BasicVideo 5.0

Visual C++ Quick Start

www.openwire.org
www.mitov.com

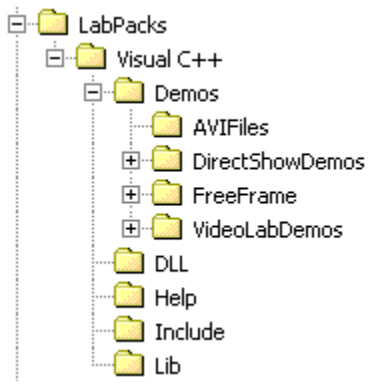
Index

Installation

VideoLab comes with an installation program. Just start the installation by double-clicking on the Setup.exe file and follow the installation instructions.

Where is BasicVideo

After the installation VideoLab is located under a single root directory. The default location is C:\Program Files\LabPacks\Visual C++. During the installation the user has the option to select alternative directory.
Here is how the directory structure should look like after the installation:



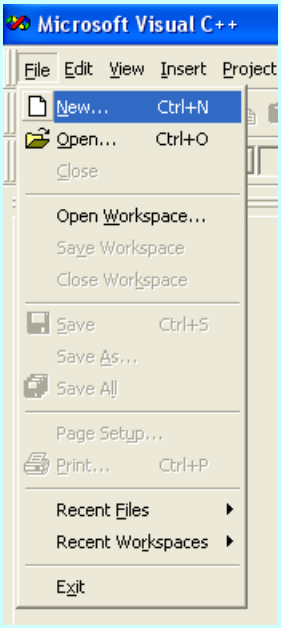
Under the VideoLabDemos and DirectShowDemos directories are located the demo files. The help files and the documentation are located under the Help directory. The DLL directory contains the redistributable DLL files. The header files needed for your projects are located under the Include directory. The Release and Debug version of the library is located under the Lib directory.
It is a great idea to start by opening and compiling the demo files. The demo projects ware designed with Visual C++ 6.0. They can be opened and compiled under Visual C++.NET as well, in this case the IDE will create the necessary solution files.

Creating a new BasicVideo project in Visual C++

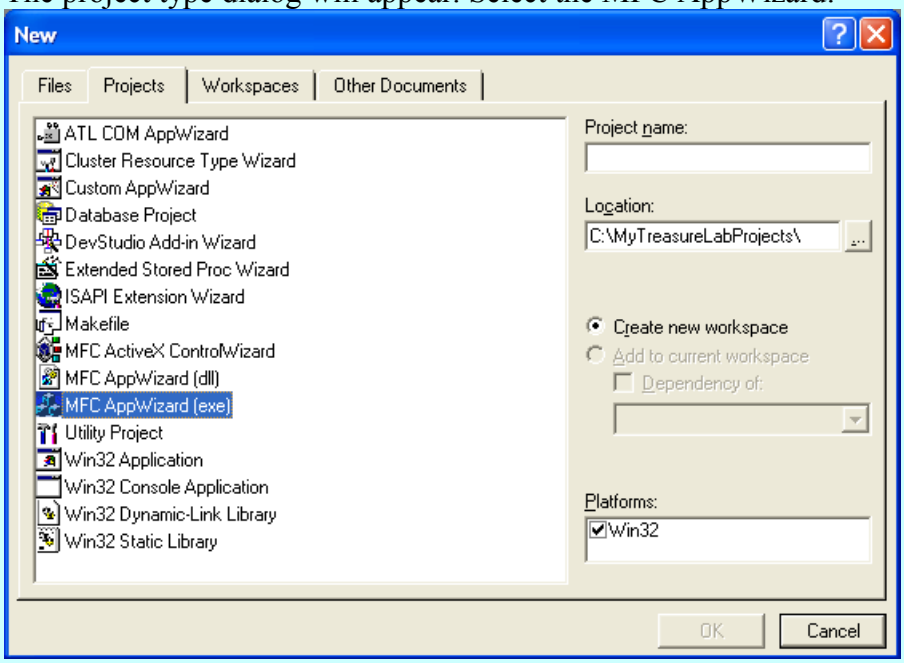
All of the examples in this manual start with creating a MFC Dialog based project. This is not a VideoLab requirement, but using the resource editor to design the application makes writing the examples much easier.
The following chapters will assume that you have created the project and will teach you how to add specific VideoLab functionality.

Visual C++ 6.0:

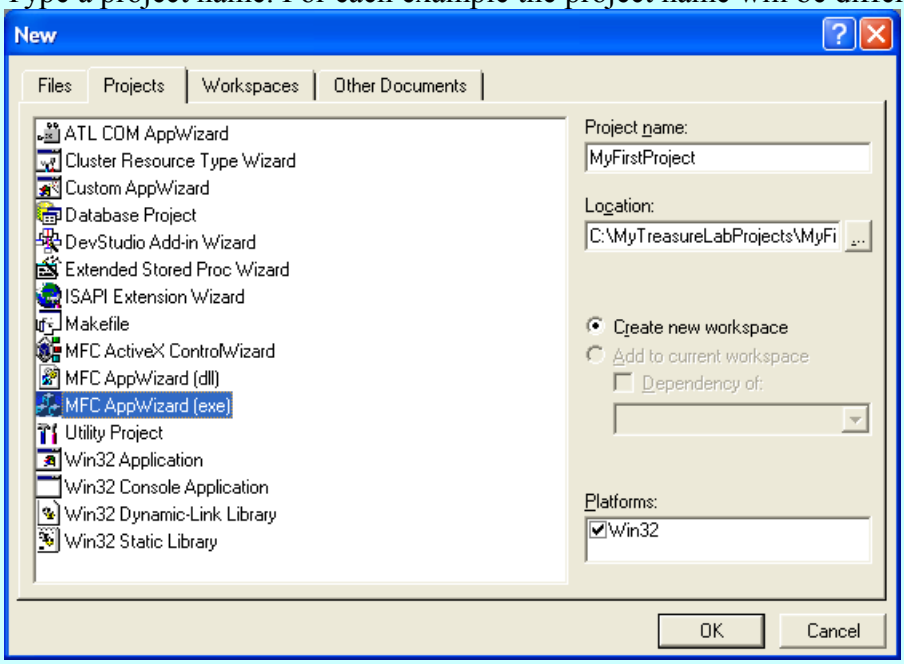
Start by creating a new project.
From the VC++ menu, select | File | New…|



The project type dialog will appear. Select the MFC AppWizard:

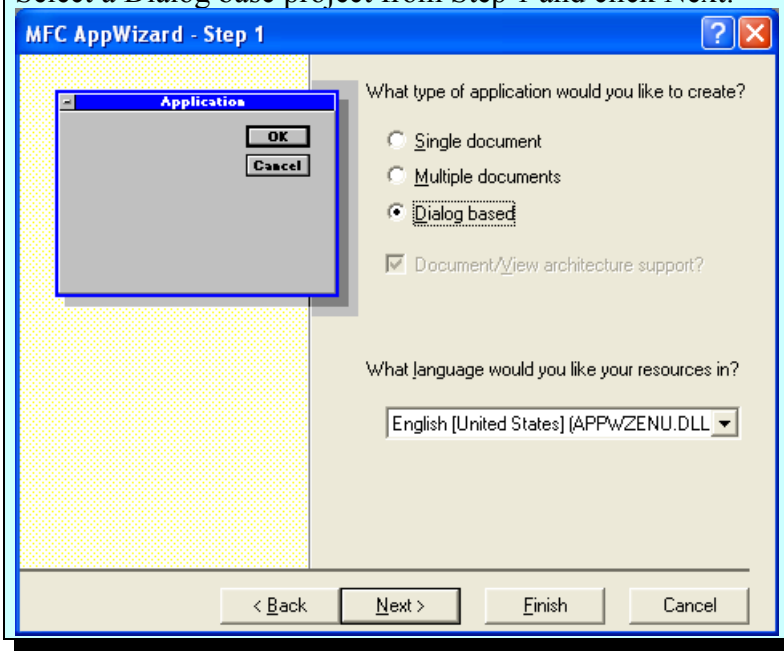


Type a project name. For each example the project name will be different:

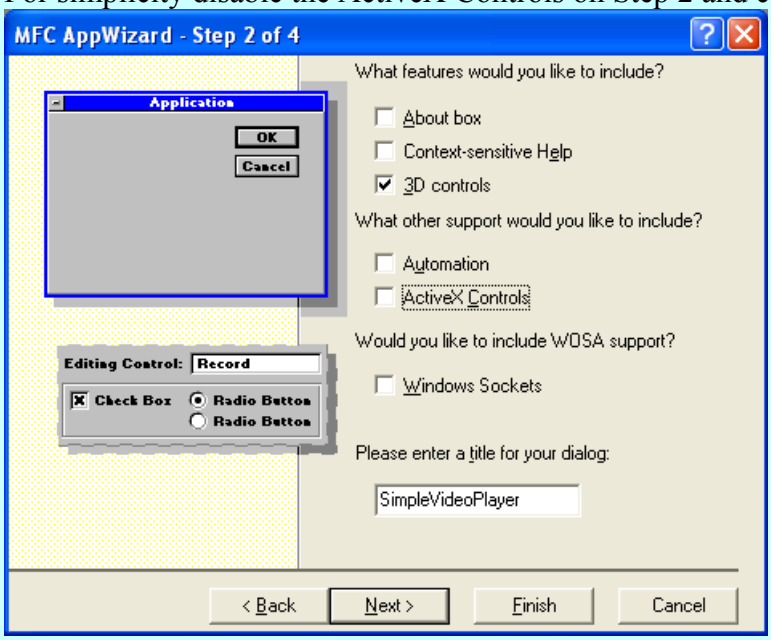


Click OK.

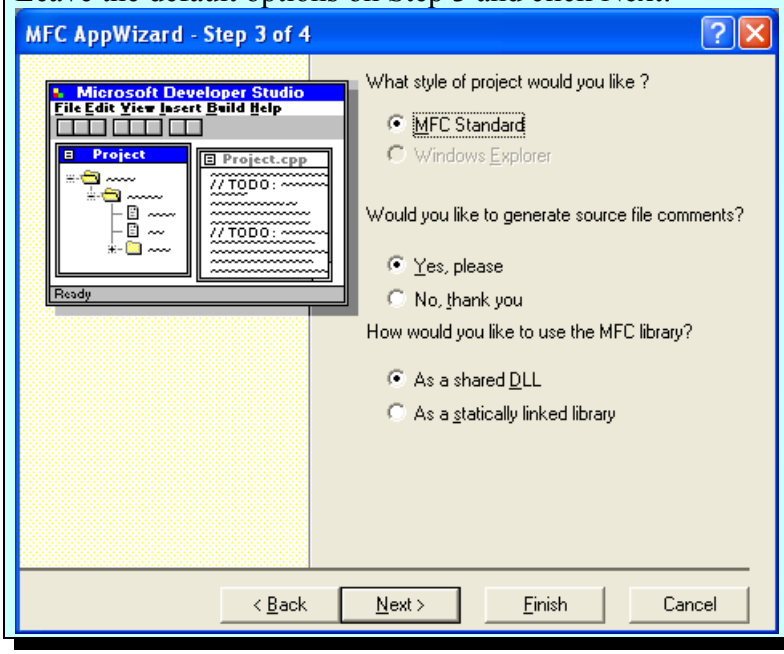
Select a Dialog base project from Step 1 and click Next:



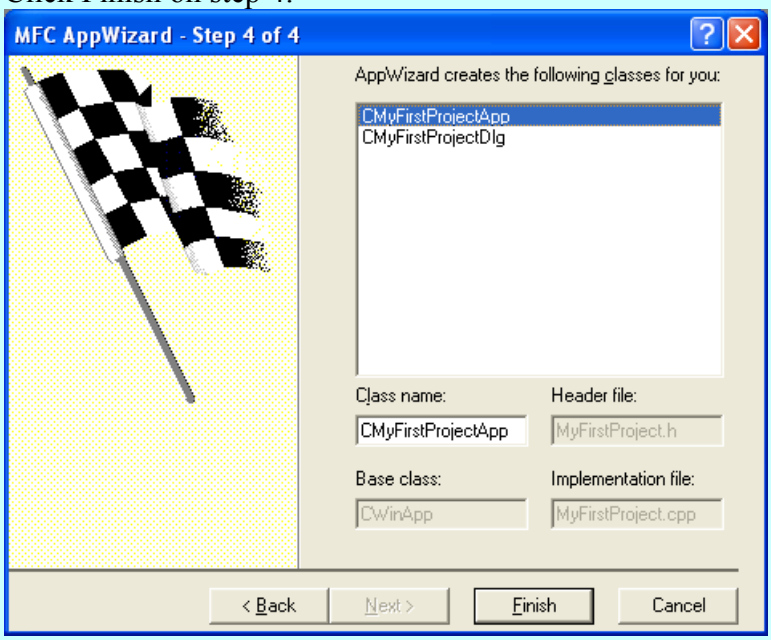
For simplicity disable the ActiveX Controls on Step 2 and click Next:



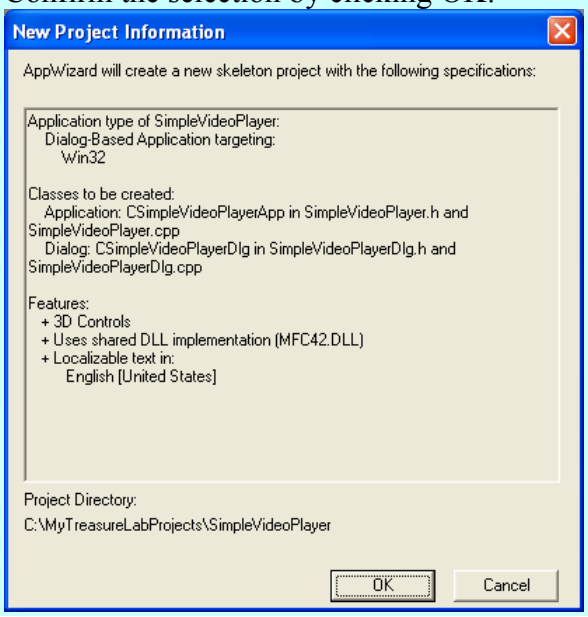
Leave the default options on Step 3 and click Next:



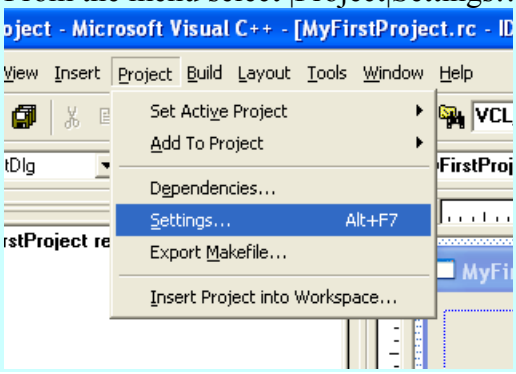
Click Finish on step 4:



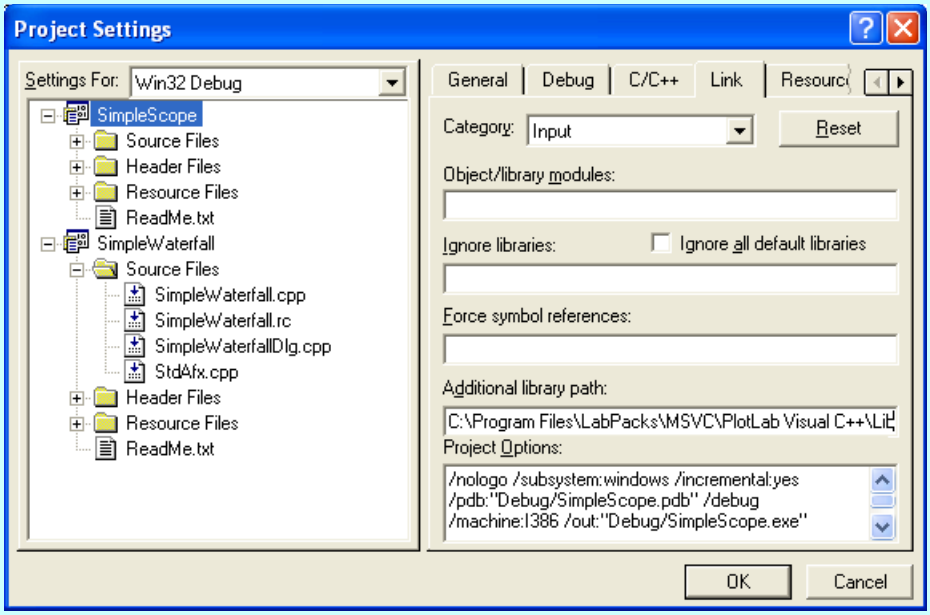
Confirm the selection by clicking OK:



At this point you should have a new project created.
From the menu select |Project|Settings…| :

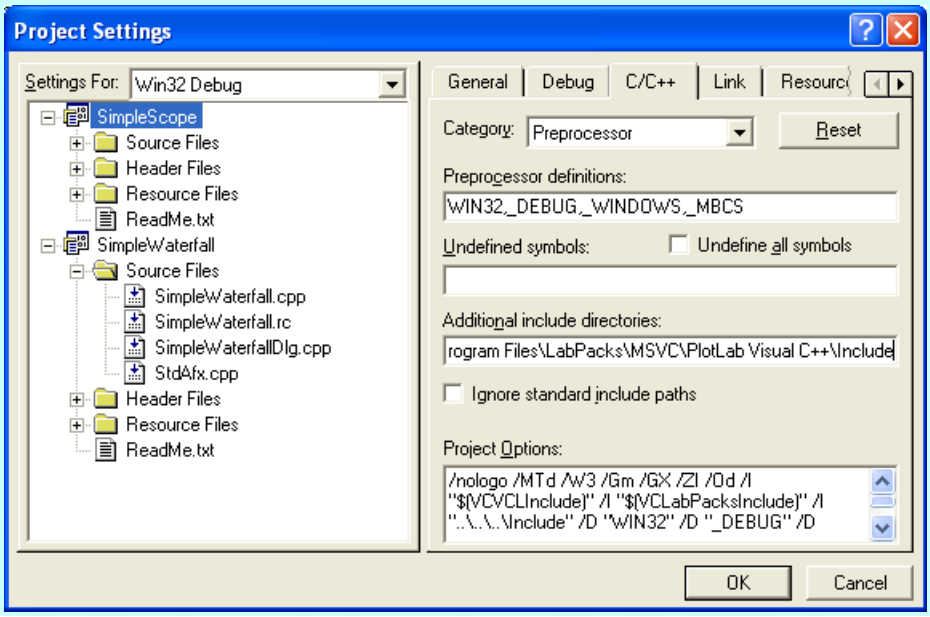


In the Project Settings dialog select the | Link | tab and in the “. Switch to the “Input” cathegory. In the “Additional library path:” edit box add the path to the library files. If you have followed the default installation it should be located at C:\Program Files\LabPacks\Visual C++\Lib:



Switch to the |C/C++| tab.
In the “Additional include directories:” edit box add the path to the header files. If you have followed the default installation they should be located at C:\Program

Files\LabPacks\Visual C++\Include:

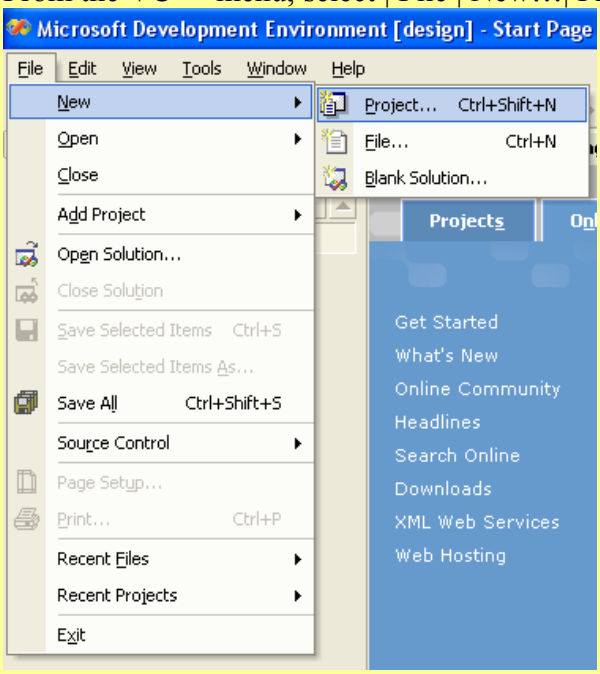


Click OK.

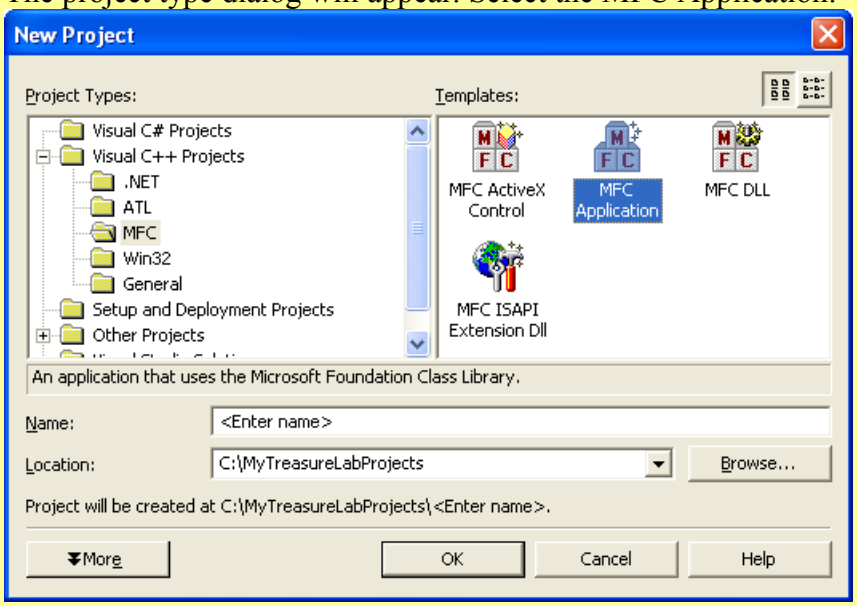
Now you have fully configured project, and you can start writing the actual code.

Visual C++ 2003:

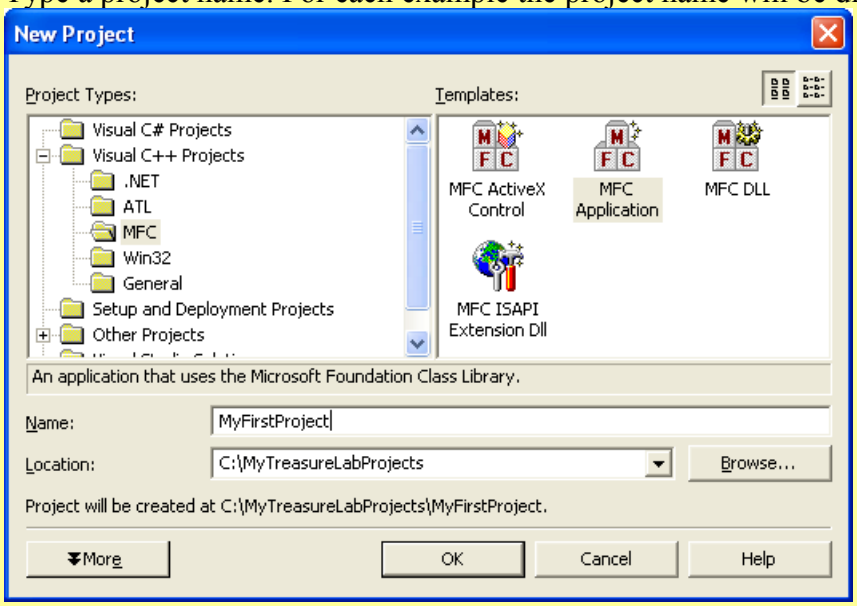
Start by creating a new project.
From the VC++ menu, select | File | New…| Project… |



The project type dialog will appear. Select the MFC Application:

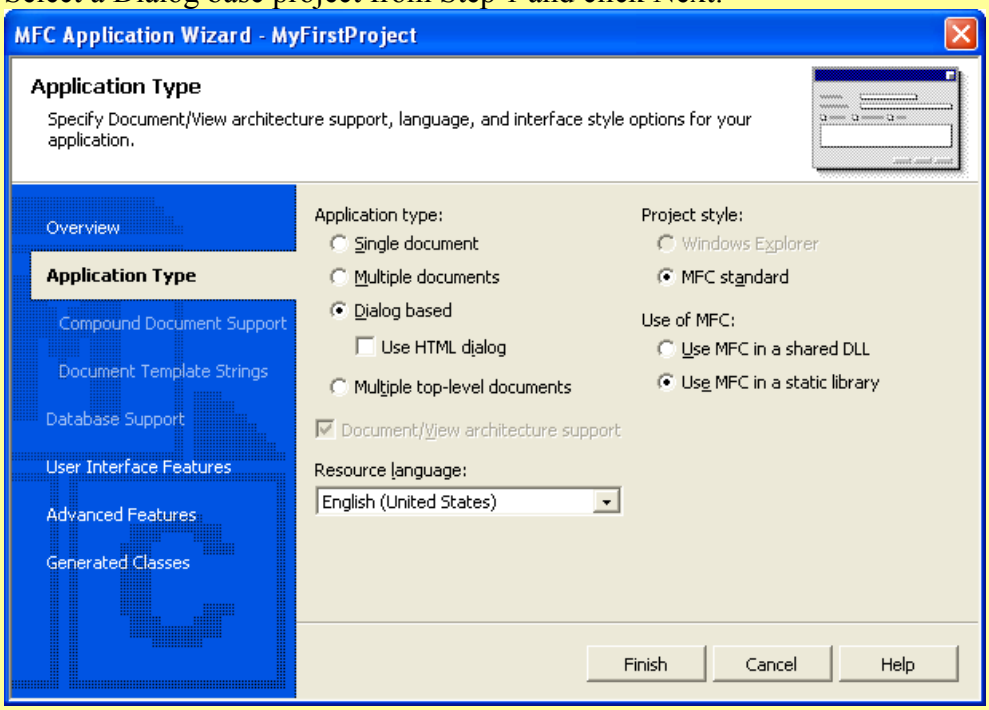


Type a project name. For each example the project name will be different:

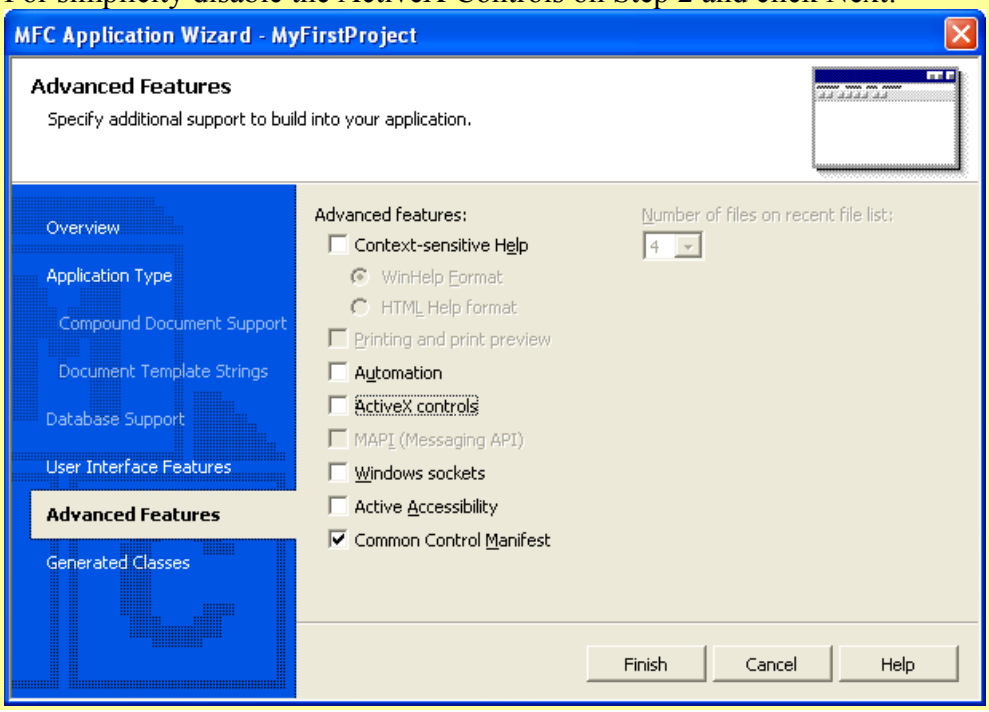


Click OK.

Select a Dialog base project from Step 1 and click Next:

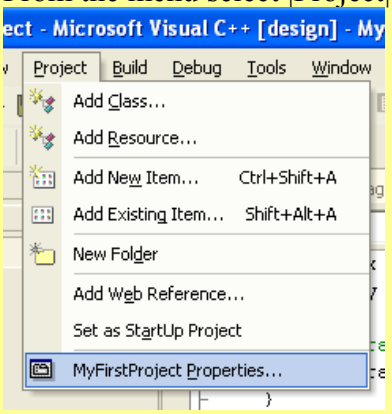


For simplicity disable the ActiveX Controls on Step 2 and click Next:

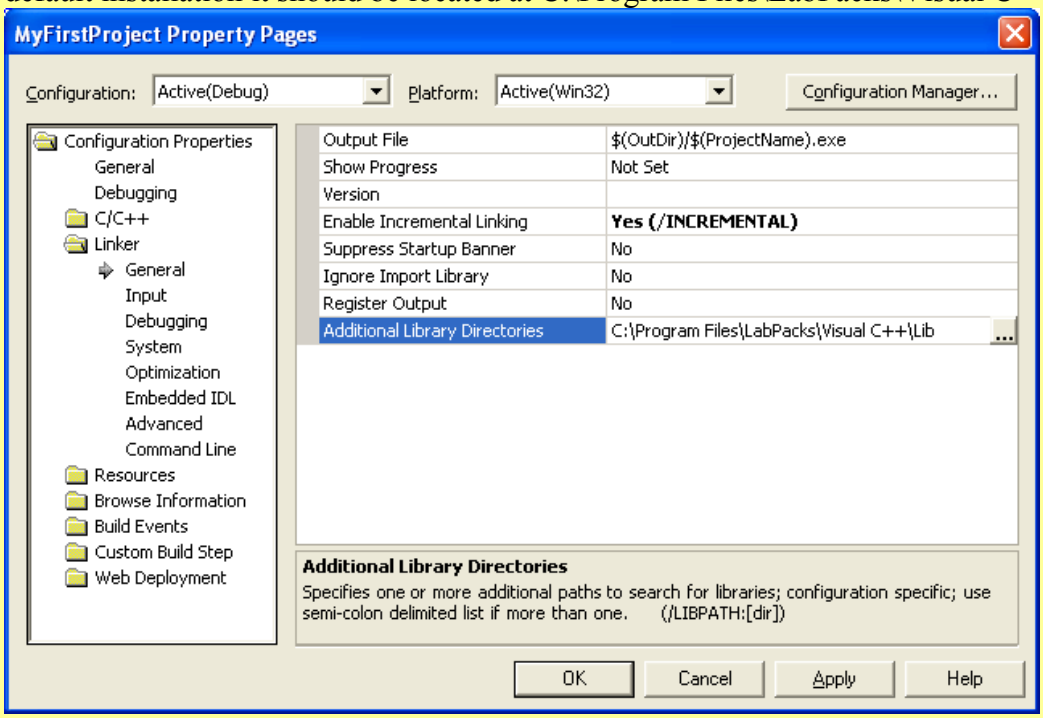


Click Finish.

At this point you should have a new project created.
From the menu select |Project|Settings…| :

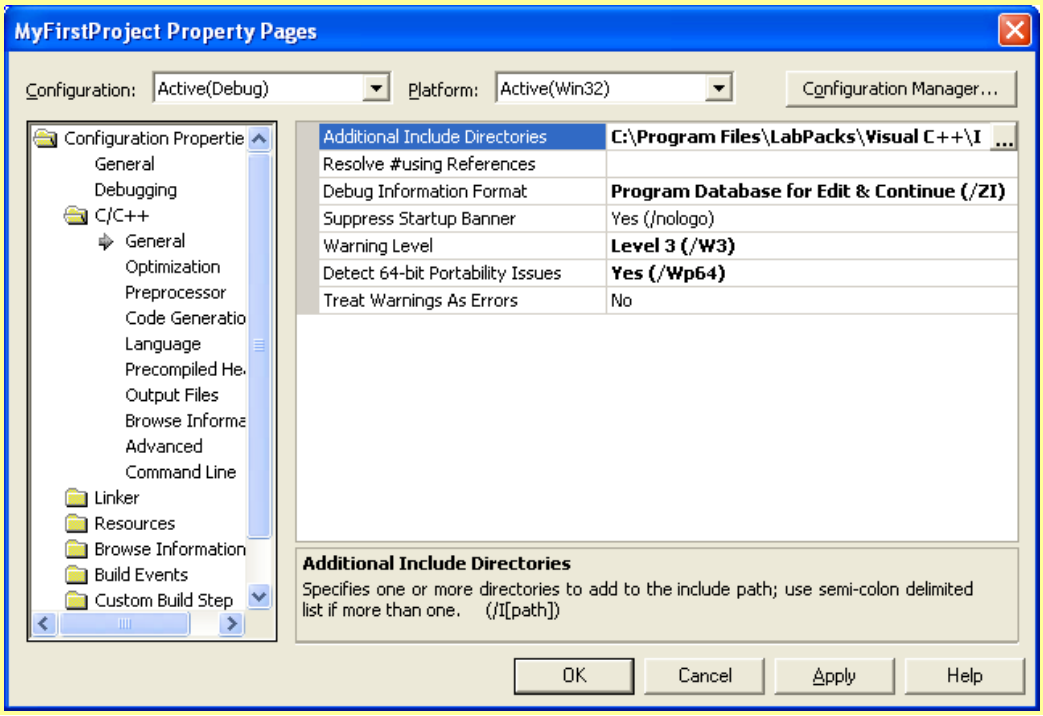


In the Project Property dialog select the Linker General page. In the "Additional library directories:" edit box add the path to the library files. If you have followed the default installation it should be located at C:\Program Files\LabPacks\Visual C++\Lib:



Switch to the C/C++ General page.
In the "Additional include directories:" edit box add the path to the header files. If you have followed the default installation they should be located at C:\Program

Files\LabPacks\Visual C++\Include:



Click OK.

Now you have fully configured project, and you can start writing the actual code.

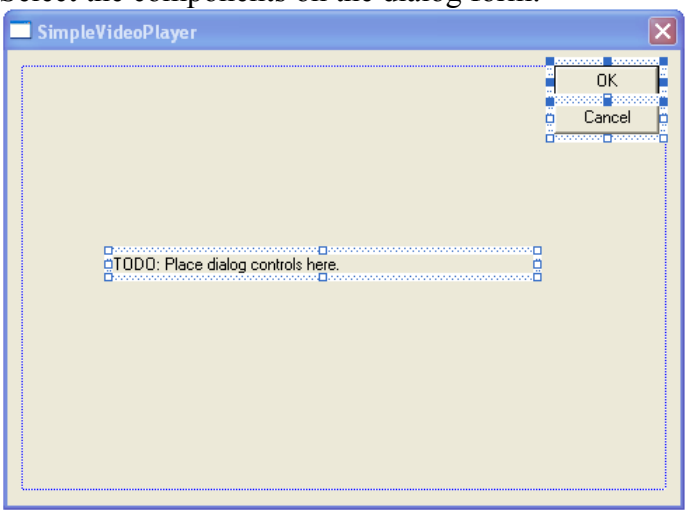
Why some of the examples don't work?

Video lab is a unique library that supports both the Win32 API's AVIFile (VFW) functions (ACM) and DirectShow. You as a developer have the ultimate choice to use either the Win32 API or DirectShow components or both at the same time.
The advantage of the Win32 API components is that hey will work on any Windows 95 and up system out of the box, however they are much less capable than the DirectShow components, and should be avoided if not necessary.
The advantage of the DirectShow components is that they will use the latest and greatest capability of DirectShow, the latest video camera devices, and TV Tuners, but they require DirectShow 9.0 or higher to be installed in order to work.
If you don't have DirectX 9.0 or higher installed on your system, you will not be able to use see the DirectShow examples working.

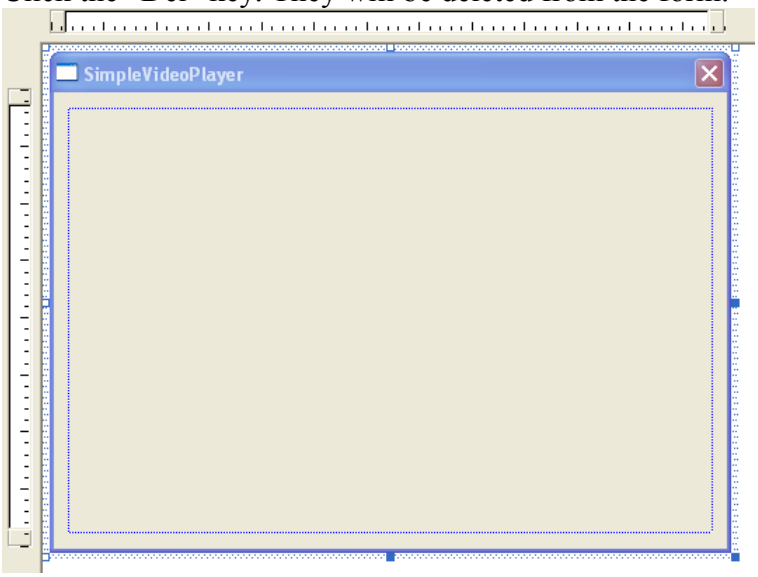
Creating a simple video player using Win32API Components

Create and setup a new project named SimpleVideoPlayer as described in the "Creating a new VideoLab project in Visual C++" chapter.

Select the components on the dialog form:



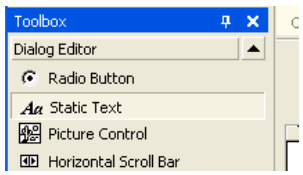
Click the “Del” key. They will be deleted from the form:



From the controls toolbar select a “Static Text” control:

VC++ 6:

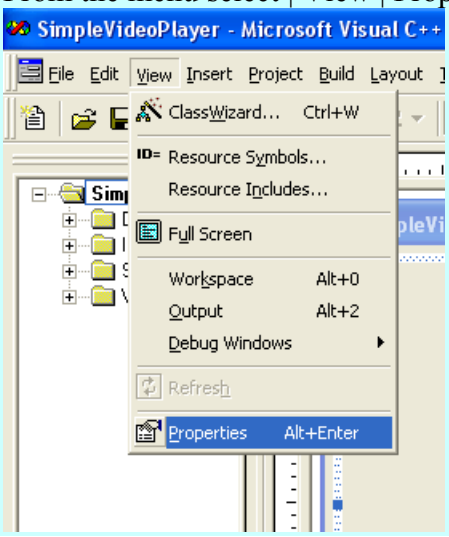
VC2003/2005:



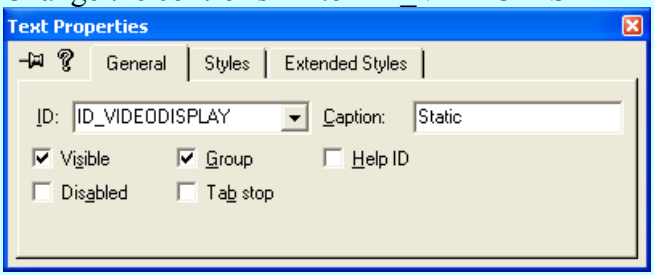
Place the control on the form:

In Visual C++ 6.0:

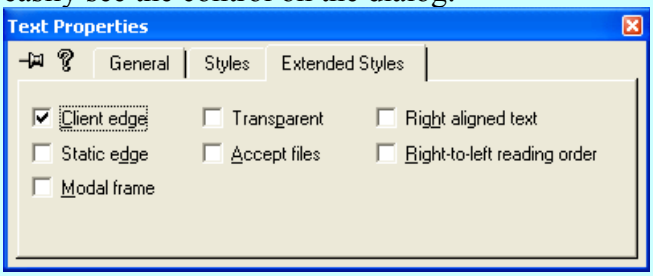
From the menu select | View | Properties |:



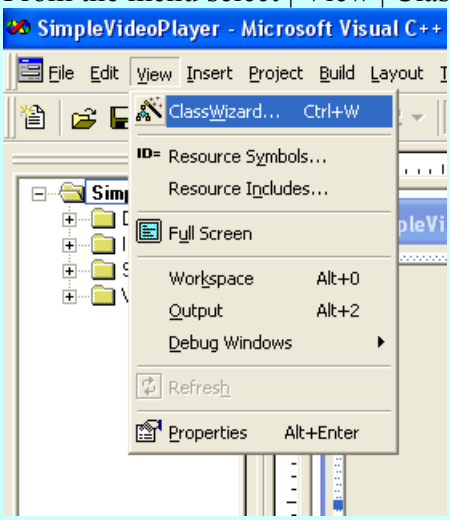
Change the control's ID to "ID_VIDEODISPLAY":



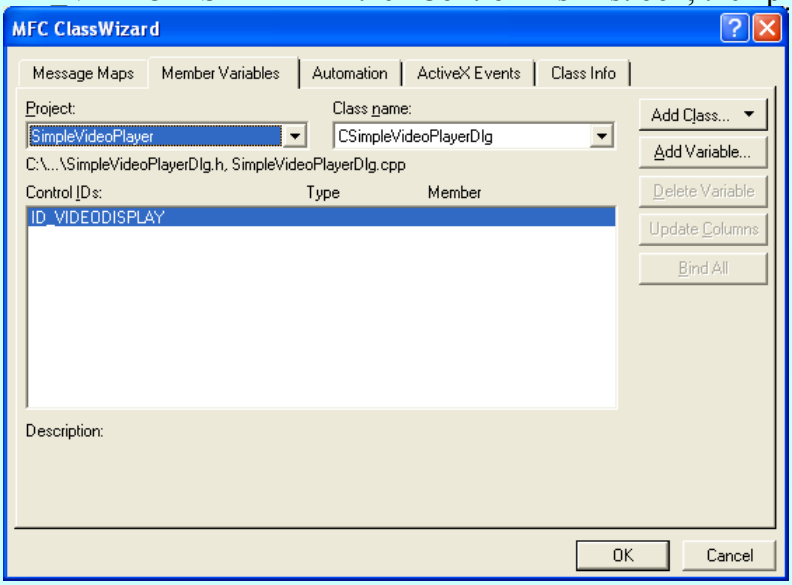
Switch to the “Extended Styles” tab and check the “Client edge” property so you can easily see the control on the dialog:



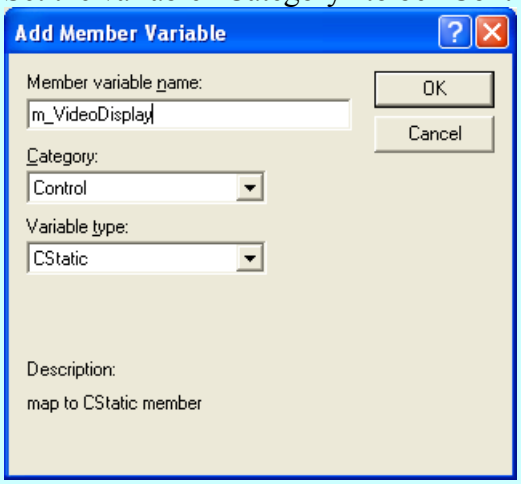
From the menu select | View | ClassWizard… |:



In the “ClassWizard” select the “Member Variables” tab and select the “ID_VIDEODISPLAY” in the “Control IDs” list box, then press “Add Variable…”:

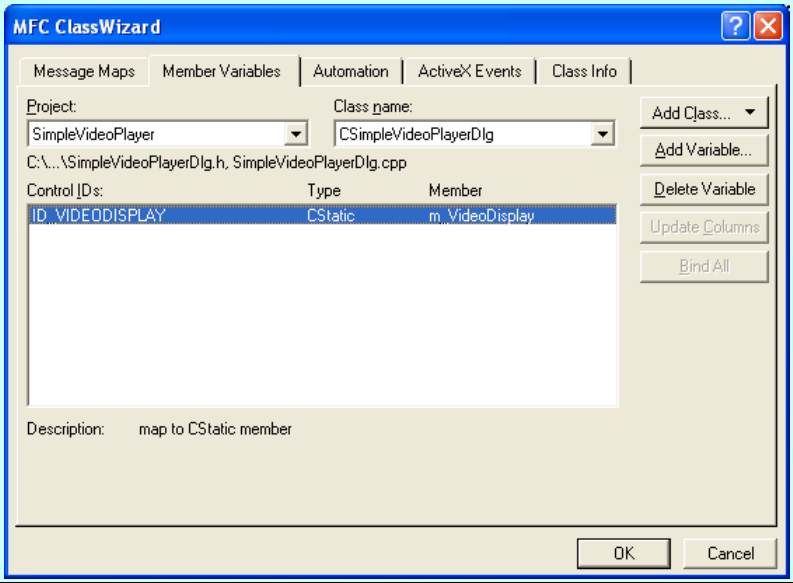


Set the variable “Category” to be “Control”, and set the name to be m_VideoDisplay:



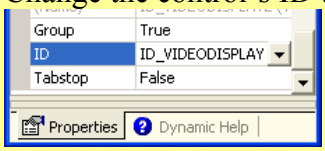
Click OK.

In the “ClassWizard” click OK

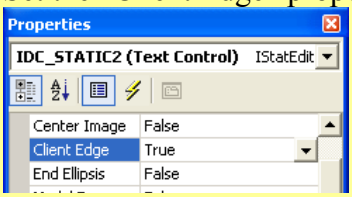


In Visual C++ 2003/2005:

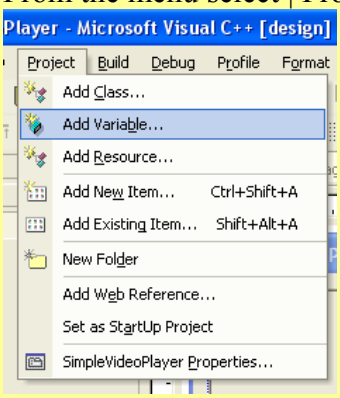
Change the control’s ID to “ID_VIDEODISPLAY”:



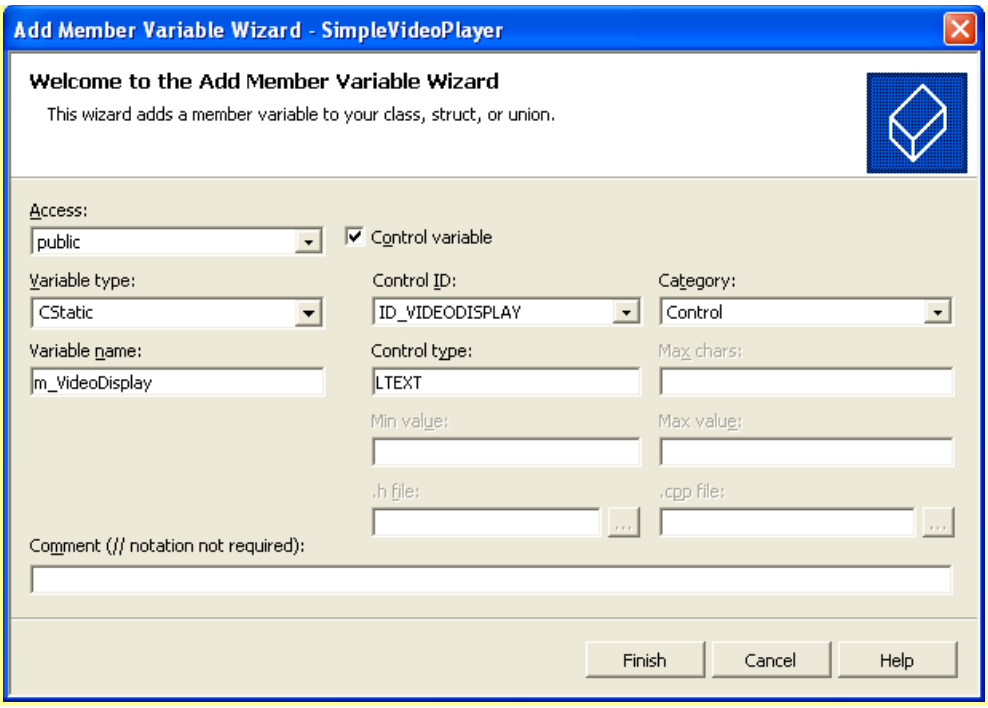
Set the “Client Edge” property to True so you can easily see the control on the dialog:



From the menu select | Project | Add Variable… |:



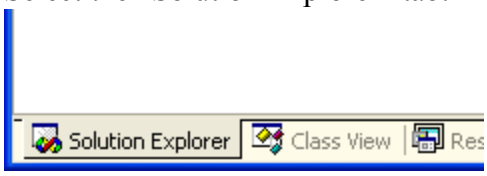
In the “Add Member Variable Wizard” set the “Variable type” to CStatic, and set the “Variable name” to be m_VideoDisplay:



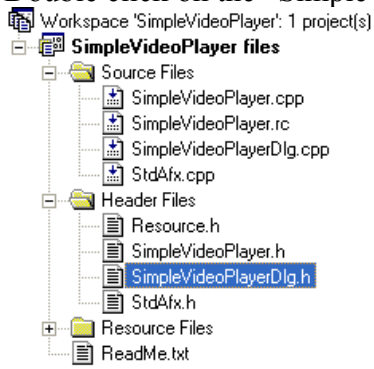
Click Finish.

In Visual C++ 6.0:
Select the “FileView” tab:

In Visual Visual C++ 2003/2005:
Select the “Solution Explorer” tab:



Double click on the “SimpleVideoPlayerDlg.h”:



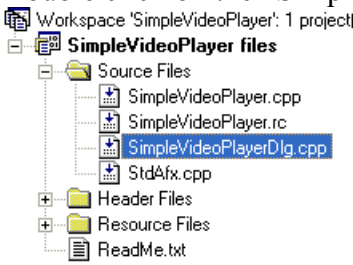
Add the highlighted lines in the header:

```
// SimpleVideoPlayerDlg.h : header file
//


#if !
defined(AFX_SIMPLEVIDEOPLAYERDLG_H__041A7871_5D55_4883_9304_616C210EA
1C2__INCLUDED_)
#define
AFX_SIMPLEVIDEOPLAYERDLG_H__041A7871_5D55_4883_9304_616C210EA1C2__INC
LUDED_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include <CVLAVIPlayer.h>
#include <CVLImageDisplay.h>


/////////////////////////////////////////////////////////////////////////
////////
// CSimpleVideoPlayerDlg dialog


class CSimpleVideoPlayerDlg : public CDialog
{
```

```
// Construction
public:
     CSimpleVideoPlayerDlg(CWnd* pParent = NULL);    // standard
constructor


// Dialog Data
     //{{AFX_DATA(CSimpleVideoPlayerDlg)
     enum { IDD = IDD_SIMPLEVIDEOPLAYER_DIALOG };
     CStatic     m_VideoDisplay;
     //}}AFX_DATA


     // ClassWizard generated virtual function overrides
     //{{AFX_VIRTUAL(CSimpleVideoPlayerDlg)
     protected:
     virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV
support
     //}}AFX_VIRTUAL


// Implementation
protected:
     CTVLImageDisplay VideoDisplay;
     CTVLAVIPlayer    AVIPlayer;

protected:
     HICON m_hIcon;


     // Generated message map functions
     //{{AFX_MSG(CSimpleVideoPlayerDlg)
     virtual BOOL OnInitDialog();
     afx_msg void OnPaint();
     afx_msg HCURSOR OnQueryDragIcon();
     //}}AFX_MSG
     DECLARE_MESSAGE_MAP()
};


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
```

```
#endif // !
defined(AFX_SIMPLEVIDEOPLAYERDLG_H__041A7871_5D55_4883_9304_616C210EA
1C2__INCLUDED_)
```

Double click on the “SimpleVideoPlayerDlg.cpp” file:

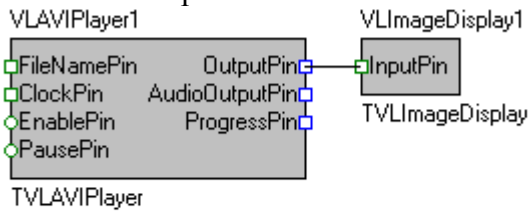


Add the highlighted lines in the CSimpleVideoPlayerDlg::OnInitDialog method:

```
BOOL CSimpleVideoPlayerDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog.  The framework does this
automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);              // Set big icon
    SetIcon(m_hIcon, FALSE);       // Set small icon

    // TODO: Add extra initialization here
    VCL_InitControls( m_hWnd );
    VideoDisplay.Open( m_VideoDisplay.m_hWnd );
    AVIPlayer.OutputPin.Connect( VideoDisplay.InputPin );

    AVIPlayer.FileName = _T( “C:\\Program Files\\Treasure
Lab\\MSVC\\Demos\\AVIFiles\\V0206-indeo3.2.avi" );
    AVIPlayer.Loop = true;
    AVIPlayer.AudioEnabled = true;

    VCL_Loaded();

    return TRUE;  // return TRUE  unless you set the focus to a
control
}
```

Compile and run the application:

Congratulations! You have just created your first VideoLab application.
Here are the OpenWire connections in the application:

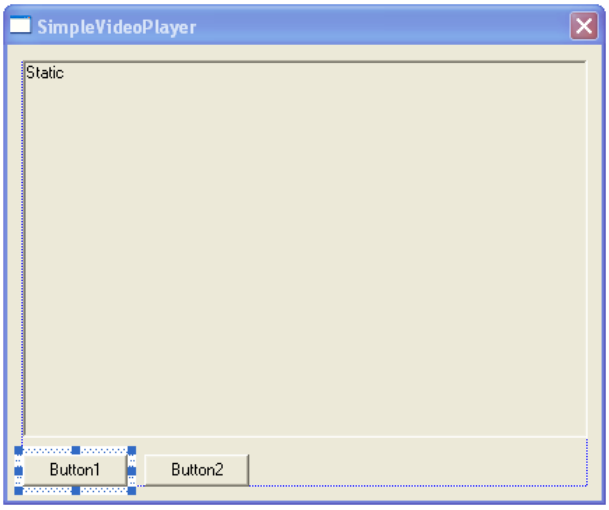


Adding Start and Stop buttons

In the previous chapter we created a simple video player. The player started playing the video as soon as we started the application. It is great for a simple application, but in a real case scenario we will need to be able to start and stop the application our self. VideoLab allows a fine control over starting, stopping, pausing, resuming and positioning the video. In this example we will cover only the simplest scenario of starting and stopping the video. For more advanced features, please refer to the VideoLab help file.
To prevent the automatic start of the video in the CSimpleVideoPlayerDlg::OnInitDialog method, add the highlighted line:

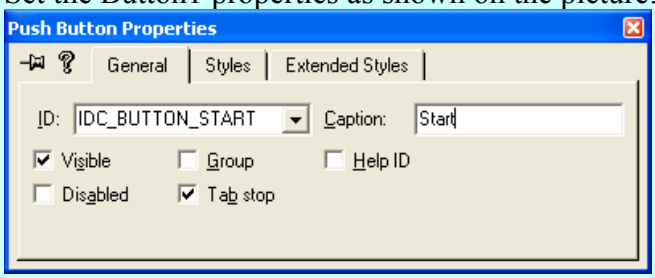
```
    VideoDisplay.Open( m_VideoDisplay.m_hWnd );
    AVIPlayer.OutputPin.Connect( VideoDisplay.InputPin );

    AVIPlayer.FileName = _T( “C:\\Program Files\\Treasure
Lab\\MSVC\\Demos\\AVIFiles\\V0206-indeo3.2.avi" );
    AVIPlayer.Loop = true;
    AVIPlayer.AudioEnabled = true;
    AVIPlayer.Enabled = false;

    VCL_Loaded();
```

Add two buttons on the dialog form:

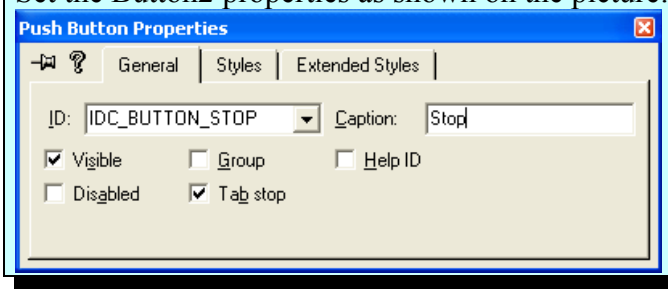


In Visual C++ 6.0:

Set the Button1 properties as shown on the picture:

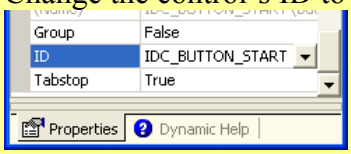


Set the Button2 properties as shown on the picture:

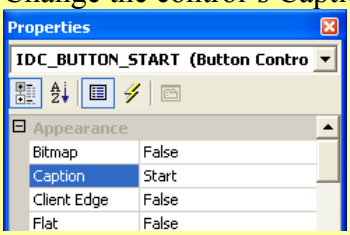


In Visual C++ 2003/2005:

Change the control's ID to "IDC_BUTTON_START":

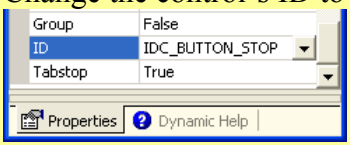


Change the control's Caption to "Start":

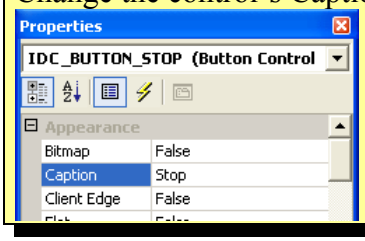


Select Button2 on the form.

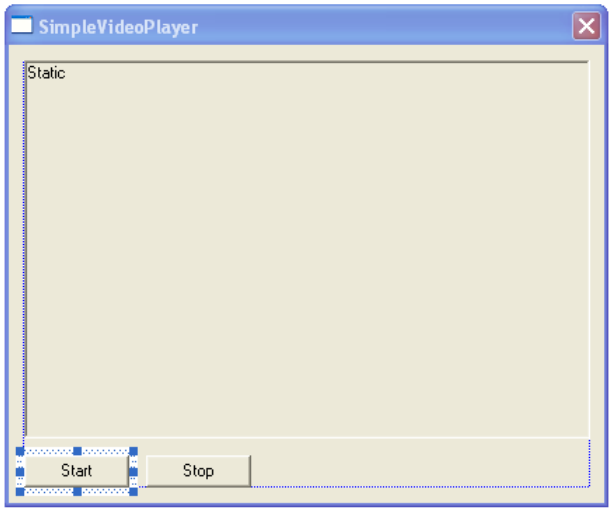
Change the control's ID to "IDC_BUTTON_STOP":



Change the control's Caption to "Stop":

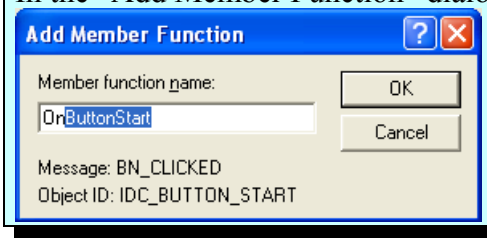


Double click on the "Start" button:



In Visual C++ 6.0:

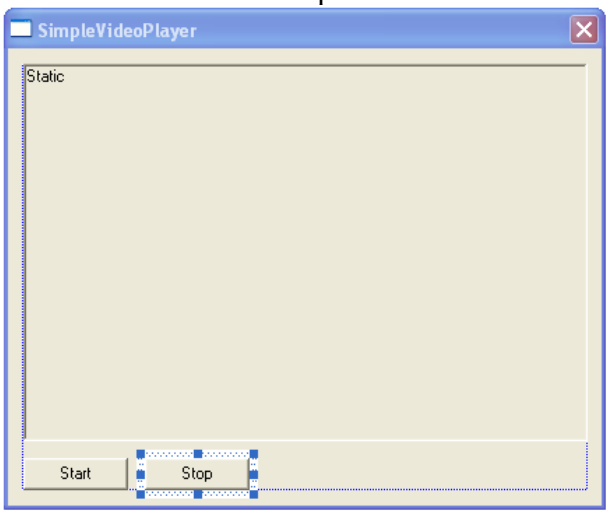
In the “Add Member Function” dialog click OK:



Add the highlighted line in the button event handler:

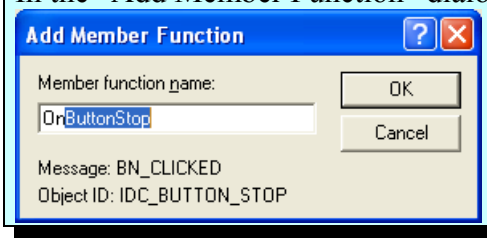
```
void CSimpleVideoPlayerDlg::OnButtonStart()
{
    // TODO: Add your control notification handler code here
    AVIPlayer.Start();
}
```

Double click on the “Stop” button:



In Visual C++ 6.0:

In the “Add Member Function” dialog click OK:



Add the highlighted line in the button event handler:

```
void CSimpleVideoPlayerDlg::OnButtonStop()
{
    // TODO: Add your control notification handler code here
```

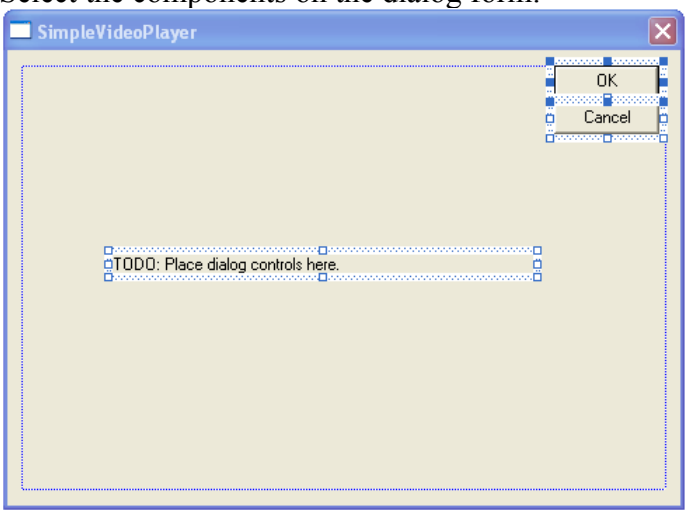
```
    AVIPlayer.Stop();
}
```

Compile and run the application, then press the “Start” button. You will see the movie playing. To stop press the “Stop” button.

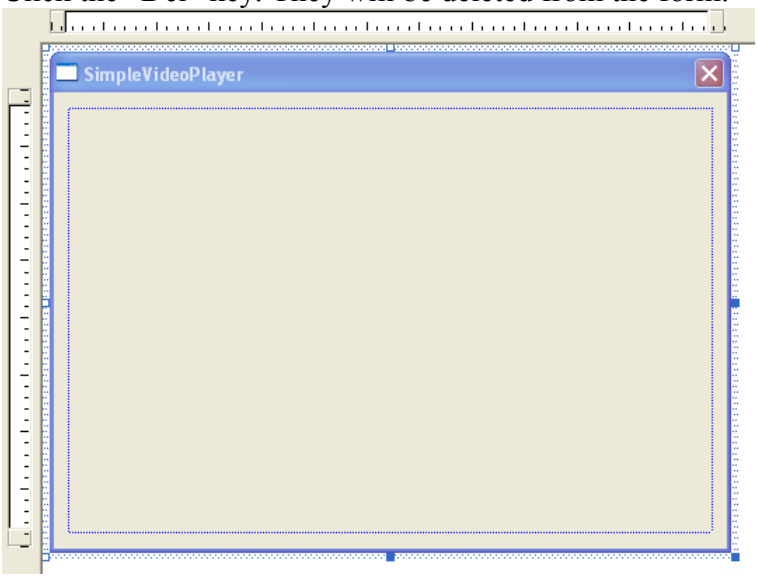
You just have learned how to add basic video control functionality to your application.

Creating a simple video capture application using Win32API

Create and setup a new project named VideoCapture as described in the “Creating a new VideoLab project in Visual C++” chapter.
After doing that you should have a project containing a Dialog resource.
Select the components on the dialog form:



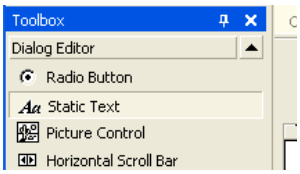
Click the “Del” key. They will be deleted from the form:



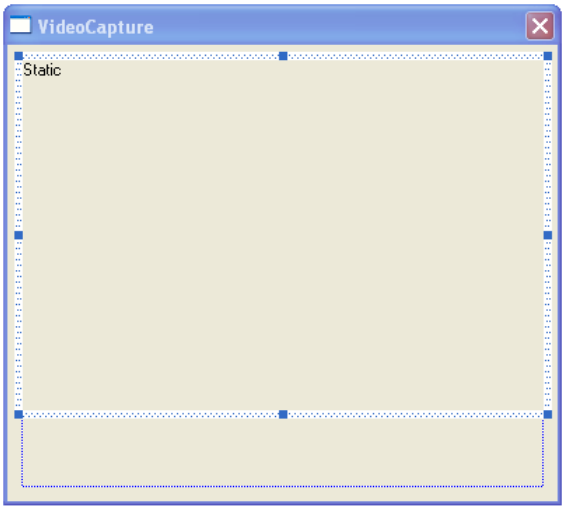
From the controls toolbar select a “Static Text” control:

VC++ 6:

VC++ 2003/2005:

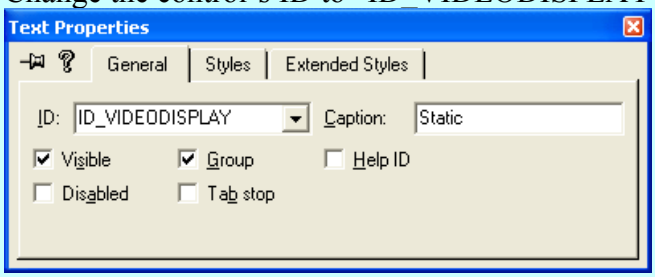


Place the control on the form:

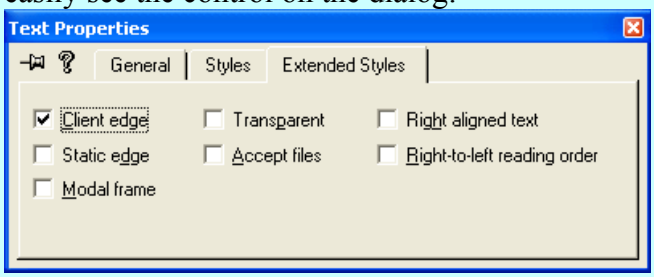


In Visual C++ 6.0:

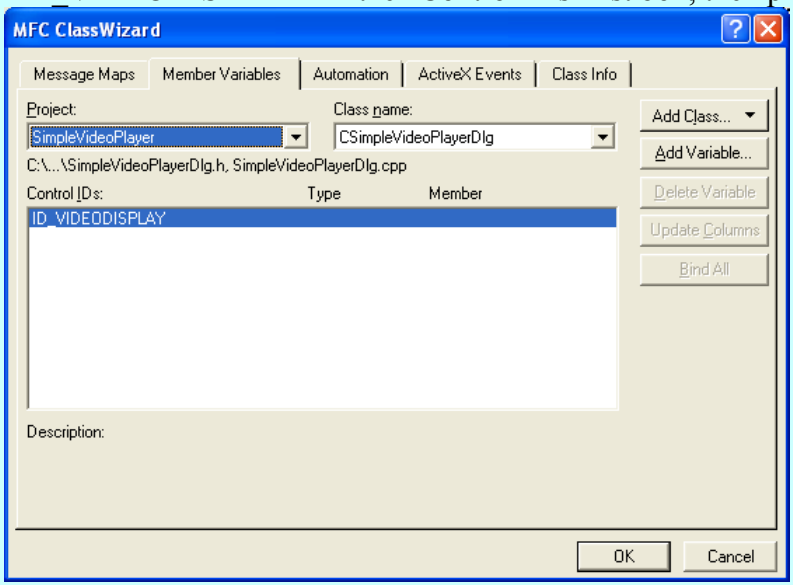
Change the control's ID to "ID_VIDEODISPLAY":



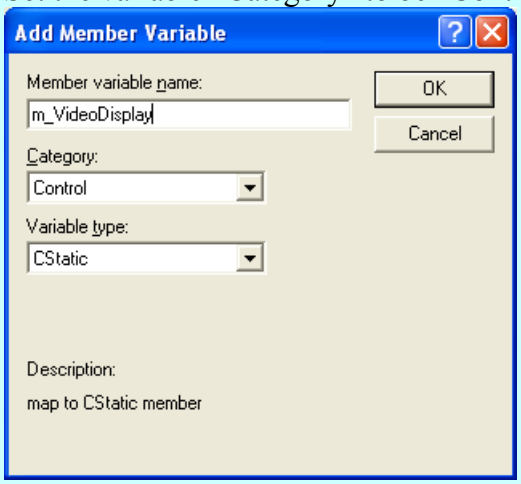
Switch to the "Extended Styles" tab and check the "Client edge" property so you can easily see the control on the dialog:



In the "ClassWizard" select the "Member Variables" tab and select the "ID_VIDEODISPLAY" in the "Control IDs" list box, then press "Add Variable…":

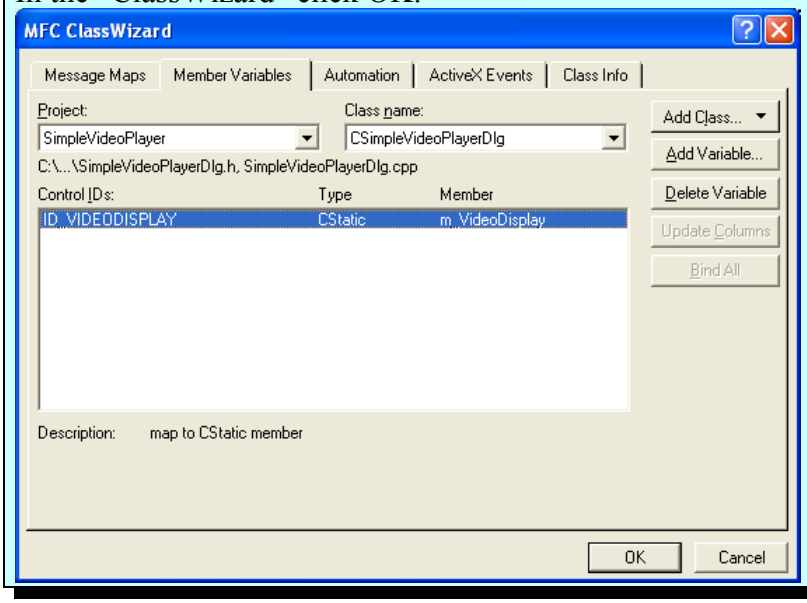


Set the variable “Category” to be “Control”, and set the name to be m_VideoDisplay:



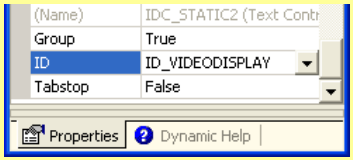
Click OK.

In the “ClassWizard” click OK:

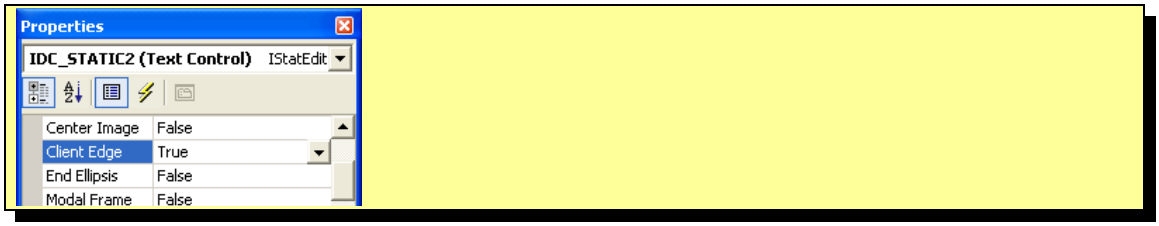


In Visual C++ 2003/2005:

Set the control’s ID to “IDC_ROTATE”:



Set the control’s “Client edge” to True:



Open the VideoCaptureDlg.h file and add the highlighted lines:

```
// VideoCaptureDlg.h : header file
//


#if !
defined(AFX_VIDEOCAPTUREDLG_H__19AB205A_6340_4413_B521_8D1F23C3844C__
INCLUDED_)
#define
AFX_VIDEOCAPTUREDLG_H__19AB205A_6340_4413_B521_8D1F23C3844C__INCLUDED
_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include <CVLImageDisplay.h>
#include <CVLCapture.h>


/////////////////////////////////////////////////////////////////////////
////////
// CVideoCaptureDlg dialog


class CVideoCaptureDlg : public CDialog
{
// Construction
public:
     CVideoCaptureDlg(CWnd* pParent = NULL);   // standard
constructor


// Dialog Data
     //{{AFX_DATA(CVideoCaptureDlg)
     enum { IDD = IDD_VIDEOCAPTURE_DIALOG };
     CStatic     m_VideoDisplay;
     //}}AFX_DATA
```

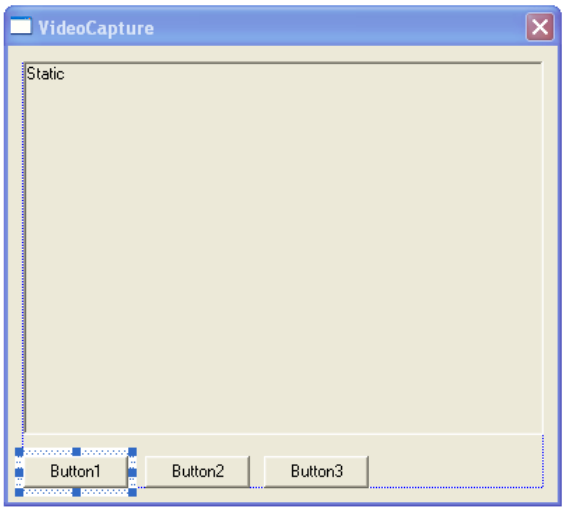
```
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CVideoCaptureDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:
	CTVLImageDisplay  VideoDisplay;
	CTVLCapture       VideoCapture;

protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CVideoCaptureDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnSourcebutton();
	afx_msg void OnStartbutton();
	afx_msg void OnStopbutton();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif // !
defined(AFX_VIDEOCAPTUREDLG_H__19AB205A_6340_4413_B521_8D1F23C3844C__
INCLUDED_)
```

Open the VideoCaptureDlg.cpp file and add the highlighted lines in the CVideoCaptureDlg::OnInitDialog method:

```
BOOL CVideoCaptureDlg::OnInitDialog()
{
```

```
      CDialog::OnInitDialog();

      // Set the icon for this dialog.  The framework does this
automatically
      //  when the application's main window is not a dialog
      SetIcon(m_hIcon, TRUE);             // Set big icon
      SetIcon(m_hIcon, FALSE);            // Set small icon

      // TODO: Add extra initialization here
      VCL_InitControls( m_hWnd );

      VideoDisplay.Open( m_VideoDisplay.m_hWnd );

      VideoCapture.OutputPin.Connect( VideoDisplay.InputPin );

      VideoCapture.DriverIndex = 0;
      VideoCapture.FramesPerSecond = 10;

      VCL_Loaded();

      return TRUE;  // return TRUE  unless you set the focus to a
control
}
```

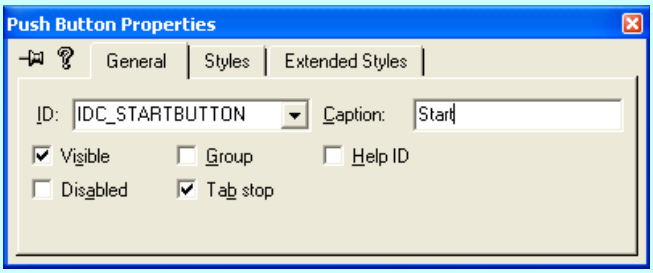
Select “Button1”:



In Visual C++ 6.0:

Change the button's ID to IDC_SOURCEBUTTON and the caption to "Select source":

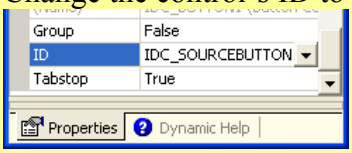
Select "Button2". Change the button's ID to IDC_STARTBUTTON and the caption to "Start":



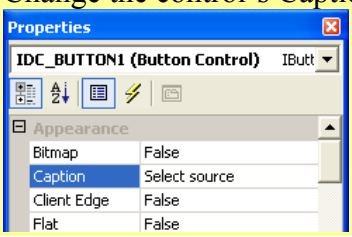
Select "Button3". Change the button's ID to IDC_STOPBUTTON and the caption to "Stop":

In Visual C++ 2003/2005:

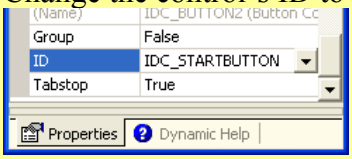
Change the control's ID to "IDC_SOURCEBUTTON":



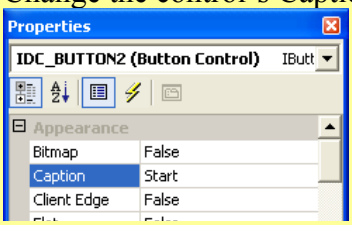
Change the control's Caption to "Select source":



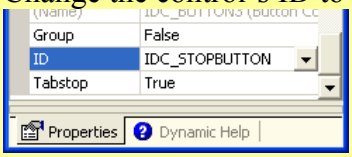
Change the control's ID to "IDC_STARTBUTTON":



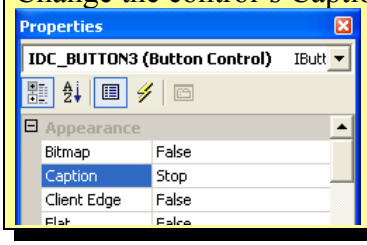
Change the control's Caption to "Start":



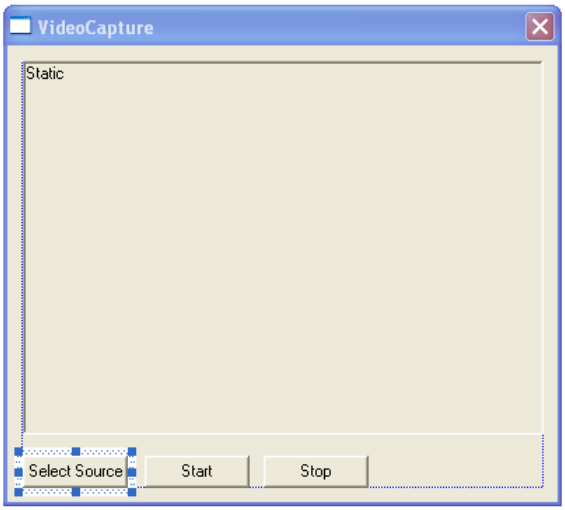
Change the control's ID to "IDC_STOPBUTTON":



Change the control's Caption to "Stop":

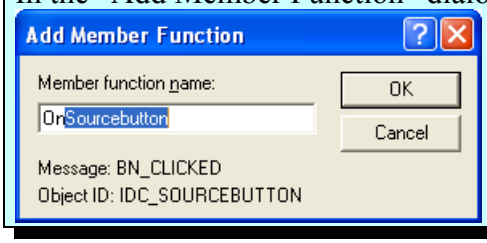


Double click on the “Select Source” button:



In Visual C++ 6.0:

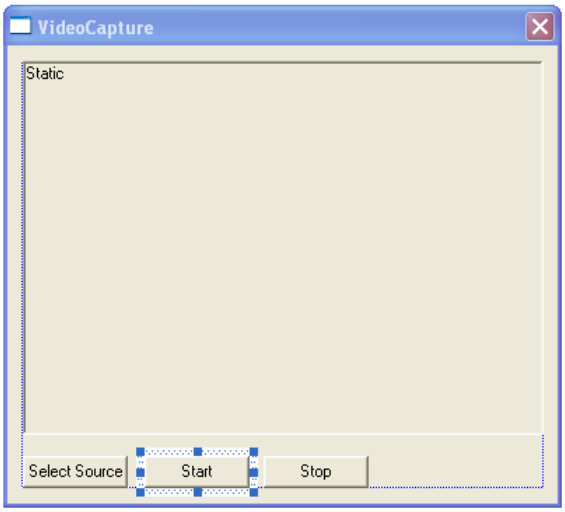
In the “Add Member Function” dialog click OK:



Add the highlighted line in the button event handler:

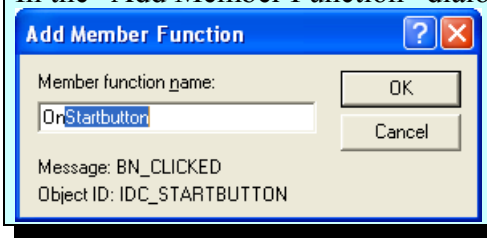
```
void CVideoCaptureDlg::OnSourcebutton()
{
      // TODO: Add your control notification handler code here
      VideoCapture.ShowVideoSourceDialog();
}
```

Double click on the “Start” button:



In Visual C++ 6.0:

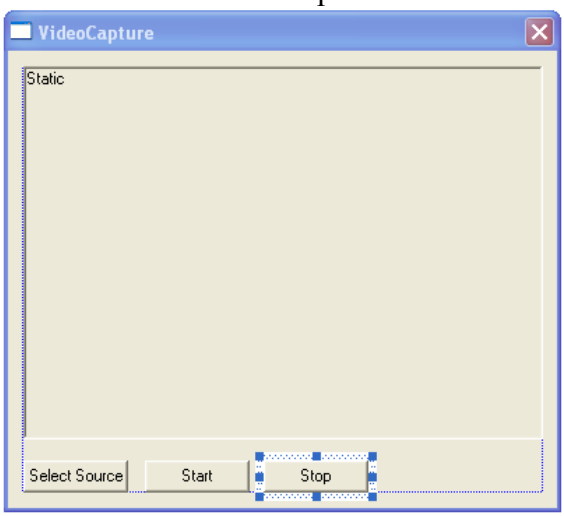
In the “Add Member Function” dialog click OK:



Add the highlighted line in the button event handler:

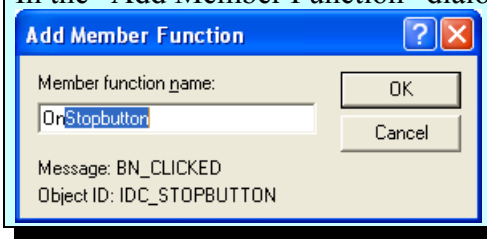
```
void CVideoCaptureDlg::OnStartbutton()
{
      // TODO: Add your control notification handler code here
      VideoCapture.Start();
}
```

Double click on the “Stop” button:



In Visual C++ 6.0:

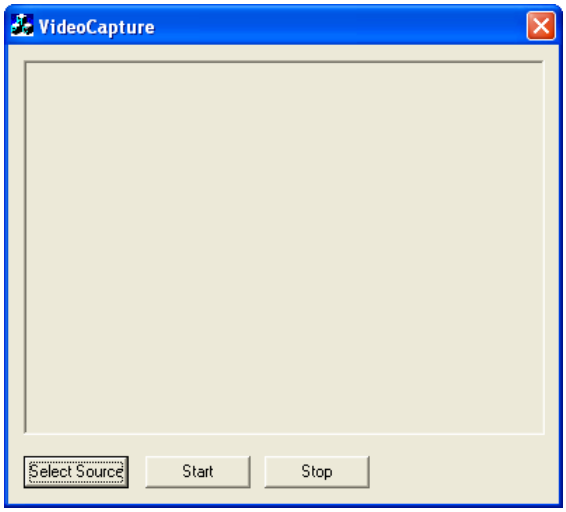
In the “Add Member Function” dialog click OK:



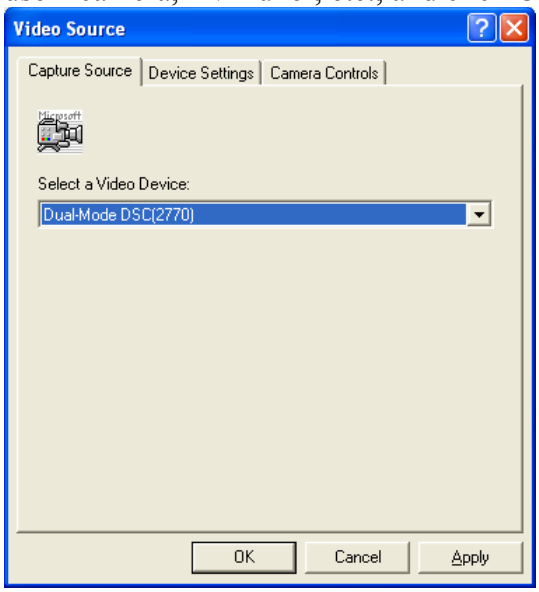
Add the highlighted line in the button event handler:

```
void CVideoCaptureDlg::OnStopbutton()
{
      // TODO: Add your control notification handler code here
      VideoCapture.Stop();
}
```

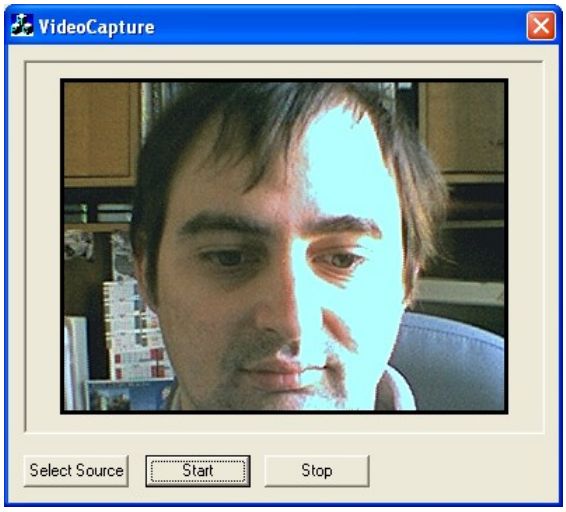
Compile and run the application.
Press the “Select Source” button:



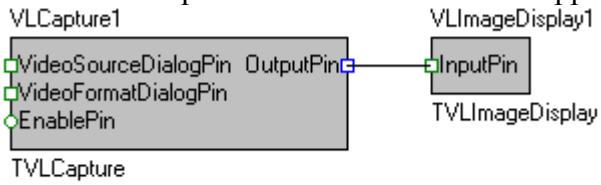
In the “Video Source” dialog select and configure the type of video source you want to use – camera, TV Tuner, etc., and click OK.



Click the “Start” button. You should see result similar to this one:



Here are the OpenWire connections in the application:

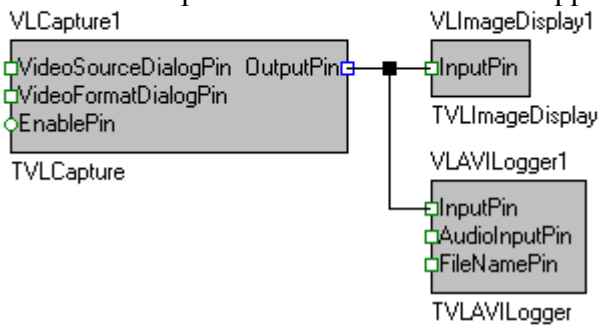


In this application, we are just displaying the captured video. If we want to store it in a file, we will have to add a TVLAVILogger component.
Stop the running application.
In the CVideoFiltersDlg::OnInitDialog method add the highlighted lines:

```
      VCL_InitControls( m_hWnd );

      VideoDisplay.Open( m_VideoDisplay.m_hWnd );

      VideoCapture.OutputPin.Connect( VideoDisplay.InputPin );

      VideoCapture.DriverIndex = 0;
      VideoCapture.FramesPerSecond = 10;

      AVILogger.InputPin.Connect( VideoCapture.OutputPin );
      AVILogger.FileName = "CapturedVideo.avi";

      VCL_Loaded();
```

Compile and run the application. Now the video will be stored in the CapturedVideo.avi file.

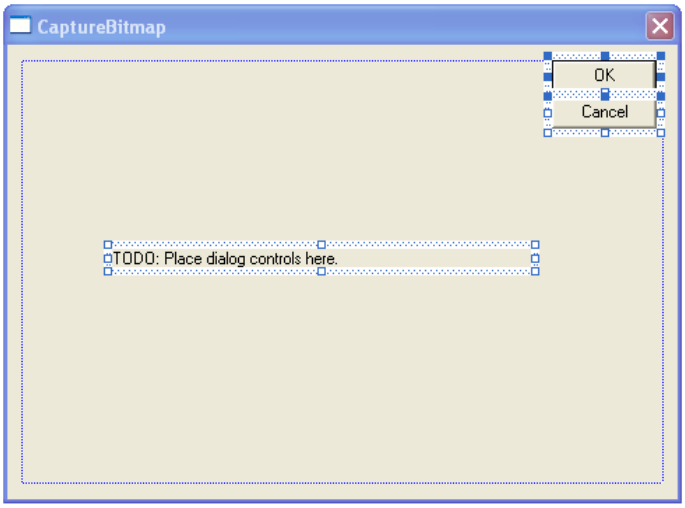
Here are the OpenWire connections in the application now:



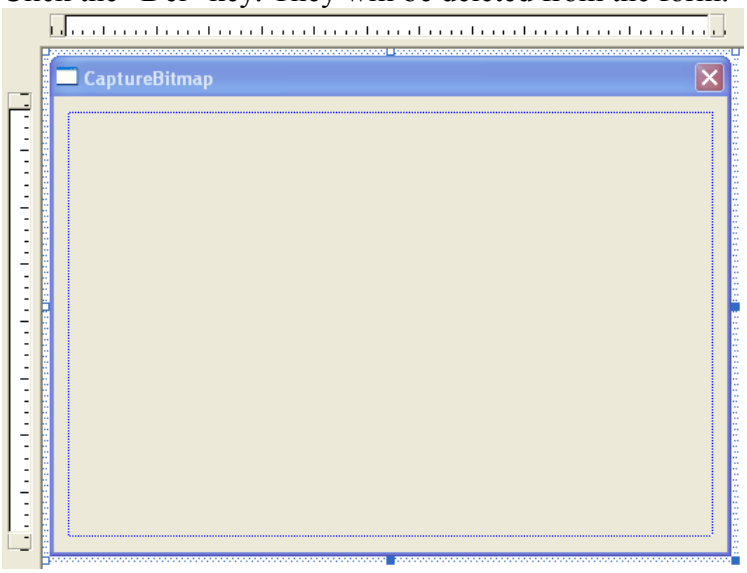
You have created your first VideoLab video capture application with VideoLab.

Capturing a frame into a Bitmap

Create and setup a new project named CaptureBitmap as described in the “Creating a new VideoLab project in Visual C++” chapter.
After doing that you should have a project containing a Dialog resource.
Select the components on the dialog form:



Click the “Del” key. They will be deleted from the form:



Open the CaptureBitmapDlg.h file and add the highlighted lines:

```
// CaptureBitmapDlg.h : header file
//


#if !
defined(AFX_CAPTUREBITMAPDLG_H__73B204DD_5BBD_44BC_A77B_4122AD6DBAB9_
_INCLUDED_)
#define
AFX_CAPTUREBITMAPDLG_H__73B204DD_5BBD_44BC_A77B_4122AD6DBAB9__INCLUDE
D_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include <CVLAVIPlayer.h>
#include <VLVideoBuffer.h>
#include <CVLGenericFilter.h>


/////////////////////////////////////////////////////////////////////////
////////
// CCaptureBitmapDlg dialog


class CCaptureBitmapDlg : public CDialog
{
// Construction
```

```
public:
    CCaptureBitmapDlg(CWnd* pParent = NULL);    // standard
constructor


// Dialog Data
    //{{AFX_DATA(CCaptureBitmapDlg)
    enum { IDD = IDD_CAPTUREBITMAP_DIALOG };
            // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA


    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCaptureBitmapDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
support
    //}}AFX_VIRTUAL


// Implementation
protected:
    CTVLGenericFilter GenericVCFilter;
    CTVLAVIPlayer  AVIPlayer;
    CBitmap ABmp;
    CSize   ImageSize;

protected:
     void __stdcall OnVideoFrameEvent( void *Sender, TVLCVideoBuffer
BufferIn, TVLCVideoBuffer &BufferOut, bool &SendOutputData );


protected:
    HICON m_hIcon;


    // Generated message map functions
    //{{AFX_MSG(CCaptureBitmapDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.


#endif // !
defined(AFX_CAPTUREBITMAPDLG_H__73B204DD_5BBD_44BC_A77B_4122AD6DBAB9_
_INCLUDED_)
```

Open the CaptureBitmapDlg.cpp file and add the highlighted lines:

```
// CaptureBitmapDlg.cpp : implementation file
//


#include "stdafx.h"
#include "CaptureBitmap.h"
#include "CaptureBitmapDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


//////////////////////////////////////////////////////////////////
////////
// CCaptureBitmapDlg dialog


CCaptureBitmapDlg::CCaptureBitmapDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CCaptureBitmapDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCaptureBitmapDlg)
            // NOTE: the ClassWizard will add member initialization
here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}


void CCaptureBitmapDlg::DoDataExchange(CDataExchange* pDX)
```

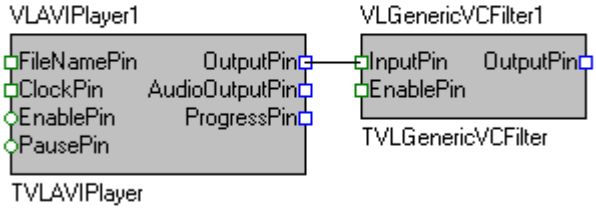
```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCaptureBitmapDlg)
           // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCaptureBitmapDlg, CDialog)
    //{{AFX_MSG_MAP(CCaptureBitmapDlg)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////
// CCaptureBitmapDlg message handlers


BOOL CCaptureBitmapDlg::OnInitDialog()
{
    CDialog::OnInitDialog();


    // Set the icon for this dialog.  The framework does this
automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);              // Set big icon
    SetIcon(m_hIcon, FALSE);       // Set small icon


    // TODO: Add extra initialization here


    ImageSize.cx = 240;
    ImageSize.cy = 180;


    VCL_InitControls( m_hWnd );


    GenericVCFilter.SynchronizeType = stQueue;
    GenericVCFilter.OnProcessData.Set( this,
&CBitmapCaptureDlg::OnVideoFrameEvent );
```

```
    AVIPlayer.FileName = _T( "C:\\Program Files\\Treasure
Lab\\MSVC\\Demos\\AVIFiles\\V0206-indeo3.2.avi" );
    AVIPlayer.OutputPin.Connect( GenericVCFilter.InputPin );
    AVIPlayer.Enabled = true;
    AVIPlayer.AudioEnabled = true;
    AVIPlayer.Loop = true;
    VCL_Loaded();


    return TRUE;  // return TRUE  unless you set the focus to a
control
}


// If you add a minimize button to your dialog, you will need the
code below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.


void CCaptureBitmapDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting


        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);


        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;


        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
```

```
            CPaintDC dc(this); // device context for painting

            CDC AnotherDC;
            AnotherDC.CreateCompatibleDC( &dc );

            AnotherDC.SelectObject( ABmp );

            dc.BitBlt( 0, 0, ImageSize.cx, ImageSize.cy, &AnotherDC,
0, 0, SRCCOPY );

        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the
user drags
//  the minimized window.
HCURSOR CCaptureBitmapDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void __stdcall CCaptureBitmapDlg::OnVideoFrameEvent(void *Sender,
TVLCVideoBuffer InBuffer, TVLCVideoBuffer &BufferOut, bool
&SendOutputData )
{
    ImageSize.cx = InBuffer.GetWidth();
    ImageSize.cy = InBuffer.GetHeight();

    ABmp.DeleteObject();
    ABmp.CreateCompatibleBitmap( GetDC(), ImageSize.cx,
ImageSize.cy );

    InBuffer.ToBitmap( ABmp );

    CRect ImageRect( 0, 0, ImageSize.cx, ImageSize.cy );

    InvalidateRect( ImageRect, FALSE );
}
```

Compile and run the application. You should see a result similar to this one:

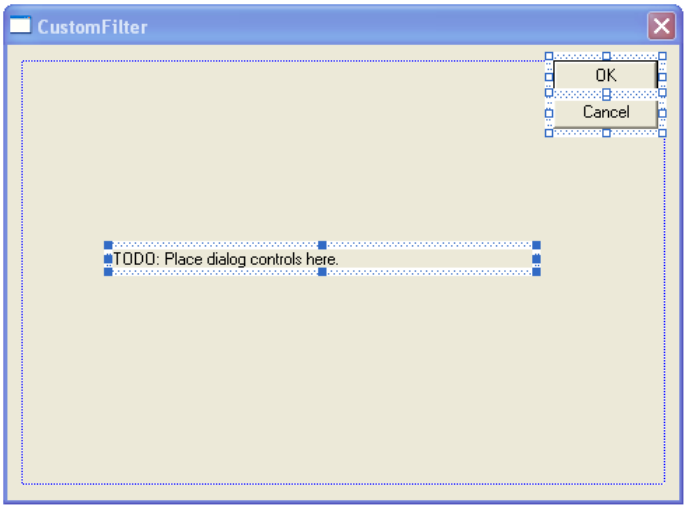
Here are the OpenWire connections in this application:



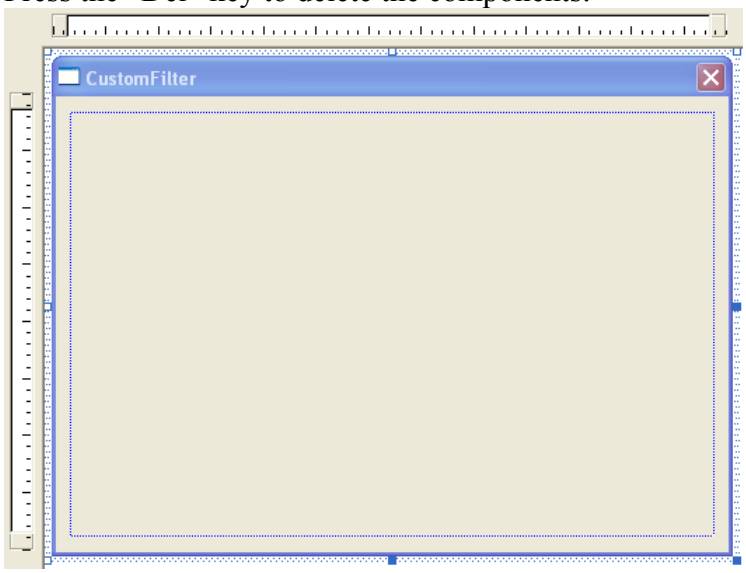
You have created your first VideoLab Bitmap capture application with VideoLab.

Creating your own filter

Create and setup a new project named CustomFilter as described in the “Creating a new VideoLab project in Visual C++” chapter.
After doing that you should have a project containing a Dialog resource.
Select the components on the dialog form:



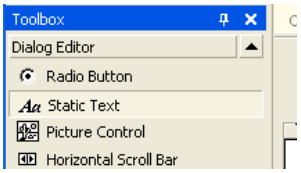
Press the “Del” key to delete the components:



From the controls toolbar select a “Static Text” control and place two of them on the dialog form:

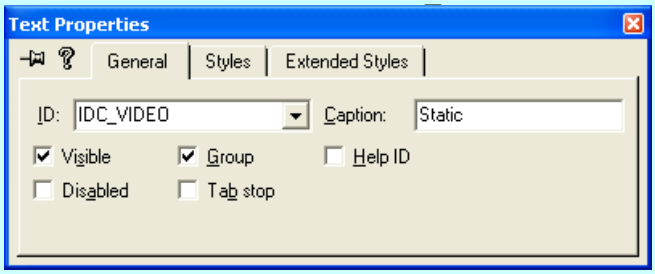
VC++ 6:

VC++ 2003/2005:

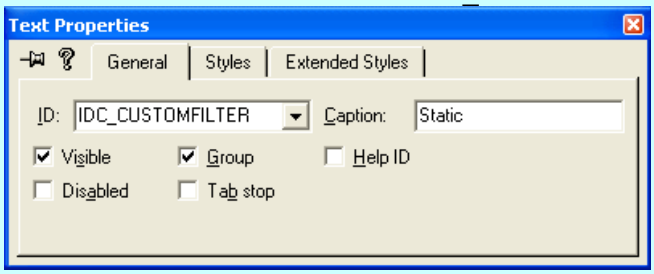


In Visual C++ 6.0:

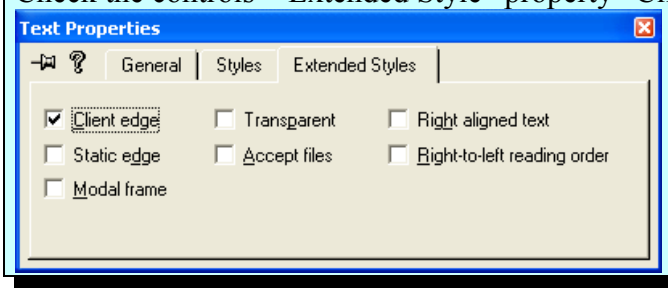
Set the first control’s ID to “IDC_VIDEO”:



Set the second control’s ID to “IDC_CUSTOMFILTER”:

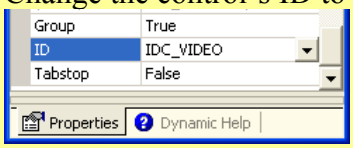


Check the controls' "Extended Style" property "Client edge":

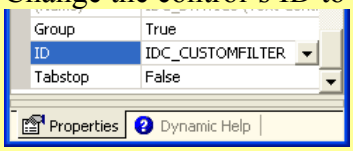


In Visual C++ 2003/2005:

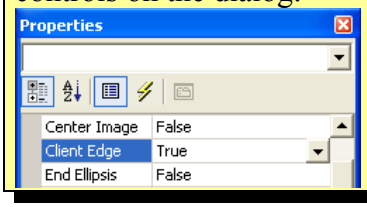
Change the control's ID to "ID_VIDEO":



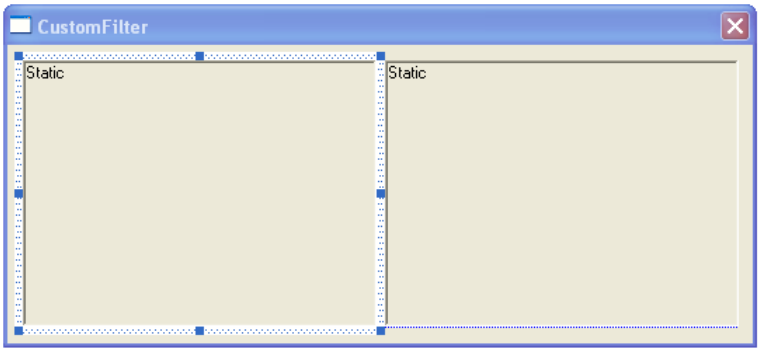
Change the control's ID to "IDC_CUSTOMFILTER":



Set the "Client Edge" property for both components to True so you can easily see the controls on the dialog:

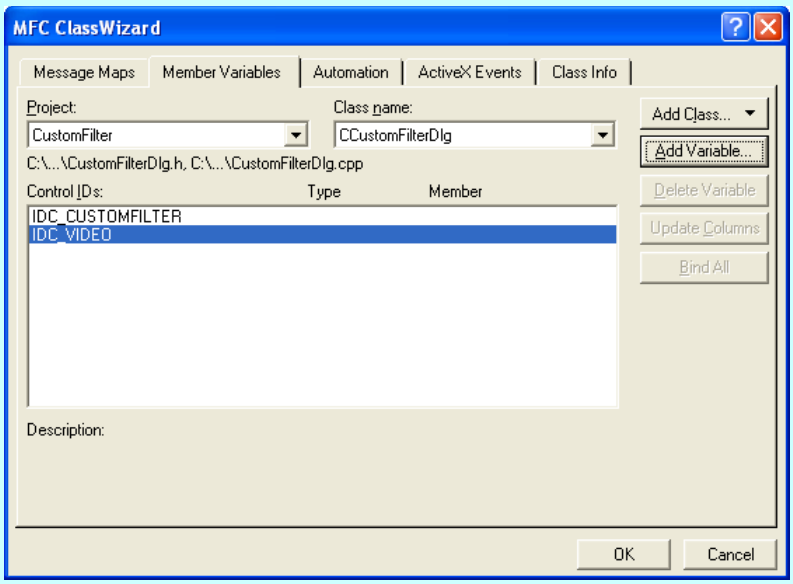


The dialog should look similar to this one:

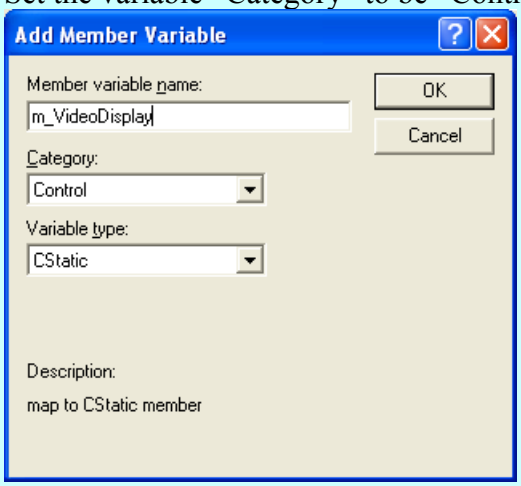


In Visual C++ 6.0:

Open the “ClassWizard”, select the “IDC_VIDEO” in the “Control ID’s” list, and click the “Add Variable…” button:

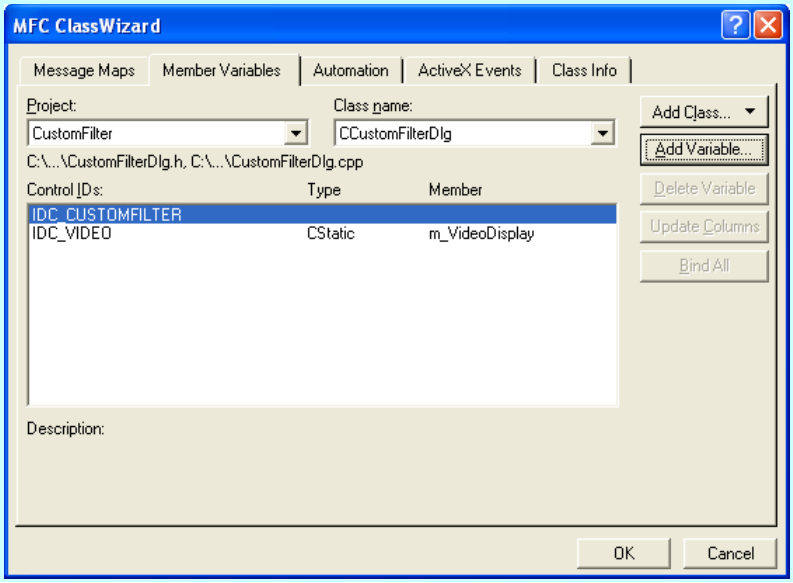


Set the variable “Category” to be “Control”, and set the name to be m_VideoDisplay:

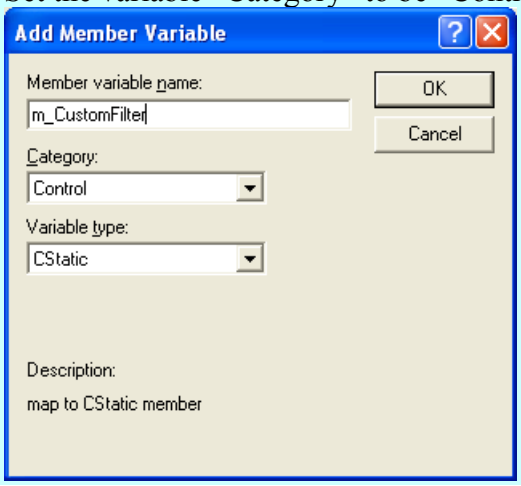


Click OK.

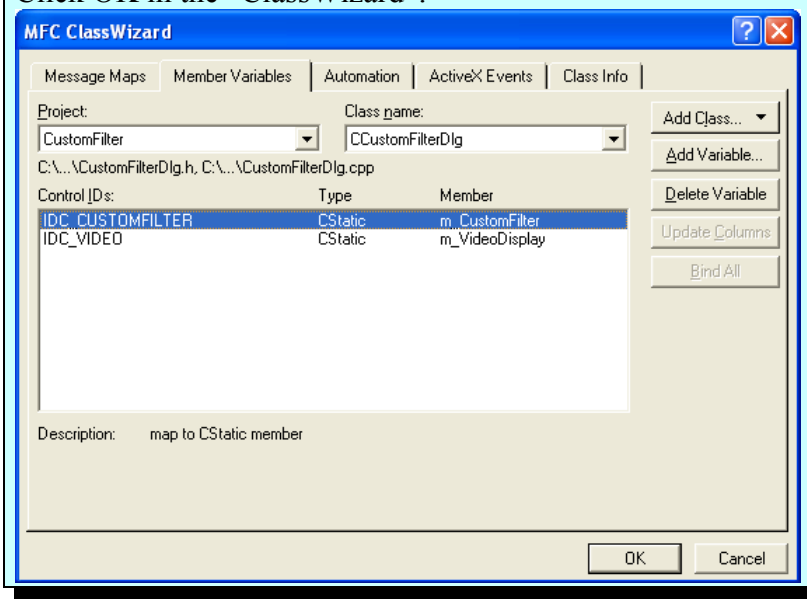
In the “ClassWizard”, select the “IDC_CUSTOMFILTER” in the “Control ID’s” list, and click the “Add Variable…” button:



Set the variable “Category” to be “Control”, and set the name to be m_CustomFilter:

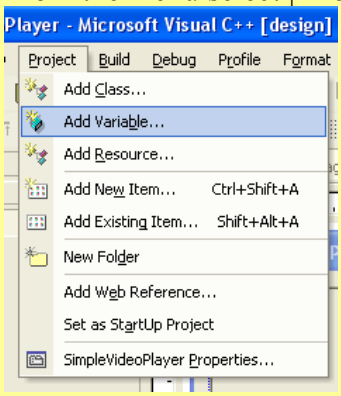


Click OK in the “ClassWizard”:

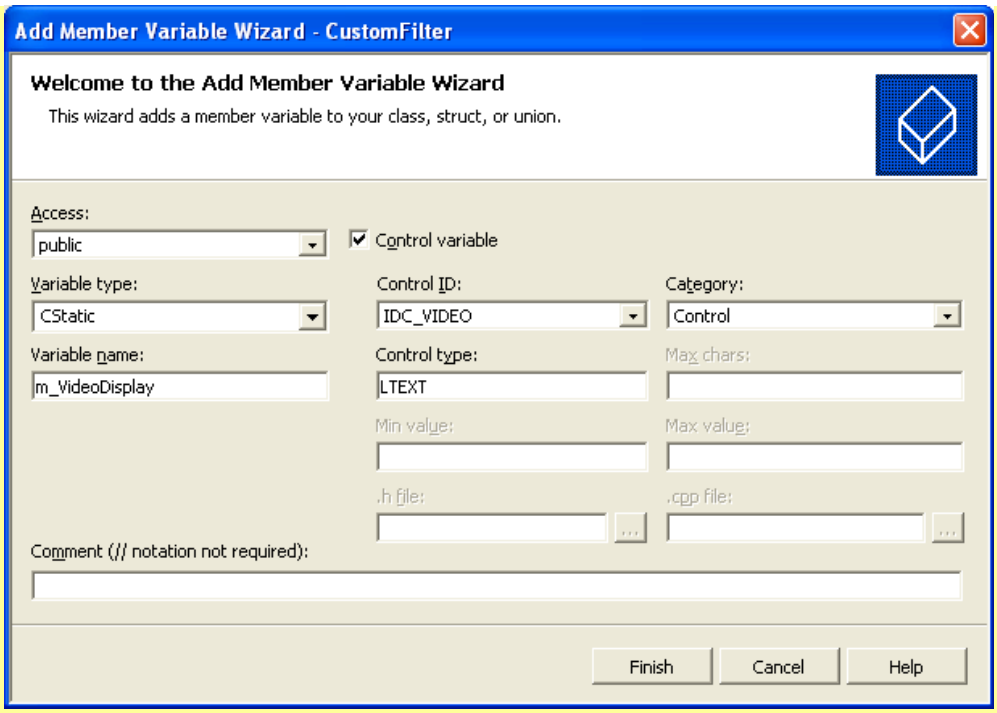


In Visual C++ 2003/2005:

From the menu select | Project | Add Variable… |:

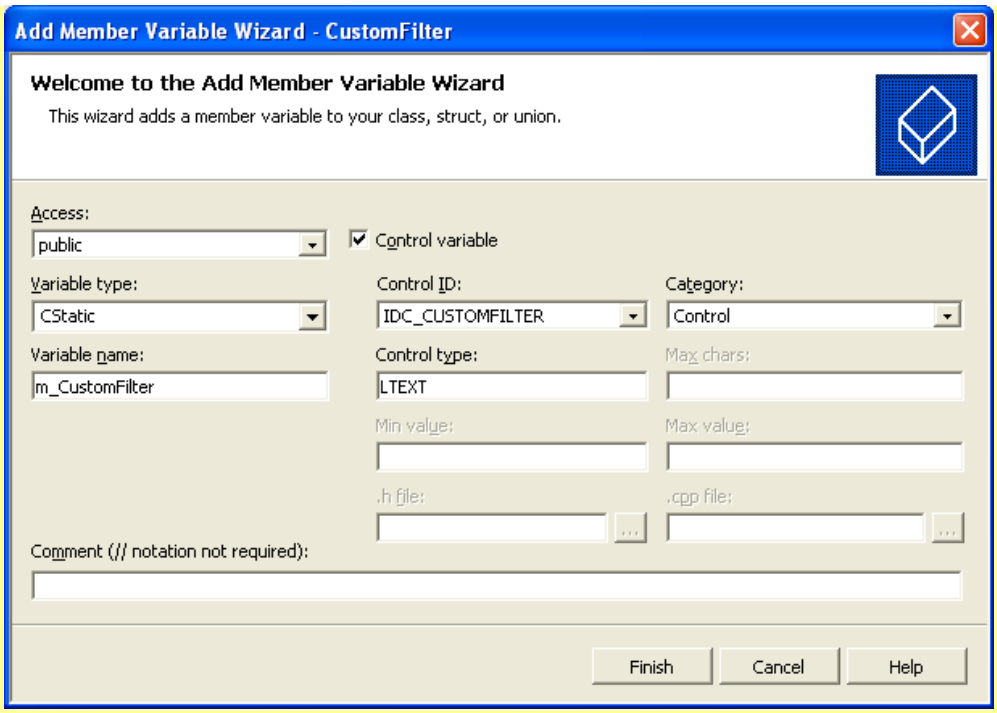


In the “Add Member Variable Wizard” set the “Variable type” to CStatic, and the “Variable name” to be m_VideoDisplay:



Click Finish.

In the “Add Member Variable Wizard” set the “Variable type” to CStatic, and the “Variable name” to be m_CustomFilter:



Click Finish.

Open the CustomFilterDlg.h file and add the highlighted lines:

```
// CustomFilterDlg.h : header file
//
#if !
defined(AFX_CUSTOMFILTERDLG_H__B6AEB381_66D7_4D8B_90FF_F6F75B55B39F__
INCLUDED_)
#define
AFX_CUSTOMFILTERDLG_H__B6AEB381_66D7_4D8B_90FF_F6F75B55B39F__INCLUDED
_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include <CVLAVIPlayer.h>
#include <CVLImageDisplay.h>
#include <VLVideoBuffer.h>
#include <CVLGenericFilter.h>
```

```
/////////////////////////////////////////////////////////////////////////
////////
// CCustomFilterDlg dialog


class CCustomFilterDlg : public CDialog
{
// Construction
public:
    CCustomFilterDlg(CWnd* pParent = NULL);    // standard
constructor


// Dialog Data
    //{{AFX_DATA(CCustomFilterDlg)
    enum { IDD = IDD_CUSTOMFILTER_DIALOG };
    CStatic m_CustomFilter;
    CStatic m_VideoDisplay;
    //}}AFX_DATA


    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCustomFilterDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV
support
    //}}AFX_VIRTUAL


// Implementation
protected:
    CTVLAVIPlayer       AVIPlayer;
    CTVLImageDisplay    AVIVideoDisplay;
    CTVLImageDisplay    FilterVideoDisplay;
    CTVLGenericFilter   GenericVCFilter;

protected:
    void __stdcall OnVideoFilterEvent( void *Sender, TVLCVideoBuffer
BufferIn, TVLCVideoBuffer &BufferOut, bool &SendOutputData );


protected:
    HICON m_hIcon;
```

```
    // Generated message map functions
    //{{AFX_MSG(CCustomFilterDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.


#endif // !
defined(AFX_CUSTOMFILTERDLG_H__B6AEB381_66D7_4D8B_90FF_F6F75B55B39F__
INCLUDED_)
```

Open the CustomFilterDlg.cpp file and add the highlighted lines:

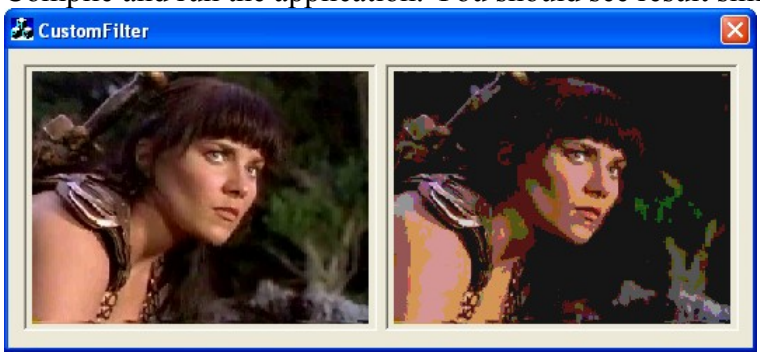
```
// CustomFilterDlg.cpp : implementation file
//


#include "stdafx.h"
#include "CustomFilter.h"
#include "CustomFilterDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


//////////////////////////////////////////////////////////////////
////////
// CCustomFilterDlg dialog


CCustomFilterDlg::CCustomFilterDlg(CWnd* pParent /*=NULL*/)
     : CDialog(CCustomFilterDlg::IDD, pParent)
{
```

```
    //{{AFX_DATA_INIT(CCustomFilterDlg)
            // NOTE: the ClassWizard will add member initialization
here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}


void CCustomFilterDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCustomFilterDlg)
    DDX_Control(pDX, IDC_CUSTOMFILTER, m_CustomFilter);
    DDX_Control(pDX, IDC_VIDEO, m_VideoDisplay);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCustomFilterDlg, CDialog)
    //{{AFX_MSG_MAP(CCustomFilterDlg)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////
////////
// CCustomFilterDlg message handlers


BOOL CCustomFilterDlg::OnInitDialog()
{
    CDialog::OnInitDialog();


    // Set the icon for this dialog.  The framework does this
automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);              // Set big icon
    SetIcon(m_hIcon, FALSE);       // Set small icon
```

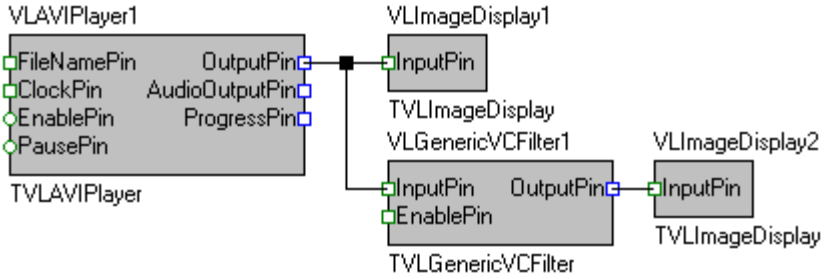
```
    // TODO: Add extra initialization here
    VCL_InitControls( m_hWnd );

    AVIVideoDisplay.Open( m_AVIDisplay.m_hWnd );
    FilterVideoDisplay.Open( m_FilterDisplay.m_hWnd );

    GenericVCFilter.OnProcessData.Set( this,
&CCustomFilterDlg::OnVideoFilterEvent );
    GenericVCFilter.OutputPin.Connect( FilterVideoDisplay.InputPin );


    AVIPlayer.OutputPin.Connect( AVIVideoDisplay.InputPin );
    AVIPlayer.OutputPin.Connect( GenericVCFilter.InputPin );
    AVIPlayer.FileName = _T( "C:\\Program Files\\Treasure
Lab\\MSVC\\Demos\\AVIFiles\\V0206-indeo3.2.avi" );
    AVIPlayer.Loop = true;
    AVIPlayer.AudioEnabled = true;


    VCL_Loaded();


    return TRUE;  // return TRUE  unless you set the focus to a
control
}


// If you add a minimize button to your dialog, you will need the
code below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.


void CCustomFilterDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting


        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
```

```
        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the
user drags
//  the minimized window.
HCURSOR CCustomFilterDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void __stdcall CCustomFilterDlg::OnVideoFilterEvent( void *Sender,
TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer, bool
&SendOutputData )
{
    const BYTE *DataIn = InBuffer.Read();
    BYTE *DataOut = OutBuffer.Write();

    int Length = OutBuffer.GetWidth() * OutBuffer.GetHeight() * 3;

    for( int i = 0; i < Length; i ++ )
       *DataOut++ = ( *DataIn++  & 0xC0C0C0 ) | 0x151515;
}
```

Compile and run the application. You should see result similar to this one:



Here are the OpenWire connections in this application:



You have learned how to create a custom video filter using VideoLab.

Using the TSLCRealBuffer in C++ Builder and Visual C++

The C++ Builder version of the library comes with a powerful data buffer class, called TSLCRealBuffer.
The TSLCRealBuffer is capable of performing basic math operations over the data as well as some basic signal processing functions. The data buffer also uses copy on write algorithm improving dramatically the application performance.
The TSLCRealBuffer is an essential part of the SignalLab generators and filters, but it can be used independently in your code.
You have seen already some examples of using TSLCRealBuffer in the previous chapters. Here we will go into a little bit more details about how TSLCRealBuffer can be used.
In order to use TSLCRealBuffer you must include SLCRealBuffer.h directly or indirectly (trough another include file):

```
#include <SLCRealBuffer.h>
```

Once the file is included you can declare a buffer:
Here is how you can declare a 1024 samples buffer:

```
TSLCRealBuffer Buffer( 1024 );
```

Version 4.0 and up does not require the usage of data access objects. The data objects are now obsolete and have been removed from the library.

You can obtain the current size of a buffer by calling the GetSize method:

```
Int ASize = Buffer.GetSize(); // Obtains the size of the buffers
```

You can resize (change the size of) a buffer:

```
Buffer.Resize( 2048 ); // Changes the size to 2048
```

You can set all of the elements (samples) of the buffer to a value:

```
Buffer.Set( 30 ); // Sets all of the elements to 30.
```

You can access individual elements (samples) in the buffer:

```
Buffer [ 5 ] = 3.7; // Sets the fifth elment to 3.7

Double AValue = Buffer [ 5 ]; // Assigns the fifth element to a
variable
```

You can obtain read, write or modify pointer to the buffer data:

```
const double *data = Buffer.Read() // Starts reading only
double *data = Buffer.Write()// Starts writing only
double *data = Buffer.Modify()// Starts reading and writing
```

Sometimes you need a very fast way of accessing the buffer items. In this case, you can obtain a direct pointer to the internal data buffer. The buffer is based on copy on write technology for high performance. The mechanism is encapsulated inside the buffer, so when working with individual items you don't have to worry about it. If you want to access the internal buffer for speed however, you will have to specify up front if you are planning to modify the data or just to read it. The TSLCRealBuffer has 3 methods for accessing the data Read(), Write(), and Modify (). Read() will return a constant pointer to the data. You should use this method when you don't intend to modify the data and just need to read it. If you want to create new data from scratch and don't intend to preserve the existing buffer data, use Write(). If you need to modify the data you should use Modify (). Modify () returns a non constant pointer to the data, but often works slower than Read() or Write(). Here are some examples:

```
const double *pcData = Buffer.Read(); // read only data pointer

double Value = *pcData; // OK!
*pcData = 3.5; // Wrong!


double *pData = Buffer.Write(); // generic data pointer

double Value = *pData; // OK!
*pData = 3.5; // OK!
```

You can assign one buffer to another:

```
Buffer1 = Buffer2;
```

You can do basic buffer arithmetic:

```
TSLCRealBuffer Buffer1( 1024 );
TSLCRealBuffer Buffer2( 1024 );
TSLCRealBuffer Buffer3( 1024 );

Buffer1.Set( 20.5 );
Buffer2.Set( 5 );

Buffer3 = Buffer1 + Buffer2;
Buffer3 = Buffer1 - Buffer2;
Buffer3 = Buffer1 * Buffer2;
Buffer3 = Buffer1 / Buffer2;
```

In this example the elements of the Buffer3 will be result of the operation (+,-,* or /) between the corresponding elements of Buffer1 and Buffer2.
You can add, subtract, multiply or divide by constant:

```
// Adds 4.5 to each element of the buffer
Buffer1 = Buffer2 + 4.5;

// Subtracts 4.5 to each element of the buffer
Buffer1 = Buffer2 - 4.5;

// Multiplies the elements by 4.5
Buffer1 = Buffer2 * 4.5;

// Divides the elements by 4.5
Buffer1 = Buffer2 / 4.5;
```

You can do “in place” operations as well:

```
Buffer1 += Buffer2;
Buffer1 += 4.5;

Buffer1 -= Buffer2;
Buffer1 -= 4.5;

Buffer1 *= Buffer2;
```

```
Buffer1 *= 4.5;

Buffer1 /= Buffer2;
Buffer1 /= 4.5;
```

Those are just some of the basic buffer operations provided by SignalLab.
If you are planning to use some of the more advanced features of TSLCRealBuffer please refer to the online help.
SignalLab also provides TSLCComplexBuffer and TSLCIntegerBuffer. They work similar to the TSLCRealBuffer but are intended to be used with Complex and Integer data. For more information on TSLCComplexBuffer and TSLCIntegerBuffer please refer to the online help.

Distributing your application

Once you have finished the development of your application you most likely will need to distribute it to other systems. In order for the built application to work, you will have to include a set of DLL files together with the distribution. The necessary files can be found under the [install path]\DLL directory([install path] is the location where the library was installed).
You can distribute them to the [Windows]\System32 ([Windows]\SysWOW64 in 64 bit Windows) directory, or to the distribution directory of your application([Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS).

Deploying your application with the IPP DLLs

The application will work, however the performance can be improved by also copying the Intel IPP DLLs provided with the library.
The DLLs are under the [install path]\LabPacks\IppDLL directory([install path] is the location where the library was installed).
In 32 bit Windows to deploy IPP, copy the files to the [Windows]\System32 directory on the target system.
In 64 bit Windows to deploy IPP, copy the files to the [Windows]\SysWOW64 directory on the target system.
[Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS
This will improve the performance of your application on the target system.