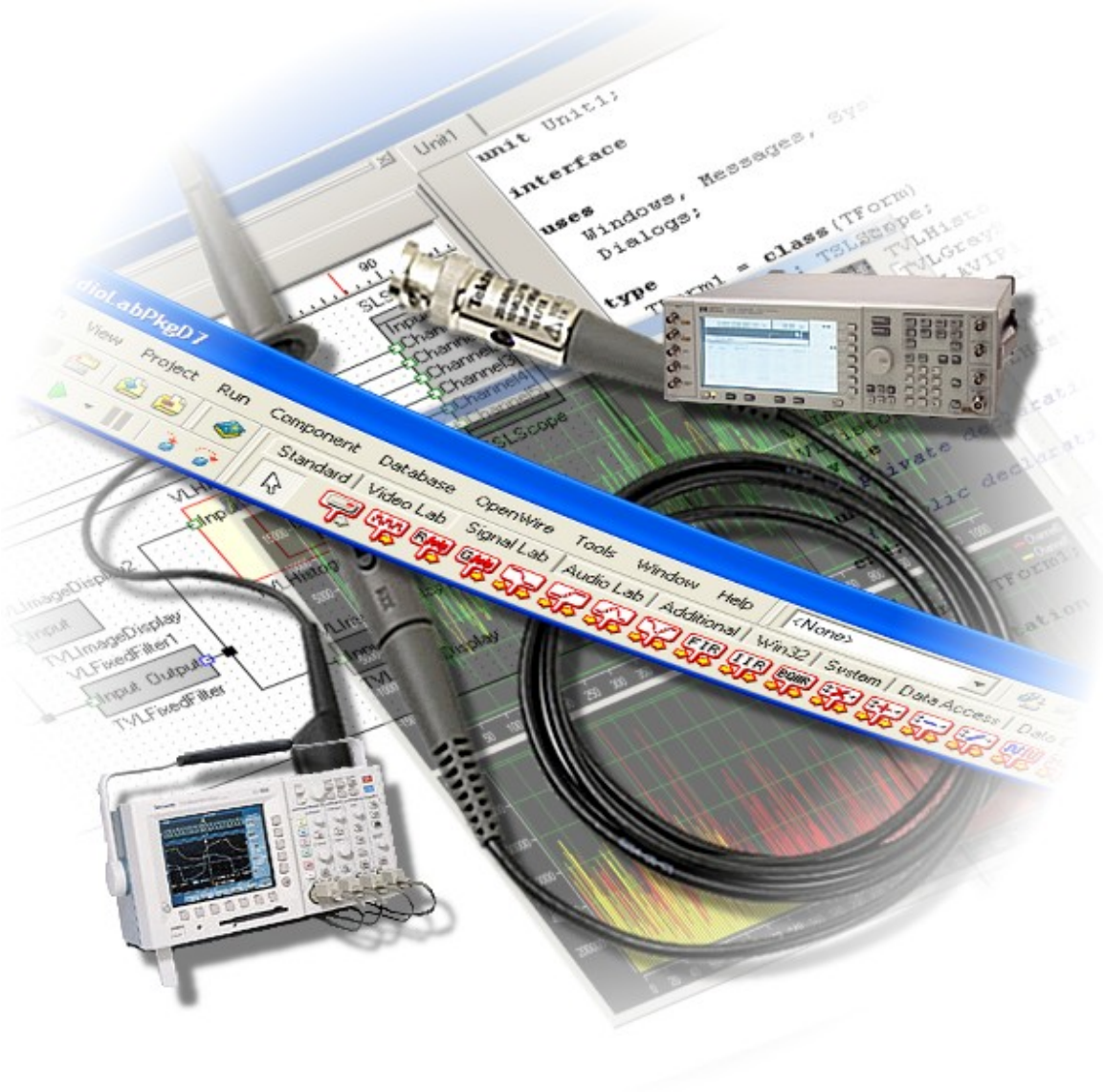
SignalLab 5.0

Visual C++ Quick Start



www.openwire.org
www.mitov.com

Index

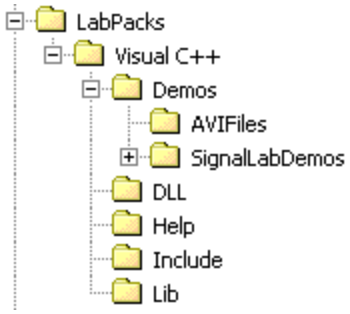
Installation....3
Where is SignalLab....3
Creating a new SignalLab project in Visual C++....3
Creating a simple signal generating application....13
Creating applications using filters, FFT, Waterfall and data logger....22
Using the TSLCRealBuffer in C++ Builder and Visual C++....50
Distributing your application....53
Deploying your application with the IPP DLLs....53

Installation

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

Where is SignalLab

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



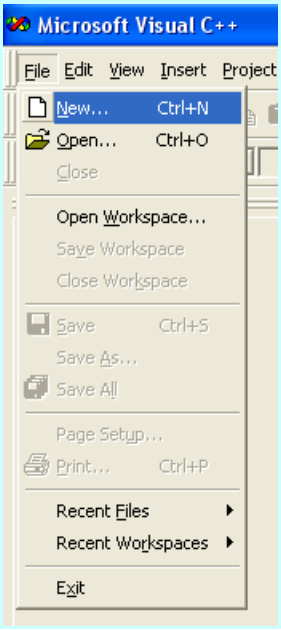
Under the SignalLabDemos directory are located the demo files. The help files and the documentation are located under the Help directory. The DLL directory contains the redistributable DLL files. The header files needed for your projects are located under the Include directory. The Release and Debug 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 SignalLab project in Visual C++

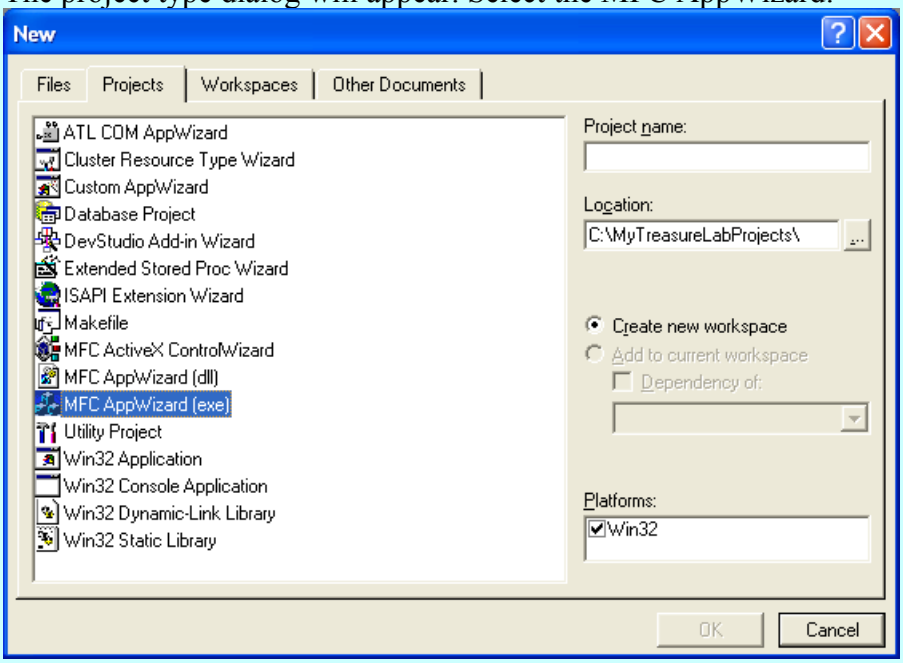
All of the examples in this manual start with creating a MFC Dialog based project. This is not a SignalLab requirement, but using the resource editor to design the application makes writing the examples much easier.
The following chapters will assume that you have created the project and will teach you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

Visual C++ 6.0:

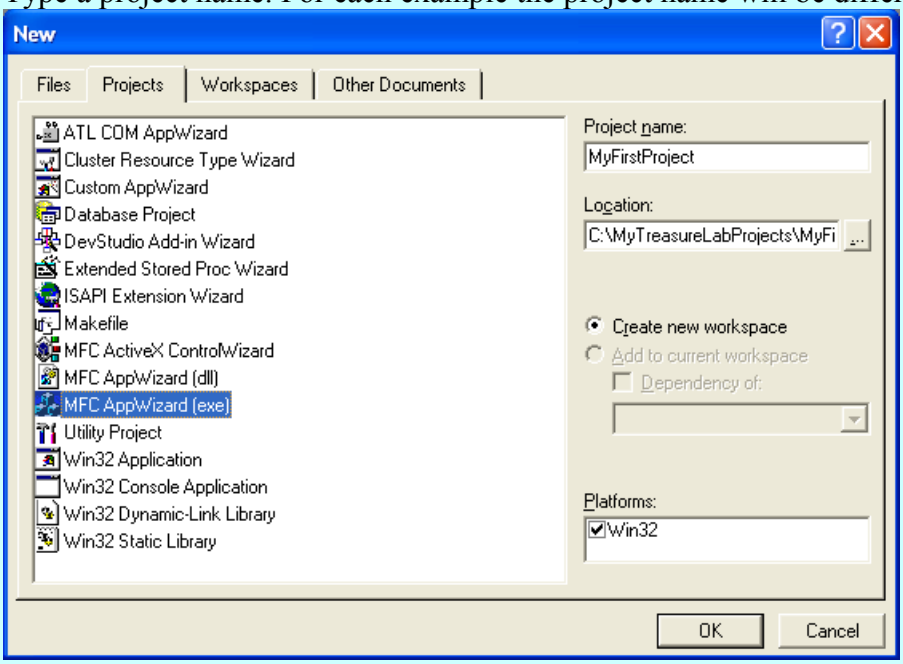
Start by creating a new project.
From the VC++ menu, select | File | New…|



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

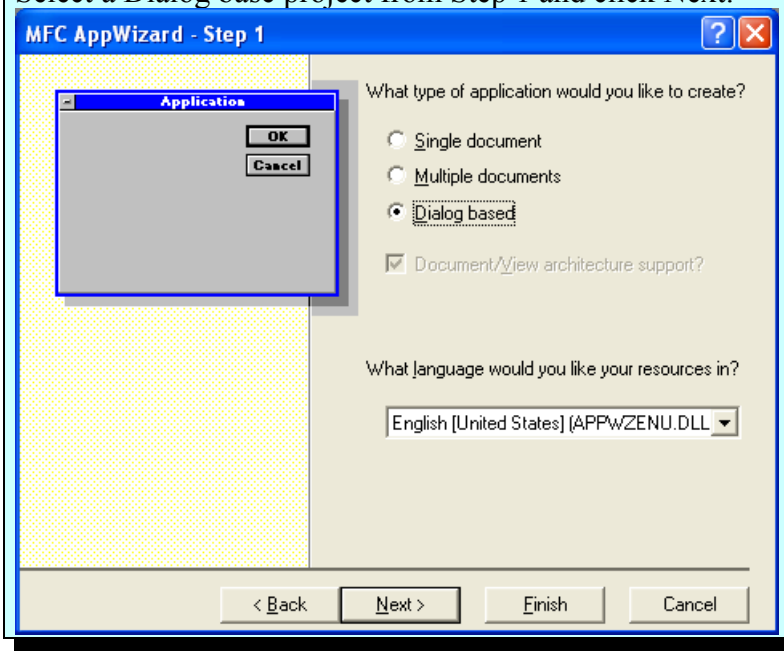


Type a project name. For each example the project name will be different:

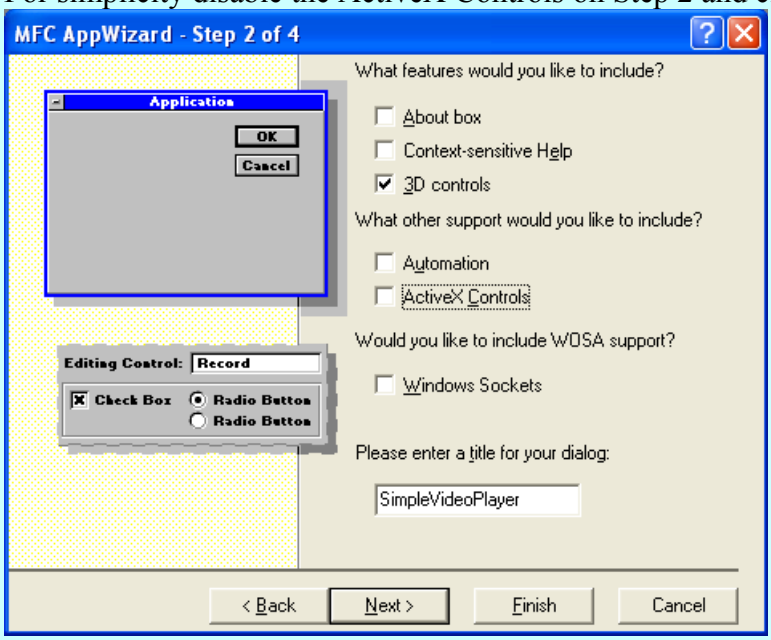


Click OK.

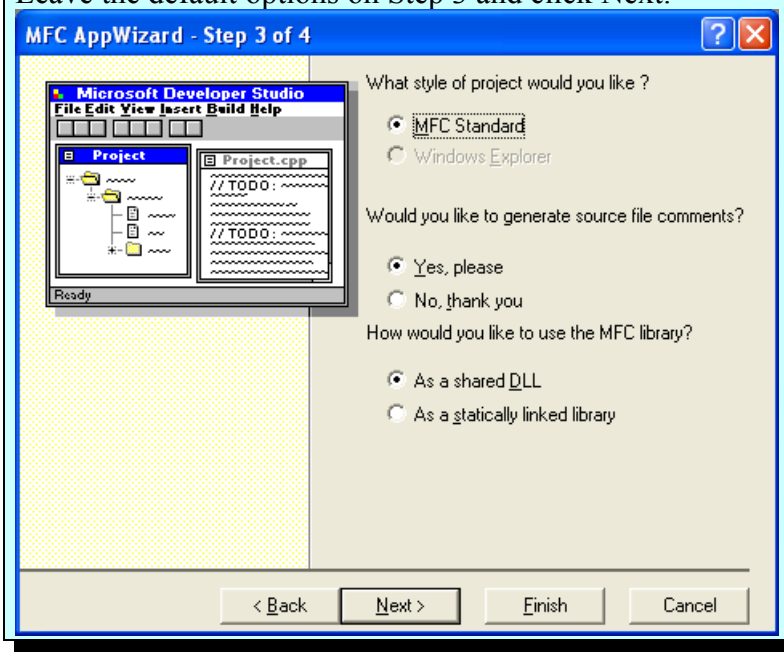
Select a Dialog base project from Step 1 and click Next:



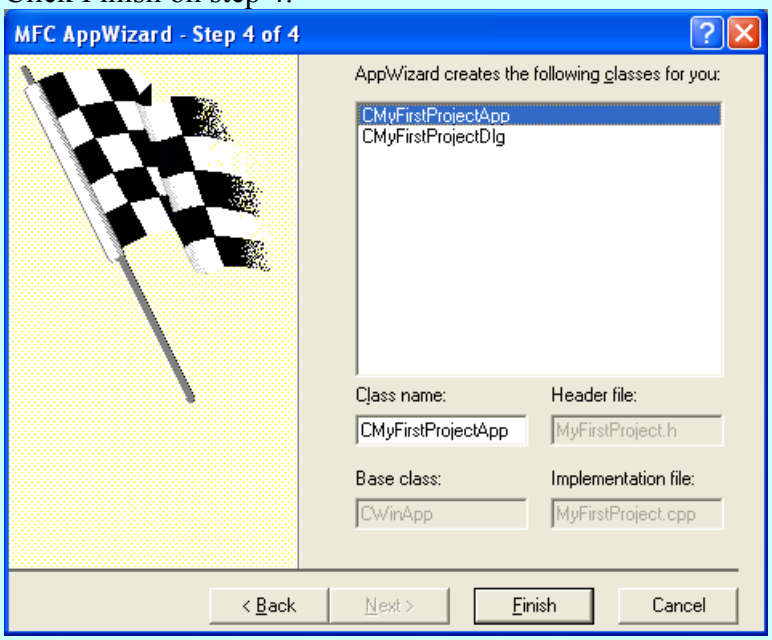
For simplicity disable the ActiveX Controls on Step 2 and click Next:



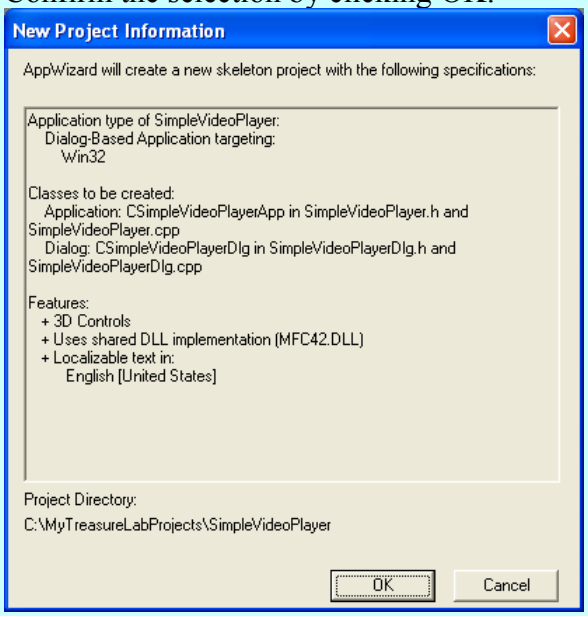
Leave the default options on Step 3 and click Next:



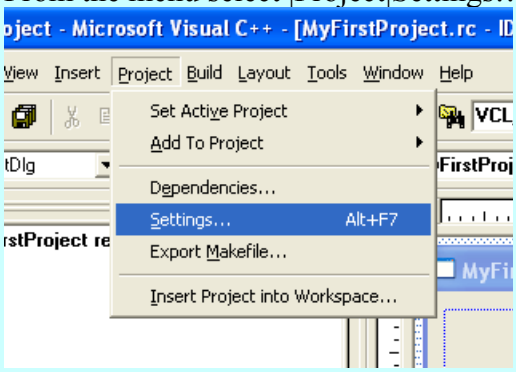
Click Finish on step 4:



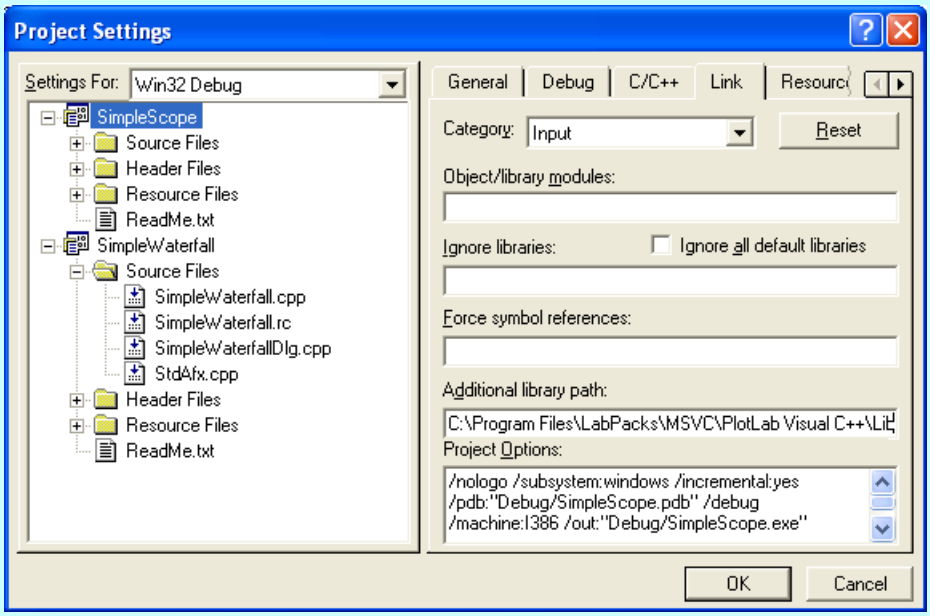
Confirm the selection by clicking OK:



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

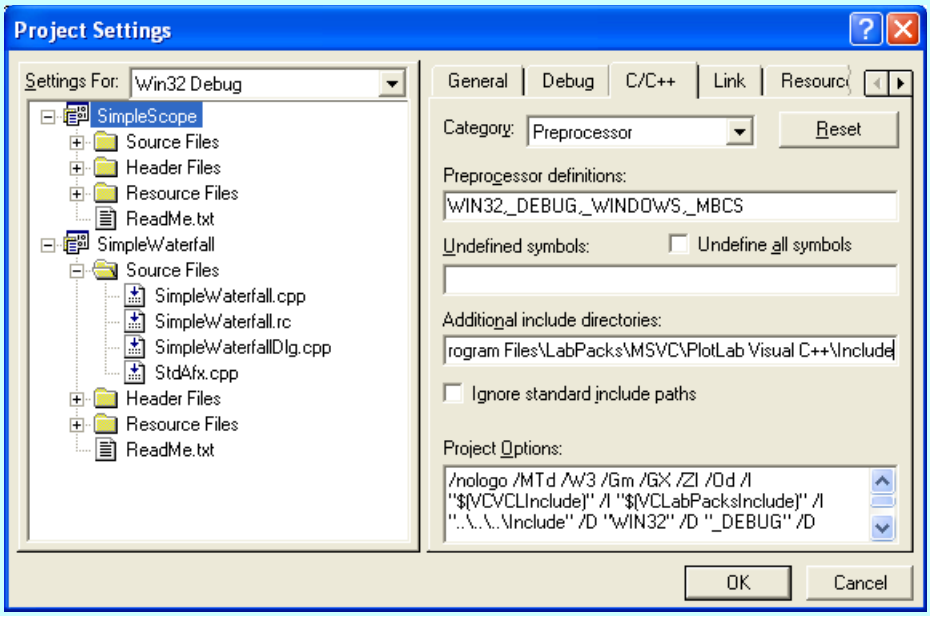


In the Project Settings dialog select the | Link | tab and in the “. Switch to the “Input” cathegory. In the “Additional library path:” edit box add the path to the library files. If you have followed the default installation it should be located at C:\Program Files\LabPacks\Visual C++\Lib:



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:

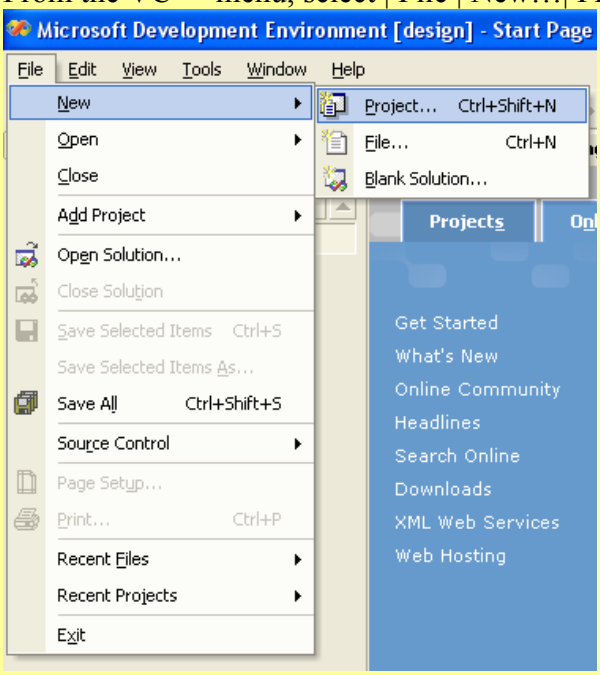


Click OK.

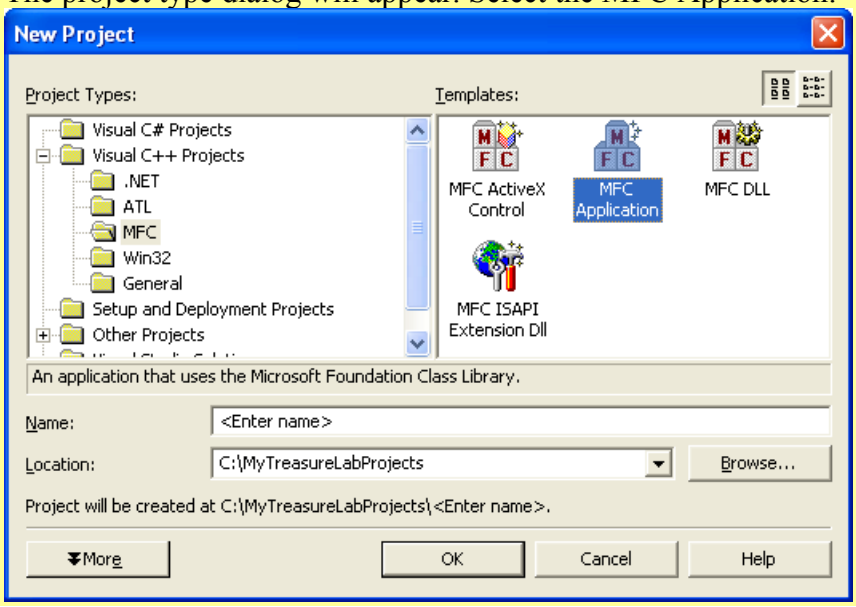
Now you have fully configured project, and you can start writing the actual code.

Visual C++ 2003:

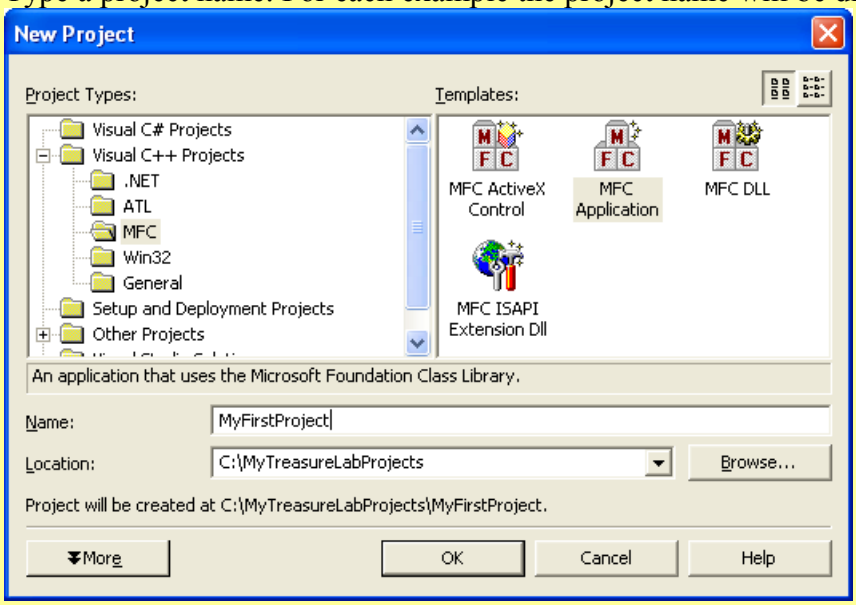
Start by creating a new project.
From the VC++ menu, select | File | New…| Project… |



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

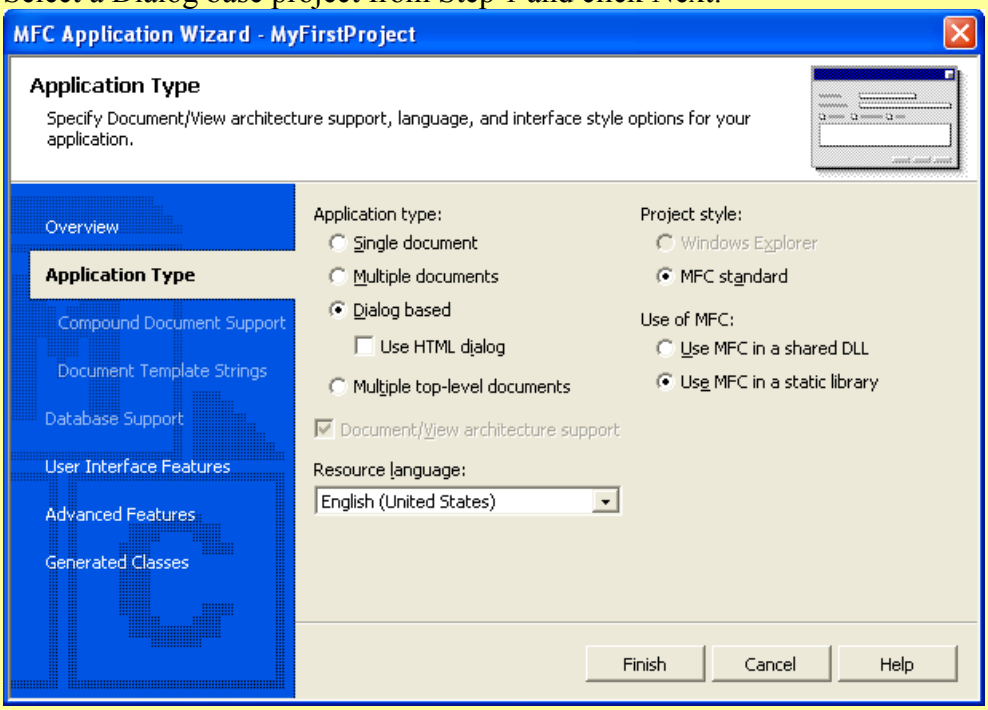


Type a project name. For each example the project name will be different:

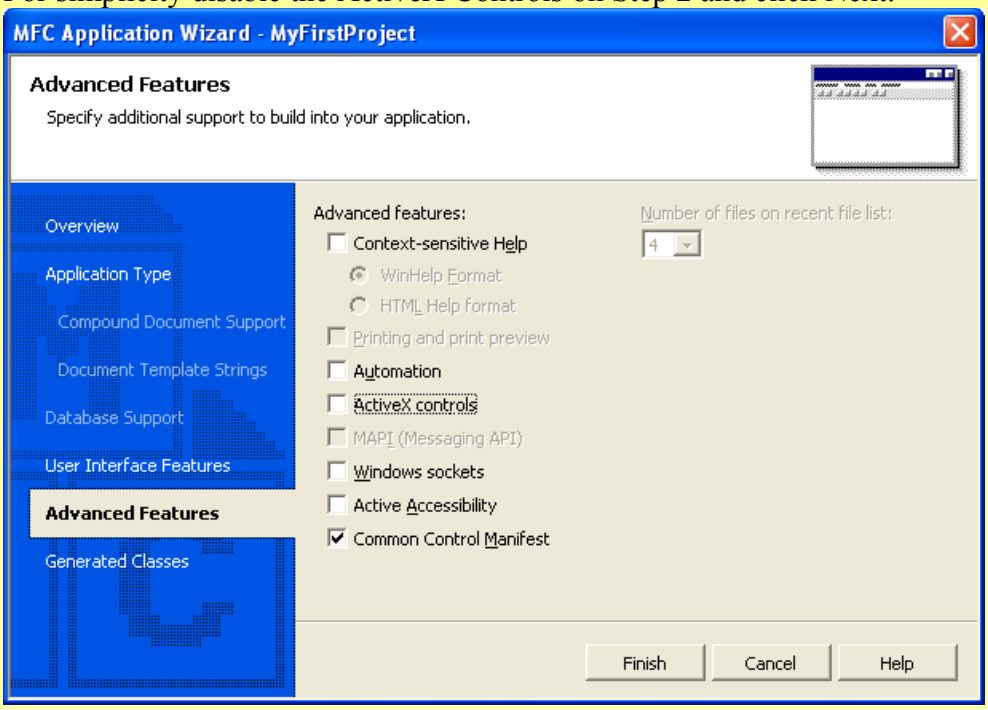


Click OK.

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

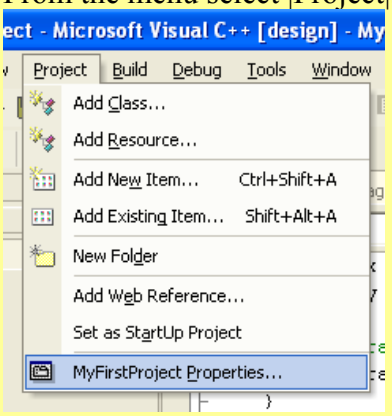


For simplicity disable the ActiveX Controls on Step 2 and click Next:

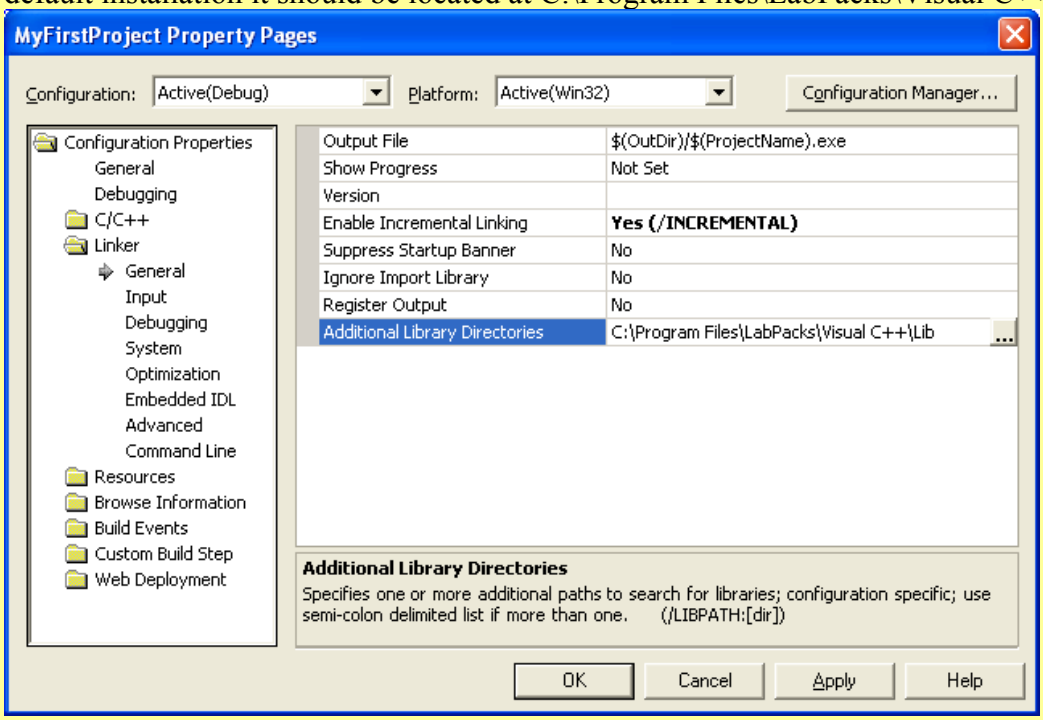


Click Finish.

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

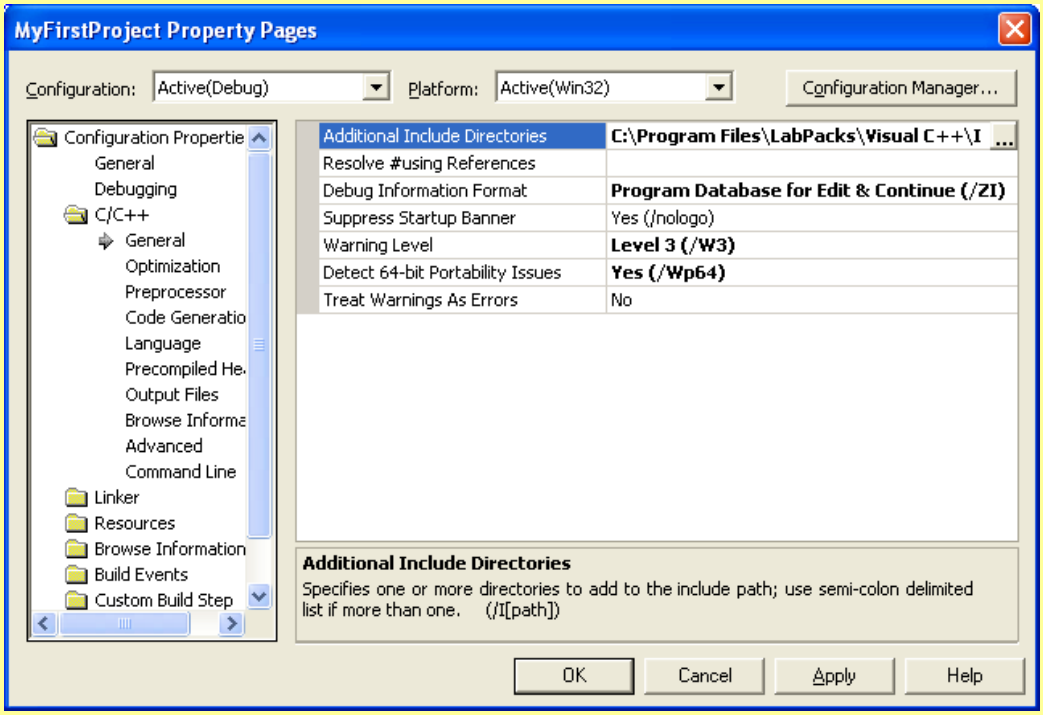


In the Project Property dialog select the Linker General page. In the "Additional library directories:" edit box add the path to the library files. If you have followed the default installation it should be located at C:\Program Files\LabPacks\Visual C++\Lib:



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:



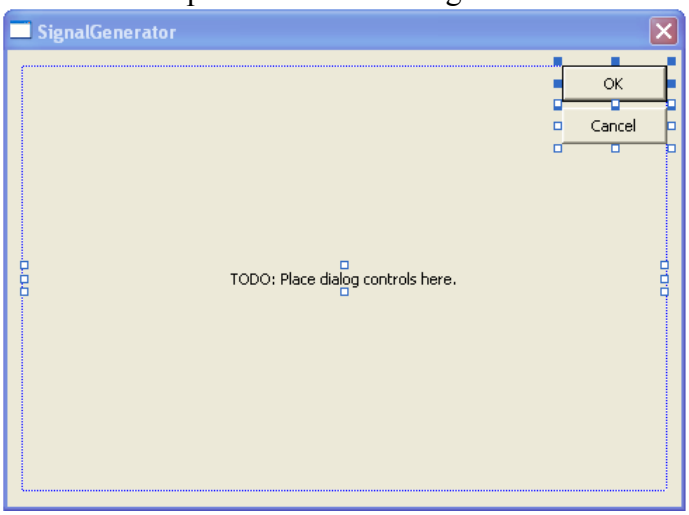
Click OK.

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

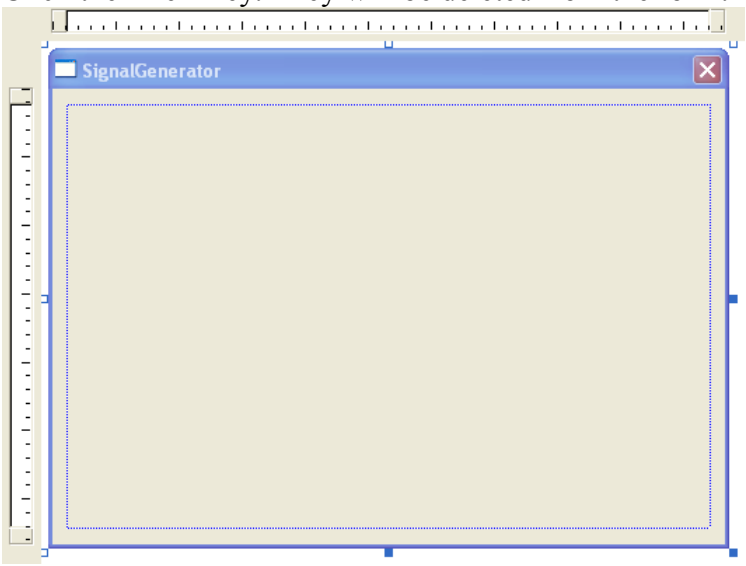
Creating a simple signal generating application

Create and setup a new project named SignalGenerator as described in the “Creating a new SignalLab project in Visual C++” chapter.

Select the components on the dialog form:



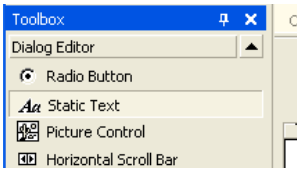
Click the “Del” key. They will be deleted from the form:



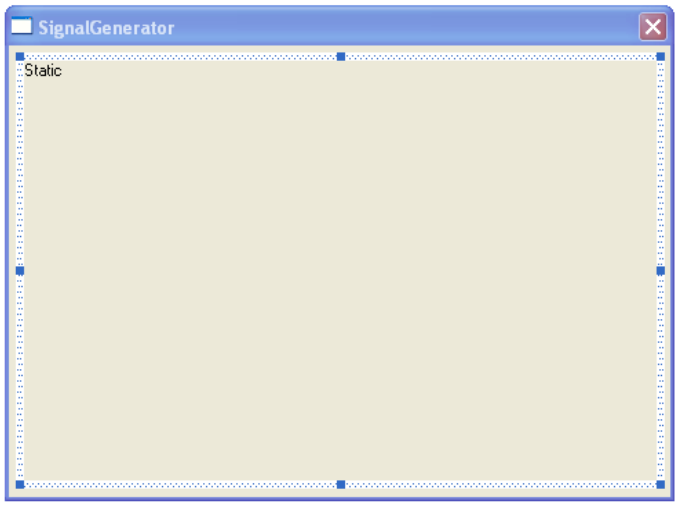
From the controls toolbar select a “Static Text” control:

VC++ 6:

VC2003/2005:

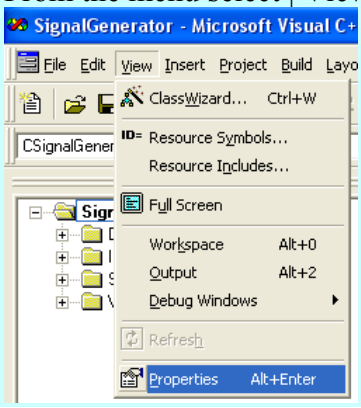


Place the control on the form:

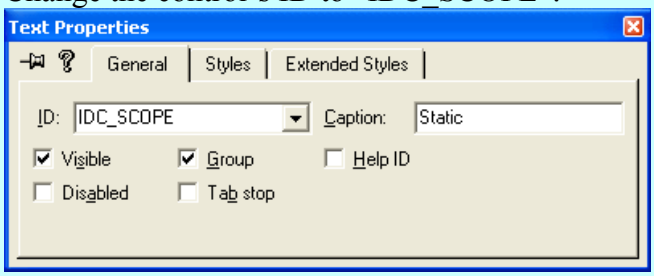


In Visual C++ 6.0:

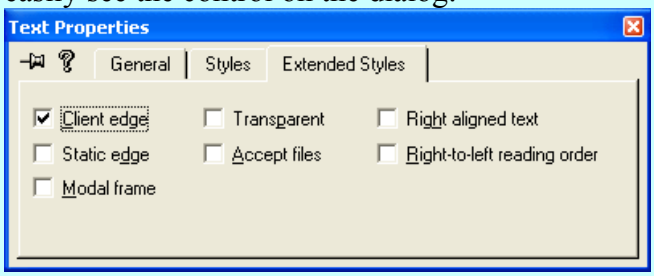
From the menu select | View | Properties |:



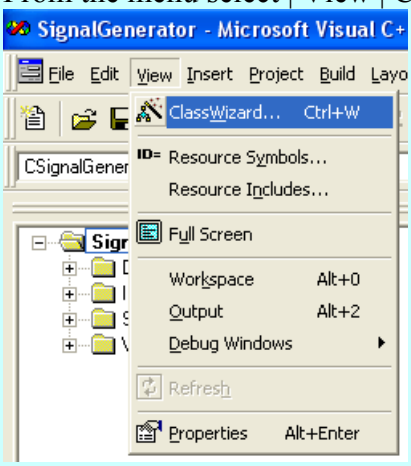
Change the control's ID to "IDC_SCOPE":



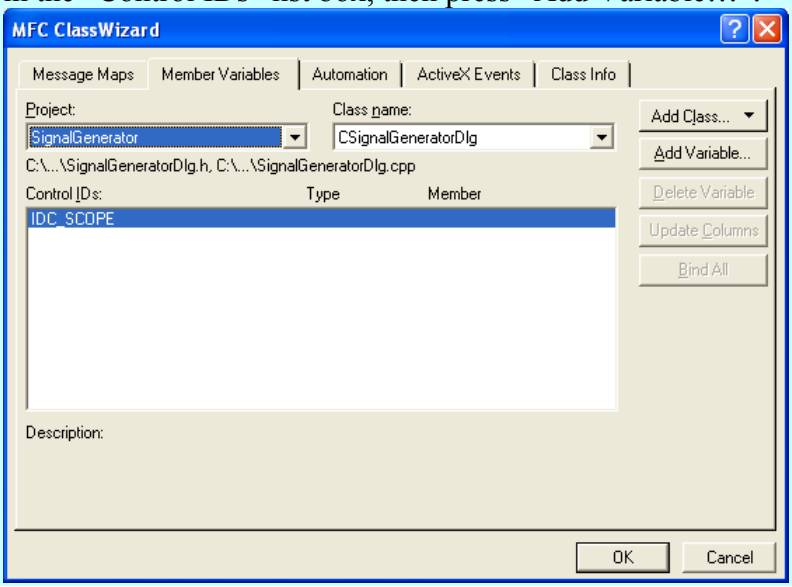
Switch to the "Extended Styles" tab and check the "Client edge" property so you can easily see the control on the dialog:



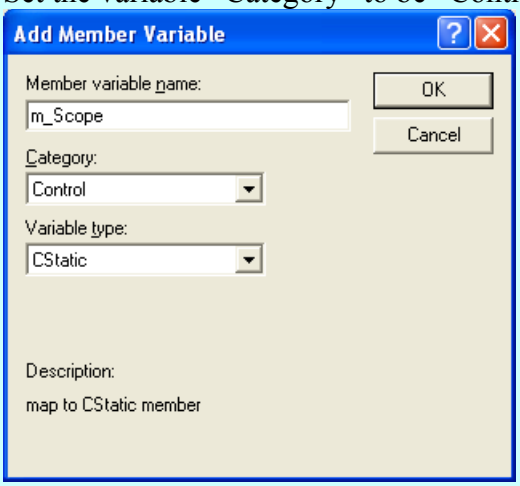
From the menu select | View | ClassWizard… |:



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

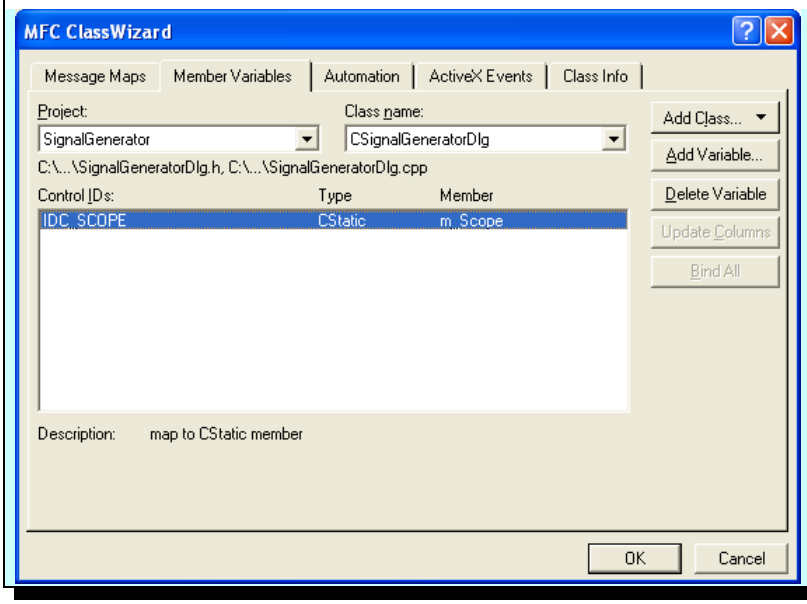


Set the variable “Category” to be “Control”, and set the name to be m_Scope:



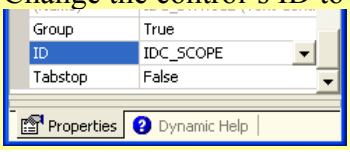
Click OK.

In the “ClassWizard” click OK:

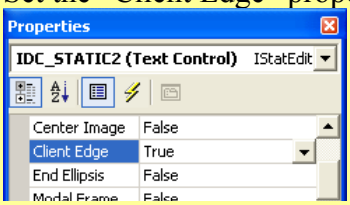


In Visual C++ 2003/2005:

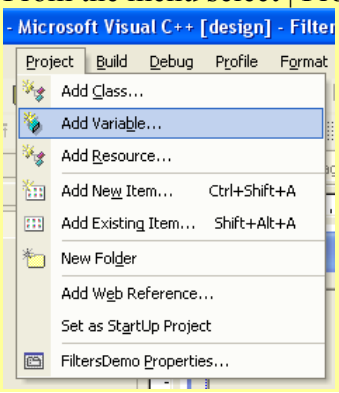
Change the control’s ID to “IDC_SCOPE”:



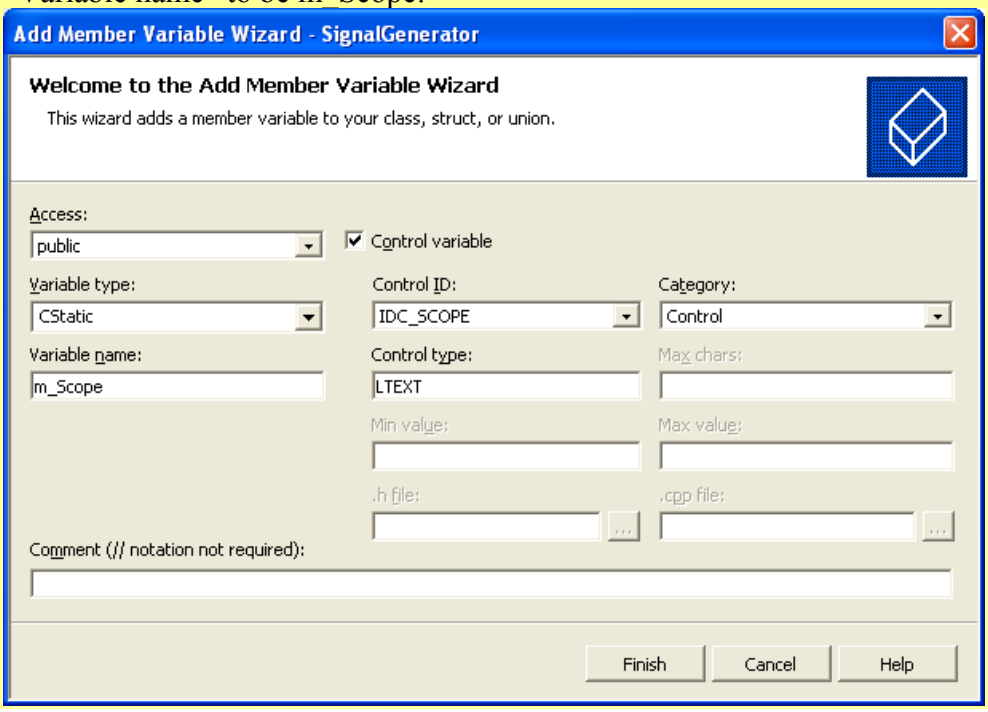
Set the “Client Edge” property to True so you can easily see the control on the dialog:



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



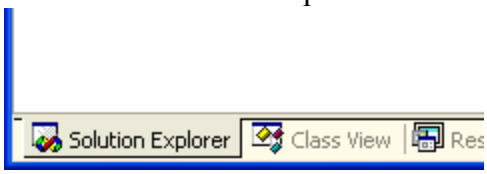
In the “Add Member Variable Wizard” set the “Variable type” to CStatic, and set the “Variable name” to be m_Scope:



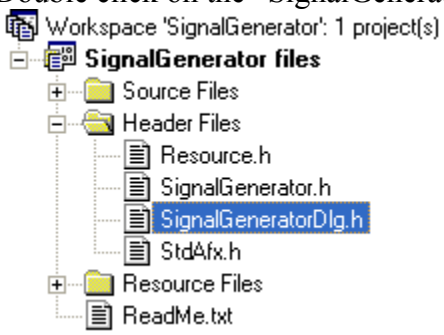
Click Finish.

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

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



Double click on the “SignalGeneratorDlg.h” file:

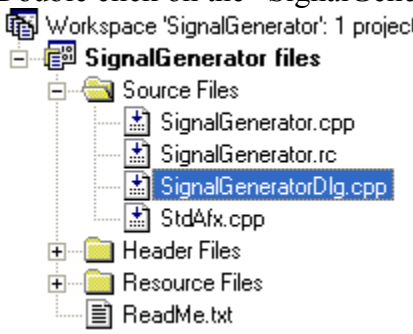


Add the highlighted lines in the header:

```
// SignalGeneratorDlg.h : header file
//

#pragma once
#include "afxwin.h"

#include <CSLScope.h>
#include <CSLSignalGen.h>

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

// Dialog Data
     enum { IDD = IDD_SIGNALGENERATOR_DIALOG };

     protected:
```

```
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support



// Implementation
protected:
	CTSLScope	SLScope;
	CTSLSignalGen SLSignalGen;

protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	CStatic m_Scope;
};
```

Double click on the "SignalGeneratorDlg.cpp" file:

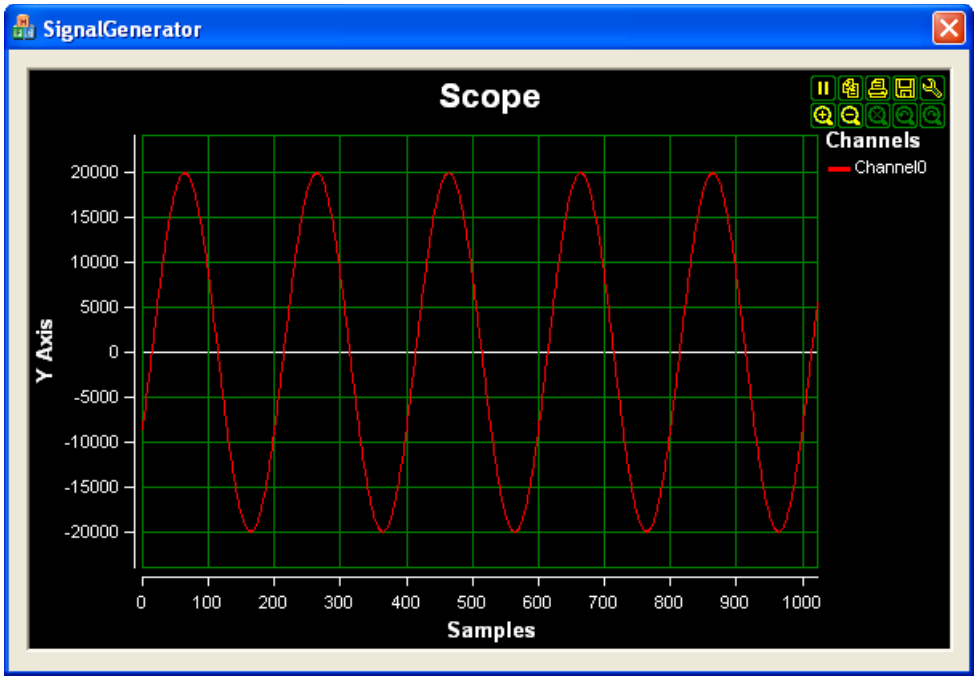


Add the highlighted lines in the CSignalGeneratorDlg::OnInitDialog method:

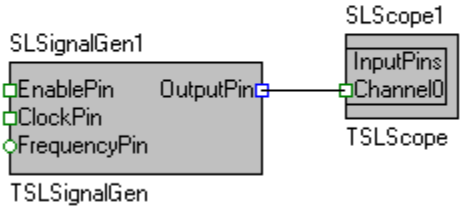
```
BOOL CSignalGeneratorDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.
```

```
	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// 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 );
	SLScope.Open( m_Scope.m_hWnd );

	SLSignalGen.OutputPin.Connect( SLScope.InputPins[ 0 ] );

	VCL_Loaded();

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

Compile and run the application:



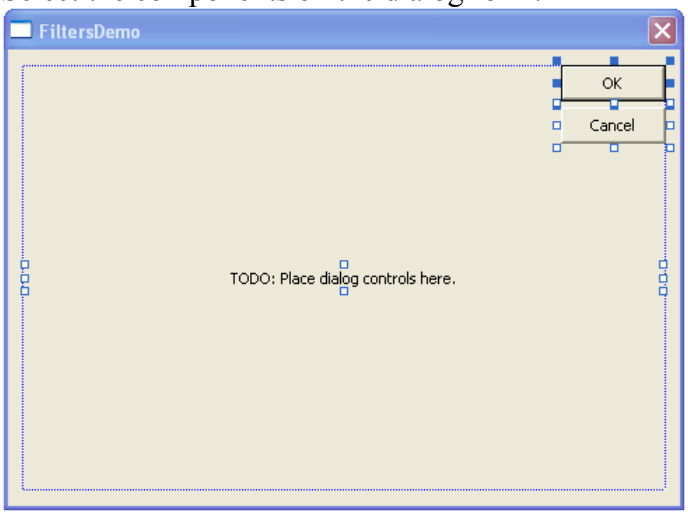
Congratulations! You have just created your first SignalLab application.
Here are the OpenWire connections in this application:



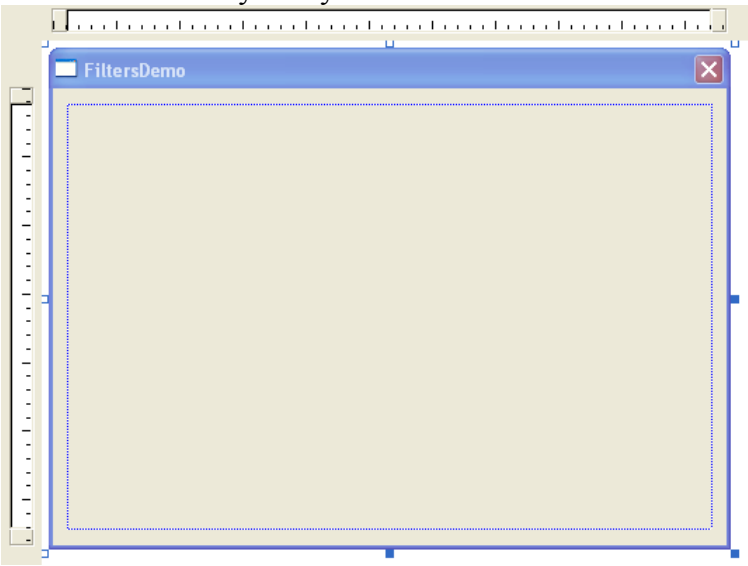
Creating applications using filters, FFT, Waterfall and data logger

Create and setup a new project named FiltersDemo as described in the “Creating a new SignalLab project in Visual C++” chapter.

Select the components on the dialog form:



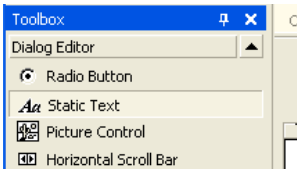
Click the “Del” key. They will be deleted from the form:



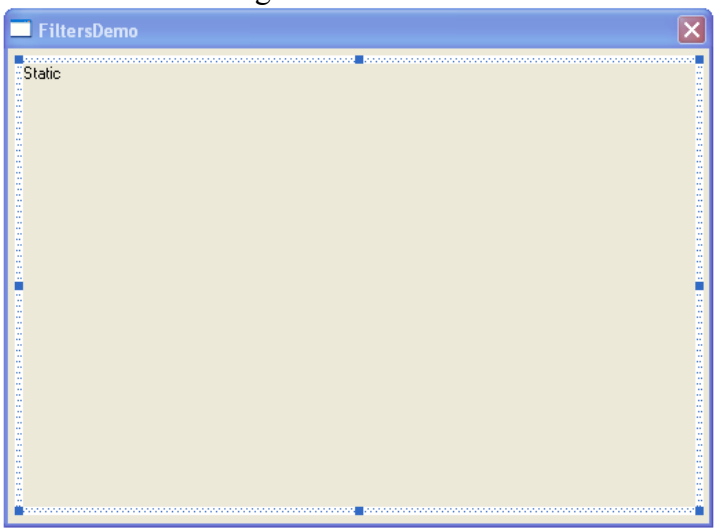
From the controls toolbar select a “Static Text” control.

VC++ 6:

VC2003/2005:

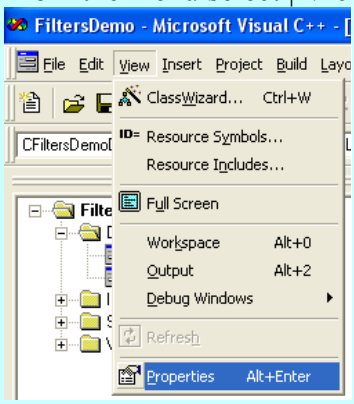


Place it on the dialog form:

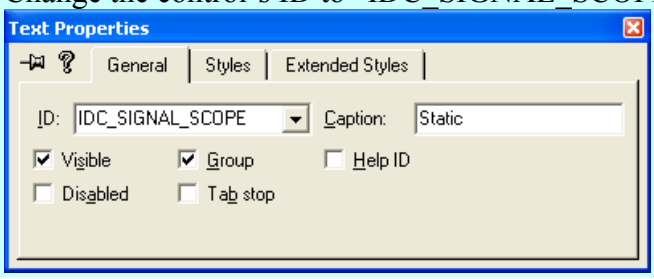


In Visual C++ 6.0:

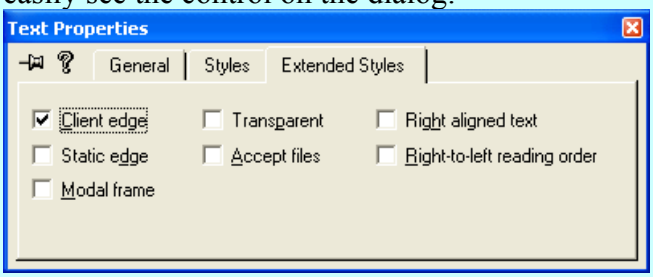
From the menu select | View | Properties |:



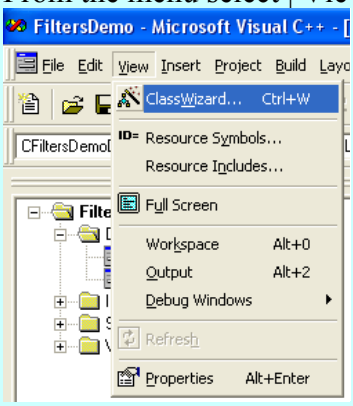
Change the control's ID to "IDC_SIGNAL_SCOPE":



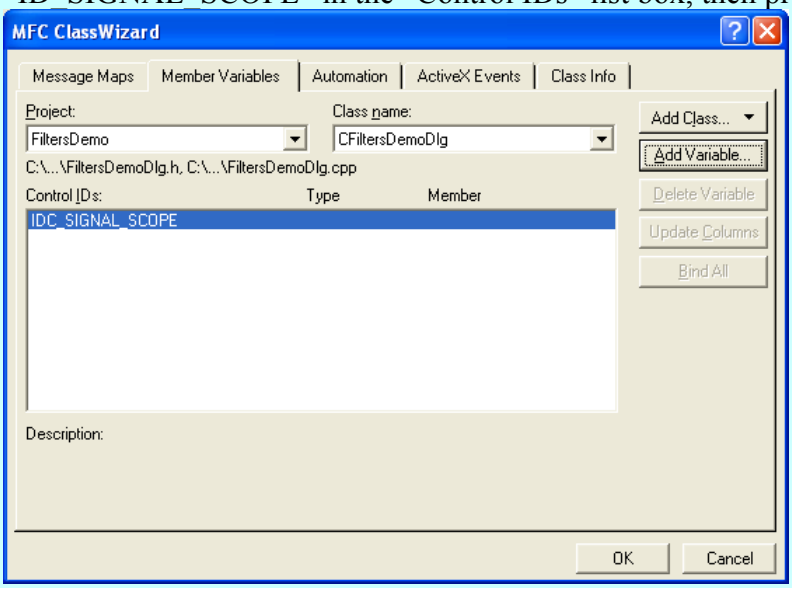
Switch to the “Extended Styles” tab and check the “Client edge” property so you can easily see the control on the dialog:



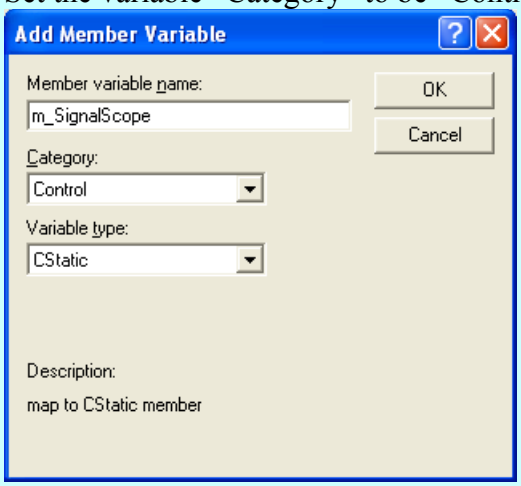
From the menu select | View | ClassWizard… |:



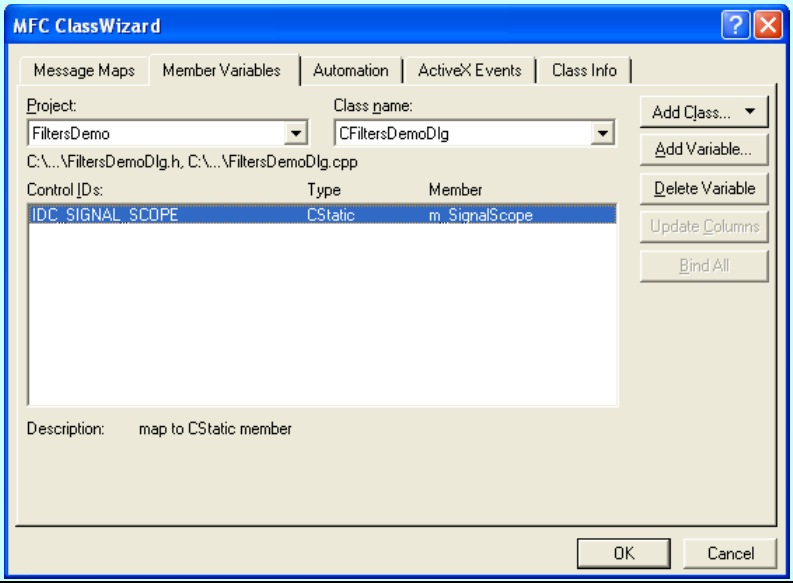
In the “ClassWizard” select the “Member Variables” tab and select the “ID_SIGNAL_SCOPE” in the “Control IDs” list box, then press “Add Variable…”:



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

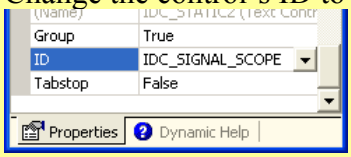


In the “ClassWizard” click OK

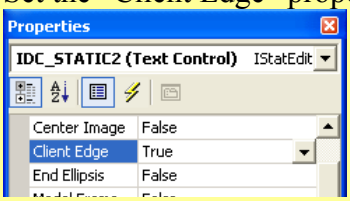


In Visual C++ 2003/2005:

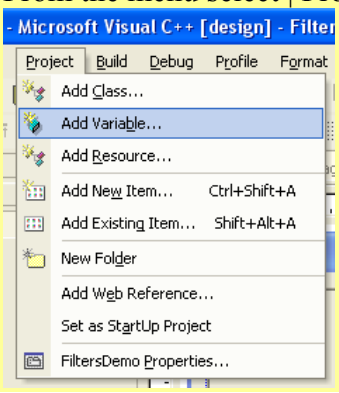
Change the control’s ID to “IDC_SIGNAL_SCOPE”:



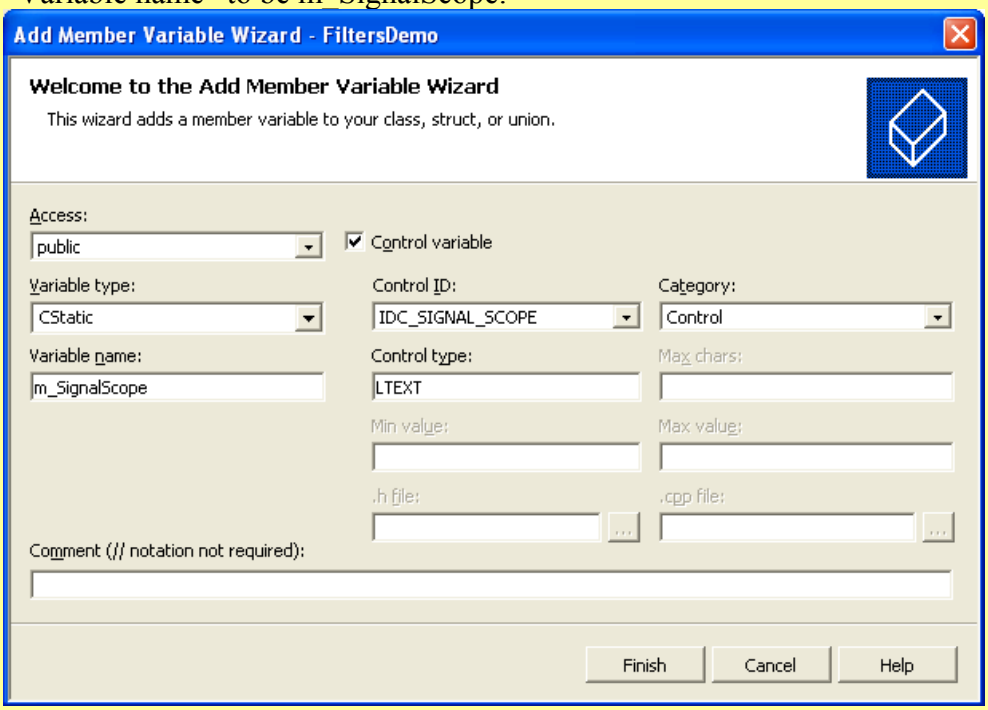
Set the “Client Edge” property to True so you can easily see the control on the dialog:



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



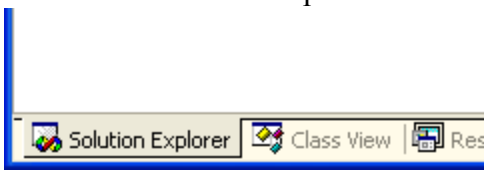
In the “Add Member Variable Wizard” set the “Variable type” to CStatic, and set the “Variable name” to be m_SignalScope:



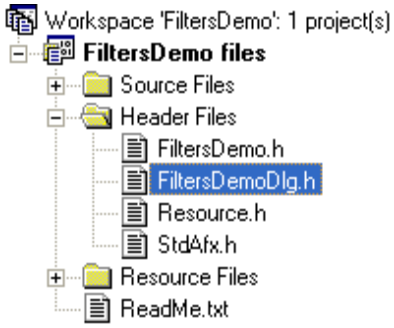
Click Finish.

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

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



Double click on the “FiltersDemoDlg.h”:

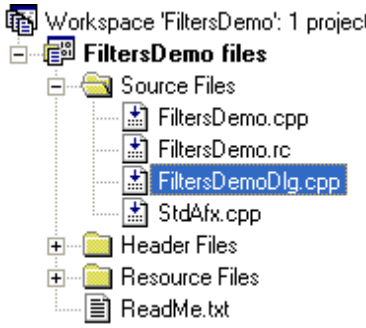


Add the highlighted lines in the header:

```
// FiltersDemoDlg.h : header file
//

#pragma once
#include "afxwin.h"

#include <CSLScope.h>
#include <CSLLowPass.h>
#include <CSLRandomGen.h>

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

// Dialog Data
     enum { IDD = IDD_FILTERSDEMO_DIALOG };

     protected:
```

```
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support



// Implementation
protected:
	CTSLScope		SLScope;
	CTSLLowPass		SLLowPass;
	CTSLRandomGen		SLRandomGen;

protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	CStatic m_SignalScope;
};
```

Double click on the "FiltersDemoDlg.cpp" file:



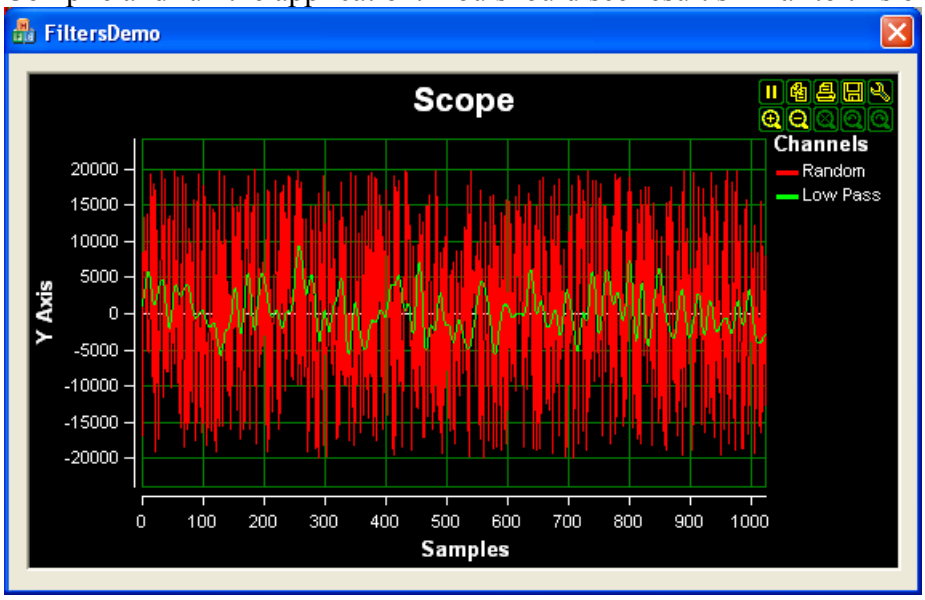
Add the highlighted lines in the CFiltersDemoDlg::OnInitDialog method:

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

	// Add "About..." menu item to system menu.
```

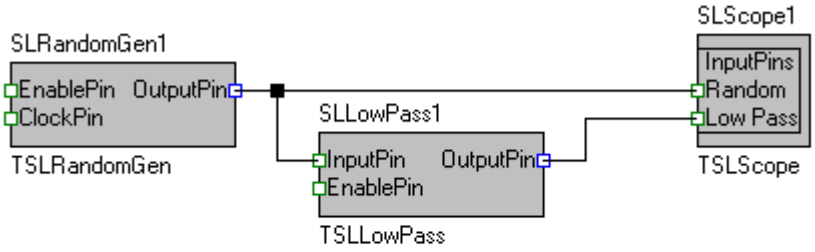
```
	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// 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 );
	SLScope.Open( m_SignalScope.m_hWnd );
	SLScope.Channels.Clear();
	SLScope.Channels.Add( 2 );

	SLScope.Channels[ 0 ].Name = "Random";
	SLScope.Channels[ 1 ].Name = "Low Pass";

	SLRandomGen.OutputPin.Connect( SLScope.InputPins[ 0 ] );
	SLRandomGen.OutputPin.Connect( SLLowPass.InputPin );
	SLLowPass.OutputPin.Connect( SLScope.InputPins[ 1 ] );
```

```
	VCL_Loaded();


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

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



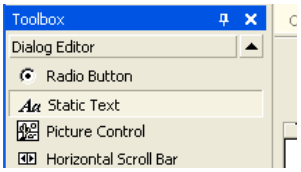
You have just created your first Low Pass filtering application with SignalLab!
Here are the OpenWire connections in this application:



Make the form big enough to accommodate another plotting component, by expanding the height. Rearrange the existing component if needed.
From the controls toolbar select a “Static Text” control and place it on the dialog form:

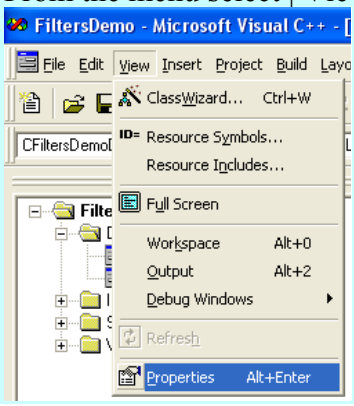
VC++ 6:

VC2003/2005:

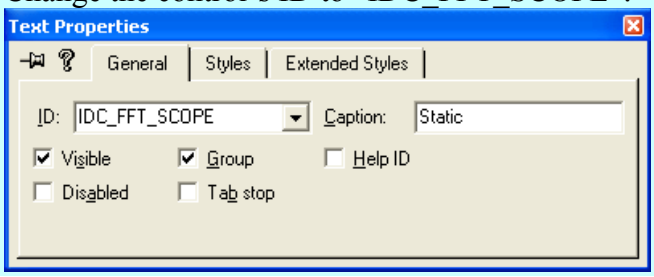


In Visual C++ 6.0:

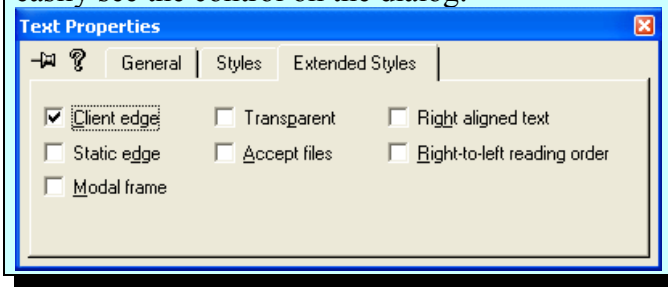
From the menu select | View | Properties |:



Change the control's ID to "IDC_FFT_SCOPE":

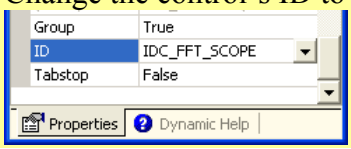


Switch to the "Extended Styles" tab and check the "Client edge" property so you can easily see the control on the dialog:

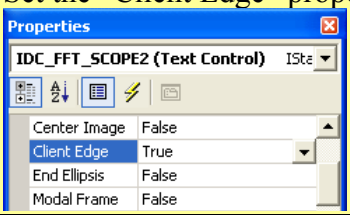


In Visual C++ 2003/2005:

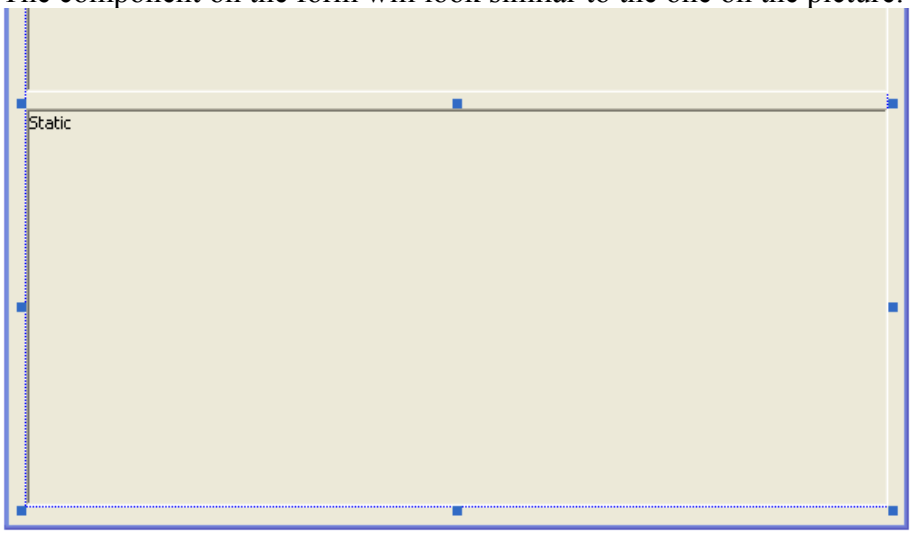
Change the control's ID to "IDC_FFT_SCOPE":



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

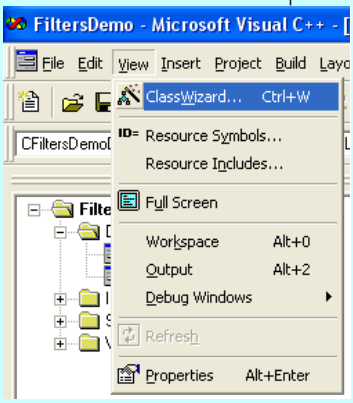


The component on the form will look similar to the one on the picture:

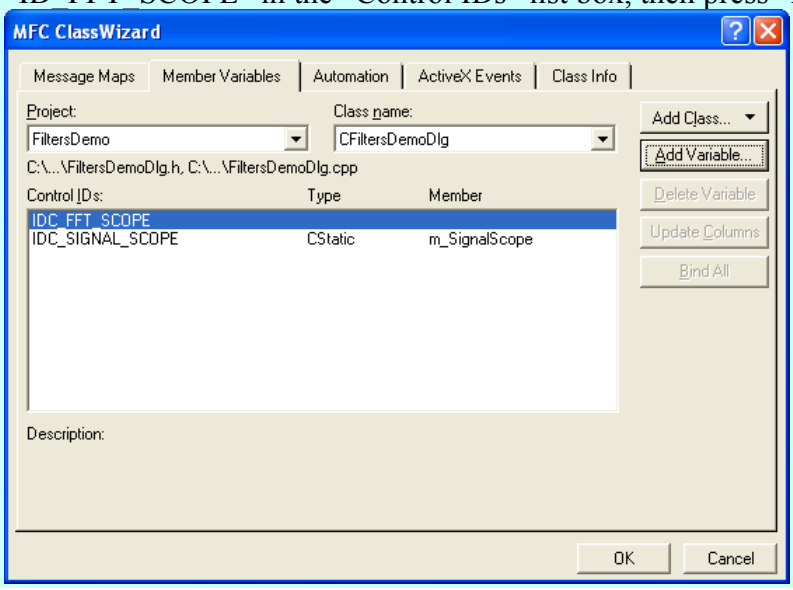


In Visual C++ 6.0:

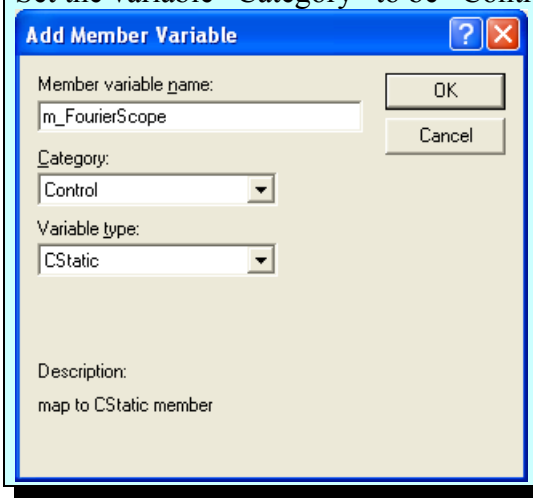
From the menu select | View | ClassWizard… |:



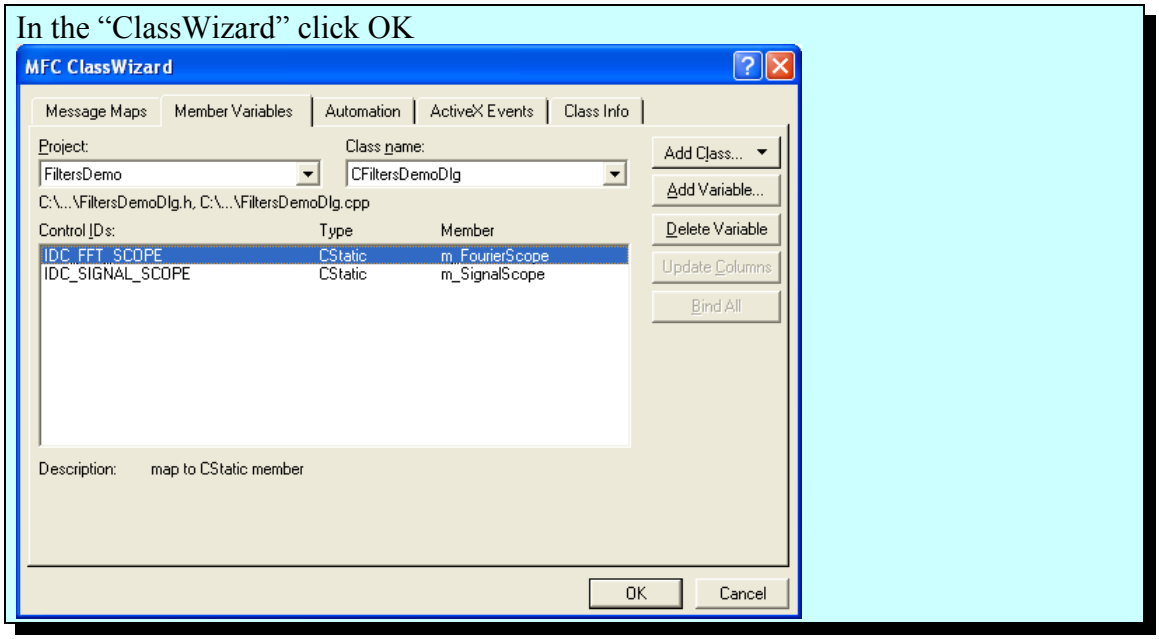
In the “ClassWizard” select the “Member Variables” tab and select the “ID_FFT_SCOPE” in the “Control IDs” list box, then press “Add Variable…”:



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

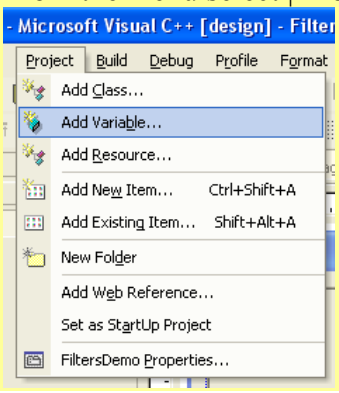


In the “ClassWizard” click OK

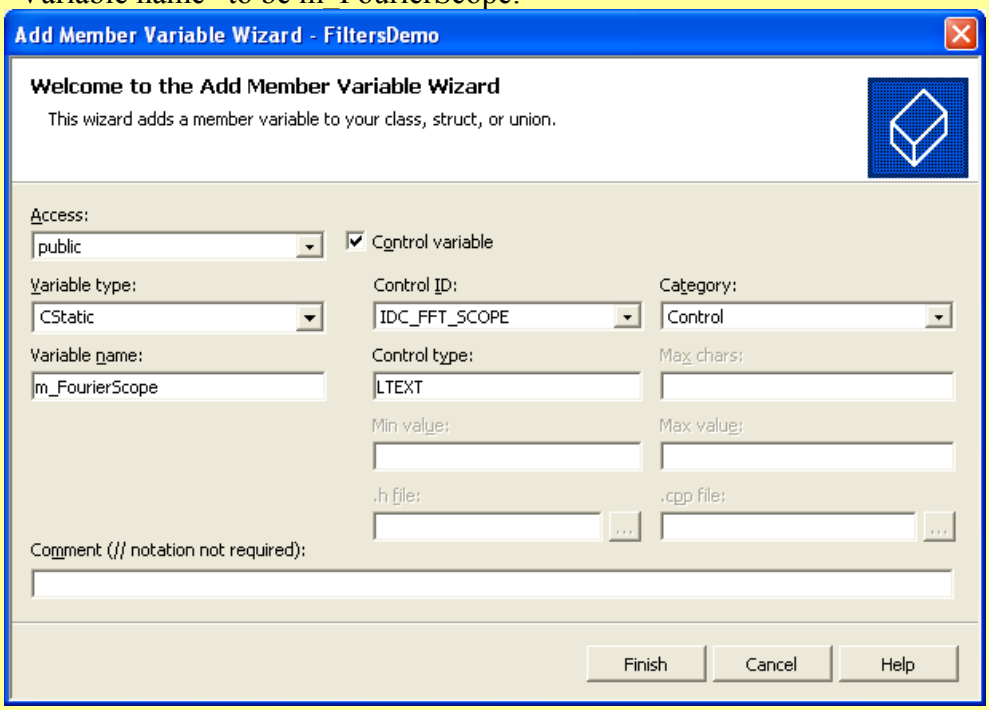


In Visual C++ 2003/2005:

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



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



Click Finish.

Add the highlighted lines in the header:

```
// FiltersDemoDlg.h : header file
//

#pragma once
#include "afxwin.h"

#include <CSLScope.h>
#include <CSLLowPass.h>
#include <CSLRandomGen.h>
#include <CSLFourier.h>

// CFiltersDemoDlg dialog
class CFiltersDemoDlg : public CDialog
{
// Construction
public:
```

```
	CFiltersDemoDlg(CWnd* pParent = NULL);	// standard
constructor


// Dialog Data
	enum { IDD = IDD_FILTERSDEMO_DIALOG };


	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support



// Implementation
protected:
	CTSLScope		SLScope;
	CTSLScope		SLScopeFourier;
	CTSLLowPass		SLLowPass;
	CTSLRandomGen		SLRandomGen;
	CTSLFourier		SLFourierRandom;
	CTSLFourier		SLFourierLowPass;


protected:
	HICON m_hIcon;


	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	CStatic m_SignalScope;
	CStatic m_FourierScope;
};
```

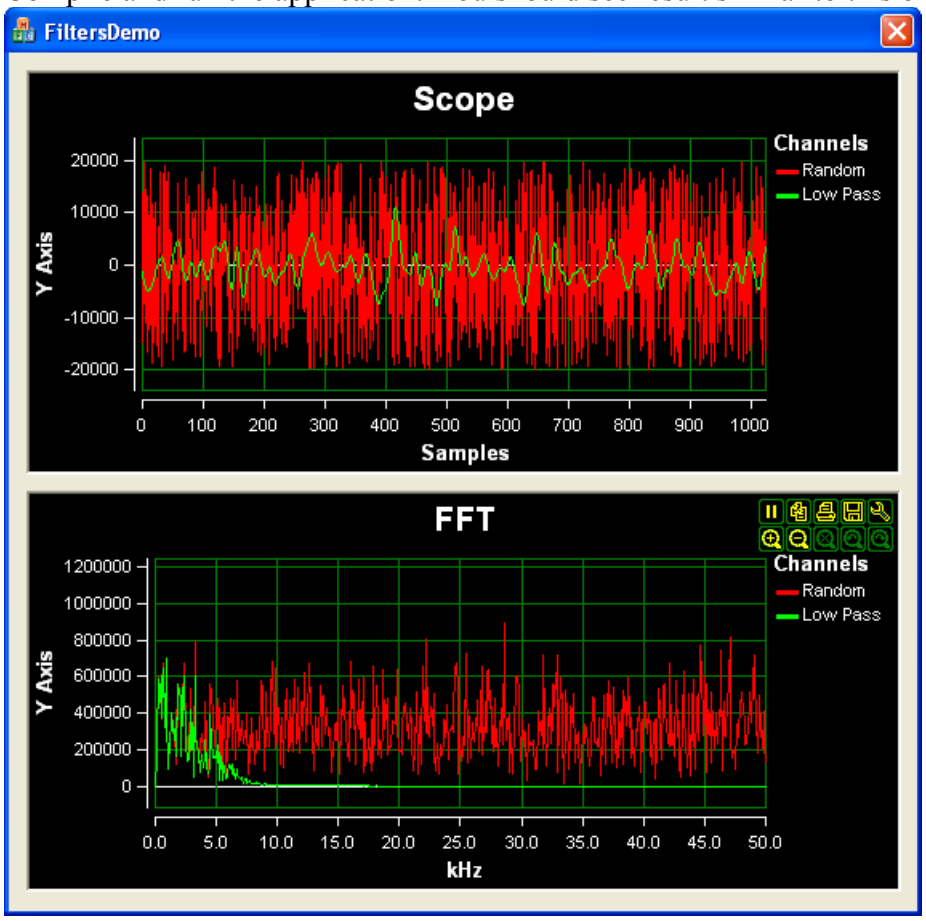
Add the highlighted lines in the CFiltersDemoDlg::OnInitDialog method:

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

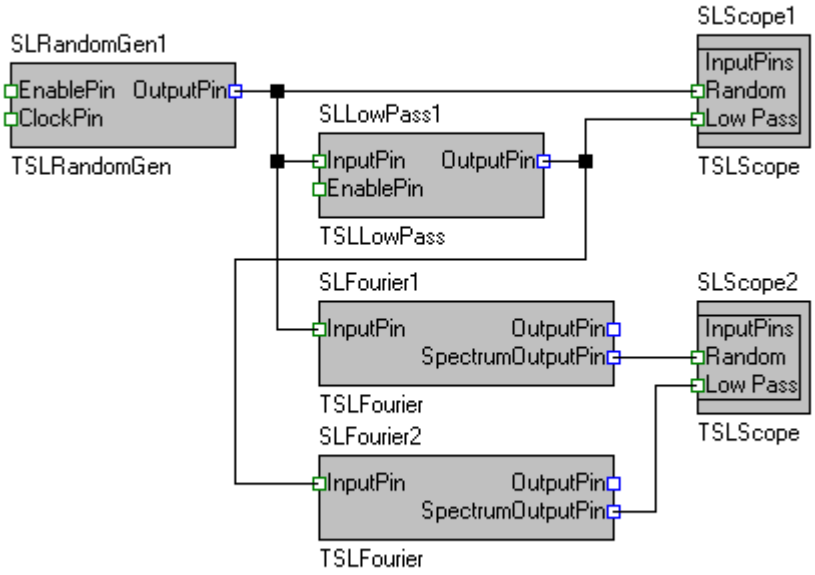
```
	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// 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 );
	SLScope.Open( m_SignalScope.m_hWnd );
	SLScope.Channels.Clear();
	SLScope.Channels.Add( 2 );

	SLScope.Channels[ 0 ].Name = "Random";
	SLScope.Channels[ 1 ].Name = "Low Pass";

	SLScopeFourier.Open( m_FourierScope.m_hWnd );
```

```
      SLScopeFourier.Title.Text = "FFT";

      SLScopeFourier.Channels.Clear();
      SLScopeFourier.Channels.Add( 2 );

      SLScopeFourier.Channels[ 0 ].Name = "Random";
      SLScopeFourier.Channels[ 1 ].Name = "Low Pass";

      SLRandomGen.OutputPin.Connect( SLScope.InputPins[ 0 ] );
      SLRandomGen.OutputPin.Connect( SLLowPass.InputPin );
      SLRandomGen.OutputPin.Connect( SLFourierRandom.InputPin );

      SLLowPass.OutputPin.Connect( SLScope.InputPins[ 1 ] );
      SLLowPass.OutputPin.Connect( SLFourierLowPass.InputPin );

SLFourierRandom.SpectrumOutputPin.Connect( SLScopeFourier.InputPins[
0 ] );

SLFourierLowPass.SpectrumOutputPin.Connect( SLScopeFourier.InputPins[
1 ] );

      VCL_Loaded();

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

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



Here are the OpenWire connections in the application now:

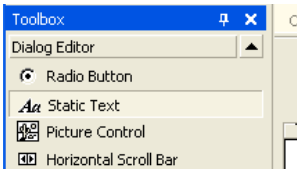


Now we will add a Waterfall component to display the FFT Spectrum of the filtered signal and a file logger to record the filtered data.

Make the form big enough to accommodate another plotting component, by expanding the height. Rearrange the existing components if needed.
From the controls toolbar select a “Static Text” control and place it on the dialog form:

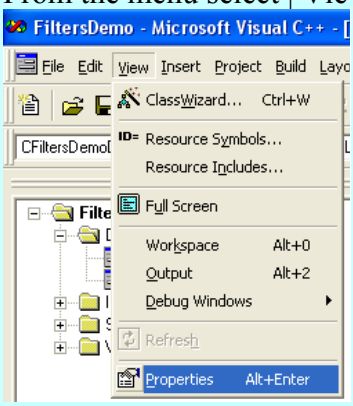
VC++ 6:

VC2003/2005:

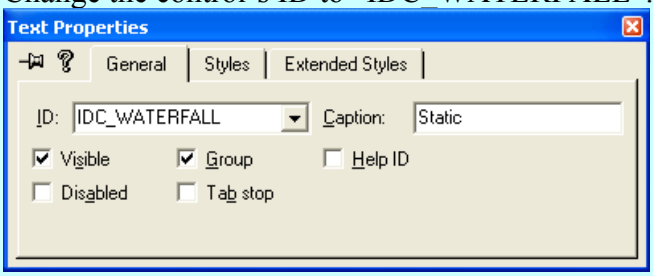


In Visual C++ 6.0:

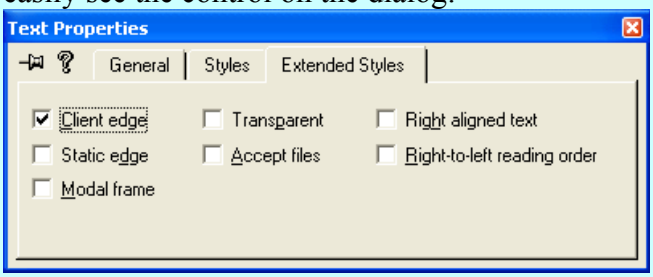
From the menu select | View | Properties |:



Change the control’s ID to “IDC_WATERFALL”:

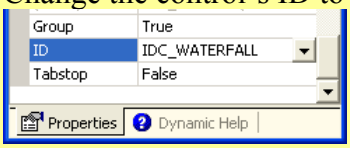


Switch to the “Extended Styles” tab and check the “Client edge” property so you can easily see the control on the dialog:

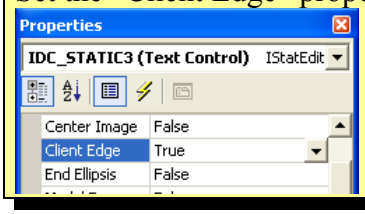


In Visual C++ 2003/2005:

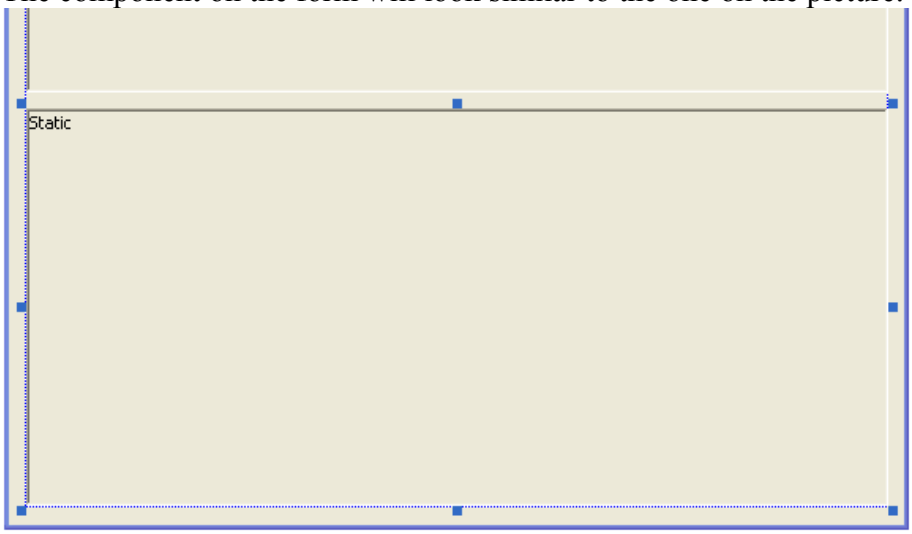
Change the control's ID to "IDC_WATERFALL":



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

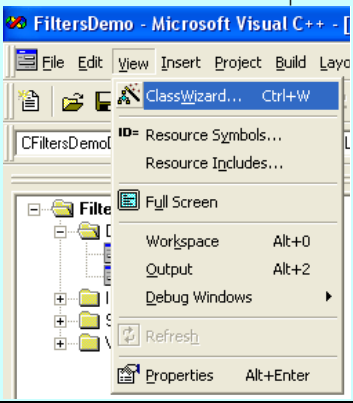


The component on the form will look similar to the one on the picture:

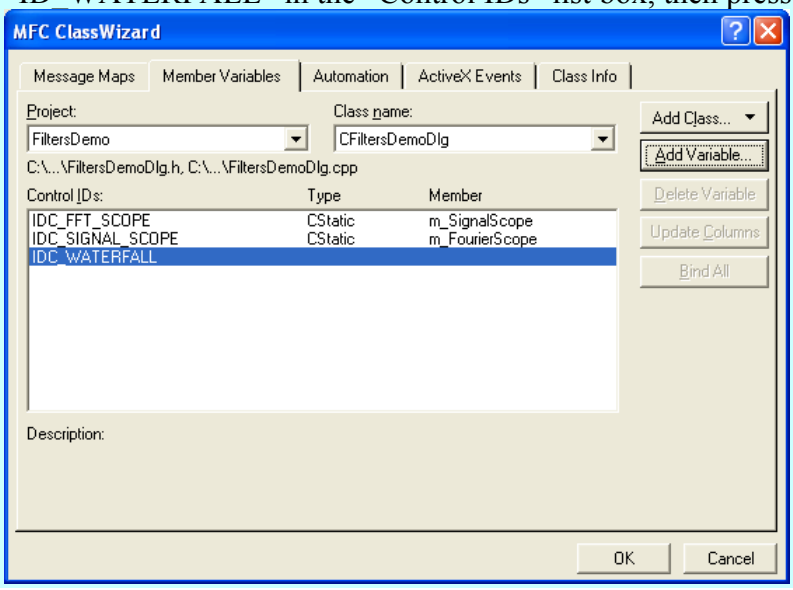


In Visual C++ 6.0:

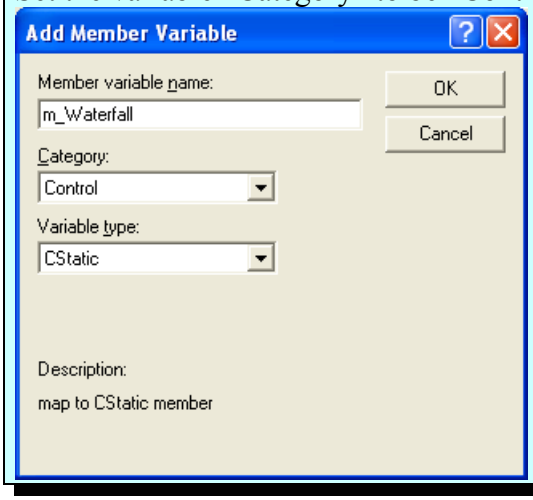
From the menu select | View | ClassWizard… |:



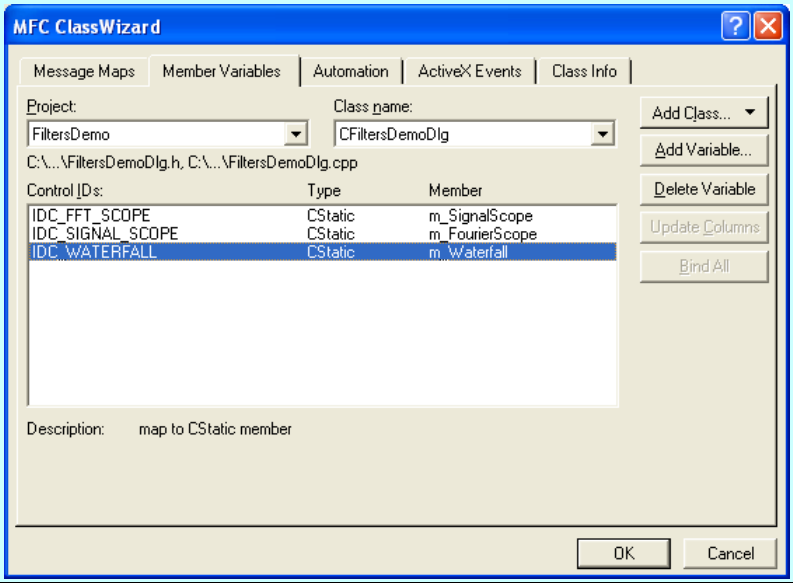
In the “ClassWizard” select the “Member Variables” tab and select the “ID_WATERFALL” in the “Control IDs” list box, then press “Add Variable…”:



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

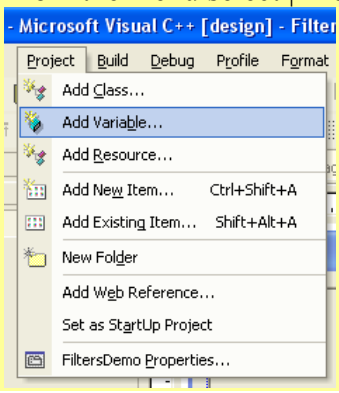


In the “ClassWizard” click OK

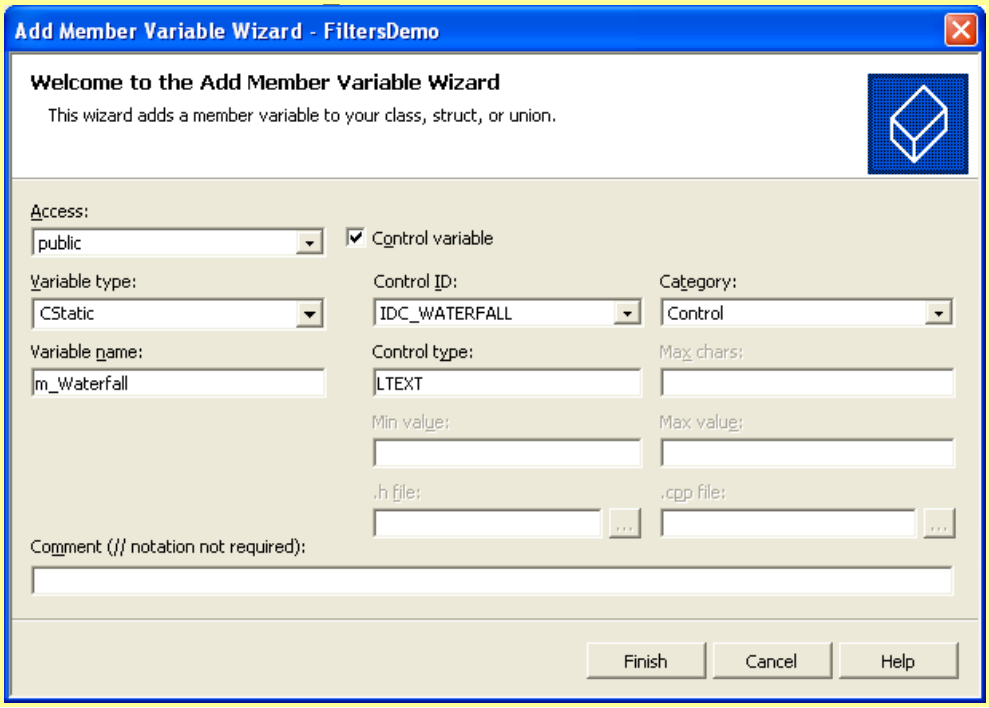


In Visual C++ 2003/2005:

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



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



Click Finish.

Add the highlighted lines in the header:

```
// FiltersDemoDlg.h : header file
//

#pragma once
#include "afxwin.h"

#include <CSLScope.h>
#include <CSLLowPass.h>
#include <CSLRandomGen.h>
#include <CSLFourier.h>
#include <CSLWaterfall.h>
#include <CSLLogger.h>

// CFiltersDemoDlg dialog
class CFiltersDemoDlg : public CDialog
{
```

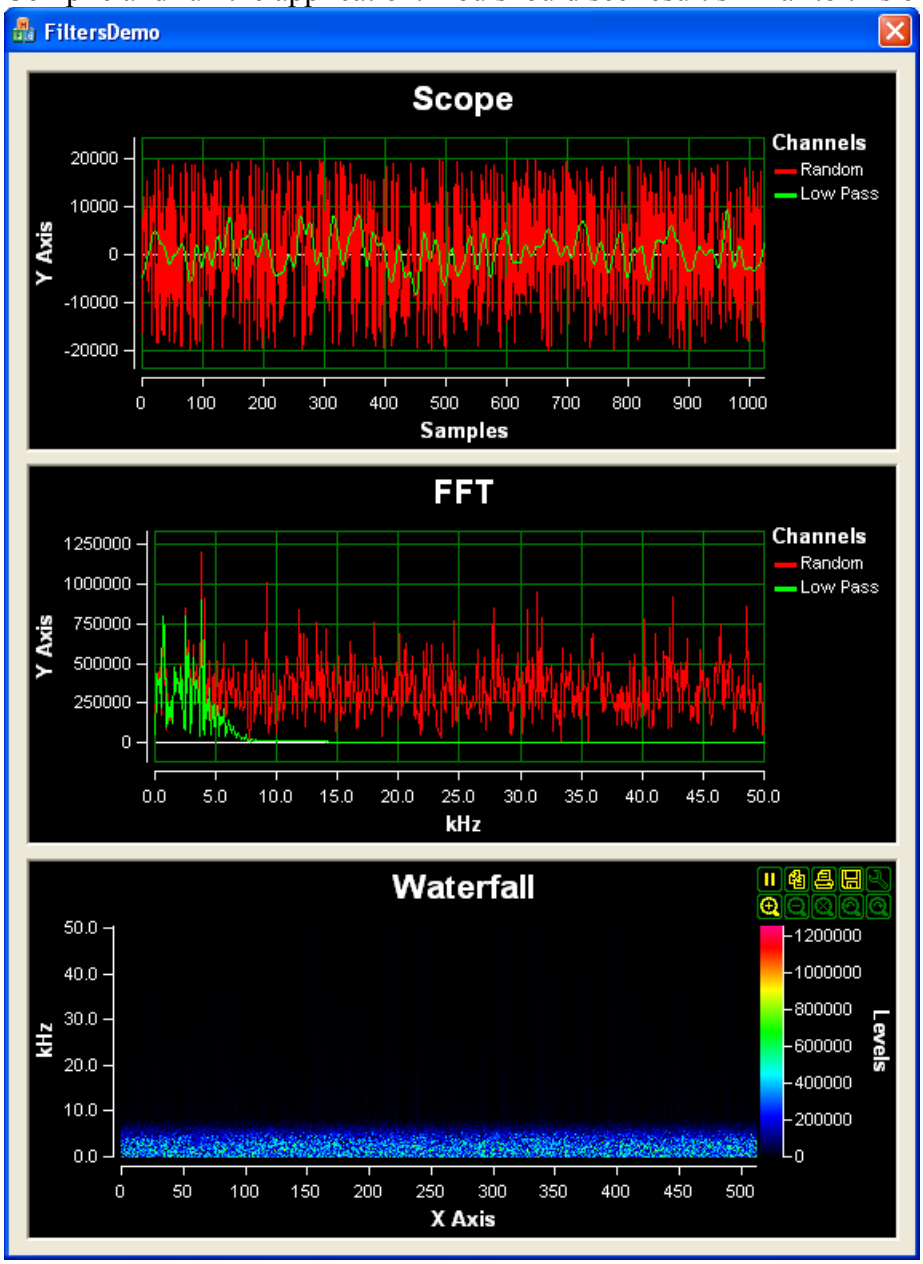
```
// Construction
public:
     CFiltersDemoDlg(CWnd* pParent = NULL);    // standard
constructor


// Dialog Data
     enum { IDD = IDD_FILTERSDEMO_DIALOG };


     protected:
     virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV
support



// Implementation
protected:
     CTSLScope          SLScope;
     CTSLScope          SLScopeFourier;
     CTSLLowPass        SLLowPass;
     CTSLRandomGen      SLRandomGen;
     CTSLFourier        SLFourierRandom;
     CTSLFourier        SLFourierLowPass;
     CTSLWaterfall      SLWaterfall;
     CTSLLogger         SLLogger;


protected:
     HICON m_hIcon;


     // Generated message map functions
     virtual BOOL OnInitDialog();
     afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
     afx_msg void OnPaint();
     afx_msg HCURSOR OnQueryDragIcon();
     DECLARE_MESSAGE_MAP()
public:
     CStatic m_SignalScope;
     CStatic m_FourierScope;
     CStatic m_Waterfall;
};
```

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

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

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// 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 );
	SLScope.Open( m_SignalScope.m_hWnd );
	SLScope.Channels.Clear();
	SLScope.Channels.Add( 2 );
```

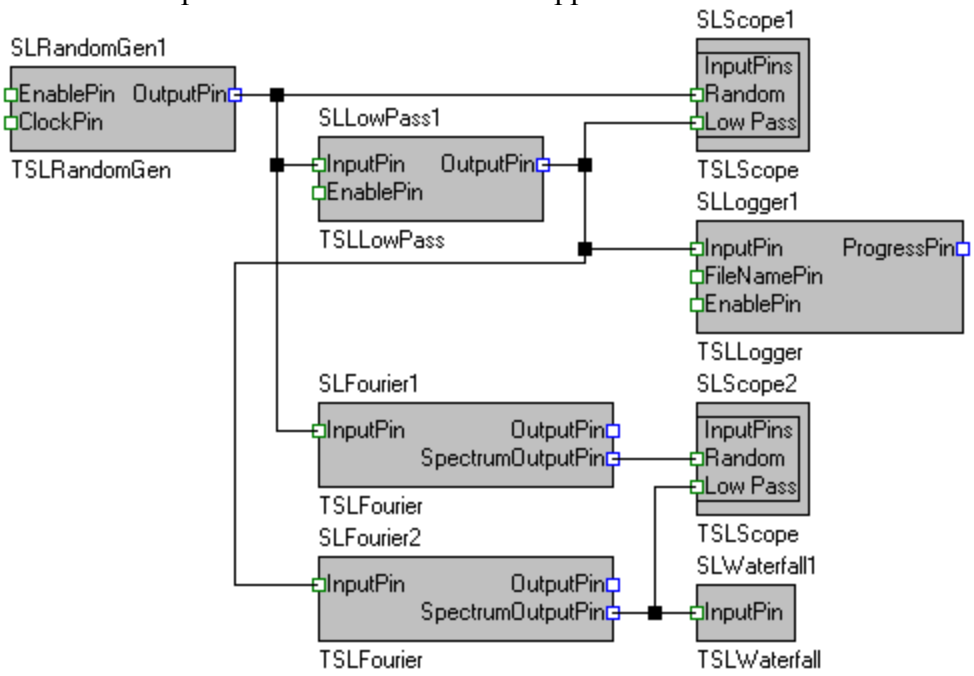
```
      SLScope.Channels[ 0 ].Name = "Random";
      SLScope.Channels[ 1 ].Name = "Low Pass";

      SLScopeFourier.Open( m_FourierScope.m_hWnd );

      SLScopeFourier.Title.Text = "FFT";

      SLScopeFourier.Channels.Clear();
      SLScopeFourier.Channels.Add( 2 );

      SLScopeFourier.Channels[ 0 ].Name = "Random";
      SLScopeFourier.Channels[ 1 ].Name = "Low Pass";

      SLWaterfall.Open( m_Waterfall.m_hWnd );

      SLLogger.FileName = "FilterOutput.bin";

      SLRandomGen.OutputPin.Connect( SLScope.InputPins[ 0 ] );
      SLRandomGen.OutputPin.Connect( SLLowPass.InputPin );
      SLRandomGen.OutputPin.Connect( SLFourierRandom.InputPin );

      SLLowPass.OutputPin.Connect( SLScope.InputPins[ 1 ] );
      SLLowPass.OutputPin.Connect( SLFourierLowPass.InputPin );
      SLLowPass.OutputPin.Connect( SLLogger.InputPin );


SLFourierRandom.SpectrumOutputPin.Connect( SLScopeFourier.InputPins[
0 ] );

SLFourierLowPass.SpectrumOutputPin.Connect( SLScopeFourier.InputPins[
1 ] );
      SLFourierLowPass.SpectrumOutputPin.Connect( SLWaterfall.InputPin
);


      VCL_Loaded();


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

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



Also a file named FilterOutput.bin will be created in the application directory, and it will contain the signal from the LowPass filter.

Here are the OpenWire connections in the application now:



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.