdFigures.vdCircle circle = new
VectorDraw.Professional.vdFigures.vdCircle();
    circle.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    circle.setDocumentDefaults();
    circle.Center = VectorDraw.Geometry.gPoint.MidPoint(refpoint, currentpoint);
    circle.Radius = circle.Center.Distance3D(refpoint);
    circle.Draw(act.Render);
}
```

Now lets see a different kind of event, the OnAfterAddItem event that fires after an entity is added to the document collection. We will use this event to add also a rectangle (vdRect Object) to the document. This rectangle will be round the line that was added with cmdLine code. First we will have to "define" this event in the form load too:

```
private void Form1_Load(object sender, EventArgs e)
{//The Form1_Load runs when the program starts. Here we will define the events
    vdFC.BaseControl.ActiveDocument.OnActionDraw += new
VectorDraw.Professional.vdObjects.vdDocument.ActionDrawEventHandler(ActiveDocument_OnActionDraw);
   vdFC.BaseControl.ActiveDocument.OnAfterAddItem += new
VectorDraw.Professional.vdObjects.vdDocument.AfterAddItemEventHandler(ActiveDocument_OnAfterAddItem);
    vdFC.BaseControl.ActiveDocument.OnGetGripPoints += new
VectorDraw.Professional.vdObjects.vdDocument.GetGripPointsEventHandler(ActiveDocument_OnGetGripPoints);
    vdFC.BaseControl.ActiveDocument.OnAfterOpenDocument += new
VectorDraw.Professional.vdObjects.vdDocument.AfterOpenDocument(ActiveDocument_OnAfterOpenDocument);
}
```

And add this code :

```
void ActiveDocument_OnAfterAddItem(object obj)
{//After a vdLine is added we will also add a rectangle.
   VectorDraw.Professional.vdFigures.vdLine line = obj as
VectorDraw.Professional.vdFigures.vdLine;
   if (line == null)
    {
      return;
    }
   else
   {   //Draw a rectangle around the vdLine that was previously added
      VectorDraw.Professional.vdFigures.vdRect onerect = new
VectorDraw.Professional.vdFigures.vdRect();
      //We set the document where the rect is going to be added.This is important for the
vdRect in order to obtain initial properties with setDocumentDefaults.
      onerect.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
      onerect.setDocumentDefaults();
      //The two previus steps are important if a vdFigure object is going to be added to a
document.
        //Now we will change some properties of the rect.
      onerect.InsertionPoint = line.StartPoint;
      onerect.Width = line.EndPoint.x - line.StartPoint.x;
      onerect.Height = line.EndPoint.y - line.StartPoint.y;
      onerect.PenColor.ColorIndex = 34;
      ////Now we will add this object to the Entities collection of the Model
Layout(ActiveLayout).
      vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(onerect);
      onerect.Invalidate();
   }

}
```

Now lets add an extra grip point to the lines we draw. In this case the OnGetGripPoints event will be use like :

```
void ActiveDocument_OnGetGripPoints(object sender, VectorDraw.Geometry.gPoints gripPoints, ref
bool cancel)
{
   VectorDraw.Professional.vdFigures.vdLine line = sender as
VectorDraw.Professional.vdFigures.vdLine;
   if (line != null)
   {
        VectorDraw.Geometry.gPoint pt = new VectorDraw.Geometry.gPoint();
        pt = VectorDraw.Geometry.gPoint.MidPoint(line.StartPoint, line.EndPoint);
        gripPoints.Add(line.StartPoint);
        gripPoints.Add(line.EndPoint);
        gripPoints.Add(pt);
        //To cancel the VDF component code that adds the startpoint and endpoint after this
event
        cancel = true;
   }
}
```

The OnAfterOpenDocument event is fire after the user opens a drawing. In the example below, a message box will be shown with the number of the entities of the ActiveLayout and also the UCS axis icon will be hidden :

```
void ActiveDocument_OnAfterOpenDocument(object sender)
{// Show the activelayouts' entities count and hide the UCS axis icon.
   MessageBox.Show("In this drawing (activelayout) are : " +
vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.Count + " entities \r\nUCS Axis will be
hidden.");
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.ShowUCSAxis = false;
}
```

Getting Started - Drag & Drop - vdFramedControl

The VectorDraw Developers Framework library has extensive Drag & Drop capability and can be fully controlled by the developer. The drag & Drop can produce a block to another VDF component, a picture in a picture box or even a VectorDraw file when it is dragged to the desktop. In the following code we have a form with 2 VDF components (Source and Destination, named vdSource and vdDestination respectively) and an ImageButton component.
Clicking on the circle in the vdSource control and dragging it to the vdDest control and then release the mouse then an insert will be added (a block also). Doing the same thing to the ImageButton control then a bitmap will be generated and added to the ImageButton.

You can also drag from the vdSource control and drop to an explorer (or to the desktop) to create a vdml file with the destination's entities. You can also drag a file from a file explorer and drop it to the the vdSource control and it will be opened there.

You can check the Drag & Drop sample for a code like :

```
private void CreateSourceEntities()
{    //create entities (in this case a circle) to drop over to the Source
BaseControl
    vdCircle circle = new vdCircle();
    circle.SetUnRegisterDocument(vdSource.ActiveDocument);
    circle.setDocumentDefaults();
    circle.Center = new VectorDraw.Geometry.gPoint();
    circle.Radius = vdSource.ActiveDocument.ViewSize / 4.0d;
    vdSource.ActiveDocument.ActiveLayOut.Entities.AddItem(circle);
    vdSource.Redraw();
}

private void DragDropSample_Load(object sender, EventArgs e)
{
    CreateSourceEntities(); //create the circle

//The events in the Source VDF component are defined here
    vdSource.vdMouseDown += new
VectorDraw.Professional.Control.MouseDownEventHandler(vdSource_vdMouseDown);
    vdSource.vdDragEnter += new
VectorDraw.Professional.Control.DragEnterEventHandler(vdSource_vdDragEnter);
    vdSource.vdDragDrop += new
VectorDraw.Professional.Control.DragDropEventHandler(vdSource_vdDragDrop);

//The events in the Destination VDF component are defined here
    vdDest.vdDragEnter += new
VectorDraw.Professional.Control.DragEnterEventHandler(vdDest_vdDragEnter);
    vdDest.vdDragDrop += new
VectorDraw.Professional.Control.DragDropEventHandler(vdDest_vdDragDrop);

//The events in the ImageButton component are defined here
    ImageButton.DragEnter += new DragEventHandler(ImageButton_DragEnter);
    ImageButton.DragDrop += new DragEventHandler(ImageButton_DragDrop);
}


// ========= The Source VDF code is here ===========
void vdSource_vdDragEnter(DragEventArgs drgevent, ref bool cancel)
{    //Show the copy drag icon to the source control. This means that also
accepts drag objects.
    drgevent.Effect = DragDropEffects.Copy;
}

void vdSource_vdDragDrop(DragEventArgs drgevent, ref bool cancel)
{
    //by default VectorDraw attaches the dropped file as xref to the current
file. We will change
    //this behaviour in order to open the dropped file.
```

```
    cancel = true;
    System.Windows.Forms.DataObject dataobject = new
DataObject(drgevent.Data);
    System.Collections.Specialized.StringCollection strings =
dataobject.GetFileDropList();
    bool success = vdSource.ActiveDocument.Open(strings[0]);
    if (!success)
    MessageBox.Show("The file could not be opened");
}

void vdSource_vdMouseDown(MouseEventArgs e, ref bool cancel)
{    //begin a drag drop to the Source BaseCotnrol
    if (e.Button == MouseButtons.Left)
    {
        cancel = true;
        vdSelection set = vdSource.ActiveDocument.Selections.Add("dragdrop");
        set.RemoveAll();
        set.Select(VectorDraw.Render.RenderSelect.SelectingMode.All, null);
        Box setBound = set.GetBoundingBox();
        gPoint dragOrigin = setBound.MidPoint;
        //The DoDragDrop (with flag = 3) command will create a stream vdml
file from the selection , an image and a vdml file to drop.
        //The stream will be used to the destination BaseControl in order to
add the entities to a block .
        //The image is going to be used in the Button Control in order to
show the image.
        //The file can be dropped in a file explorer (desktop) in order to
create a vdml file.
        vdSource.ActiveDocument.CommandAction.DoDragDrop(set,
vdSelection.DragDropEffects.Copy, dragOrigin, ImageButton.Width,
ImageButton.Height, 3);
    }
}


// ========== The Destination VDF code is here ===========
void vdDest_vdDragEnter(DragEventArgs drgevent, ref bool cancel)
{    //allow drop to the Destination BaseControl
    drgevent.Effect = DragDropEffects.Copy;
}

void vdDest_vdDragDrop(DragEventArgs drgevent, ref bool cancel)
{    //drop the stream as block to the Destination BaseControl
    cancel = true;

    System.IO.MemoryStream stream = drgevent.Data.GetData("VectorDraw.6") as
System.IO.MemoryStream;
    if (stream == null) return;
    vdBlock block = vdDest.ActiveDocument.Blocks.AddFromStream("dragdrop",
stream);
    if (block == null) return;
    vdDest.ActiveDocument.CommandAction.CmdInsert(block.Name,
vdDest.ActiveDocument.CCS_CursorPos(), 1.0d, 1.0d, 0.0d);
}


// ========== The ImageButton code is here ===========
void ImageButton_DragDrop(object sender, DragEventArgs e)
{    //drop the bitmap image to the Button control
    System.Drawing.Bitmap bmp = e.Data.GetData(typeof(System.Drawing.Bitmap))
as System.Drawing.Bitmap;
    if (bmp == null) return;
    ImageButton.Image = bmp;
}

void ImageButton_DragEnter(object sender, DragEventArgs e)
{    //allow drop for the Image to the Button control
    e.Effect = DragDropEffects.Copy;
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdDragEnter" and "vdDragDrop".

Getting Started - Globalization of Resources - vdFramedControl

The VectorDraw Developers Framework version 6 .NET component can be localized in any language. Basically the words that are being localized are the prompt messages, the grid property names and also the forms. If a customer wants to localize his VectorDraw component can contact with us and we will provide the following:

1. **vdRes.txt**
   vdRes.txt is a text document that contains all prompt messages and also all property names that are being displayed in the property list. There he can translate these messages in any language.
2. **vdFormsRes.dll, vdFormsRes project**
   vdFormsRes is a project that contains all forms of VDF component. In this project all labels and buttons can be localized to any language. Also someone can change the appearance of the forms.

Limitations:
-The constants in the property grid cannot be changed (e.g. PlFlagOPEN ,PlFlagCLOSE constant of the polyline's flag property cannot be localized).
-You cannot add new buttons, labels at the vdFormsRes project.

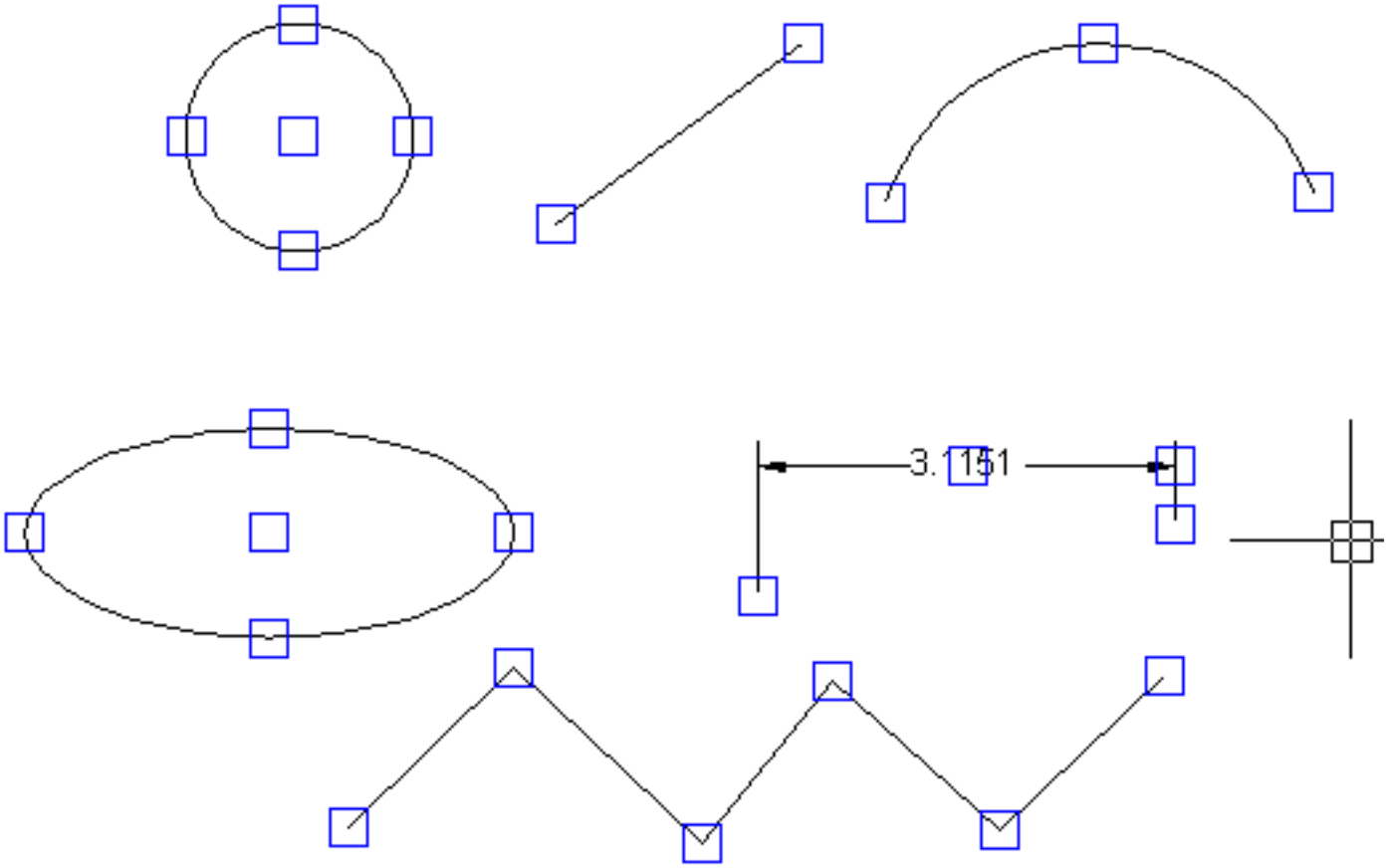
New method added to get/set resource directory for commands.txt, menu.txt, vdres.txt
VectorDraw.Serialize.Activator.GetResourcesDirectory
VectorDraw.Serialize.Activator.SetResourcesDirectory
This directory is set by application (when the application is loaded) and the component searches this directory for the resource files Commands.txt, Menu.txt and vdres.txt.

Also in vdres.txt the first name represent a property name and if the second value is '**-**' (minus symbol) then the property is hidden from property grid.

If you want to use the vdFormsRes.dll in order to load your own forms then you should set the directory that this dll is located in the vdres.txt at GlobalizedFormsDirectory with full path and dll name at the beginning of the text document.

Getting Started - Grips - vdFramedControl

Grips are small, squares that are displayed at strategic points on objects that you have selected with your mouse. You can drag these grips to edit the object(s) (stretch, move, rotate etc) quickly. *The picture below shows the default Grip points that are displayed on some vdFigures:*



VectorDraw automatically shows the Grips on objects that are click if the EnableAutoGripOn is true. The EnableAutoGripOn property is a value representing if the mouse click will show the grips of the entities and also if the window Select command will start when the user click's to the control and there is no Entity below.

You can also show the grips of an entity if the EnableAutoGripOn is false, using code in this article.

You can change the size or the color of the grips that are displayed using the GripColor and GripSize of vdRenderGlobalProperties.

VectorDraw has the ability to create and display your own grips. For example you can add a mid-point grip on vdLine object. See the following code :

```
void ActiveDocument_OnGetGripPoints(object sender,
VectorDraw.Geometry.gPoints gripPoints, ref bool cancel)
{
    VectorDraw.Professional.vdFigures.vdLine line=sender as
VectorDraw.Professional.vdFigures.vdLine;
    if (line != null)
    {
        VectorDraw.Geometry.gPoint pt = new VectorDraw.Geometry.gPoint();
        pt = VectorDraw.Geometry.gPoint.MidPoint(line.StartPoint,
line.EndPoint);
        gripPoints.Add(line.StartPoint);
        gripPoints.Add(line.EndPoint);
        gripPoints.Add(pt);
        //To cancel the code that adds the startpoint and endpoint after this
event
        cancel = true;
    }
```

```
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "ShowGrips Method".

Getting Started - Handle - vdFramedControl

Handle is a unique hexadecimal number for every VectorDraw graphical object (like lines arcs etc) and for VectorDraw collections (like Layers, Dimstyles etc). This number which is a read only property of string type is persistent (stays the same) in a drawing for the life time of the object. Every time a drawing is opened the drawing's object's Handle are the same.

It is very useful when you want to "connect" a graphical object of your drawing to a data object in your database, for fast retrieving like when you use FindFromHandle of vdDocument. See the simple example below :

```
private void button1_Click(object sender, EventArgs e)
{
    string Handle_circ = "";
    vdFC.BaseControl.ActiveDocument.New();
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(new VectorDraw.Geometry.gPoint(2.0d, 3.0d,
4.0d), 3.0d);
    VectorDraw.Professional.vdFigures.vdCircle circ;
    circ =
vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities[vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.Count
- 1] as VectorDraw.Professional.vdFigures.vdCircle;
    if (circ != null)
    {
        MessageBox.Show("A circle created with unique Handle : " + circ.Handle.ToStringValue() + "\n" +
"\n" + "This is the string that can be stored in a database.");
        Handle_circ = circ.Handle.ToStringValue(); //save the handle to the database
    }
    vdFC.BaseControl.ActiveDocument.SaveAs(@"c:\test\test1.vdcl"); //Drawing saved
    vdFC.BaseControl.ActiveDocument.New();// the active document cleared
    vdFC.BaseControl.ActiveDocument.Redraw(true);
    MessageBox.Show("Document saved and cleared (is empty)");

    //Now lets open again the drawing and GET the entity from the stored Handle
    vdFC.BaseControl.ActiveDocument.Open(@"c:\test\test1.vdcl"); //Drawing opened
    vdFC.BaseControl.ActiveDocument.Redraw(true);
    VectorDraw.Professional.vdObjects.vdObject obj = vdFC.BaseControl.ActiveDocument.FindFromHandle(new
VectorDraw.Professional.vdObjects.vdHandle(Handle_circ),
typeof(VectorDraw.Professional.vdObjects.vdPrimary));
    if (obj != null)
    {
        VectorDraw.Professional.vdObjects.vdPrimary prim = obj as
VectorDraw.Professional.vdObjects.vdPrimary;
        MessageBox.Show("I got back the : " + prim._TypeName.ToString() + " with handle : " +
Handle_circ + " that I read from the database.");
    }
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "Handle Property".

Getting Started - Hatches - vdFramedControl

Hatches and Solid fills are used when you want to show an area different from another area. For example in a drawing that shows a house you can use a "GRASS" hatch to draw the garden of the house and a "CONCRETE" hatch to show the road outside this house. There are some predefined hatches in the hatches collection that you can use but you can also add your own hatches. Hatches consist of lines that can be Dashes (-), Dots (.) and Spaces ( ). Beside hatching you can also fill an area with a specific fillcolor that can also be transparent, with a block or with an image.

Below is a sample code that will add a hatch pattern (named TEST_HATCH) with two lines to the existing hatch collection:

```
private void AddHatchItems() {
    VectorDraw.Professional.vdPrimaries.vdHatchPattern myHatch = new
VectorDraw.Professional.vdPrimaries.vdHatchPattern();
    myHatch.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    myHatch.setDocumentDefaults();
    myHatch.Name = "TEST_HATCH"; //hatch is created and it is empty
    //Add 2 lines with dashes to the hatch pattern
    myHatch.PatternLines.AddItem(new VectorDraw.DrawElements.grPatternLine(0.0d,
0.0d, 0.0d, 0.0d, 5.0d, new double[] { 3.0d, -3.0d }, false));
    myHatch.PatternLines.AddItem(new
VectorDraw.DrawElements.grPatternLine(VectorDraw.Geometry.Globals.HALF_PI, 0.0d,
10.0d, 6.0d,0.0d, new double[] { 3.0d, -2.0d }, false));

    // OR you can do the same as below:
    //myHatch.PatternLines.AddItem(new VectorDraw.DrawElements.grPatternLine(0.0d,
0.0d, 0.0d, 0.0d, 5.0d, new double[] { 3.0d, -3.0d }, false));
    //myHatch.PatternLines.AddItem(new VectorDraw.DrawElements.grPatternLine(90.0d,
0d, 10d, 0.0d, 6.0d, new double[] { 3, -2 }, true));

    // Important Notes: grPatternLine(angle, originX, originY, offsetX, offsetY,
double[] dashes, bool applytransforms)
    // - When you use the applytransforms = true then you have to use degrees (like
90 degrees) and not radians (like PI/2) in the angle
    // - The point(OriginX,OriginY) remains the same as when you use
applytransforms = false
    // - When you use the applytransforms = true then the offset distance is
translated accordingly with the angle. In the above example the offset distance x:6
& y:0 when is transformed by angle=90 becomes the opposite; x:0 & y:6
    //
    // - If you like to add a dot in the 1st line above then use something like :
    //      myHatch.PatternLines.AddItem(new
VectorDraw.DrawElements.grPatternLine(0.0d, 0.0d, 0.0d, 0.0d, 5.0d, new double[] {
3.0d, -1.5d,0.0d,-1.5d }, false));

    myHatch.Update(); //Update the Hatch
    vdFC.BaseControl.ActiveDocument.HatchPatterns.AddItem(myHatch); //add the hatch
to the collection
}
```

After this lets some some entities with hatches and fillmode solid. See the sample code below :

```
private void AddHatchedEntities() {
    VectorDraw.Professional.vdFigures.vdCircle circ1 = new
VectorDraw.Professional.vdFigures.vdCircle();
    circ1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    circ1.setDocumentDefaults();
    circ1.Center = new VectorDraw.Geometry.gPoint(10.0, 10.0);
    circ1.Radius = 7.0;
    circ1.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    circ1.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModeSolid;
    circ1.HatchProperties.FillColor.ColorIndex = 1;
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(circ1);
    circ1 = new VectorDraw.Professional.vdFigures.vdCircle();
    circ1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
```

```
    circ1.setDocumentDefaults();
    circ1.Center = new VectorDraw.Geometry.gPoint(20.0, 10.0);
    circ1.Radius = 5.0;
    circ1.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    circ1.HatchProperties.Solid2dTransparency = 100;
    circ1.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModeSolid;
    circ1.HatchProperties.FillColor.ColorIndex = 2;
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(circ1);

    VectorDraw.Professional.vdFigures.vdPolyline poly = new
VectorDraw.Professional.vdFigures.vdPolyline();
    poly.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    poly.setDocumentDefaults();
    poly.VertexList.Add (new VectorDraw.Geometry.gPoint (10.0,-10.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(20.0, -10.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(25.0, -20.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(5.0, -20.0));
    poly.Flag = VectorDraw.Professional.Constants.VdConstPlineFlag.PlFlagCLOSE;
    poly.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    poly.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModePattern;
    poly.HatchProperties.FillColor.ColorIndex = 0;
    poly.HatchProperties.FillBkColor.ColorIndex = 2;
    poly.HatchProperties.HatchPattern =
vdFC.BaseControl.ActiveDocument.HatchPatterns.FindName("GRASS");
    poly.HatchProperties.HatchScale = 0.05;
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(poly);

    // This is the polyline that will be hatched with the hatch we made in the
previous sample
    poly = new VectorDraw.Professional.vdFigures.vdPolyline();
    poly.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    poly.setDocumentDefaults();
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(25.0, -10.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(35.0, -10.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(40.0, -20.0));
    poly.VertexList.Add(new VectorDraw.Geometry.gPoint(30.0, -20.0));
    poly.Flag = VectorDraw.Professional.Constants.VdConstPlineFlag.PlFlagCLOSE;
    poly.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    poly.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModePattern;
    poly.HatchProperties.FillColor.ColorIndex = 1;
    poly.HatchProperties.FillBkColor.ColorIndex = 10;
    poly.HatchProperties.HatchPattern =
vdFC.BaseControl.ActiveDocument.HatchPatterns.FindName("TEST_HATCH");
    poly.HatchProperties.HatchScale = 0.15;
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(poly);

    VectorDraw.Professional.vdFigures.vdArc arc = new
VectorDraw.Professional.vdFigures.vdArc();
    arc.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    arc.setDocumentDefaults();
    arc.Center = new VectorDraw.Geometry.gPoint(30, -4);
    arc.Radius = 10.0;
    arc.StartAngle = VectorDraw.Geometry.Globals.DegreesToRadians(13.0);
    arc.EndAngle = VectorDraw.Geometry.Globals.DegreesToRadians(175.0);
    arc.HatchProperties = new
VectorDraw.Professional.vdObjects.vdHatchProperties();
    arc.HatchProperties.FillMode =
VectorDraw.Professional.Constants.VdConstFill.VdFillModeHatchBlock;
    arc.HatchProperties.FillColor.SystemColor = Color.Blue;
    arc.HatchProperties.HatchBlock =
vdFC.BaseControl.ActiveDocument.Blocks.VDDIM_DEFAULT;
    arc.HatchProperties.HatchAngle =
VectorDraw.Geometry.Globals.DegreesToRadians(33.0);
    arc.HatchProperties.HatchScale = 0.3;
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(arc);

    vdFC.BaseControl.ActiveDocument.Model.ZoomExtents();
    vdFC.BaseControl.ActiveDocument.Redraw(true);
```

```
}
```

You can call the hatch pattern dialog with a code like :

```
private void OpenHatchPatternsDialog()
{
    VectorDraw.Professional.Dialogs.GetHatchPatternsDialog frm =
VectorDraw.Professional.Dialogs.GetHatchPatternsDialog.Show(vdFC.BaseControl.ActiveDocument,
vdFC.BaseControl.ActiveControl,
vdFC.BaseControl.ActiveDocument.HatchPatterns.Solid);
   if (frm.finalSelected!=null)
      MessageBox.Show(frm.finalSelected.Name + " pattern selected");
   else
      MessageBox.Show("Cancel button pressed");
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdHatchProperties Class" "vdHatchPatterns Class" and "vdHatchPattern Class".

Getting Started - Images - vdFramedControl

Image is a basic object for inserting images in the drawing. Inserted Images can be BMP, GIF, JPG, PNG and TIFF.

Image has a Scale property that is used to resize keeping the aspect ratio of the image. The image is defined by InsertionPoint, Rotation, Width and Height.

ImageDefs is the collection of the images used in the drawing. Every new image is added to this collection. You can try a code like :

```
private void AddImageItems()
{
    //We will create an image and add it to the imageDef
    //First we create a black circle with radius 5.0 at (0.0,0.0,0.0).
    VectorDraw.Professional.vdFigures.vdCircle circle = new
VectorDraw.Professional.vdFigures.vdCircle();
    circle.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    circle.setDocumentDefaults();
    circle.Center = new VectorDraw.Geometry.gPoint();
    circle.Radius = 5.0;
    circle.PenColor.SystemColor = Color.Black;
    circle.PenWidth = 0.3;
    //Then we create a layout to add this circle.Note that this is a
temporary layout and it is not added to the document at all.
    VectorDraw.Professional.vdPrimaries.vdLayout lay = new
VectorDraw.Professional.vdPrimaries.vdLayout();
    lay.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    lay.setDocumentDefaults();
    lay.Entities.AddItem(circle);
    lay.BkColorEx = Color.Red;
    //This property overides the general DisableShowPrinterPaper property
that is located in the ActiveDocument.
    lay.DisableShowPrinterPaper = true;
    //We zoom the layout where we want to show the circle.
    lay.ZoomWindow (new VectorDraw.Geometry.gPoint(-6.0,-6.0),new
VectorDraw.Geometry.gPoint(6,6));
    //We create a 250,250 image and a Graphics from that image.
    System.Drawing.Bitmap image = new Bitmap(250, 250);
    System.Drawing.Graphics gr = System.Drawing.Graphics.FromImage(image);
    //We draw the layout to that image.We first disable the layout paper and
enable it again.
    lay.RenderToGraphics(gr, null, image.Width, image.Height);
    //From that image we create the imageDefinition.
    VectorDraw.Professional.vdPrimaries.vdImageDef imagedef = new
VectorDraw.Professional.vdPrimaries.vdImageDef();
    imagedef.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    imagedef.setDocumentDefaults();
    imagedef.Name = "Image1";
    System.IO.MemoryStream stream = new System.IO.MemoryStream();
    image.Save(stream, System.Drawing.Imaging.ImageFormat.Bmp);
    imagedef.InternalSetBytes(new
VectorDraw.Geometry.ByteArray(stream.ToArray()));
    vdFC.BaseControl.ActiveDocument.Images.AddItem(imagedef);
    //We dispose any objects that we don't want.
    stream.Close();
    stream.Dispose();
    gr.Dispose();
    //We could have saved the image to the drive using the Save function.
    string path = System.IO.Path.GetDirectoryName(Application.ExecutablePath)
+ "\\" + "vdimage.bmp";
    image.Save(path, System.Drawing.Imaging.ImageFormat.Bmp);
    //And create a second imageDef from this filename.
    VectorDraw.Professional.vdPrimaries.vdImageDef imagedef1 = new
```

```
VectorDraw.Professional.vdPrimaries.vdImageDef();
    imagedef1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    imagedef1.setDocumentDefaults();
    imagedef1.Name = "Image2";
    imagedef1.FileName = path;
    vdFC.BaseControl.ActiveDocument.Images.AddItem(imagedef1);
    image.Dispose();
    MessageBox.Show ("An image was created (silently) from rendered objects
and has been added to the Images collection of the document.");
}

private void AddImagesEntities()
{
    if (vdFC.BaseControl.ActiveDocument.Images.Count == 0)
    {
        MessageBox.Show("Please run AddImageItems procedure first");
        return;
    }
    //We create an image and give the precreated image definition.
    VectorDraw.Professional.vdFigures.vdImage image = new
VectorDraw.Professional.vdFigures.vdImage();
    image.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    image.setDocumentDefaults();
    //We know that the first image is what we want so we can just get the [0]
item from the collection else we could have used the Find methods.
    image.ImageDefinition = vdFC.BaseControl.ActiveDocument.Images[0];
    image.InsertionPoint = new VectorDraw.Geometry.gPoint(1.0,1.0);
    vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(image);
    vdFC.BaseControl.ActiveDocument.Redraw(true);
}
```

The You can see the image definition dialog using a code like:

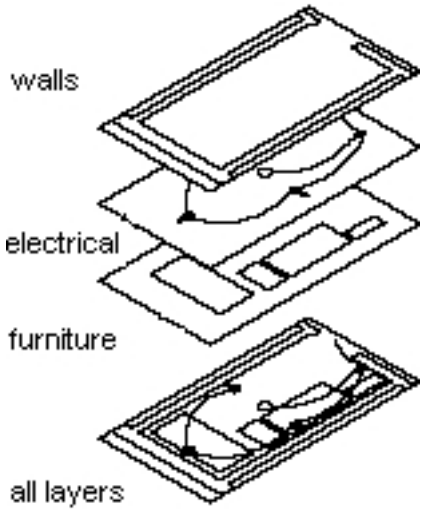
```
private void OpenImageDefinitionsDialog()
{
    VectorDraw.Professional.Dialogs.FrmImageDefs.Show
(vdFC.BaseControl.ActiveDocument);
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdImages Class".

Getting Started - Layers - vdFramedControl

Layer is the equivalent of the overlay used in paper-based drafting. It is the primary organizational tool in VectorDraw, and you can use it to group information by function and to enforce linetype, color, and other standards.
Organizing Layers and the objects on Layers make it easier to manage the information in your Drawings.

When you put one layer over another then the result is the complete drawing.



Having kindred objects on the same layer it is very helpfull in order to organise the drawing.

Properties:

- layer name
- color of the entities
- line type of the entities
- line weight of the entities
- if it is participate or not in the drawing (thawed or frozen)
- locked or not

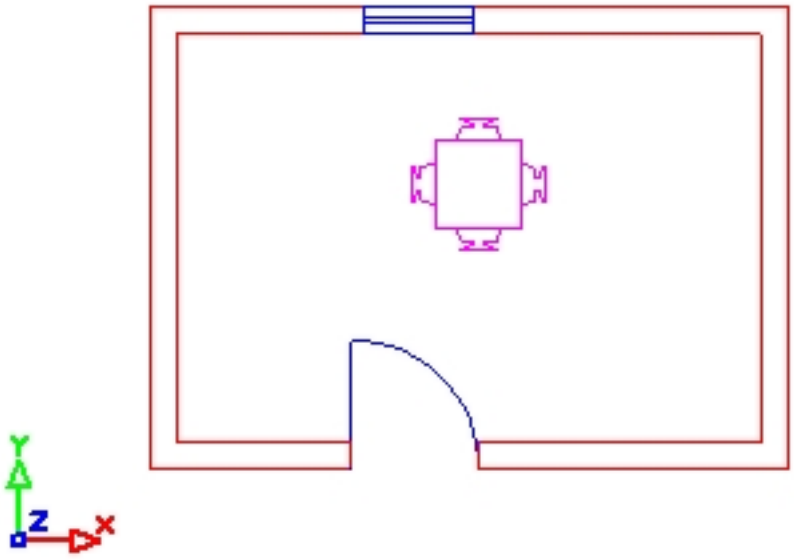
When you begin a new drawing, VectorDraw creates a special layer named 0. By default, layer 0 is assigned color number 6 (Foreground color : white or black depending upon your background color), the CONTINUOUS linetype and a lineweight of Default (the default setting is .01 inch or .25 mm). Layer 0 cannot be deleted or renamed.

All new objects are added to the active layer if no layer is specified. Using VectorDraw you can Freeze (Hide), Thaw (Show) and Lock layers. By controlling whether a Layer's state is Thaw or Frozen you can change the appearance of your drawing to display only the information on the Layers that are visible. Freezing unused Layers will help the performance of VectorDraw.

In the drawing below (Picture 1) there are 3 types of items : **walls** (the lines and Polylines with red color), **doors&windows** (Blue color) and **furniture** (Magenta).
These objects are teamed and drawn in different layers. Walls placed on layer "WALLS", Doors&windows are placed on layer "WIN_DOORS" and furniture are placed on layer "FURNITURE".

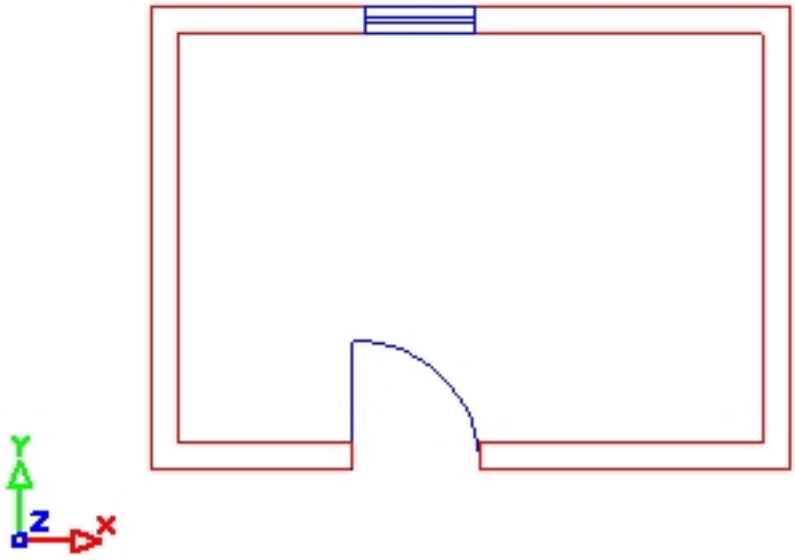
Picture 1

All layers are Thawed. All information is shown.
In **big** drawings this might confuse the user.



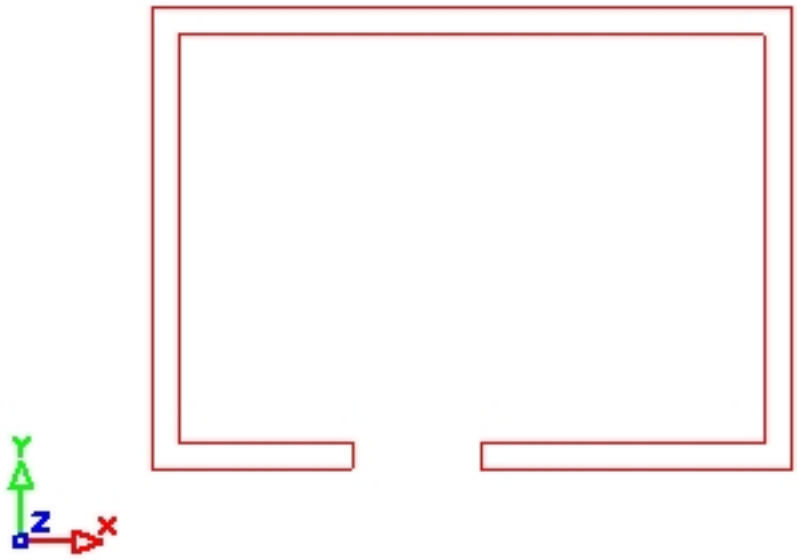
Picture 2

Layer "FURNITURE" is Frozen. The furniture is not shown.
A civil engineer would like to view the drawing this way.



Picture 3

All layers except the "WALLS" are frozen.
This is the basic information.



You can see them using the Layers dialog :

```
private void OpenLayersDialog()
{
```

```
    VectorDraw.Professional.Dialogs.LayersDialog.Show
(vdFC.BaseControl.ActiveDocument );
}
```

You can also create your own custom Layers. Like in the code below :

```
VectorDraw.Professional.vdPrimaries.vdLayer lay = new
VectorDraw.Professional.vdPrimaries.vdLayer();
lay.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
lay.setDocumentDefaults();
lay.Name = "Layer1";
lay.PenColor.SystemColor = Color.Red;
vdFC.BaseControl.ActiveDocument.Layers.AddItem(lay);
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdLayer Class".

Getting Started - Layouts & Viewports - vdFramedControl

Layout is used to compose or lay out your model drawing for printing. A layout may consist of a title block, one or more viewports, and annotations. As you create a layout, you can design floating Viewport configurations to visualize different details in your drawing.

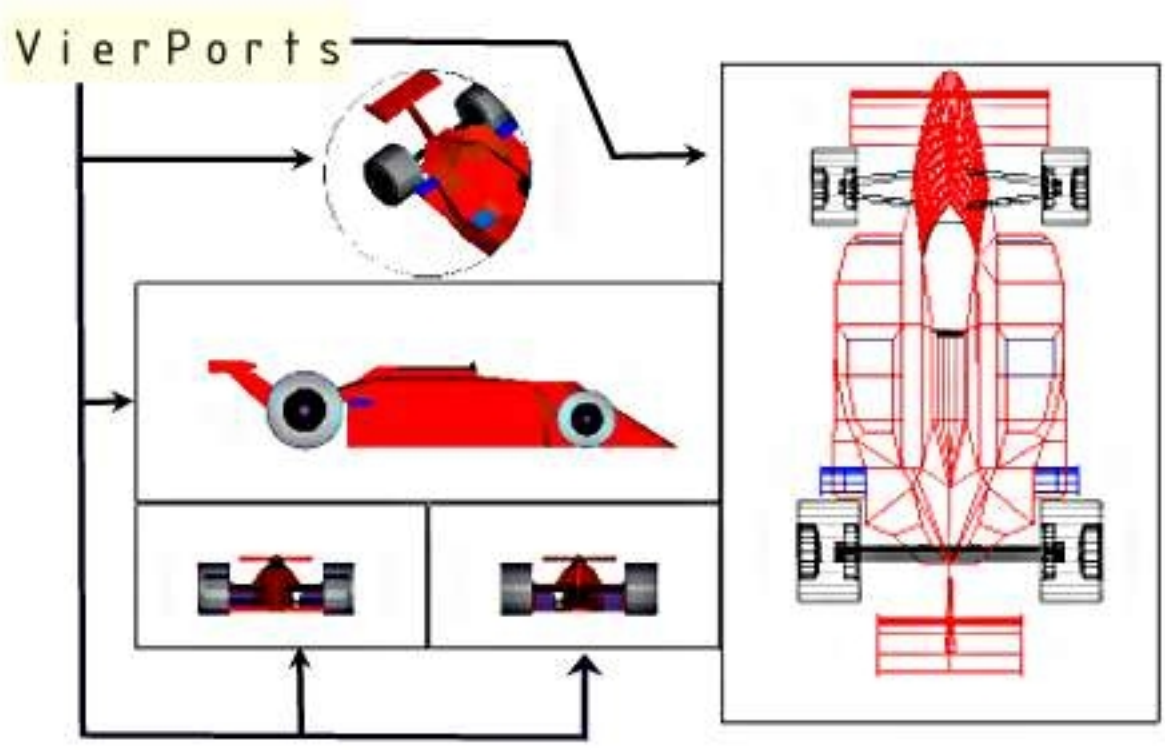
A layout is a paper space environment that simulates a sheet of paper. In a layout, you can create and position viewport objects, and you can add a title block or other geometry. You can create multiple layouts in a drawing to display various views. Each layout displays the drawing as it will be printed on the sheet of paper.

Typically, when you begin designing a layout environment, you step through the following process:
- Create a model drawing.
- Activate or create a layout.
- Insert a title block if is necessary.
- Create floating viewports and position them in the layout.
- Set the view scale and 3D view angle of the floating viewports.
- Print your layout.

Viewports are areas that display different views of your model. As you work, you can split the drawing area into one or more adjacent rectangular views known as *model viewports*. In large or complex drawings, displaying different views reduces the time needed to zoom or pan in a single view. Also, errors you might miss in one view may be apparent into others. ViewPorts are treated as rectangle drawing objects which display views and can be moved or resized.

They can be created only in entity list collection of a layout and not in Model. Those rectangles filled with the image of the model space objects in different scales depend from the ViewSize and the ViewCenter property. This way in one paper you can print out different views (with different scales) of the same drawing (Model Space) or parts of this drawing. In viewports commands like Pan, Zoom and View3D can be applied.

A sample picture of a Layout (Paper Space) with some Viewports:



Below is a sample code that will open an existing drawing and add layouts and viewports. The AddLayout code should be run before the AddLayoutEntities code :

```
private void AddLayout()
{
    //We open a drawing to the document that contains the info we like to
show. This file "wmap.vdml" is installed along with our setup in the zip file
that contains the vdFrameWork samples, named "vdfexamples.zip" in a folder
```

```
like "c:\Program Files\VectorDraw\VectorDraw Developer
Framework\vdFramedControl Samples"
    string path = Application.ExecutablePath.Substring(0,
Application.ExecutablePath.LastIndexOf('\\')) + "\\..\\..\\wmap.vdml";
    vdFC.BaseControl.ActiveDocument.Open(path);
    //We will add two Layouts to the document.
    //Create a vdLayout object and add it to the layouts collection.
    VectorDraw.Professional.vdPrimaries.vdLayout lay = new
VectorDraw.Professional.vdPrimaries.vdLayout();
    lay.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    lay.setDocumentDefaults();
    lay.Name = "MyLayout1";
    lay.ShowUCSAxis = false;
    vdFC.BaseControl.ActiveDocument.LayOuts.AddItem(lay);
    //Or we can add a layout like this:
    VectorDraw.Professional.vdPrimaries.vdLayout lay2 =
vdFC.BaseControl.ActiveDocument.LayOuts.Add("MyLayout2");
    lay2.DisableShowPrinterPaper = true;
    //In order to see the background color you must disable the printer paper
draw.
    lay2.BkColorEx = Color.BlueViolet;
    //The file we opened has already two layouts that we will delete.
    vdFC.BaseControl.ActiveDocument.LayOuts[0].Deleted = true;
    //or this can be done like this:
    VectorDraw.Professional.vdPrimaries.vdLayout lay3 =
vdFC.BaseControl.ActiveDocument.LayOuts.FindName("PAPER_SPACE");
    if (lay3 != null) lay3.Deleted = true;
}

private void AddLayoutEntities()
{
    //We check if the layouts have already been added
    if (vdFC.BaseControl.ActiveDocument.LayOuts.Count <=
2){MessageBox.Show("Please add some layouts by pressing the above button
first");return;}
    //Then we add a rectangular viewport to the first added layout.
    VectorDraw.Professional.vdFigures.vdViewport view = new
VectorDraw.Professional.vdFigures.vdViewport();
    view.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    view.setDocumentDefaults();
    view.ShowUCSAxis = false;
    view.Height = 100;
    view.Width = 150;
    view.Center = new VectorDraw.Geometry.gPoint(100.0, 230.0);
    view.ViewCenter = new VectorDraw.Geometry.gPoint(4.4008,6.8233);
    view.ViewSize = 0.252;

    //And add this viewport to the entities of the first layout.
    VectorDraw.Professional.vdPrimaries.vdLayout lay =
vdFC.BaseControl.ActiveDocument.LayOuts.FindName("MyLayout1");
    //If the entities of the layout are not 0 then this means that this
button is already pressed and the viewport already exists so we do not add it
again.
    if (lay.Entities.Count > 0) return;
    if (lay!=null) lay.Entities.AddItem(view);
    //We also add a text entity to the layout.
    VectorDraw.Professional.vdFigures.vdText text = new
VectorDraw.Professional.vdFigures.vdText();
    text.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    text.setDocumentDefaults();
    text.InsertionPoint = new VectorDraw.Geometry.gPoint(55, 155);
    text.TextString = "GREECE";
    text.Height = 15.0;
    if (lay != null) lay.Entities.AddItem(text);
    //We set as active layout the first addede layout to show it.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut =
vdFC.BaseControl.ActiveDocument.LayOuts.FindName("MyLayout1");
    vdFC.BaseControl.ActiveDocument.Redraw(true);
    //Now we will add some viewports with reference objects to the second
```

```
added layout MyLayout2.
    lay = vdFC.BaseControl.ActiveDocument.LayOuts.FindName("MyLayout2");
    //We will create two polylines and a circle to be used as viewports.
    VectorDraw.Professional.vdFigures.vdPolyline poly1 = new
VectorDraw.Professional.vdFigures.vdPolyline();
    poly1.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    poly1.setDocumentDefaults();
    poly1.VertexList.Add (new VectorDraw.Geometry.gPoint(10.0d,280.0d));
    poly1.VertexList.Add(new VectorDraw.Geometry.gPoint(10.0d, 180.0d));
    poly1.VertexList.Add(new VectorDraw.Geometry.gPoint(190.0d, 180.0d));
    poly1.VertexList.Add(new VectorDraw.Geometry.gPoint(190.0d, 280.0d));
    poly1.Flag =
VectorDraw.Professional.Constants.VdConstPlineFlag.PlFlagCLOSE;
    if (lay!=null) lay.Entities.AddItem(poly1);

    //Another polyline
    VectorDraw.Professional.vdFigures.vdPolyline poly2 = new
VectorDraw.Professional.vdFigures.vdPolyline();
    poly2.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    poly2.setDocumentDefaults();
    poly2.VertexList.Add(new VectorDraw.Geometry.gPoint(50.0d, 115.0d));
    poly2.VertexList.Add(new VectorDraw.Geometry.gPoint(10.0d, 65.0d));
    poly2.VertexList.Add(new VectorDraw.Geometry.gPoint(50.0d, 15.0d));
    poly2.VertexList.Add(new VectorDraw.Geometry.gPoint(90.0d, 65.0d));
    poly2.Flag =
VectorDraw.Professional.Constants.VdConstPlineFlag.PlFlagCLOSE;
    if (lay!=null) lay.Entities.AddItem(poly2);
    //A circle
    VectorDraw.Professional.vdFigures.vdCircle circle = new
VectorDraw.Professional.vdFigures.vdCircle();
    circle.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    circle.setDocumentDefaults();
    circle.Center = new VectorDraw.Geometry.gPoint(160.0d,65.0d);
    circle.Radius = 50.0d;
    if (lay!=null) lay.Entities.AddItem(circle);

    //Now that we have the entities we can create 3 different viewports for
these entities.
    VectorDraw.Professional.vdFigures.vdViewport vp = new
VectorDraw.Professional.vdFigures.vdViewport();
    vp.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    vp.setDocumentDefaults();
    vp.ShowUCSAxis = false;
    vp.ClipObj = poly1 as VectorDraw.Professional.vdFigures.vdCurve;
    vp.ZoomExtents();
    vp.PenColor.SystemColor = Color.Red;
    if (lay!=null) lay.Entities.AddItem(vp);

    //Another viewport
    vp = new VectorDraw.Professional.vdFigures.vdViewport();
    vp.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    vp.setDocumentDefaults();
    vp.ShowUCSAxis = false;
    vp.SetClipHandle(poly2.Handle);
    vp.PenColor.SystemColor = Color.Blue;
    vp.ViewCenter = new VectorDraw.Geometry.gPoint(7.1531, 5.0466);
    vp.ViewSize = 2.6962;
    if (lay!=null) lay.Entities.AddItem(vp);

    //And a viewport for the circle
    vp = new VectorDraw.Professional.vdFigures.vdViewport();
    vp.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    vp.setDocumentDefaults();
    vp.ShowUCSAxis = false;
    vp.SetClipHandle(circle.Handle);
    vp.PenColor.SystemColor = Color.Yellow;
    vp.ViewCenter = new VectorDraw.Geometry.gPoint(4.1837, 5.9294);
    vp.ViewSize = 2.2468;
    if (lay!=null) lay.Entities.AddItem(vp);
    //And add some texts for fun :)
```

```
        text = new VectorDraw.Professional.vdFigures.vdText();
        text.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
        text.setDocumentDefaults();
        text.InsertionPoint = new VectorDraw.Geometry.gPoint(80.0d, 165.0d);
        text.TextString = "WORLD";
        text.Height = 10.0;
        if (lay != null) lay.Entities.AddItem(text);
        text = new VectorDraw.Professional.vdFigures.vdText();
        text.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
        text.setDocumentDefaults();
        text.InsertionPoint = new VectorDraw.Geometry.gPoint(15.0d, 120.0d);
        text.TextString = "AUSTRALIA";
        text.Height = 10.0;
        if (lay != null) lay.Entities.AddItem(text);
        text = new VectorDraw.Professional.vdFigures.vdText();
        text.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
        text.setDocumentDefaults();
        text.InsertionPoint = new VectorDraw.Geometry.gPoint(133.0d, 120.0d);
        text.TextString = "AFRICA";
        text.Height = 10.0;
        if (lay != null) lay.Entities.AddItem(text);
        MessageBox.Show("By Default you can double click the viewport entity to
    enter inside and pan or zoom");
    }
```

With the following code you can see the Layout control :

```
    private void OpenLayoutsDialog()
    {
        vdFC.SetLayoutStyle(vdControls.vdFramedControl.LayoutStyle.LayoutTab,
    true);
       MessageBox.Show("The Layout tab has been activated.You can navigate and
    see the added viewports at the bottom left of the application.");
    }
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdLayout Class" and "vdViewport Class".

Getting Started - Linetypes - vdFramedControl

Linetypes are different ways to display a line. This is useful when you want to distinct different lines, for example a line that represents a 12Volt wire and another line that represents a 24 Volt wire in an electrical drawing. VectorDraw library have already some linetypes that can be used.

You can see them using the Linetype's dialog :

```
private void OpenLinetypesDialog() {
    VectorDraw.Professional.Dialogs.GetLineTypeDialog frm =
VectorDraw.Professional.Dialogs.GetLineTypeDialog.Show(vdFC.BaseControl.ActiveDocument,vdFC.BaseControl.ActiveControl,
"Solid", true);
   DialogResult res = frm.DialogResult;
   if (res == DialogResult.OK )
       MessageBox.Show(frm.finalSelected + " Linetype Selected");
   else
      MessageBox.Show("Cancel button pressed");
    }
```

You can also create your own custom Linetypes using Dashes (-), Dots (.) and Spaces ( ). Like in the code below :

```
VectorDraw.Professional.vdPrimaries.vdLineType linetype = new
VectorDraw.Professional.vdPrimaries.vdLineType();
linetype.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
linetype.setDocumentDefaults();
linetype.Name = "MyCustomLinetype";
linetype.Comment = "Custom _ _ . . _ _ . . _ _ . . _ _ . . _ _ . . _ _ . . ";
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(0.5));//Dash
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-0.5));//Blank
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(0.5));//Dash
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-0.5));//Blank
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(0.0)); //Dot
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-0.5));//Blank
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(0.0)); //Dot
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-0.5));//Blank
linetype.Segments.UpdateLength();
vdFC.BaseControl.ActiveDocument.LineTypes.AddItem(linetype);
MessageBox.Show("We created a custom Linetype named MyCustomLinetype and looks like this: _ . . _");
```

Beside this you can create a Linety that will also contain text:

```
//We will add a second Custom linetype that contains text.
linetype = new VectorDraw.Professional.vdPrimaries.vdLineType();
linetype.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
linetype.setDocumentDefaults();
linetype.Name = "MyCustomLinetype2";
linetype.Comment = "Custom2 _ VD _ _ VD _ _ VD _ _ VD _ _ VD _ _ VD _ ";
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(2.0));//Dash
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-1.0));//Blank
VectorDraw.Render.LineTypeSegment seg = new VectorDraw.Render.LineTypeSegment();
seg.Flag = VectorDraw.Render.LineTypeSegment.LineTypeElementType.TTF_TEXT;
seg.ShapeScale = 0.7;
seg.ShapeStyle = vdFC.BaseControl.ActiveDocument.TextStyles.Standard.GrTextStyle;
seg.ShapeText = "VD";
linetype.Segments.AddItem(seg); //Text "VD"

linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-2.0));//Blank
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(2.0)); //Dash
linetype.Segments.AddItem(new VectorDraw.Render.LineTypeSegment(-1.0));//Blank
linetype.Segments.UpdateLength();
vdFC.BaseControl.ActiveDocument.LineTypes.AddItem(linetype);
MessageBox.Show("We created a custom Linetype named MyCustomLinetype2 and looks like this: _ VD _");
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdLinetype Class".

Getting Started - MText & Text - vdFramedControl

The Text (vdText object) and MultilineText (vdMText) object are use to add text strings in your drawings. These texts follow the TextStyle properties (which is a collection of settings) that determines the appearance of text strings. You can also specify if the texts (vdText/vdMText) will be underlined,bold etc. When you enter text, it uses the current text style, which sets the font, size, and other text characteristics. If you want to create text using a different text style, you can make another text style active.

In the TextStyle you can use Windows TTF fonts or CAD fonts named SHX fonts.

You can easily add a text (vdText Object) using a code like :

```
vdFC.BaseControl.ActiveDocument.CommandAction.CmdText(null, null, null);
```

and a MultilineText like :

```
vdFC.BaseControl.ActiveDocument.CommandAction.CmdMText(null, null, null);
```

The above commands can be used for simple texts or for user-input texts. You can add more complex texts and multiline texts using vdText and vdMText objects like below.

You can add a vdText object with code like below:

```
private void butText_Click(object sender, EventArgs e)
{
    //We will create a vdText object and add it to the Active Layout which is the
basic
    //   Model Layout always existing in a Document.
    VectorDraw.Professional.vdFigures.vdText onetext = new
VectorDraw.Professional.vdFigures.vdText();
    //We set the document where the text is going to be added. This is important
for the vdText
    //   in order to obtain initial properties with setDocumentDefaults.
    onetext.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    onetext.setDocumentDefaults();
    //The two previous steps are important if a vdFigure object (like vdText) is
going to be
    //   added to a document. Now we will change some properties of the text.
    onetext.PenColor.ColorIndex = 3;
    onetext.TextString = "Vectordraw Development Framework";
    //vdText object with setDocumentDefaults has the STANDARD TextStyle.
    //   We will change the font of this textstyle to Verdana.
    vdFC.BaseControl.ActiveDocument.TextStyles.Standard.FontFile = "Verdana";
    //We set the insertion point depending the width of the Text from the
vdFigure's BoundingBox
    onetext.InsertionPoint = new VectorDraw.Geometry.gPoint(40.0 -
onetext.BoundingBox.Width, 2);
    onetext.TextLine = VectorDraw.Render.grTextStyleExtra.TextLineFlags.OverLine;
    //Now we will add this object to the Entities collection of the Model
Layout(ActiveLayout).
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(onetext);
    //Zoom in order to see the object.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.ZoomWindow(new
VectorDraw.Geometry.gPoint(-30.0, -10.0), new VectorDraw.Geometry.gPoint(80.0,
50.0));
    //Redraw the document to see the above changes.
    vdFC.BaseControl.ActiveDocument.Redraw(true);
}
```

You can also create a MultiLine Text with code like below:

```
private void butMtext_Click(object sender, EventArgs e)
{
    //We will create a vdMText object and add it to the Active Layout which is the
basic
    //   Model Layout always existing in a Document.
```

```
    VectorDraw.Professional.vdFigures.vdMText onemtext = new
VectorDraw.Professional.vdFigures.vdMText();
    //We set the document where the Mtext is going to be added. This is important
    //  for the vdMText in order to obtain initial properties with
setDocumentDefaults.
    onemtext.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    onemtext.setDocumentDefaults();
    //Now we will change some properties of the text.
    onemtext.InsertionPoint = new VectorDraw.Geometry.gPoint(20, -15);
    onemtext.HorJustify =
VectorDraw.Professional.Constants.VdConstHorJust.VdTextHorCenter;
    onemtext.BoxWidth = 45;

    //The Text string can contain some special control characters like "\P" or
"\C1" which
    //   control the way that this multiLine text is going to be displayed.
    onemtext.TextString = @"\C1;\H5.0;This is\P\C2;\H5.5;\La
MultiText\l\P\C3;\H3.5;\Q20.0;\OObject\o\Q0.0;\P\T2.0;SPACED\H1.5;\A0;down\A1;middle\A2;up";
    //Now we will add this object to the Entities collection of the Model
Layout(ActiveLayout).
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.AddItem(onemtext);
    //Zoom in order to see the object.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.ZoomWindow(new
VectorDraw.Geometry.gPoint(-30.0d, -10.0d), new VectorDraw.Geometry.gPoint(80.0d,
50.0d));
    //Redraw the document to see the above changes.
    vdFC.BaseControl.ActiveDocument.Redraw(true);
}
```

The code above will have a result like :

In the Text String some special control characters/strings can be used in order to change the appearence of the text. These are :

\0...\o	Turns overline on and off
\L...\l	Turns underline on and off
****	Inserts a backslash
\{...\}	Inserts an opening and closing brace
\C*index*;	Changes to the specified color
\H*value*;	Changes to the text height specified in drawing units
\H*value*x;	Changes the text height to a multiple of the current text height
\T*value*;	Adjusts the space between characters, from .75 to 4 times
\Q*angle*;	Changes obliquing angle
\W*value*;	Changes width factor to produce wide text
\F*file_name*;	Changes to the specified font file
\A*x*;	Sets the alignment value; valid values: 0, 1, 2
\P	Ends paragraph
\S...^...;	Stacks the subsequent text at the \, #, or ^ symbol

For more detailed information about properties/methods/events etc of the above object(s), you can also check our

vdf.chm help file, topic(s): "vdText Class" and "vdMText Class".

Getting Started - Object Snaps (OSnaps) - vdFramedControl

Object options allow you to draw in the Object Snap (OSnap). An OSnap mode specifies a snap point at an exact location on an object (like the mid-point of a line). The OSnap mode helps the user to draw an object in a position relative to a previously drawn object.

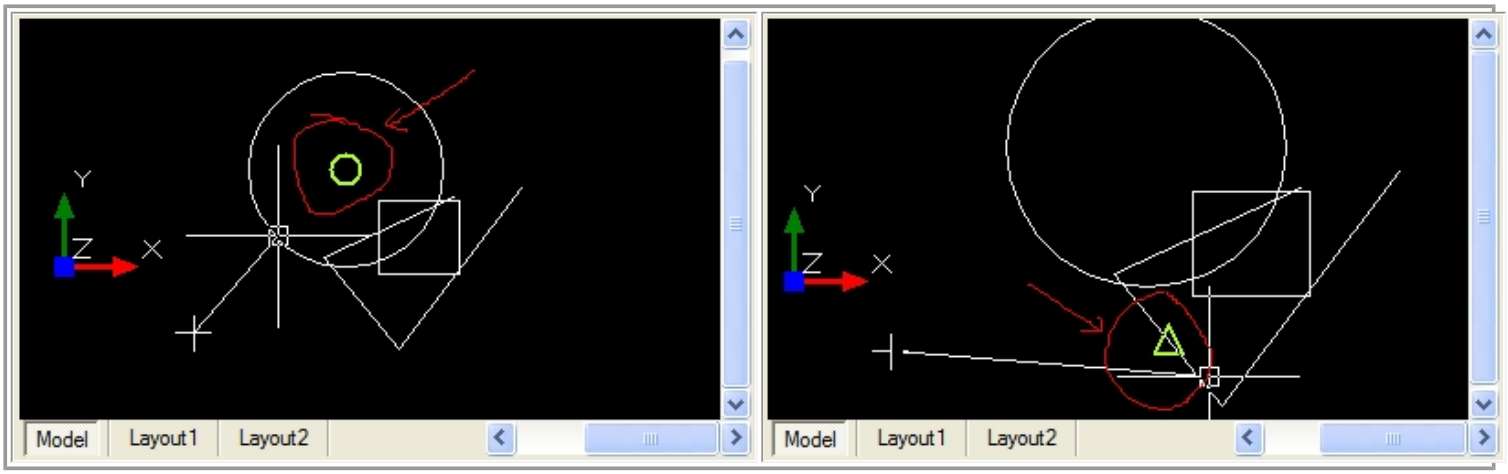
The snap point on an object can be; End-point, Mid-point, Center point, Insertion point, Perpendicular, Nearest, Apparent Intersection, Node (point), Quadrant, Tangent and Intersection.

The OSnap mode remains active until you turn it off, and there is an OSnap indicator in status bar if you choose to set it visible.

You can create a simple project, a form with a vdFramedControl and a button and open adrawing that contains some enities like lines, points inserts etc. Then in the button click code you can set the OSnaps to Mid point plus Center point and start a user command like cmdLine that will ask the use to give the line's points with mouse :

```
vdFC.BaseControl.ActiveDocument.osnapMode = VectorDraw.Geometry.OsnapMode.CEN;
vdFC.BaseControl.ActiveDocument.osnapMode |= VectorDraw.Geometry.OsnapMode.MID;
vdFC.BaseControl.ActiveDocument.CommandAction.CmdLine("USER");
```

The code above will highlight the end points and center points while the cmdLine is active:




```
vdFC.BaseControl.ActiveDocument.osnapMode |= VectorDraw.Geometry.OsnapMode.MID;
vdFC.BaseControl.ActiveDocument.osnapMode ^= VectorDraw.Geometry.OsnapMode.MID;
```

The code above will remove the mid points from OSnaps.

There are also some extra snap features that can be used in command line. These can be used in the command line and override the current OsnapMode value until the user input the point he wants or cancel the command.

These are :

Command	What do they do	Example
	These commands also overide the current OsnapMode and previous typed osnap commands. After user finishes input the OsnapMode returns to the previous value.	*Open the VDFCad sample, in a new drawing add a circle and some lines and in the command line start a user command, like line or polyline and while the program waits for user to input a point :*
.end	This enable the **End Point** object snap	By typing *.end* the end point osnap is activated.
.cen	This enable the **Center Point** object snap	By typing *.cen* the center point osnap is activated.
.ins	This enable the **Insertion Point** object snap	By typing *.ins* the insertion point osnap is activated.
.int	This enable the **Intersection Point** object snap	By typing *.int* the intersection point osnap is activated.
.mid	This enable the **Middle Point** object snap	By typing *.mid* the middle point osnap is activated.
.nea	This enable the **Nearest Point** object snap	By typing *.nea* the nearest point osnap is activated.
.node	This enable the **Node Point** (vdPoint) object snap	By typing *.node* the node (for vdPoints) point osnap is activated.
.per	This enable the **Perpendicular Point** object snap	By typing *.per* the perpendicular point osnap is activated.
.qua	This enable the **Quadratic Point** object snap	By typing *.qua* the quadratic point osnap is activated.
.tang	This enable the **Tangent Point** object snap	By typing *.tang* the tangent point osnap is activated
.x	This enable the **same X** coordinate object snap, after that is waiting for Y,Z	By typing *.x* the user selects the X coordinate of the point and the Y,Z coordinates should be given in the next step. In VDFCad you can type .x and then .end to get the x coordinate of the endpoint of a line and then choose the Y & Z coordinates.
.y	This enable the **same Y** coordinate object snap, after that is waiting for X,Z	By typing *.y* the user selects the Y coordinate of the point and the X,Z coordinates should be given in the next step.

.z	This enable the **same Z** coordinate object snap, after that is waiting for X,Y	By typing *.z* the user selects the Z coordinate of the point and the X,Y coordinates should be given in the next step.
.xy	This enable the **same X & Y** coordinate object snap, after that is waiting for Z	By typing *.xy* the user selects the X & Y coordinates of the point and the Z coordinate should be given in the next step.
.xz	This enable the **same X & Z** coordinate object snap, after that is waiting for Y	By typing *.xz* the user selects the X & Z coordinates of the point and the Y coordinate should be given in the next step.
.yz	This enable the **same Y & Z** coordinate object snap, after that is waiting for X	By typing *.yz* the user selects the Y & Z coordinates of the point and the X coordinate should be given in the next step.
.from	This enable the same X coordinate object snap	By typing *.from* the user can enter a temporary reference or base point from which he can specify an offset to locate the next point.
.par	This enable the **Parallel Point** to a given line segment object snap	By typing *.par* the user selects a line segment and then a parallel snap to this line segment will be enabled.

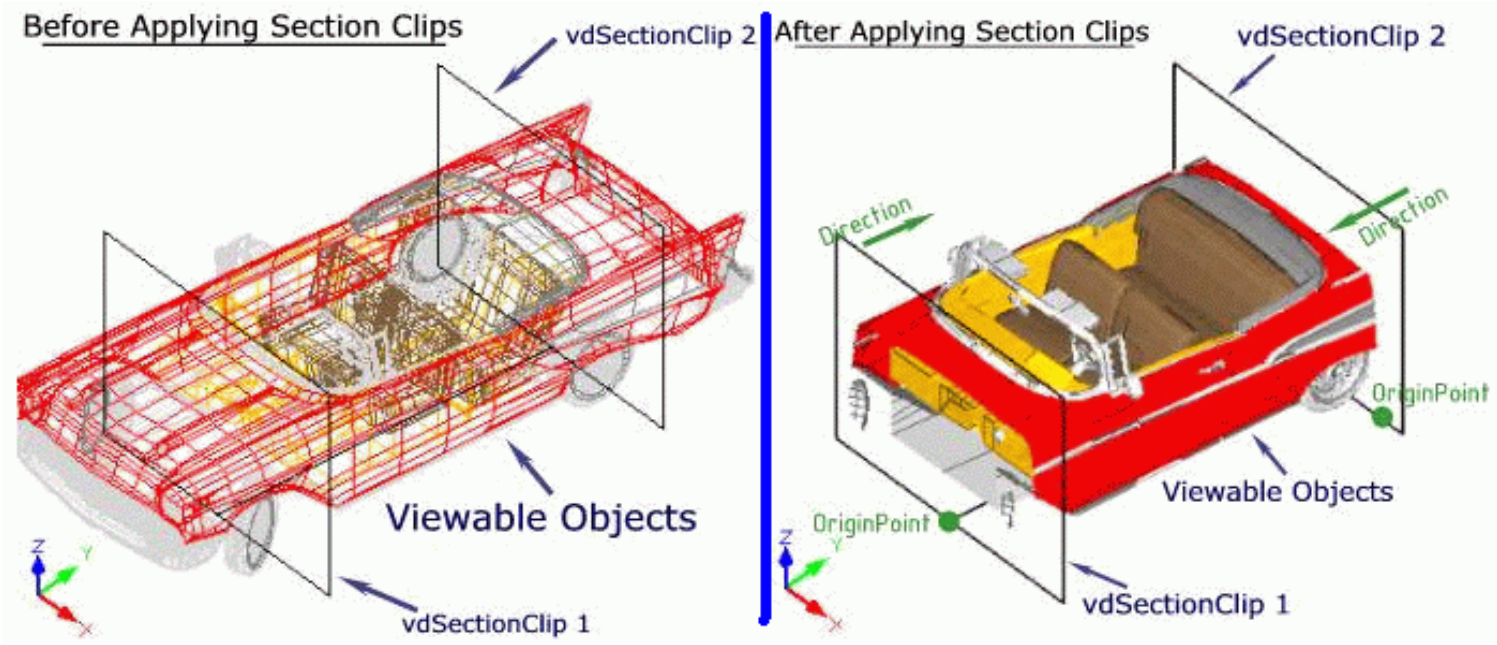
For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "OsnapMode Enumeration (VectorDraw.Geometry)" and "osnapMode Property (vdDocument)".

Getting Started - Section Clipping - vdFramedControl

VectorDraw uses section clipping planes to hide an area of the drawing. This SectionClip defines the side of the drawing that will be visible. The rest will be hidden and this applies to the Render mode. The visible plane is defined in the vdSectionClip object by defining the OriginPoint and the Direction vector. The direction of that vector defines the visible area. The area in the opposite direction will be hidden.

Direction with the OriginPoint define the plane that will do the section clipping. Their values are always in World Coordinating System.

Multiple vdSectionClip objects will show the common viewable area of all. This allows an unobstructed interior view of a 3D drawing by hiding the defined area from view.



Below is a sample code that will draw a sphere and will add two section clips :

```
private void TestClippingSections()
{
    vdFC.BaseControl.ActiveDocument.New(); // We create a new document
    //We change the active pen color so the created (and added to the document by the cmdSphere
function) spheres will take this color.
    vdFC.BaseControl.ActiveDocument.ActivePenColor.ColorIndex = 9;
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdSphere(new VectorDraw.Geometry.gPoint(5, 5),
2.0, 40, 40);
    //Zoom in order to see the object.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.ZoomAll();
    vdFC.BaseControl.ActiveDocument.CommandAction.View3D("RENDER");
    //Redraw the document to see the above changes.
    vdFC.BaseControl.ActiveDocument.Redraw(true);
    MessageBox.Show("Sphere created and is shown in RENDER mode.\r\n We will create a clip (center of
the sphere) to hide the left half of the sphere.");

    //We will create a clip (center of the sphere) to hide the left half of the sphere.
    VectorDraw.Professional.vdObjects.vdSectionClip clip = new
VectorDraw.Professional.vdObjects.vdSectionClip();
    clip.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    clip.Name = "SLICE1";
    clip.Enable = true;
    clip.OriginPoint = new VectorDraw.Geometry.gPoint(5.0, 5.0, 0);
    clip.Direction = new VectorDraw.Geometry.Vector(1, 0, 0); //This is the direction where we want
the visible objects to be.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.Sections.AddItem(clip);
    vdFC.BaseControl.Redraw();

    //We create another clip that will hide the bottom half of the sphere.
    MessageBox.Show("Now we will create a second clip to hide the bottom half of the sphere.");
    clip = new VectorDraw.Professional.vdObjects.vdSectionClip();
    clip.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    clip.Name = "SLICE2";
    clip.Enable = true;
    clip.OriginPoint = new VectorDraw.Geometry.gPoint(5.0, 5.0, 0.0);
    clip.Direction = new VectorDraw.Geometry.Vector(0, 1, 0); //This is the direction where we want
the visible objects to be.
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.Sections.AddItem(clip);
    vdFC.BaseControl.Redraw();

    //We will remove the sections now.
    MessageBox.Show("The sphere is not altered but only a fourth is shown. \r\n Now we will remove
all clippings to show it as a complete sphere again.");
```

```
            vdFC.BaseControl.ActiveDocument.ActiveLayOut.Sections.RemoveAll();
            vdFC.BaseControl.Redraw();
        }
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdSectionClipping Class".

Getting Started - Selections - vdFramedControl

The Selections objects (vdSelection) can be used in order to temporary (not saved in the drawing) create a group of drawing objects (like lines, circles etc) in order to apply properties (like pencolor) or to transform them (like using cmdMove):

1) A simple code:

```
private void butGetSelection_Click(object sender, EventArgs e)
{
    vd.BaseControl.ActiveDocument.New();
    AddSomeEntities(); // This function adds some entities like lines and cirlces
    vd.BaseControl.ActiveDocument.Prompt("Select Entities:");
   vdSelection selset = vd.BaseControl.ActiveDocument.ActionUtility.getUserSelection();
    vd.BaseControl.ActiveDocument.Prompt(null);
   if (selset != null)  {
        MessageBox.Show("You have selected " + selset.Count + " figures");
       foreach (vdFigure var in selset)
                var.HighLight = true;
        vd.BaseControl.ActiveDocument.Redraw(true);    }
}
```

The code above starts an new empty document, call a function to create some entities, and prompts the user to select some entities by choosing a window. If there are entities selected then displays the number of them and set the HighLight property of each of them to true.

2) If we want to move (transform the location of) the entites that are selected then the code can be :

```
private void butGetSelection_Click(object sender, EventArgs e)
{
    vd.BaseControl.ActiveDocument.New();
    AddSomeEntities(); // This function adds some entities like lines and cirlces
    vd.BaseControl.ActiveDocument.Prompt("Select Entities:");
   vdSelection selset = vd.BaseControl.ActiveDocument.ActionUtility.getUserSelection();
    vd.BaseControl.ActiveDocument.Prompt(null);
   if (selset != null)  {
        gPoint pt1 = new gPoint(0, 0, 0);
        gPoint pt2 = new gPoint(2, 2, 0);
        vd.BaseControl.ActiveDocument.CommandAction.CmdMove(selset, pt1 , pt2);
        vd.BaseControl.ActiveDocument.Redraw(true);    }
}
```

The code above will move the selected entities from point {0,0,0} to point {2,2,0}.

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdSelection Class".

Getting Started - SupportPath - vdFramedControl

SupportPath is a very useful property of the VectorDraw Developers Framework. This property specifies the directories in which VDF searches for files when using methods for drawings like insert, file open, images and XRef (External Reference) attach, SHX font files. It is a string with multi paths separated by semicolon < ; > like:

c:\;c:\temp;c:\Program Files/VectorDraw;

The order when VectorDraw searches is :

1) The folder that drawing is placed.
2) The Support path.

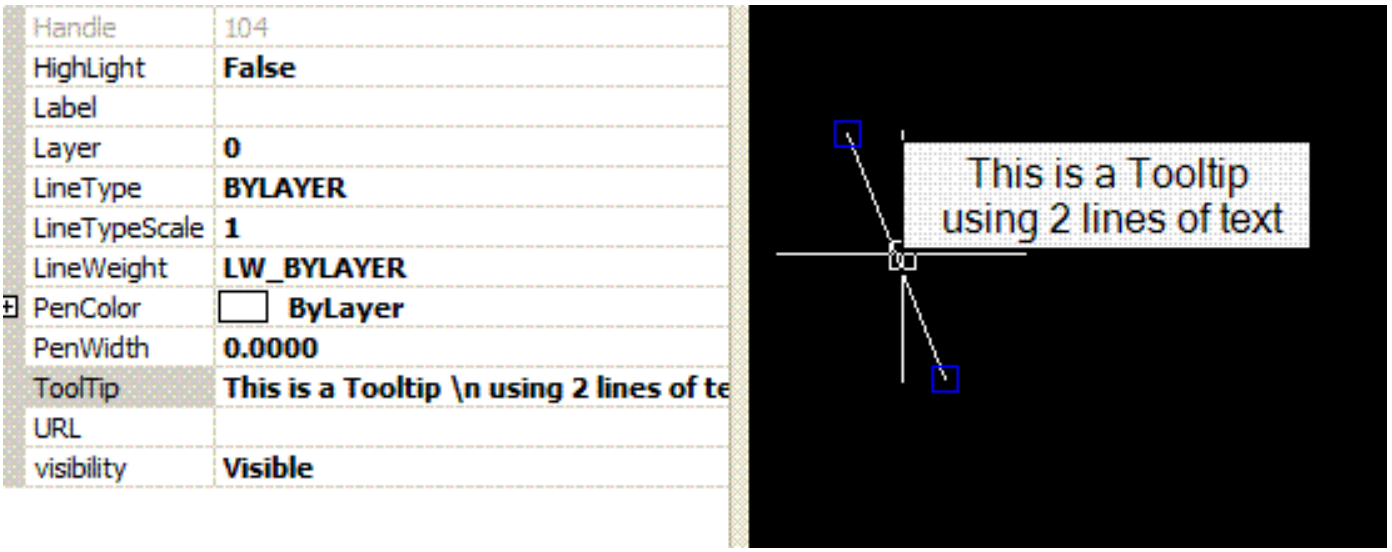
You can get/set the support path property like :

```
// adds the c:\temp folder to the SupportPath property
string SupPath;
SupPath = vdFC.BaseControl.ActiveDocument.SupportPath;// Get
SupPath = SupPath + @"c:\temp;";
vdFC.BaseControl.ActiveDocument.SupportPath = SupPath;// Set
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s):"SupportPath Property".

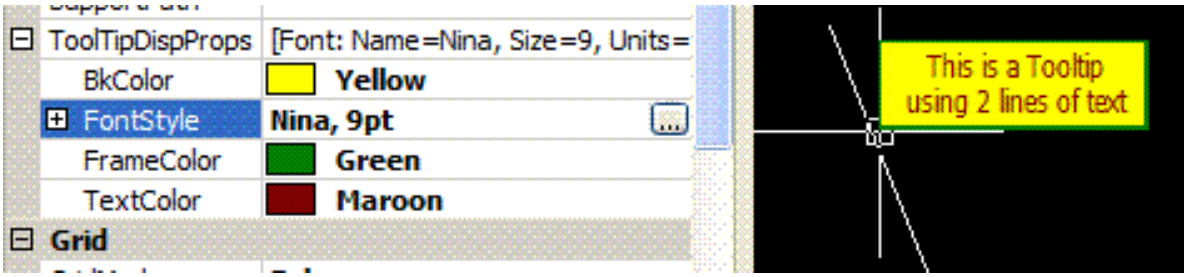
Getting Started - Tooltips & URL - vdFramedControl

The ToolTips is a property that applies to vdFigures, it is used to display a text when the mouse is over this object if EnableToolTips property of vdDocument is true. Like :



The tooltip can be a multi line text like in the above example that "\n" is used to break the text to 2 lines.

You can change the way that the tooltips appear using the properties that you can find under the in "ToolTipDisplayProps Class, Members" topic of our help. Like:



Beside tooltips you can use URLs that can be used in order to guide the user to an external link like a website. URLs apply to an vdFigure if this object has an non-empty URL property and also the EnableUrls Property is true. For example you can set the URL property of a vdFigure to "http://www.vdraw.com" in order to open this web page when the user clicks (holding down CTRL keyboard key) this figure. If CTRL keyboard key is not down then the web page will not open.

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "EnableUrls Property", "EnableToolTips Property", "ToolTip Property" and "URL Property".

Getting Started - Undo & Redo - vdFramedControl

The VectorDraw Developer Framework has Undo and Redo functionality which is unlimited. You can undo the last actions one by one or group them and undo the whole group. The same is for redo previously undo actions. A simple code will be like adding an entity (using for example cmdCircle) and undo the last action. Like :

```
private void Button_cmdCircle(object sender, EventArgs e)
{ //This will draw one user-defined (center point and radius) circle. This
code must be run before the UNDO code
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(null, null);
}
private void button_Undo(object sender, EventArgs e)
{ //This will Undo the last action if exists. cmdCircle code must run before
this.
    vdFC.BaseControl.ActiveDocument.CommandAction.Undo("");
}
```

After the Button_cmdCircle code run and the button_Undo run then the Redo can run and bring back the circle, like :

```
private void button_Redo(object sender, EventArgs e)
{ //This will REDO the last action if exists
    vdFC.BaseControl.ActiveDocument.CommandAction.Redo();
}
```

Except this simple undo/redo calls, you can also group some code so this can be undo/redo as a group. See the code below :

```
private void Button_Circles(object sender, EventArgs e)
{ //This will draw 4 circles inside a undo Group
    vdFC.BaseControl.ActiveDocument.CommandAction.Undo("BEGIN"); //The Undo-
Group begins
    VectorDraw.Geometry.gPoint pt = new VectorDraw.Geometry.gPoint(1.0d,
1.0d, 0.0d);
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(pt, 0.4d);
    pt.x = 2.0d; pt.y = 2.0d;
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(pt, 0.4d);
    pt.x = 3.0d; pt.y = 3.0d;
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(pt, 0.4d);
    pt.x = 4.0d; pt.y = 4.0d;
    vdFC.BaseControl.ActiveDocument.CommandAction.CmdCircle(pt, 0.4d);
    vdFC.BaseControl.ActiveDocument.CommandAction.Undo("END");//The Undo-
Group ends here
}
```

After this code run, 4 circles will be created. One undo call and these 4 circles will disappear like they were never drawn. After this undo, one Redo will bring back thes four circles.

Beside this, you can also clear the undo & redo history, and after that previous undo/redo actions will not be available. You can do this using a code like :

```
private void button_ClearUndo(object sender, EventArgs e)
{ //This will Clear the Undo & Redo history. No Undo/redo will be available
from this point
   vdFC.BaseControl.ActiveDocument.ActiveLayOut.UndoHistory.Clear();
}
```

You can also get the undo & redo actions count using a code like:

```
private void button_GetURCount(object sender, EventArgs e)
{ //This will write to a textbox the undo/redo that are available
   textBox1.Text = " Undo : " +
vdFC.BaseControl.ActiveDocument.UndoHistory.UndoStackLength.ToString() + "
Redo : " +
vdFC.BaseControl.ActiveDocument.UndoHistory.RedoStackLength.ToString();
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "Undo Method (vdCommandAction)".

Getting Started - Units (Linear - Angle) - vdFramedControl

Coordinates are being expressed in drawing units (D.U.). Drawing units are not expressing particular units (meters, inches etc). In this part user have to make some assumptions in order to define that the coordinates of the drawing mean particular units (meters, inches etc).

For example: For a mechanical drawing we can make the assumption for example: where one drawing unit defines one millimeter(1 D.U. =1mm). For a architectural/technical drawing we can make the assumption for example: where one drawing unit defines one meter(1 D.U. =1m). This can be very helpful in designing, dimensioning, retrieving information from the drawing (distances, area calculations etc)

You can specify the type of the current unit of measurement (linear and angle) and the precision for the current units as also the current angle format and the precision for the current angle display. The type of measurement can be:

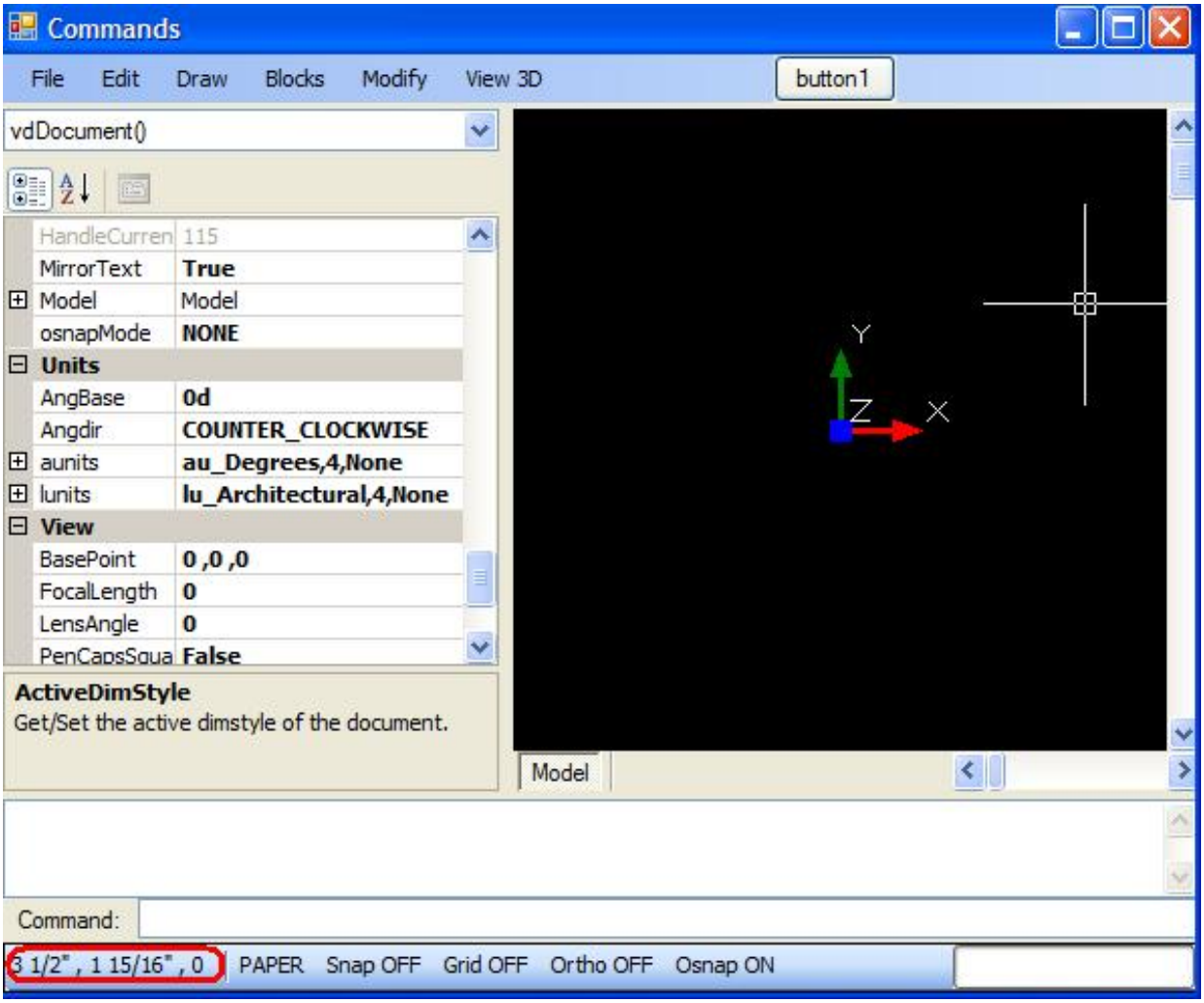
Linear LUNITS		**Angle AUNITS**	
Scientific like 1.6449E+001	lu_Scientific = 1	Decimal degrees	au_Decdegrees = 0
Decimal like 16.4492	lu_Decimal = 2	Degrees/minutes/seconds	au_Degrees = 1
Engineering like 1' - 4.4492"	lu_Engineering = 3	Gradians	au_Gradians = 2
Architectural like 1' - 4 7/16"	lu_Architectural = 4	Radians	au_Radians = 3
Fractional like 16 7/16	lu_Fractional = 5	Surveyor	au_Surveyor = 4
Windows desktop (It takes the settings of the regional settings)	lu_windesk = 6		

These Properties change only the display and **not** the values which are in Drawing Units.

Running a code like :

```
private void button1_Click(object sender, EventArgs e)
{
    vdFC.BaseControl.ActiveDocument.ActiveLayOut.lunits.UType =
VectorDraw.Geometry.LUnits.LUnitType.lu_Architectural;
   vdFC.BaseControl.ActiveDocument.ActiveLayOut.aunits.UType =
VectorDraw.Geometry.AUnits.AUnitType.au_Radians;
}
```

You will get a display like :



The Angular & Linear units also apply to the vdDimensions and Dimension Styles. See the following code :

```
private void button2_Click(object sender, EventArgs e)
{
    vdFC.BaseControl.ActiveDocument.New();

vdFC.BaseControl.ActiveDocument.CommandAction.CmdDim(VectorDraw.Professional.Constants.VdConstDimType.dim_Angular,
"USER", "USER", 0);
    VectorDraw.Professional.vdFigures.vdDimension dime;
    dime =
vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities[vdFC.BaseControl.ActiveDocument.ActiveLayOut.Entities.Count
- 1] as VectorDraw.Professional.vdFigures.vdDimension;
   if (dime != null)
    {
        MessageBox.Show("The Angular units will now change from au_Degrees to au_Radians.");
        dime.Invalidate();
        dime.DimAunit = VectorDraw.Geometry.AUnits.AUnitType.au_Radians;
        dime.Update();
        dime.Invalidate();
    }
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "LUnits Class" and "AUnits Class".

Getting Started - XProperties - vdFramedControl

The XProperties are used to store extended data in a object. For example you might add the part number as a string in a vdInsert object that represents graphically the part. This can be connected to a database with more information for this object.

For example if you like the user to input a point in the drawing the you can call a code like :

```
//We will add a circle (assume that the circle represents a part) to the
model with a xproperty in it's collection that will represent the part number
in our database.
VectorDraw.Professional.vdFigures.vdCircle circle = new
VectorDraw.Professional.vdFigures.vdCircle();
circle.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
circle.setDocumentDefaults();
circle.Radius = 5.0;
circle.Center = new VectorDraw.Geometry.gPoint(2, 2, 0);
//Add string value xproperty that will represent in our applicatio the part
number of this part.
VectorDraw.Professional.vdObjects.vdXProperty xprop = new
VectorDraw.Professional.vdObjects.vdXProperty();
xprop.Name = "Part No.";
xprop.PropValue = "FG 127-34566";
circle.XProperties.AddItem(xprop);
vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(circle);
vdFC.BaseControl.ActiveDocument.Redraw(true);
```

You can also see topic "vdLinetype Class" in our VDF help (VDF.CHM).

The XProperties can also be stored in other objects than graphical entities. The objects can be vdLayouts, vdLayers etc.

There are also some Xproperties that are dynamic and are binded to the geometrical information of the objects. For example you can have the length of the line binded to the line and when this line is stretched or scaled the value of the XProperty is changed accordingly. See the code below:

```
VectorDraw.Professional.vdObjects.vdXProperty xprop = new
VectorDraw.Professional.vdObjects.vdXProperty();
VectorDraw.Professional.vdFigures.vdLine line = new
VectorDraw.Professional.vdFigures.vdLine();
line.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
line.setDocumentDefaults();
line.StartPoint = new VectorDraw.Geometry.gPoint(0, 0, 0);
line.EndPoint = new VectorDraw.Geometry.gPoint(2, 2, 0);
xprop.Name = "ThisLineLength";
xprop.PropValue = line.EndPoint.Distance3D(line.StartPoint);
xprop.TransformID =
VectorDraw.Professional.vdObjects.vdXProperty.TransformationType.WorldSpaceDist;
line.XProperties.AddItem(xprop);
vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(line);
vdFC.BaseControl.ActiveDocument.Redraw(true);
MessageBox.Show("Line's length : " + line.Length().ToString() + " Line's
XProperty value : " + xprop.PropValue + "\n  Line will now be scaled.");
vdFC.BaseControl.ActiveDocument.CommandAction.CmdScale(line, new
VectorDraw.Geometry.gPoint(0, 0, 0), 1.5);
MessageBox.Show("Line is scaled. \n  Line's length : " +
line.Length().ToString() + " Line's XProperty value : " +
xprop.PropValue);
line.Invalidate();
vdFC.BaseControl.ActiveDocument.Redraw(true);
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "vdXProperty Class".

Getting Started - External Reference (Xrefs) - vdFramedControl

External Reference files (XRefs) are drawings that can be attached for reference in a drawing. When you attach a drawing as an xref, you link that referenced drawing to the current drawing; any changes to the referenced drawing are displayed in the current drawing when it is opened or reloaded. A drawing can be attached as an XRefs to multiple drawings at the same time and multiple drawings can be attached as referenced drawings to a single drawing.

Note that the objects in the XRefs attached to the current drawing are model space objects only. You can insert the xref into the current drawing in either model space or paper space. The XRefs can be attached at any location, scale, and rotation. The XRefs is a vdBlock object that is inserted (instance) in the drawing as a vdInsert object.

Below is a sample code that will add an existing drwaing as an XRef file. This should be defined as a vbBlock with ExternalReferencePath property :

```
private void AddExternalReferences()   {
//We will add a vdblock object as an external reference to the blocks dialog.
   string path = Application.ExecutablePath.Substring(0,
Application.ExecutablePath.LastIndexOf('\\')) + "\\..\\..\\wmap.vdml";//the
name and path of the file that will be added as XRef
    VectorDraw.Professional.vdPrimaries.vdBlock xref = new
VectorDraw.Professional.vdPrimaries.vdBlock();
    xref.SetUnRegisterDocument(vdFC.BaseControl.ActiveDocument);
    xref.setDocumentDefaults();
    xref.Name = "wmap";
    xref.ExternalReferencePath = path;
   //With update the file is opened and the document of the file is added to
the External references of the file.
   xref.Update();
    vdFC.BaseControl.ActiveDocument.Blocks.AddItem(xref);
}
```

After this the XRef (vdBlock) should be inserted in the drawing. See the sample code below :

```
private void AddReferencesInserts()  {
    //Now we will add the vdinsert object that will show the external
reference that we created.
    VectorDraw.Professional.vdFigures.vdInsert ins = new
VectorDraw.Professional.vdFigures.vdInsert();
    ins.SetUnRegisterDocument(vdFramedControl1.BaseControl.ActiveDocument);
    ins.setDocumentDefaults();
    //We check if the block exists and then give it to the insert.
    VectorDraw.Professional.vdPrimaries.vdBlock blk =
vdFC.BaseControl.ActiveDocument.Blocks.FindName("wmap");
    if (blk != null)
    {
        ins.Block = blk;
        vdFC.BaseControl.ActiveDocument.Model.Entities.AddItem(ins);
    }

    //Note that the Same operation like above could have been done with the
following function cmdXref.
   //string path = Application.ExecutablePath.Substring(0,
Application.ExecutablePath.LastIndexOf('\\')) + "\\..\\..\\wmap.vdml";
    //vdFC.BaseControl.ActiveDocument.CommandAction.CmdXref("A", path, new
VectorDraw.Geometry.gPoint(), new double[] { 1.0, 1.0 }, 0.0, 0);
   //Zoom the model to show the entity.
    vdFC.BaseControl.ActiveDocument.Model.ZoomExtents();
    vdFC.BaseControl.ActiveDocument.Redraw(true);
}
```

For more detailed information about properties/methods/events etc of the above object(s), you can also check our vdf.chm help file, topic(s): "ExternalReferencePath Property".