IntelligenceLab 5.0

Visual C++ Quick Start

www.openwire.org
www.mitov.com

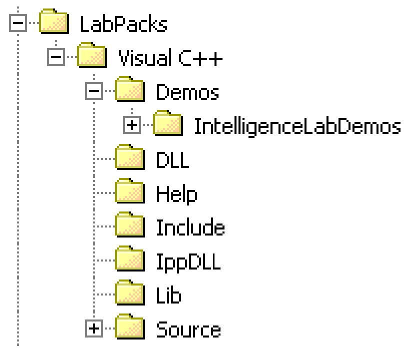
Index

Installation

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

Where is IntelligenceLab

After the installation IntelligenceLab 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:



```
LabPacks
  Visual C++
    Demos
      IntelligenceLabDemos
    DLL
    Help
    Include
    IppDLL
    Lib
    Source
```

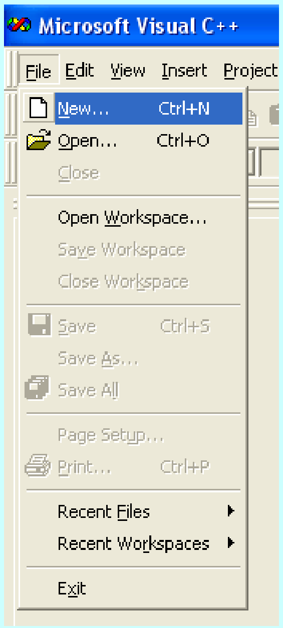
Under the IntelligenceLabDemos directory are located the demo files. The help files and the documentation are located under the Help directory. The DLL directory contains the redistributable DLL files. The header files needed for your projects are located under the Include directory. The Release and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

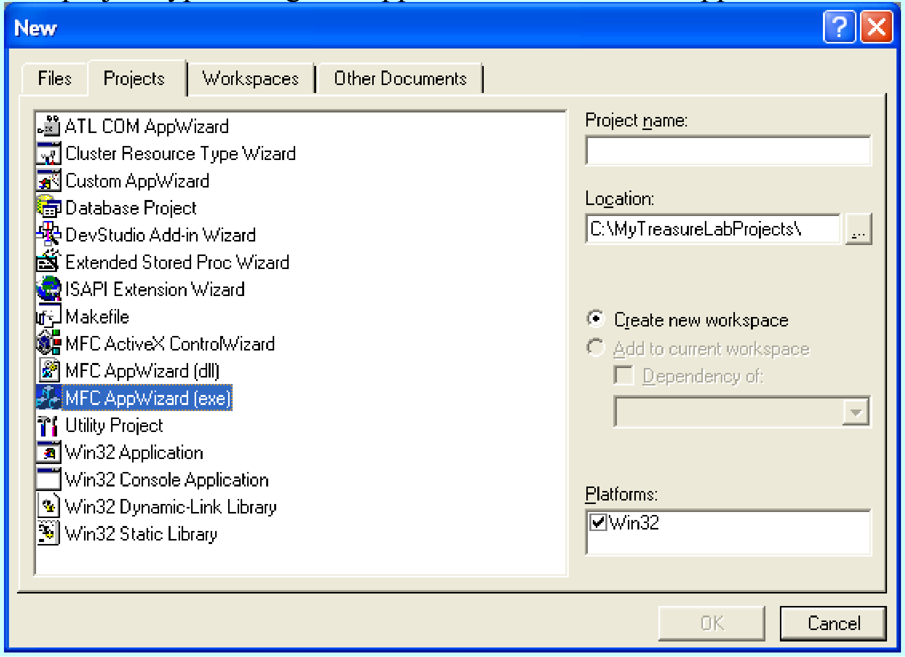
All of the examples in this manual start with creating a MFC Dialog based project. This is not an IntelligenceLab requirement, but using the resource editor to design the application makes writing the examples much easier.
The following chapters will assume that you have created the project and will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

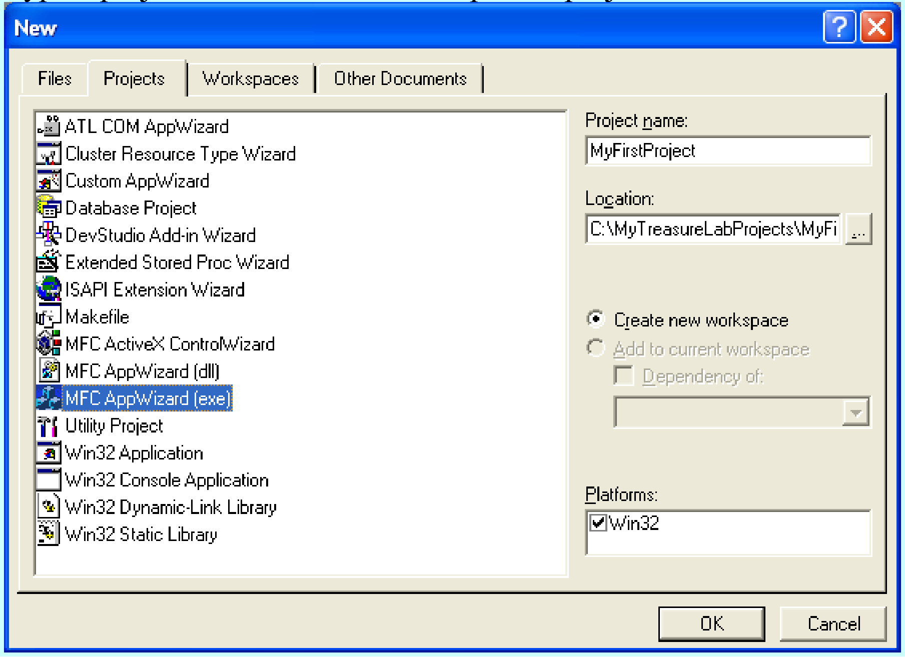


Location: C:\MyTreasureLabProjects\

[✓ Create new workspace]

Platforms: [✓ Win32]

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



Project name: MyFirstProject

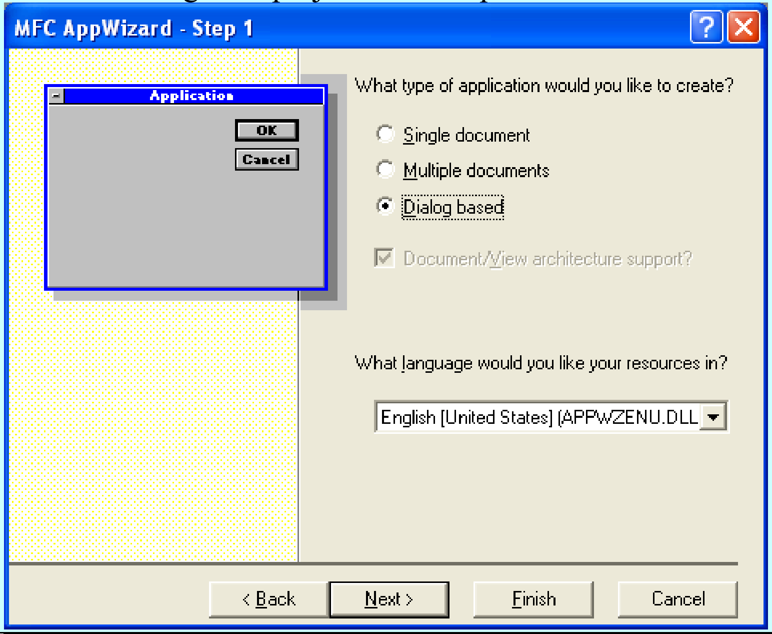
Location: C:\MyTreasureLabProjects\MyFi

(●) Create new workspace

Platforms: [✓] Win32

Click OK.

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



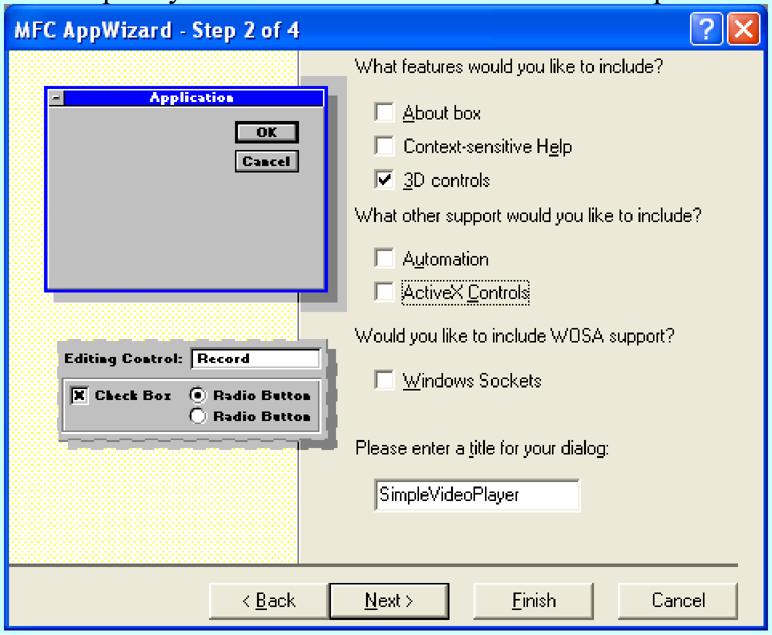
What type of application would you like to create?

(●) Dialog based

What language would you like your resources in?

English [United States] (APPWZENU.DLL

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


MFC AppWizard - Step 2 of 4

What features would you like to include?

- [ ] About box
- [ ] Context-sensitive Help
- [x] 3D controls

What other support would you like to include?

- [ ] Automation
- [ ] ActiveX Controls

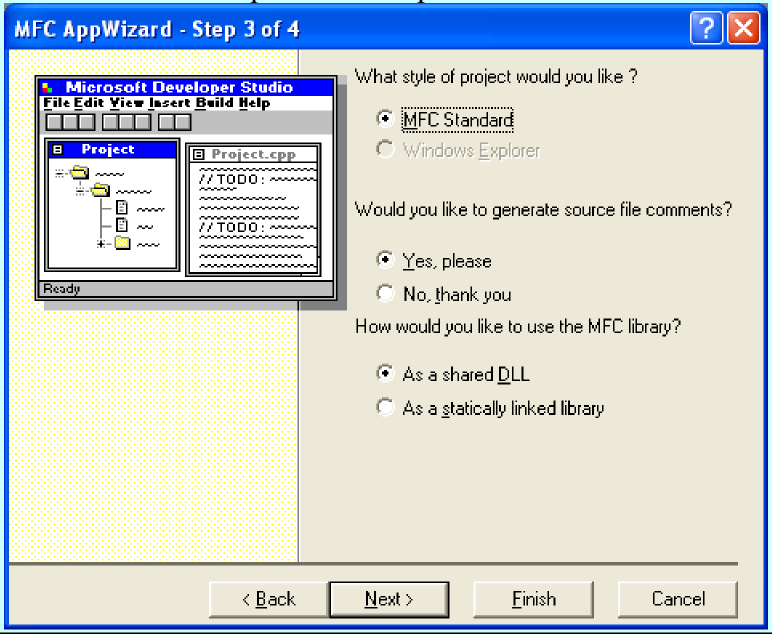
Would you like to include WOSA support?

- [ ] Windows Sockets

Please enter a title for your dialog:

SimpleVideoPlayer

Leave the default options on Step 3 and click Next:


MFC AppWizard - Step 3 of 4

What style of project would you like ?

- ● MFC Standard
- ○ Windows Explorer

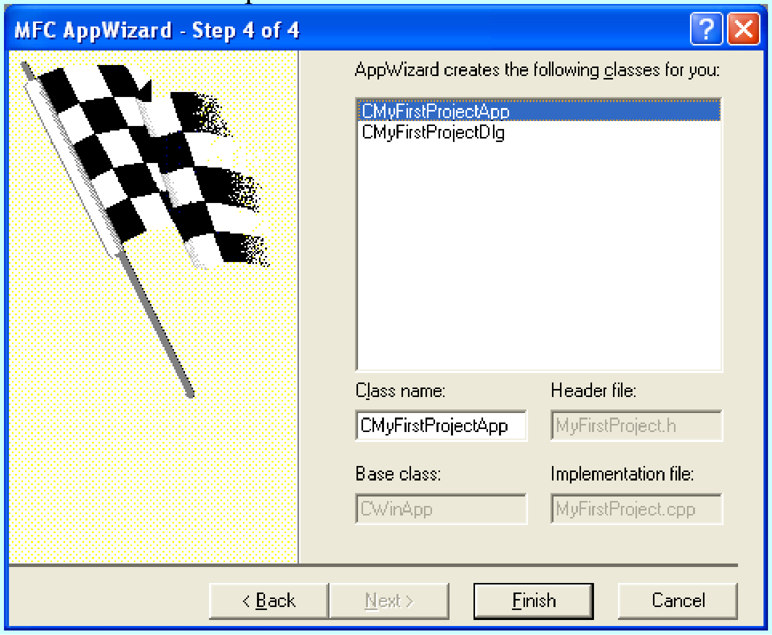
Would you like to generate source file comments?

- ● Yes, please
- ○ No, thank you

How would you like to use the MFC library?

- ● As a shared DLL
- ○ As a statically linked library

Click Finish on step 4:

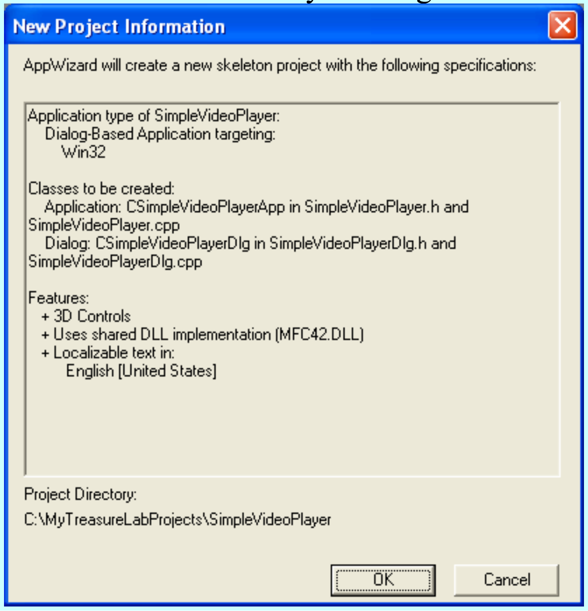


AppWizard creates the following classes for you:

CMyFirstProjectApp
CMyFirstProjectDlg

Class name: CMyFirstProjectApp
Header file: MyFirstProject.h
Base class: CWinApp
Implementation file: MyFirstProject.cpp

Confirm the selection by clicking OK:

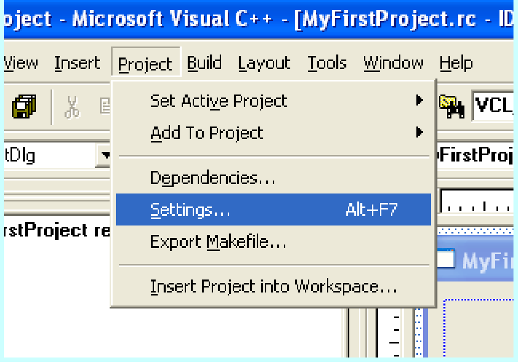


AppWizard will create a new skeleton project with the following specifications:

```
Application type of SimpleVideoPlayer:
   Dialog-Based Application targeting:
      Win32

Classes to be created:
   Application: CSimpleVideoPlayerApp in SimpleVideoPlayer.h and
SimpleVideoPlayer.cpp
   Dialog: CSimpleVideoPlayerDlg in SimpleVideoPlayerDlg.h and
SimpleVideoPlayerDlg.cpp

Features:
  + 3D Controls
  + Uses shared DLL implementation (MFC42.DLL)
  + Localizable text in:
       English [United States]
```

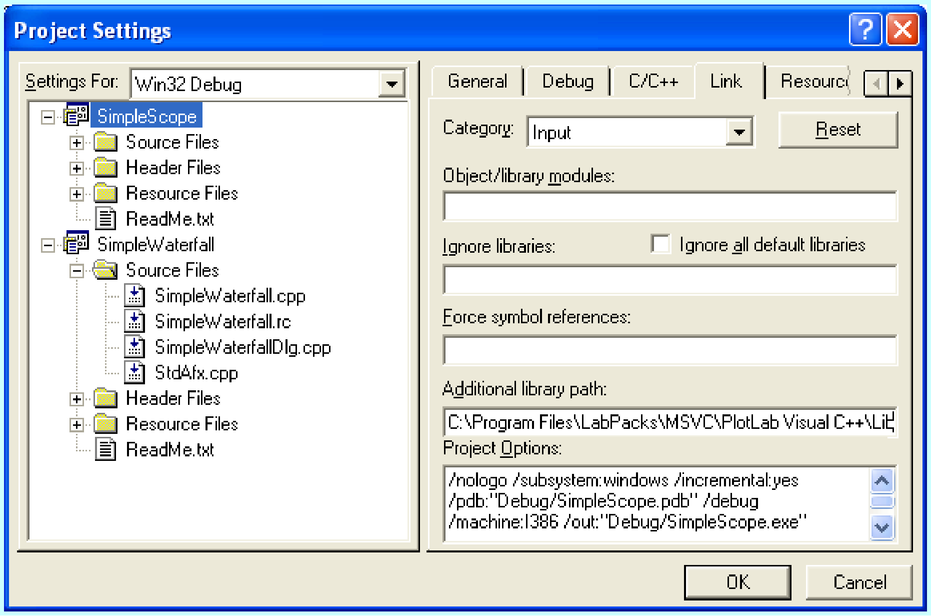
Project Directory:
C:\MyTreasureLabProjects\SimpleVideoPlayer

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



oject - Microsoft Visual C++ - [MyFirstProject.rc - ID
View Insert Project Build Layout Tools Window Help
Set Active Project
Add To Project
Dependencies…
Settings… Alt+F7
Export Makefile…
Insert Project into Workspace…

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:



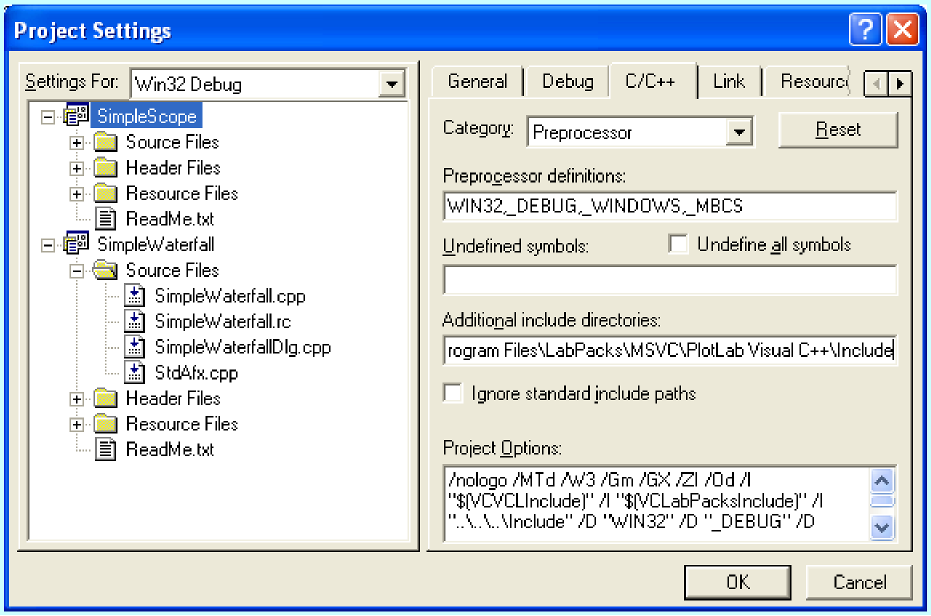
Additional library path: C:\Program Files\LabPacks\MSVC\PlotLab Visual C++\Lib

Project Options:

```
/nologo /subsystem:windows /incremental:yes
/pdb:"Debug/SimpleScope.pdb" /debug
/machine:I386 /out:"Debug/SimpleScope.exe"
```

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:



Category: Preprocessor

Preprocessor definitions: WIN32,_DEBUG,_WINDOWS,_MBCS

Undefined symbols:

Undefine all symbols

Additional include directories: rogram Files\LabPacks\MSVC\PlotLab Visual C++\Include

Ignore standard include paths

Project Options:

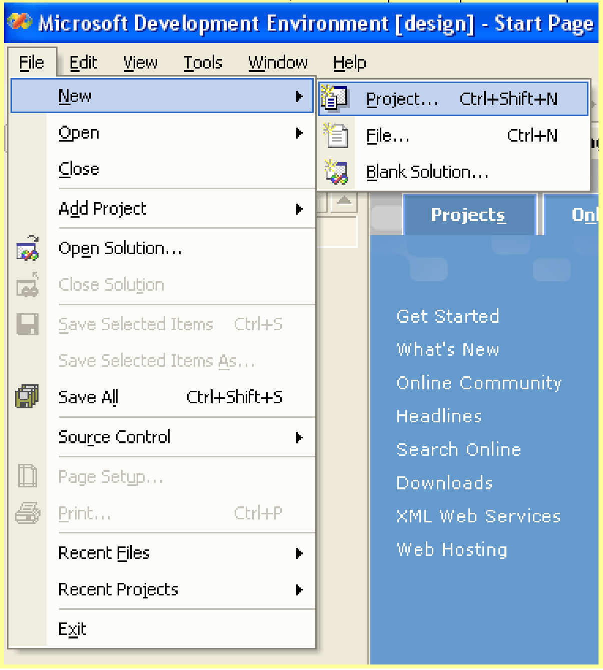
```
/nologo /MTd /W3 /Gm /GX /ZI /Od /I
"$(VCVCLInclude)" /I "$(VCLabPacksInclude)" /I
"..\..\..\Include" /D "WIN32" /D "_DEBUG" /D
```

Click OK.

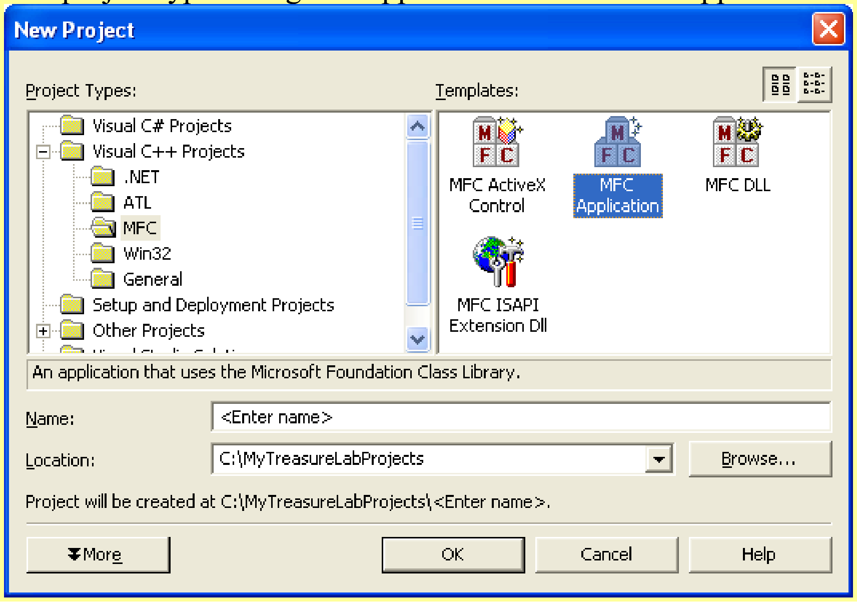
Now you have fully configured project, and you can start writing the actual code.

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:


New Project

Project Types:
- Visual C# Projects
- Visual C++ Projects
  - .NET
  - ATL
  - MFC
  - Win32
  - General
- Setup and Deployment Projects
- Other Projects

Templates: MFC ActiveX Control, MFC Application, MFC DLL, MFC ISAPI Extension Dll

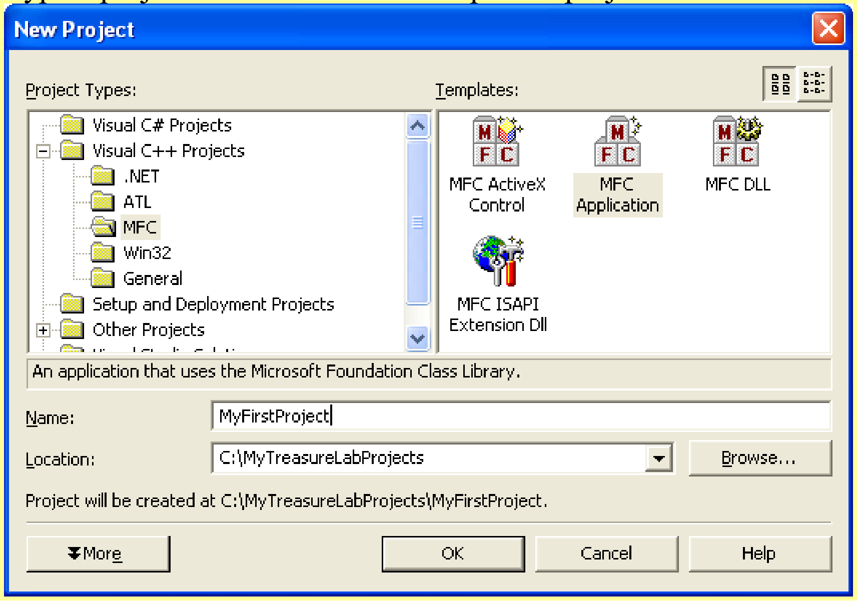
An application that uses the Microsoft Foundation Class Library.

Name: <Enter name>

Location: C:\MyTreasureLabProjects

Project will be created at C:\MyTreasureLabProjects\<Enter name>.

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


New Project

Project Types:
- Visual C# Projects
- Visual C++ Projects
  - .NET
  - ATL
  - MFC
  - Win32
  - General
- Setup and Deployment Projects
- Other Projects

Templates: MFC ActiveX Control, MFC Application, MFC DLL, MFC ISAPI Extension Dll

An application that uses the Microsoft Foundation Class Library.

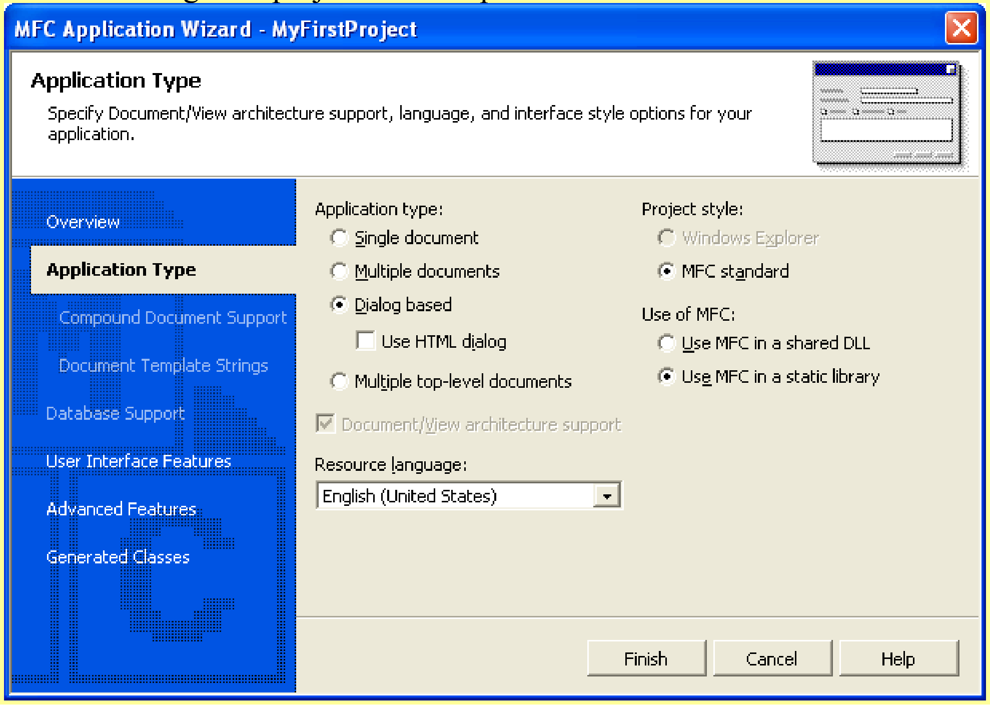
Name: MyFirstProject

Location: C:\MyTreasureLabProjects

Project will be created at C:\MyTreasureLabProjects\MyFirstProject.

Click OK.

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


MFC Application Wizard - MyFirstProject

Application Type

Specify Document/View architecture support, language, and interface style options for your application.

Application type:
- ○ Single document
- ○ Multiple documents
- ● Dialog based
  - ☐ Use HTML dialog
- ○ Multiple top-level documents

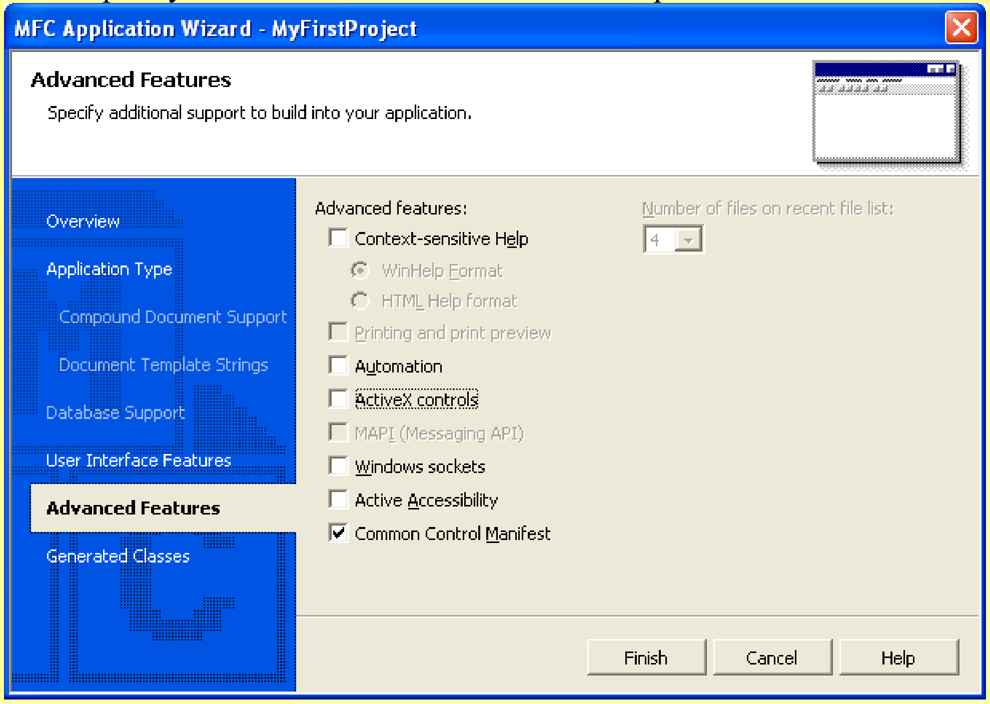
☑ Document/View architecture support

Resource language: English (United States)

Project style:
- ○ Windows Explorer
- ● MFC standard

Use of MFC:
- ○ Use MFC in a shared DLL
- ● Use MFC in a static library

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


MFC Application Wizard - MyFirstProject

Advanced Features

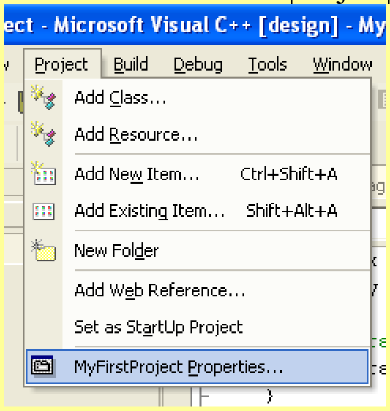
Specify additional support to build into your application.

Advanced features:
- ☐ Context-sensitive Help
  - ● WinHelp Format
  - ○ HTML Help format
- ☐ Printing and print preview
- ☐ Automation
- ☐ ActiveX controls
- ☐ MAPI (Messaging API)
- ☐ Windows sockets
- ☐ Active Accessibility
- ☑ Common Control Manifest

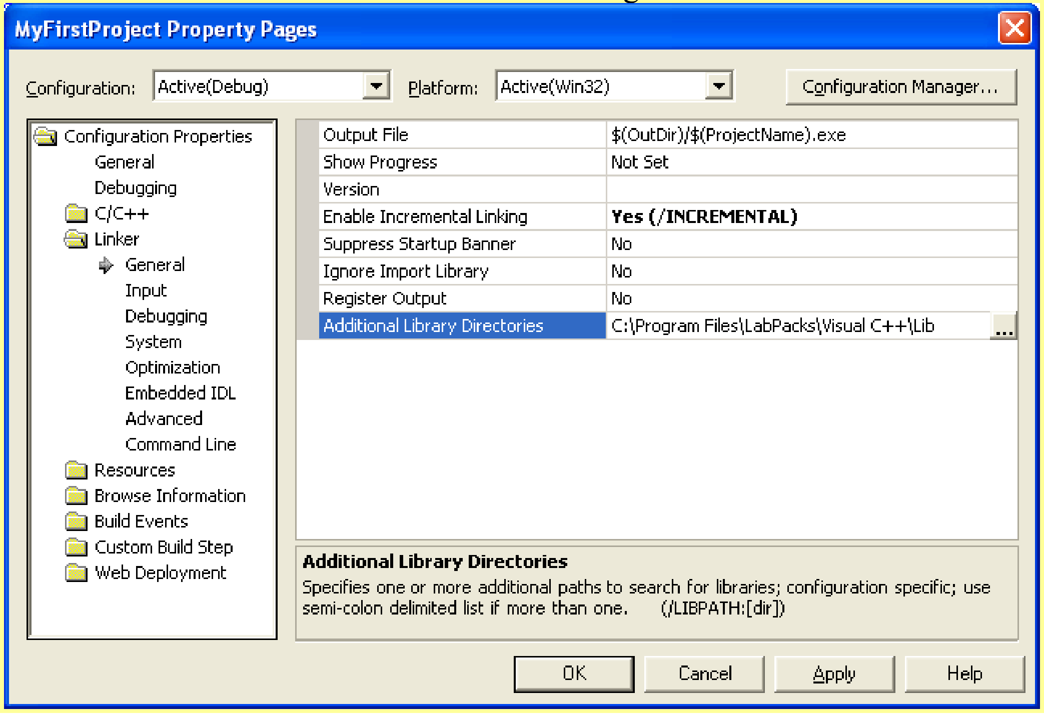
Number of files on recent file list: 4

Click Finish.

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


ect - Microsoft Visual C++ [design] - My
Project Build Debug Tools Window
Add Class...
Add Resource...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
New Folder
Add Web Reference...
Set as StartUp Project
MyFirstProject Properties...

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:

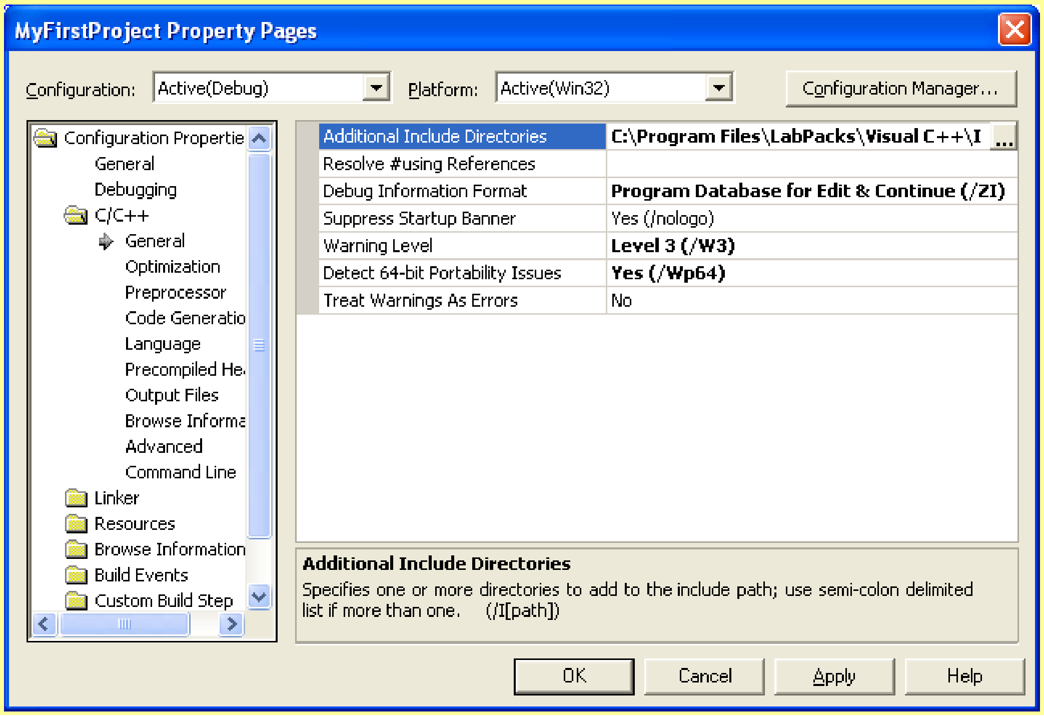


Output File	$(OutDir)/$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	No
Ignore Import Library	No
Register Output	No
Additional Library Directories	C:\Program Files\LabPacks\Visual C++\Lib

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration specific; use semi-colon delimited list if more than one. (/LIBPATH:[dir])

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:


MyFirstProject Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

Additional Include Directories	C:\Program Files\LabPacks\Visual C++\I
Resolve #using References	
Debug Information Format	Program Database for Edit & Continue (/ZI)
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	Yes (/Wp64)
Treat Warnings As Errors	No

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

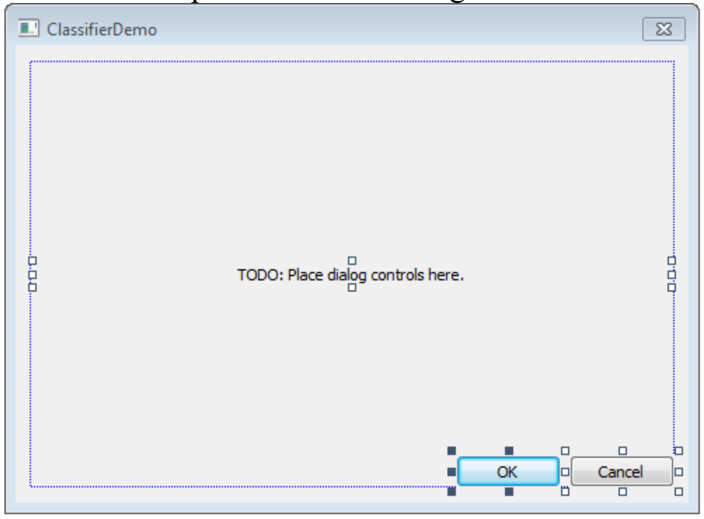
Click OK.

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

Creating classifier application

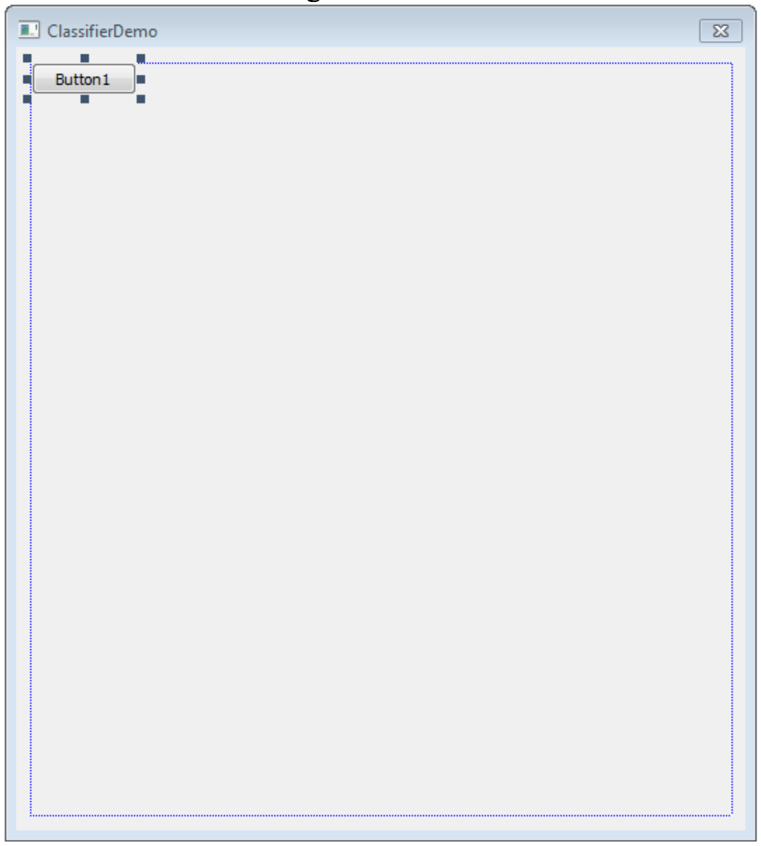
Create and setup a new project named ClassifierDemo as described in the “Creating a new IntelligenceLab project in Visual C++” chapter.

Select the components on the dialog form:



Click the "Del" key. They will be deleted from the form.

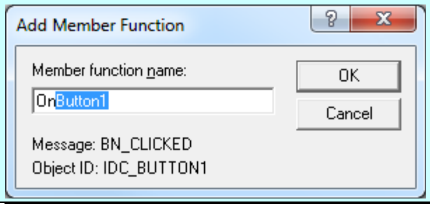
Add button on the dialog form:



Double click on the "Button1" button.

In Visual C++ 6.0:

In the "Add Member Function" dialog click OK:



Member function name: OnButton1

Message: BN_CLICKED

Object ID: IDC_BUTTON1

Add the highlighted lines in the OnBnClickedButton1 event handler:

```
void CClassifierDemoDlg::OnBnClickedButton1()
{
      // TODO: Add your control notification handler code here
      TSLCRealMatrixBuffer ATrainingData( 150, 2 );
      TSLCRealBuffer AResposes( 150 );

      for( i = 0; i < 50; i ++ )
      {
            AResposes[ i ] = 1;
            ATrainingData[ i ][ 0 ] = rand() % 250;
            ATrainingData[ i ][ 1 ] = rand() % 200;
      }

      for( i = 50; i < 100; i ++ )
      {
            AResposes[ i ] = 2;
            ATrainingData[ i ][ 0 ] = 300 + rand() % 200;
            ATrainingData[ i ][ 1 ] = 100 + rand() % 200;
      }

      for( i = 100; i < 150; i ++ )
      {
            AResposes[ i ] = 3;
            ATrainingData[ i ][ 0 ] = rand() % 300;
            ATrainingData[ i ][ 1 ] = 300 + rand() % 200;
      }

      NaiveBayes.Train( ATrainingData, AResposes, false );

      CPaintDC dc(this); // device context for painting
      DrawDC.CreateCompatibleDC( &dc );

      DrawDC.SelectObject( MapImage );

      TSLCRealBuffer ATestData( 2 );

      for( i = 0; i < MAP_WIDTH; i ++ )
            for( j = 0; j < MAP_HEIGHT; j ++ )
            {
                  ATestData[ 1 ] = i;
                  ATestData[ 0 ] = j;
                  NaiveBayes.Predict( ATestData );
            }

      // display the original training samples
      CBrush ARedBrush;
      ARedBrush.CreateSolidBrush( RGB( 255, 0, 0 ));

      CBrush AGreenBrush;
      AGreenBrush.CreateSolidBrush( RGB( 0, 255, 0 ));

      CBrush ABlueBrush;
      ABlueBrush.CreateSolidBrush( RGB( 0, 0, 255 ));

      for( i = 0; i < 150 / 3; i ++ )
      {
            CPoint pt1( (int)( ATrainingData[ i ][ 0 ] + 0.5 ), (int)( 
ATrainingData[ i ][ 1 ] + 0.5 ) );
            DrawDC.FillRect( CRect( pt1.x - 2, pt1.y - 2, pt1.x + 2, 
pt1.y + 2 ), &ARedBrush);

            CPoint pt2( (int)( ATrainingData[ i + 50 ][ 0 ] + 0.5 ), 
(int)( ATrainingData[ i + 50 ][ 1 ] + 0.5 ) );
            DrawDC.FillRect( CRect( pt2.x - 2, pt2.y - 2, pt2.x + 2, 
pt2.y + 2 ), &AGreenBrush);

            CPoint pt3( (int)( ATrainingData[ i + 100 ][ 0 ] + 0.5 ), 
(int)( ATrainingData[ i + 100 ][ 1 ] + 0.5 ) );
            DrawDC.FillRect( CRect( pt3.x - 2, pt3.y - 2, pt3.x + 2, 
pt3.y + 2 ), &ABlueBrush);
      }

      DrawDC.DeleteDC();
     Invalidate();
}
```

Add the highlighted lines in the ClassifierDemoDlg.h header file:

```

// ClassifierDemoDlg.h : header file
//
#pragma once

#include <CILNaiveBayes.h>

// CClassifierDemoDlg dialog
class CClassifierDemoDlg : public CDialogEx
```

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

// Dialog Data
     enum { IDD = IDD_CLASSIFIERDEMO_DIALOG };

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

// Implementation
protected:
     CBitmap                    MapImage;

     CTILNaiveBayes    NaiveBayes;

     CDC                        DrawDC;
     int                        i, j;

protected:
    void __stdcall OnResult(VCLHANDLE ASender, VCLHANDLE AFeatures,
VCLHANDLE AResult);

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:
     afx_msg void OnBnClickedButton1();
};
```

Add the highlighted lines in the CClassifierDemoDlg::OnInitDialog of the ClassifierDemoDlg.cpp source file:

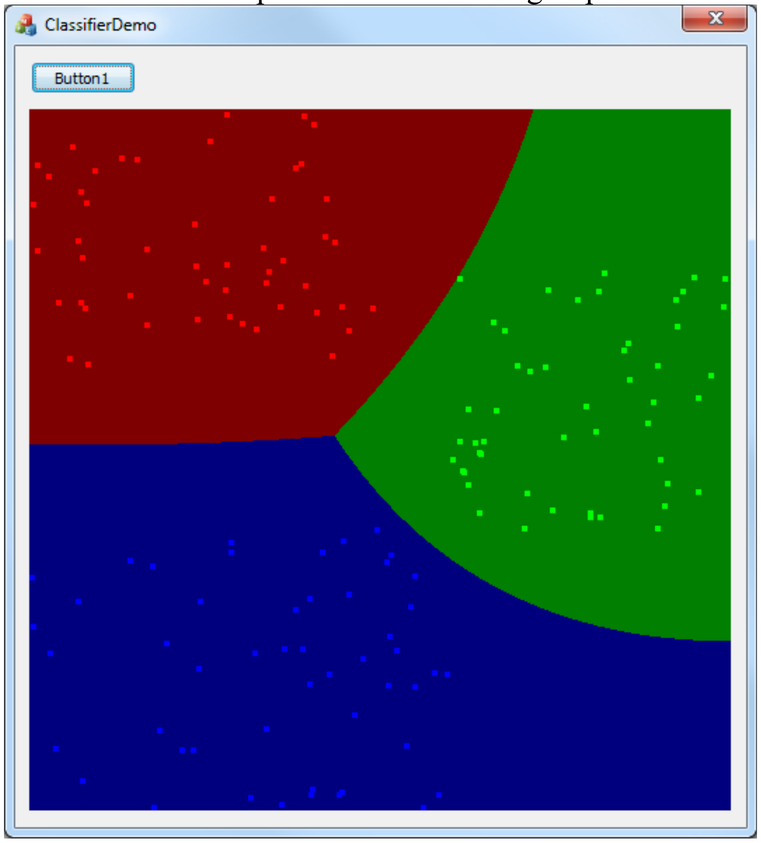
```
BOOL CClassifierDemoDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// 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)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		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
	MapImage.CreateCompatibleBitmap( GetDC(), MAP_WIDTH,
MAP_HEIGHT );
	NaiveBayes.OnResult.Set( this, &CClassifierDemoDlg::OnResult );

	VCL_Loaded();
```

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

Add the highlighted lines in the CClassifierDemoDlg::OnPaint of the ClassifierDemoDlg.cpp source file:

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

        SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<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( MapImage );
        dc.BitBlt( 10, 45, 500, 500, &AnotherDC, 0, 0, SRCCOPY );

        CDialogEx::OnPaint();
    }
}
```

Compile and run the application.
Click on the “Button1” button.
You should see the pixels classified in 3 groups:



You have just learned how to use IntelligenceLab classifier.

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.