PlotLab 7.5

Quick Start

www.openwire.org
www.mitov.com

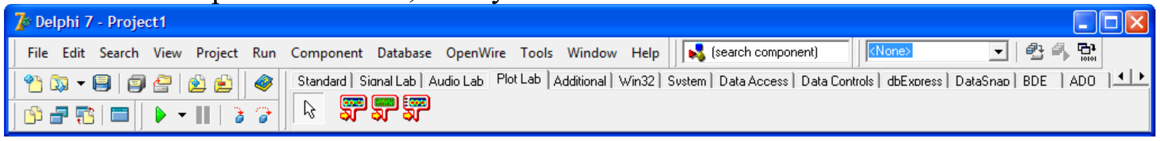
Index

Installation

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

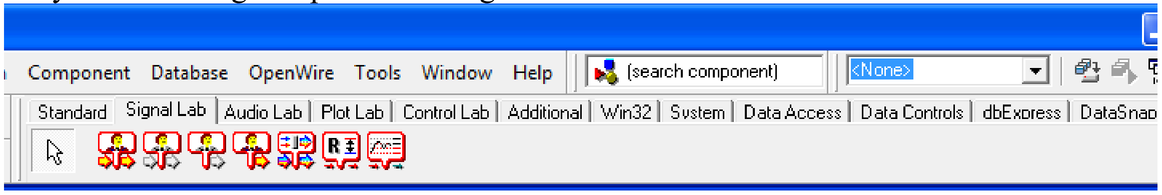
Where is PlotLab?

After the installation, start your Delphi or C++ Builder.
Scroll the "Component Palette", until you see the last two tabs:


Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Plot Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap | BDE | ADO

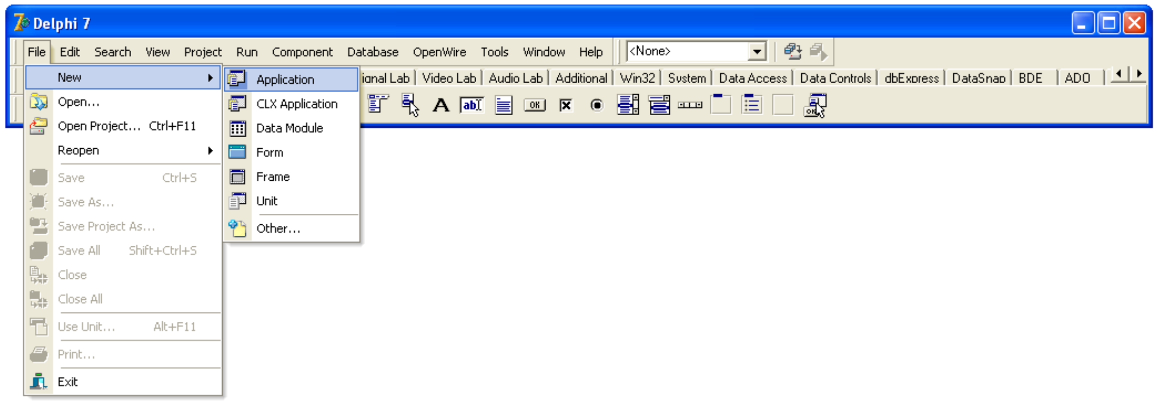
If the installation was successful, it should be named "Signal Lab" and "Plot Lab".

Only the following components of SignalLab will be available:


Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Plot Lab | Control Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap

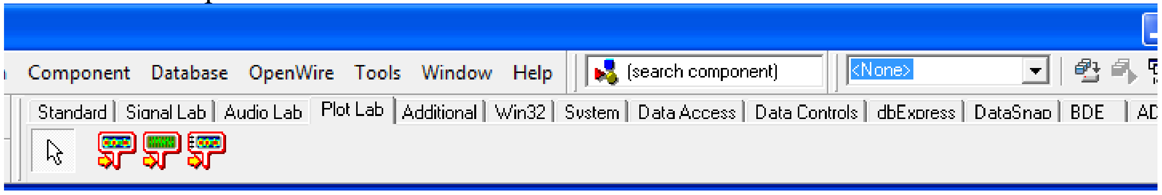
Creating a simple Scope application

From the Delphi/C++Builder menu select | File | New | Application |.



An empty form will appear on the screen.

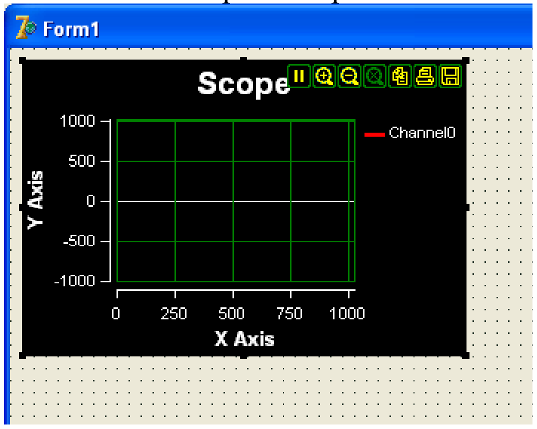
From the "Component Palette" select the "Plot Lab" tab:


Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Plot Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap | BDE | AD

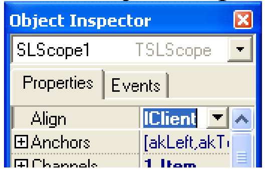
From the tab select and drop on the form the following component:

- TSLScope

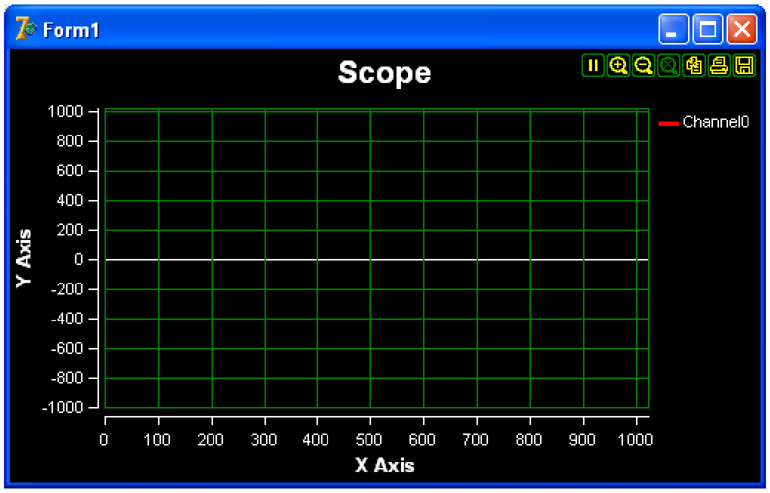
Select the SLScope1 component on the form.



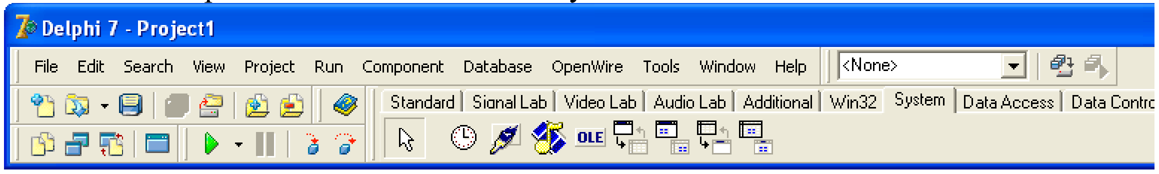
In the Object Inspector set the Align property to alClient:



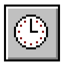
Make the form relatively small.
The form will look similar to this one:

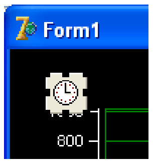


From the “Component Palette” select the “System” tab:

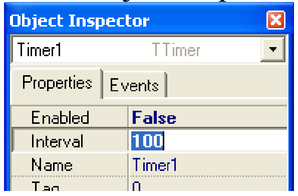


From the tab select and drop on the form a TTimer component.

 - TTimer



In the Object Inspector set the Enabled to False and the interval to 100:


Object Inspector

Timer1 TTimer

Properties | Events

Enabled	False
Interval	100
Name	Timer1
Tag	0

Double click on the timer to generate event handler. Change the code in the form to look like this one. The code works as a simple hardware simulator:

If you are using Delphi add the highlighted code to the Unit1.pas file:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, SLScope, ExtCtrls;

type
  TForm1 = class(TForm)
    SLScope1: TSLScope;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
    Counter : Integer;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
```

```
var
  Buffer : array [ 0..99 ] of Single;
  I      : Integer;

begin
  for I := 0 to 99 do
    begin
    Buffer[ I ] := Counter mod 15 - 7;
    Inc( Counter );
    end;

  SLScope1.Channels[ 0 ].Data.SetYData( PSingle( @Buffer[ 0 ] ),
100 );
end;

end.
```

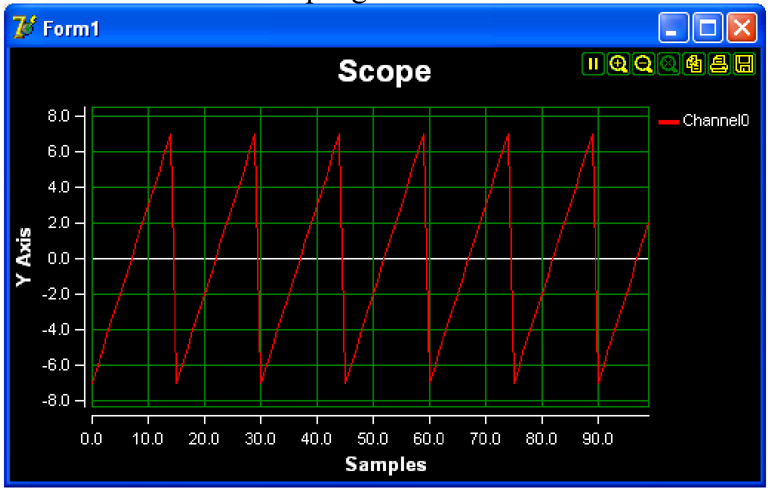
If you are using C++ Builder add the highlighted line to the Unit1.h file:

```
//---------------------------------------------------------------------
--------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------
--------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "SLScope.h"
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------
--------
class TForm1 : public TForm
{
__published:      // IDE-managed Components
        TSLScope *SLScope1;
        TTimer *Timer1;
        void __fastcall Timer1Timer(TObject *Sender);
private:    // User declarations
        int Counter;

public:           // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------
--------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------
--------
#endif
```

If you are using C++ Builder add the highlighted code to the Unit1.cpp file:

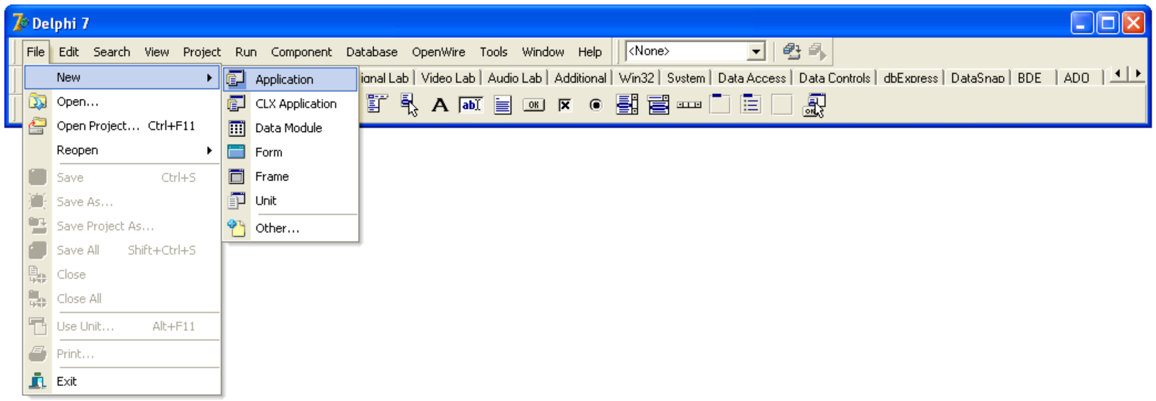
```
//---------------------------------------------------------------------
--------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------
--------
#pragma package(smart_init)
#pragma link "SLScope"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
--------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
--------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  float Buffer[ 100 ];
  for( int i = 0; i < 100; i ++ )
    {
    Buffer[ i ] = Counter % 15 - 7;
    ++Counter;
    }

  SLScope1->Channels->Channels[ 0 ]->Data->SetYData( Buffer, 100 );
}
//---------------------------------------------------------------------
--------
```

Compile and run the application.
You should see the ramp signal:



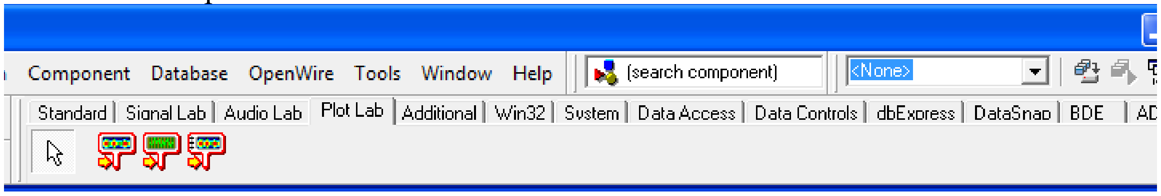
Creating a simple Waterfall application

From the Delphi/C++Builder menu select | File | New | Application |.



An empty form will appear on the screen.

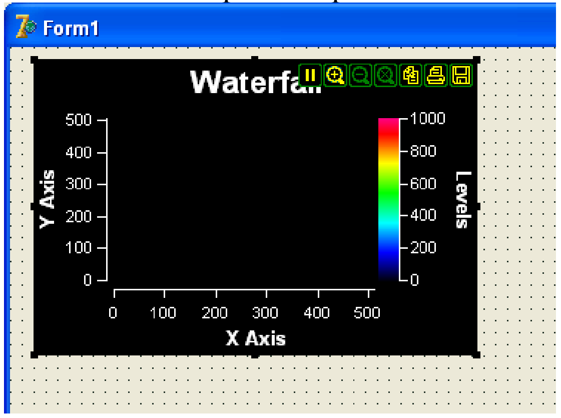
From the “Component Palette” select the “Plot Lab” tab:


Component Database OpenWire Tools Window Help
(search component)
<None>
Standard Signal Lab Audio Lab Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap BDE

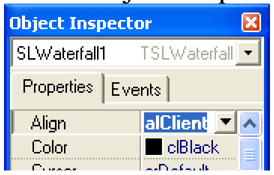
From the tab select and drop on the form the following component:

- TSLWaterfall

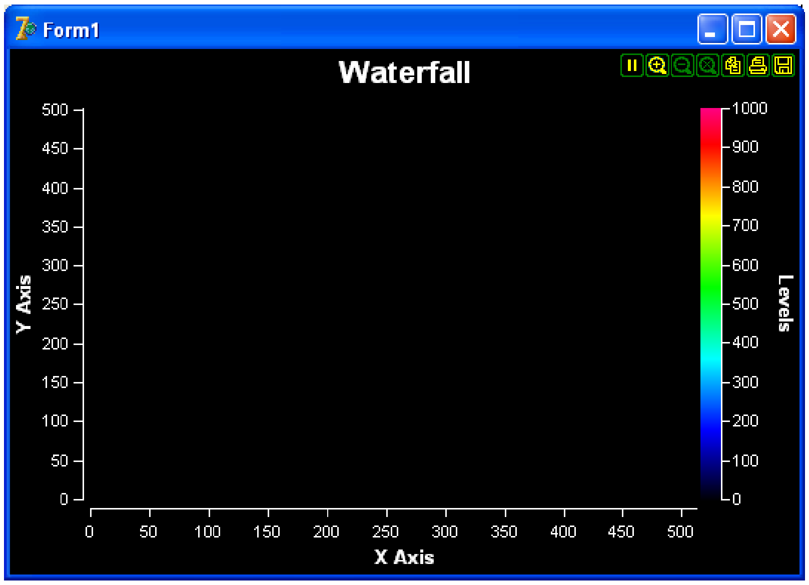
Select the SLScope1 component on the form.



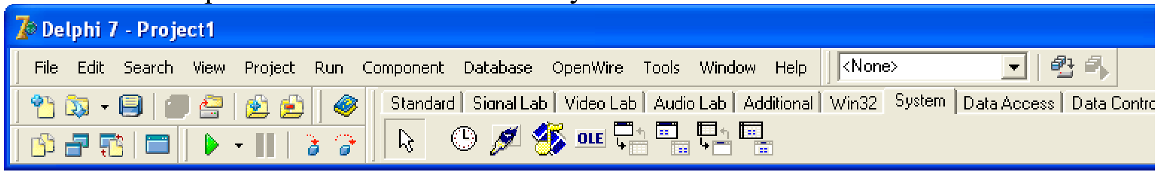
In the Object Inspector set the Align property to alClient:


Object Inspector
SLWaterfall1 TSLWaterfall
Properties | Events
Align alClient
Color clBlack

Make the form relatively small.
The form will look similar to this one:

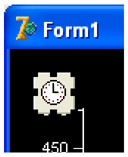


From the “Component Palette” select the “System” tab:

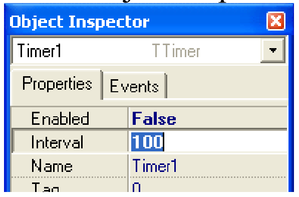


From the tab select and drop on the form a TTimer component.

- TTimer



In the Object Inspector set the Enabled to False and the interval to 100:



Timer1 TTimer

Enabled	False
Interval	100
Name	Timer1
Tag	0

Double click on the timer to generate event handler. Change the code in the form to look like this one. The code works as a simple hardware simulator:

If you are using Delphi add the highlighted code to the Unit1.pas file:

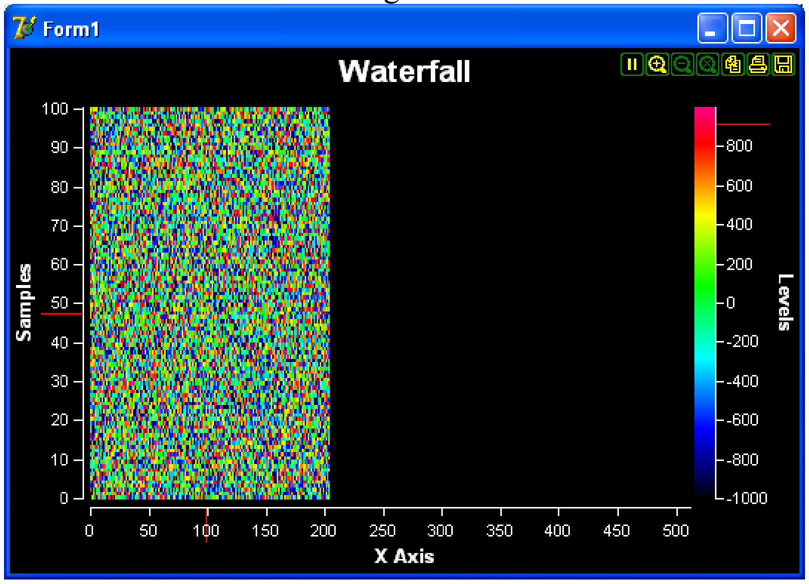
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, SLScope, SLLevelDisplay, SLWaterfall, ExtCtrls;

type
  TForm1 = class(TForm)
    SLWaterfall1: TSLWaterfall;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Math;

{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
var
  Buffer : array [ 0..99 ] of Single;
  I      : Integer;

begin
  for I := 0 to 99 do
    Buffer[ I ] := RandomRange( -1000, 1000 );

  SLWaterfall1.Data.AddData( PSingle( @Buffer[ 0 ] ), 100 );
end;

end.
```

If you are using C++ Builder add the highlighted code to the Unit1.cpp file:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <Math.hpp>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SLScope"
#pragma link "SLWaterfall"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  float Buffer[ 100 ];
  for( int i = 0; i < 100; i ++ )
    Buffer[ i ] = RandomRange( -1000, 1000 );

  SLWaterfall1->Data->AddData( Buffer, 100 );
}
//---------------------------------------------------------------------------
```

Compile and run the application.

You should see the random signal:



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.

Deploying your 32 bit application with the IPP DLLs

The compiled applications can be deployed to the target system by simply copying the executable. 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\Win32 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.

Deploying your 64 bit application

The current version of the library requires when deploying 64 bit applications, the Intel IPP DLLs to be deployed as well.

The DLLs are under the [install path]\LabPacks\IppDLL\Win64 directory([install path] is the location where the library was installed).

To deploy IPP, copy the files to the [Windows]\System32 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.