

Quick Start



www.openwire.org www.mitov.com

Copyright Boian Mitov 2004 - 2014

Index

Installation
Where is SignalLab?
Creating a simple signal generating application
Creating applications using filters, FFT, Waterfall and data logger
Using the TSLPlayer
Creating custom filters
Creating custom data generators
Sending data to third party plot components or using the results of the process into your application37
Manual data pumping
Using SignalLab with data acquisition board
Using the TSLCRealBuffer in C++ Builder and Visual C++
Deploying your 32 bit application with the IPP DLLs
Deploying your 64 bit application

Installation

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

Where is SignalLab?

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

🥻 Delphi 7 - Project1	
File Edit Search View Project Run Component Database OpenWire Tools Window Help 🛛 📢 (search component) 🛛 🕅 🛃 🖓 😭	
🔭 🕼 🕶 🗐 🚰 😰 😰 🥔 Standard Signal Lab Audio Lab Plot Lab Additional Win32 Sustem Data Access Data Controls dbExpress DataSnap BDE ADO	141
: 유유화 28 26 날날 25 26 26 26 28 29 28 26 26 28 29 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	: +

If the installation was successful, they should be named "Signal Lab" and "PlotLab".

The following two PlotLab components will be available.

Component Database OpenWire Tools	Window Help
Standard Signal Lab Audio Lab Video Lab	Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap

Creating a simple signal generating 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 "Signal Lab" tab:



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

- TSLSignalGen

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

Component Database OpenWire Tools	Window Help
Standard Signal Lab Audio Lab Video Lab	Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap
k 20 20 20 20 20 20 20 20 20 20 20 20 20	

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

ም - TSLScope

Select the SLScope1 component on the form.



In the Object Inspector set the Align property to alClient:



Make the form relatively small. Select the SLSignalGen1.



The form will look similar to this one:

In the Object inspector select the OutputPin property and click the 🔤 button.

Object Inspector 🛛 🛛					
SLSignalGen1	TSLSignalGen 💌				
Properties E	vents				
Frequency	500 🔺				
FrequencyPi	(Disconnected)				
Name	SLSignalGen1				
Offset	0				
OutputPin	(Disconnected) 🚥				
Phase	n				

You should see the Pin Editor:

🅻 Connections - Source Pin : SLSignalGen1.Outp 🔳 🗖 🔀							
Form	(7)						
Sink	pin	👎 Comp	Connected to	Conn	OpenWire		
	InputPins	SLScope1			opennine		
0	Frequency	SLSignalGen1			🔁 <u>R</u> estore		
					<mark>}–{</mark> Link to all		
					<mark>≩€ ∐</mark> nlink all		
					🖌 ОК		
					🗙 Cancel		
Link	s:2 🔽	Show all compa	tible pins				

Click on the check box to make it look as in the picture, and then click OK.

Compile and run the application.

You should see the sine wave:



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

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

7	🖻 Delphi 7								
	File Edit Search View Project Run Component Database OpenWire Tools Window Help 🕢 (None> 💌 🔮 🚳								
1		New >	۲J	Application	ianal Lab Video Lab Audio Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap BDE ADO 💶 🕨				
Ť	3	Open	<u>ر</u>	CLX Application	I 🖁 A 💵 📄 🗷 🖉 🖷 🔜 📰 🗋 🚍 📖				
-	<u> </u>	Open Project Ctrl+F11	:::	Data Module					
		Reopen 🕨		Form					
		Save Ctrl+S		Frame					
) () (;	Save As	P	Unit					
		Save Project As	*	Other					
		Save All Shift+Ctrl+S	_						
	₿. ₩₽	Close							
	1948 1948	Close All							
	٦	Use Unit Alt+F11							
	۲	Print							
	ñ.	Exit							

An empty form will appear on the screen.

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



From the tab select and drop on the form the following four components:

One - TSLRandomGen Two - TSLLowPass

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



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





Double click on the SLScope1 component, to open the Channels editor:

🥻 Editing S 🔀					
🎦 🍇 📥 🔸					
0 - ChannelO					

Click on the "Add New" 🛅 button to create a second channel:



By using the Object Inspector rename the channels to "Random" and "Low Pass":

🥻 Editing S 🔀				
🏠 🏡 🔶 🕈				
0 - Random 1 - LowPass				

Select the SLRandomGen1 on the form.



In the Object Inspector select the OutputPin property and click the 🛄 button.

Object Inspe	ctor 🛛 🔀
SLRandomGer	n1 TSLRandomGe💌
Properties E	vents
EnablePin	(Disconnected)
High	20000
Low	-20000
Name	SLRandomGen1
OutputPin	(Disconnected) 🚥
PumpPriority	0 🔳

You should see the Pin Editor:

🅻 Connections - Source Pin : SLRandomGen1.Out 🔳 🗖 🗙								
Form	6							
Sink	pin	🐥 Comp	Connected to	Cor	OpenWire			
	InputPin	SLLowPass1			openwire			
	InputPins.Random	SLScope1			S Restore			
	InputPins.LowPass	SLScope1						
					<mark>}⊸{</mark> Link to all			
					💥 <u>U</u> nlink all			
					🗸 ОК			
<				>	🗶 Cancel			
Links	Links : 3 🦷 Show all compatible pins							

Make the following selections:

🥻 Connections - Source Pin : SLRandomGen1.Out 🔳 🗖 🗙						
Form	Form : Form1 (Current)					
Sink	Sink pin 🦊 Comp Connected to Cor				OpenWire	
Ø	InputPin SLLowPass1				openwire	
	InputPins.Random	SLScope1			S Restore	
	InputPins.LowPass	SLScope1				
					<mark>}⊶{</mark> Link to all	

Click OK.

Select the SLLowPass1 on the form.



In the Object Inspector select the OutputPin property and click the 🛄 button.

Object Inspector 🛛 🔀			
SLLowPass1	TSLLowPass 💌		
Properties E	vents		
EnablePin	(Disconnected)		
Frequency	5000		
InputPin	SLRandomGen1.0.		
Name	SLLowPass1		
NumTaps	41 🔳		
OutputPin	(Disconnected) 🚥		
Teo	0		

In the Pin Editor make the following selection and click OK:

🅻 Connections - Source Pin : SLLowPass1.Output 🔳 🗖 🔀					
Form	6				
Sink	. pin	👎 Comp	Connected to	Cot	OpenWire
	InputPins.Random	SLScope1	SLRandom	Bi-c	openwire
	InputPins.LowPass	SLScope1			🔁 <u>R</u> estore

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:



Now let's show the FFT Spectrum of the two signals.

Stop the running application, make the form bigger and drop the following components on it:



The form should look something like this:



Double click on the SLScope2 and add a second channel as you did with the SLScope1. By using the Object Inspector rename the channels to "Random" and "Low Pass":



Select the SLScope2 on the form. In the Object Inspector click on the ⊞ button to expand the "Title":

Object Inspector		
SLScope2	TSLScope	•
Properties Events		
RefreshInter	100	^
Tag	0	
⊞ Title	(TSLDisplayTitle)	
⊞ ToolBar	(TSLDisplayToolBar	

Change the "Text" sub property to "FFT":

Object Inspector 🛛 🔀				
SLScope2	TSLScope 💌			
Properties Events				
RefreshInter	100 🔼			
Tag	0			
🗆 Title	(TSLDisplayTitle			
⊞ Font	(TFont)			
Text	FFT			
Visible	True 📃			

The SLScope2 should look like this:



Select the SLFourier1 component on the form:



Select the InputPin property and double click on it.

Object Inspector			
SLFourier1	TSLFourier	•	
Properties E	ies Events		
Alpha	1	^	
IgnoreDC	False		
InputPin	(Disconnected)		
Name	SLFourier1	=	
0.4	10		

In the Pin Editor make the following selection:

🕻 с				
Form	(\overline{a})			
Sour	ce pin	🐥 Component	Connections	OpenWire
0	OutputPin	SLFourier2		opennie
0	SpectrumOutput	SLFourier2		🔁 Restore
0	OutputPin	SLLowPass1	SLScope1.I	
\odot	OutputPin	SLRandomGen1	(2 Links)	
				🗸 ОК
<			>	🗙 Cancel
Link	s:4 🔽 Shov	v all compatible pins		

Click OK.

In the Object Inspector select the SpectrumOutputPin property, and click the 🛄 button.

Object Inspector 🛛 🛛			
SLFourier1	TSLFourier 💽		
Properties Events			
Alpha	1 🔥		
IgnoreDC	False		
InputPin	SLRandomGen1.0		
Name	SLFourier1 🗧		
Order	10		
OutputPin	(Disconnected)		
SpectrumOu	(Disconnected) 📟 💳		
Tao	0 🗸		

In the pin editor check the InputPins.Random of the SLScope2 and click OK:

才 с	🅻 Connections - Source Pin : SLFourier1.Spectru 🔳 🗖 🔀				
Form	G				
Sink	pin	🐥 Component	Connected to	OpenWire	
	InputPin	SLFourier2		opennine	
	InputPin	SLLowPass1	SLRandom	🔁 Restore	
	InputPins.Random	SLScope1	SLRandom		
	InputPins.LowPass	SLScope1	SLLowPass	<mark>}⊶{</mark> Link to all	
	InputPins.Random	SLScope2			
	InputPins.LowPass	SLScope2		🍑 Unlink all	

Select the SLFourier2 component on the form:



Select the InputPin property and double click on it.

Object Inspector 🛛 🛛 🔀		
SLFourier2	TSLFourier	-
Properties E	vents	
Alpha	1	^
IgnoreDC	False	
InputPin	(Disconnected) 💌	

In the Pin Editor make the following selection:

7 0 c				
Form	6			
Sour	ce pin	🐥 Component	Connections	OpenWire
0	OutputPin	SLFourier1		openwire
0	SpectrumOutput	SLFourier1	SLScope2.1	🔁 Restore
\odot	OutputPin	SLLowPass1	SLScope1.I	
0	OutputPin	SLRandomGen1	(3 Links)	

Click OK.

In the Object Inspector select the SpectrumOutputPin property, and click the 🛄 button.

Object Inspector 🛛 🛛 🛛		
SLFourier2	TSLFourier	•
Properties E	vents	
Alpha	1	^
IgnoreDC	False	
InputPin	SLLowPass1.Outpu	
Name	SLFourier2	≡
Order	10	
OutputPin	(Disconnected)	
SpectrumOu	(Disconnected) 😐	-
Tag	0	~

In the pin editor check the InputPins.Low Pass of the SLScope2 and click OK:

🅻 Connections - Source Pin : SLFourier2.Spectru 🔳 🗖 🔀				
Form : Form1 (Current)				$\boldsymbol{\omega}$
Sink	; pin	🐥 Component	Connected to	OpenWire
	InputPin	SLFourier1	SLRandom	opennie
	InputPins.Random	SLScope1	SLRandom	S Restore
	InputPins.LowPass	SLScope1	SLLowPass	
	InputPins.Random	SLScope2	SLFourier1	<mark>}⊶{</mark> Link to all
	InputPins.LowPass	SLScope2		
				🏽 🍑 Unlink all



Compile and run the application. You should see a 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 Scope size component, and add the following two component to the form:



The form now should look something like this:



Select the SLFourier2 on the form:



In the Object Inspector select the SpectrumOutputPin property, and click the 🛄 button.

Object Inspector 🛛 🔀						
SLFourier2	TSLFourier 💽					
Properties Events						
Order	10 🔼					
OutputPin	(Disconnected)					
SpectrumOu	(Disconnected) 🚥 💼					

In the pin editor check the InputPin of the SLWaterfall1 and click OK:

🅻 Connections - Source Pin : SLFourier2.Spectru 🔳 🗖 🔀								
Form	()							
Sink	OpenWire							
	InputPin	SLFourier1	SLRandom	openwire				
	InputPins.Random	SLScope1	SLRandom	S Restore				
	InputPins.LowPass	SLScope1	SLLowPass					
	InputPins.Random	SLScope2	SLFourier1	<mark>}⊸{</mark> Link to all				
\square	InputPins.LowPass	SLScope2	SLFourier2					
\square	InputPin	SLWaterfall1		💥 <u>U</u> nlink all				

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





Here are the OpenWire connections in the application now:

Now we will change the LowPass filter's properties and will add a Data Logger to save the filtered data.

Add the following component to the form:





Select the InputPin property and double click on it.

Object Inspector 🛛 🛛 🔀					
SLLogger1	TSLLogger 💽				
Properties E	vents				
Enabled	True				
EnablePin	(Disconnected)				
FileName					
FileNamePin	(Disconnected)				
InputPin	(Disconnected)				
Name	SLLogger1				

In the pins editor select the OutputPin of the SLLowPass1 and click OK:

🕻 с				
Form	$\boldsymbol{\omega}$			
Sour	ce pin	🐥 Component	Connections	OpenWire
0	OutputPin	SLFourier1		openwire
0	SpectrumOutput	SLFourier1	SLScope2.1	S Restore
0	OutputPin	SLFourier2		
0	SpectrumOutput	SLFourier2	(2 Links)	
\odot	OutputPin	SLLowPass1	(2 Links)	
0	OutputPin	SLRandomGen1	(3 Links)	
				0 M

In the Object Inspector set the FileName to FilterOutput.bin:

Object Inspector 🛛 🔀						
SLLogger1	TSLLogger 💽					
Properties Events						
Enabled	True					
EnablePin	(Disconnected)					
FileName	FilterOutput.bin 📃 🛄					
FileNamePin	(Disconnected)					
InnutPin	SLLowPass1 OutputPir					

Select the SLLowPass1 on the form.



In the object inspector set the "Frequency" to 10000 and the "NumTaps" to 15:

Object Inspector 🛛 🔀							
SLLowPass1	TSLLowPass 💌						
Properties Events							
Alpha	1 🔥						
Enabled	True						
EnablePin	(Disconnected)						
Frequency	5000						
InputPin	SLRandomGen1.0						
Name	SLLowPass1						
NumTaps	15						
OutputPin	(3 Links) 🛛 🔽						

This will set the filter frequency to a higher one, and also the smaller number of taps will make the filter less efficient(but with much more "interesting" FFT).



Compile and run the application. You should see a 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:

Now you have all the necessary knowledge to build complex signal processing, visualization and logging applications.

Using the TSLPlayer

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



An empty form will appear on the screen.

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

Co	mponent	Database	OpenWire	Tools 1	Window	Help	<none< th=""><th>></th><th>•</th><th>e 4.</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></none<>	>	•	e 4.							
T	Standard	j Signal Lat	Video Lab	Audio I	_ab Ad	ditional	Win32	System]	Data Access	: Data (Controls	dbExpress	DataSna	BDE	ADO	InterBase	WebServices
	2		<u>2225</u>		귀리 (T 귀리 100 신 (9 epp	52 57		? % %		1	3 😪 🔐			38 S		

select and drop on the form the following component:



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

		Ŀ
1	Component Database OpenWire Tools Window Help	ç
Τ	Standard Signal Lab Audio Lab Video Lab Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSna	5

select and drop on the form the following components:



Arrange the form to look something like this, and select the SLPlayer1 component on the form:





Object Inspector 🛛 🔀						
SLPlayer1	TSLPlayer 💽					
Properties Events						
Enabled	True 🔥					
EnablePin	(Disconnected)					
FileName						
FileNamePin	(Disconnected)					
Mode	nmSton =					

A file selection dialog will appear. Select a file to play. There is a DemoData.bin file available in the AVIFiles subdirectory of the SignalLab demos:

Open					? 🗙
Look jn:	AVIFiles		•	(= 🗈 💣 🖽	•
My Recent Documents Desktop My Documents	DemoData.bin				
S					
My Network Places	File <u>n</u> ame: Files of <u>t</u> ype:	DemoData.bin Binary files (*.bin)		• •	<u>O</u> pen Cancel

Click Open.

In the Object inspector select the OutputPin property and click the 🛄 button.

Object Inspector 🛛 🔀						
SLPlayer1	TSLPlayer 🔽					
Properties E	vents					
EnablePin	(Disconnected)					
FileName	C:\Program Files					
FileNamePin	(Disconnected)					
Mode	pmStop					
Name	SLPlayer1 🔤					
OutputPin	(Disconnected) 🚥					
ProgressPin	(Disconnected)					

In the Pin Editor make the following selection and click OK:

🅻 Connections - Source Pin : SLPlayer1.OutputPin 📃 🗖 🔀							
Form : Form1 (Current)							
Sink pin	🐥 Component	Connected to	OpenWire				
🗹 InputPins.Channel0	SLScope1		opennie				
			S Restore				



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

Here are the OpenWire connections in the application:

		SLScope1
SLPlayer1		Lunge at Dings
eEnablePin eClockPin eFileNamePin	OutputPin <mark>g</mark> ProgressPing	TSLScope
TSLPlayer		

Next we will add an FFT and a Waterfall to analyze the signal.

Add the following two components to the form:



- TSLWaterfall



Arrange the form to look something like this and select the SLFourier1 component:

In the Object inspector double-click on the InputPin property over the word (Disconnected) :

Object Inspector 🛛 🛛 🔀					
SLFourier1	TSLFourier 🔽				
Properties E	vents				
Alpha	1 🔥				
IgnoreDC	False				
InputPin	(Disconnected) 💌				
Name	SLFourier1 🔤				

In the Pin editor make the following selection and click OK:

7 0 c	🅻 Connections - Sink Pin : SLFourier1.InputPin								
Form	: Form1 (Current)		-	(\overline{a})					
Sour	rce pin	🐥 Component	Connections	OpenWire					
\odot	OutputPin	SLPlayer1	SLScope1.I	opennie					
				Bestore					

In the Object inspector select the SpectrumOutputPin property and click the 🛄 button.

Object Inspe	ctor 🛛 🔀
SLFourier1	TSLFourier 🔄
Properties E	vents
IgnoreDC	False 🔥
InputPin	SLPlayer1.OutputPi
Name	SLFourier1
Order	10
OutputPin	(Disconnected)
SpectrumOu	(Disconnected) 🚥
Tag	0

In the Pin editor make the following selection and click OK:

	🥻 Connections - Source Pin : SLFourier1.Spectru 🔳 🗖 🗙						
	Form	Form1 (Current)		-	6		
l	Sink	pin	🐥 Component	Connected to	OpenWire		
		InputPins.Channel0	SLScope1	SLPlayer1	openwire		
	\square	InputPin	SLWaterfall1		Bestore		

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



Here are the OpenWire connections in the application now:



You have learned how to use the TSLPlayer component to play recorded files. Now you can experiment adding filter to the project, and modifying the playback signal. You can also add a TSLLogger and record the signal after being processed. This way you will have a file data processing application.

Creating custom filters

SignalLab includes a fare amount of filters and converters, however often you may need to process the data from inside your code. SignalLab offers a set of generic filters that can be used to implement a custom filter by writing a small amount of C++ or Pascal(Delphi) code.

Here is an example of how you can do this:

2	De	lphi 7				
	File	Edit Search View Project	Ru	n Component Da	atabase OpenWire Tools Window Help 🔣 🔿	
T		New 🕨		Application	ianal Lab Video Lab Audio Lab Additional Win32 Svstem Data Access Data Controls dbExpress DataSnap BDE ADD	<u> </u>
Ť		Open	()	CLX Application	🖺 🖁 A 💵 📄 📧 🗵 🛛 📲 🚍 🚥 🗋 🚍 🔛	
1	<u></u>	Open Project Ctrl+F11		Data Module		
		Reopen 🕨		Form		
		Save Ctrl+S		Frame		
) () (;	Save As	P	Unit		
		Save Project As	*	Other		
		Save All Shift+Ctrl+S	_			
		Close				
	1 1978	Close All				
	Ъ	Use Unit Alt+F11				
	۲	Print				
	i.	Exit				

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

An empty form will appear on the screen.

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

Componer	t Database	OpenWire	Tools	Window H	telp <no< td=""><td>ne></td><td>- 6</td><td>3 🖷</td><td></td><td></td><td></td><td></td><td></td><td></td></no<>	ne>	- 6	3 🖷						
Standa	ard Signal Lat	Video Lab	Audio	Lab Addit	ional Win3/	2 System	Data Access	Data Con	trols dbExc	ress Data	Snap BDB	E ADO	InterBase	WebServices
k		629			R .	<u>, </u>	₽₩ ₩	쭔댨	3555			9 <mark>9 9</mark> 9	.	

From the tab select and drop on the form the following two components:

- TSLSignalGen



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



select and drop on the form the following component:



Select the SLScope1 component on the form.



In the Object Inspector set the Align property to alClient:



Make the form relatively small. Select the SLSignalGen1.



The form will look similar to this one:

In the Object inspector select the OutputPin property and click the **button**.

ctor 🛛 🔀
TSLSignalGen 💌
vents
(Disconnected)
SLSignalGen1
0
(Disconnected) 😐
0

You should see the Pin Editor:

7 0 c	🅻 Connections - Source Pin : SLSignalGen1.Outp 🔳 🗖 🗙					
Form	Form1 (Current)		•	()		
Sink	. pin	🐥 Component	Connected to	OpenWire		
	InputPin	SLGenericReal1		openwire		
	InputPins.Channel0	SLScope1		S Restore		
\diamond	FrequencyPin	SLSignalGen1				
				<mark>}{</mark> Link to all		

Click on the check box to make it look as in the picture, and then click OK.

Select the SLGenericReal1 on the form:



In the Object inspector select the OutputPin property and click the 🛄 button.

Object Inspector 🛛 🔀					
SLGenericRea	1 TSLGenericRe 🔽				
Properties Events					
Enabled	True				
EnablePin	(Disconnected)				
InputPin	SLSignalGen1.OutputP				
Name	SLGenericReal1				
OutputPin	(Disconnected) 📃 🛄				
Synchronize	stNone				
Т	•				

In the Pin Editor check the following pin and click OK:

] ⊘ c	🅻 Connections - Source Pin : SLGenericReal1.0u 🔳 🗖 🔀					
Form	: Form1 (Current)		•	6		
Sink	; pin	🐥 Component	Connected to	OpenWire		
🗹 InputPins.Channel0		SLScope1		openwire		
FrequencyPin		SLSignalGen1		Bestore		
				🍋 Link to all		

In the "Object Inspector" select the "Events" tab and double click on the OnProcessData event:

Object Inspector 🛛 🛛 🔀						
SLGenericReal1 TSLGenericRe						
Properties	Even	ts				
OnFilterDa	ata					
OnProces	sData	•				
OnStart						
OnStop						

In the event handler add the following code:

If you are using Delphi:

```
procedure TForm1.SLGenericReal1ProcessData(Sender: TObject;
  InBuffer: ISLRealBuffer; var OutBuffer: ISLRealBuffer;
  var SendOutputData: Boolean);
var
  InputRawDataPointer : PReal;
  OutputRawDataPointer : PReal;
  Т
         : Integer;
begin
  InputRawDataPointer := InBuffer.Read();
 OutputRawDataPointer := OutBuffer.Write();
  for I := 0 to OutBuffer.GetSize() - 1 do
   begin
    if ( InputRawDataPointer^ > 5000 ) then
    OutputRawDataPointer^ := InputRawDataPointer^ / 10 + 5000
    else if( InputRawDataPointer^ < -5000 ) then</pre>
     OutputRawDataPointer^ := InputRawDataPointer^ / 10 - 5000
    else
```

```
OutputRawDataPointer^ := InputRawDataPointer^;
Inc( InputRawDataPointer );
Inc( OutputRawDataPointer );
end;
end;
```

If you are using C++ Builder:

```
fastcall TForm1::SLGenericReal1ProcessData(TObject *Sender,
void
     bool &SendOutputData)
{
 const double *InputRawDataPointer = InBuffer.Read();
 double *OutputRawDataPointer = OutBuffer.Write();
 for( int i = 0; i < OutBuffer.GetSize(); i ++ )</pre>
   if( *InputRawDataPointer > 5000 )
    *OutputRawDataPointer = *InputRawDataPointer / 10 + 5000;
   else if( *InputRawDataPointer < -5000 )</pre>
    *OutputRawDataPointer = *InputRawDataPointer / 10 - 5000;
   else
    *OutputRawDataPointer = *InputRawDataPointer;
   InputRawDataPointer ++;
   OutputRawDataPointer ++;
   }
```

Compile and run the application. The result should be similar to this one:



Here are the OpenWire connections in this application:



Congratulations! You have just created your first SignalLab custom filter.

Creating custom data generators

SignalLab comes with a set of signal generators such as TSLSignalGen and TSLRandomGen, but very often you will need to generate your own data. Here is an example of how you can do that:

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



An empty form will appear on the screen.

From the "Component Palette" select the "Standard" tab:

Proje	ct1													
earch	View	Project	Run	Component	Database	OpenWire	Tools	Window	Help	<none< th=""><th>></th><th>•</th><th>h 🖡</th><th></th></none<>	>	•	h 🖡	
] 🍯) 🚑	🖄 🖄	0	Standard	Signal Lab) Video Lat) Audio	Lab Ad	ditional	Win32	Svstem	Data Access	Data Controls	dt
		•	3 🕝	- R -		🖁 A 🗖	I 📄	OK 🗙	۲				R	

From the tab select and drop on the form three TButton components:

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

Component Database OpenWire Tools Window Help	Kone>
Standard Signal Lab Video Lab Audio Lab Additional W	Win32 Sustem Data Access Data Controls dbExpress DataSnap BDE ADO InterBase WebService

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



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

select and drop on the form the following component:



Change the button names to be StartButton, DataButton and StopButton. Rename the button captions to be Start, Data, and Stop. Arrange the components to look like the picture bolow.

Select the SLGenericReal1:



In the Object inspector select the OutputPin property and click the **button**.

Object Inspector 🛛 🛛 🔀					
SLGenericReal1	TSLGenericRe				
Properties Events					
Enabled	True				
EnablePin	(Disconnected)				
InputPin	(Disconnected)				
Name	SLGenericReal1				
OutputPin	(Disconnected)				
SynchronizeTypestNone					
Tag	0				

In the Pin Editor check the following pin and click OK:

🅻 Connections - Source Pin : SLGenericReal1.0u 🔳 🗖 🔀							
Form	(7)						
Sink	pin	🐥 Component	Connected to	OpenWire			
🗹 InputPins.Channel0		SLScope1		opennite			
				Bestore			

Double click on the StartButton.

In the event handler write the following code:

If you are using Delphi:

```
procedure TForm1.StartButtonClick(Sender: TObject);
begin
    SLGenericReal1.SendStartCommand( 1000 );
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::StartButtonClick(TObject *Sender)
{
   SLGenericReal1->SendStartCommand( 1000 );
}
```

With this code we are sending a start command from the SLGenericReal1 to any component connected to its OutputPin. Here for the purpose of the example we are stating that the expected data rate is 1 KHz. The data rate is of no importance for this application, but if there are any filters of FFT components involved in the process they will need the data rate to adjust it self.

Double click on the DataButton. In the event handler write the following code:

If you are using Delphi:

```
procedure TForml.DataButtonClick(Sender: TObject);
var
DataBuffer : ISLRealBuffer;
RawDataPtr : PReal;
I : Integer;
begin
// Create a buffer to hold the data.
DataBuffer := TSLRealBuffer.CreateSize( 1024 );
// Obtain pointer to the locked data as a ^Real
RawDataPtr := DataBuffer.Write();
// Fill the buffer with simulated data.
for I := 0 to 1024 - 1 do
begin
RawDataPtr^ := I mod 300;
```

```
Inc( RawDataPtr );
end;
SLGenericReal1.SendData( DataBuffer );
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::DataButtonClick(TObject *Sender)
{
    // Create a buffer to hold the data.
    TSLCRealBuffer DataBuffer( 1024 );
    // Obtain pointer to the data as a ^Real
    double *RawDataPtr = DataBuffer.Write();
    // Fill the buffer with simulated data.
    for( int i = 0; i < 1024; i ++ )
        *RawDataPtr ++ = i % 300;
    SLGenericReal1->SendData( DataBuffer );
}
```

Here we are generating a buffer and sending it via the SLGenericReal1. In this application each buffer will look the same for simplicity. In a real application however each buffer will most likely contain different data. The data can be generated on the fly or it can be copied from a memory location.

Double click on the StopButton. In the event handler write the following code:

If you are using Delphi:

```
procedure TForm1.StopButtonClick(Sender: TObject);
begin
    SLGenericReal1.SendStopCommand();
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::StopButtonClick(TObject *Sender)
{
   SLGenericReal1->SendStopCommand();
}
```

Here we are just sending a Stop command to indicate the end of the data feeding process.

Here is how your full Delphi source code should look like:

```
unit Unit1;
```

interface

```
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
 Dialogs, SLScope, SLCommonFilter, SLGenericReal, StdCtrls;
type
 TForm1 = class(TForm)
    StartButton: TButton;
   DataButton: TButton;
   StopButton: TButton;
   SLGenericReal1: TSLGenericReal;
   SLScope1: TSLScope;
   procedure StartButtonClick(Sender: TObject);
   procedure DataButtonClick(Sender: TObject);
   procedure StopButtonClick(Sender: TObject);
  private
   { Private declarations }
  public
   { Public declarations }
 end;
var
 Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.StartButtonClick(Sender: TObject);
begin
 SLGenericReal1.SendStartCommand( 1000 );
end;
procedure TForm1.DataButtonClick(Sender: TObject);
var
 DataBuffer : ISLRealBuffer;
 RawDataPtr : PReal;
 Ι
             : Integer;
begin
 // Create a buffer to hold the data.
  DataBuffer := TSLRealBuffer.CreateSize( 1024 );
 // Obtain pointer to the locked data as a ^Real
 RawDataPtr := DataBuffer.Write();
  // Fill the buffer with simulated data.
  for I := 0 to 1024 - 1 do
   begin
   RawDataPtr^ := I mod 300;
   Inc( RawDataPtr );
   end;
  SLGenericReal1.SendData( DataBuffer );
end;
```

```
procedure TForm1.StopButtonClick(Sender: TObject);
begin
   SLGenericReal1.SendStopCommand();
end;
end.
```

Compile and run the application.

Press the "Start" button. This will send a start command for the data processing, and will initialize the Scope or any other components connected via OpenWire such as filters or format converters.

Press few times the "Data" button to send few buffers of data. All of the buffers will look the same so don't expect to see any differences in the Scope.

Press the "Stop" button to send a stop command to the Scope indicating the end of the data transfer.



The result should be similar to this one:

Here are the OpenWire connections in this application:



Congratulations! You have just learned how to feed custom data into a SignalLab application.

Sending data to third party plot components or using the results of the process into your application

SignalLab comes with a Scope and Waterfall components, but there are cases when you will need to plot the data on a more sophisticated and advanced third party component, or simply to feed the result data of the processing into your application.

In this case TSLGenericReal comes again into play.

Here we will create a simple application sending feeding the data into a TChart component.

The TChart component comes for free with the Professional and higher versions of Delphi and C++ Builder. If you have a Standard version of Delphi or C++ Builder you can use another component or use the data in the application as it fits your needs. The code will look almost the same.

You can download a trial version of the TChart (TeeChart) component from <u>http://www.steema.com/</u>.



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

An empty form will appear on the screen.

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

C	omponent	Database	OpenWire	Tools Wind	low Help	<none></none>	,	-	2 🐴							
	Standar	d Signal Lab	Video Lab	Audio Lab	Additional	Win32	Svstem	Data Access	Data Co	ontrols	dbExpress	DataSnap	BDE	ADO	InterBase	WebServices
-	L3	GEN IIR	222 5			S 7		<u>᠊</u> ,	3505	55				7	1 R R	

From the tab select and drop on the form the following two components:



From the "Component Palette" select the "Additional" tab:

Component Database OpenWire Tools Window Help	KNone>
Standard Signal Lab Video Lab Audio Lab Additional	Win32 System Data Access Data Controls dbExpress DataSnap BDE
	🗷 + a 🗈 🛠 🏾 🕷 🖉 s 🦻 🏧 🖏 🖷

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



The form now will look something like this:

🅻 Form1		
	TChart	
•		

In the Object Inspector set the Chart1 Align property to alClient:



Select the SLSignalGen1 on the form:

🅻 Form1		
	TChart	

In the Object inspector set the Frequency to 50:

Object Inspector 🛛 🔀					
SLSignalGen1	TSLSignalGen 💌				
Properties Events					
EnablePin	(Disconnected)				
Frequency	50				
FrequencyPin	n (Disconnected)				
Name	SLSionalGen1 💦 👘				

In the Object inspector set the SampleRate to 10000:

Object Inspector 🛛 🛛 🔀					
SLSignalGen1	TSLSignalGen 💌				
Properties Events					
Name	SLSignalGen1 📃 🔥				
Offset	0				
OutputPin	(Disconnected)				
Phase	0				
PumpPriority	0				
SampleRate	10000				
SignalType	stTone				

In the Object inspector select the OutputPin property and click the 🛄 button.

Object Inspector 🛛 🛛 🔀						
SLSignalGen1 TSLSignalGen 💌						
Properties Events						
Name	SLSignalGen1 📃 🔥					
Offset	0					
OutputPin	Disconnected)					
Phase	0					

In the Pin Editor check the following pin and click OK.

7 0 c	🏅 Connections - Source Pin : SLSignalGen1.Outp 📃 🗖 🔀								
Form	6								
Sink	pin	🐥 Component	Connected to	OpenWire					
	InputPin	SLGenericReal1		openwire					
FrequencyPin		SLSignalGen1		🔁 Restore					

On the form double click on the Chart1 component to open the component editor:



Click the "Add.." button to add a new series, and select Line, then click OK:



Click Close:





Now your form will be similar to the one shown on the picture:

In the "Object Inspector" set the SLGenericReal1 SynchronizeType property to stSingleBuffer:

U				
Object Inspector 🛛 🛛 🛛				
SLGenericReal1	TSLGenericRe 🔽			
Properties Ev	rents			
Enabled	True			
EnablePin	(Disconnected)			
InputPin	SLSignalGen1.Outputl			
Name	SLGenericReal1			
OutputPin	(Disconnected)			
SynchronizeT	stSingleBuffer 💌			
Tag	0			

In the "Object Inspector" select the "Events" tab and double click on the OnProcessData event:

Object Inspector 🛛 🛛 🛛					
SLGenericR	eal1 TSLGenericRea				
Properties	Events				
OnFilterDa	ata				
OnProces	sDa 💌				
OnStart					

In the event handler add the following code:

If you are using Delphi:

```
procedure TForm1.SLGenericReal1ProcessData(Sender: TObject;
    InBuffer: ISLRealBuffer; var OutBuffer: ISLRealBuffer;
    var SendOutputData: Boolean);
var
    I : Integer;
begin
    Chart1.Series[ 0 ].Clear();
    for I := 0 to InBuffer.GetSize() - 1 do
```

```
Chart1.Series[ 0 ].Add(InBuffer.Items[ I ], '', clRed );
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::SLGenericReallProcessData(TObject *Sender,
	TSLCRealBuffer InBuffer, TSLCRealBuffer &OutBuffer,
	bool &SendOutputData)
{
	Chart1->Series[ 0 ]->Clear();
	for( unsigned int i = 0; i < InBuffer.GetSize(); i ++ )
		Chart1->Series[ 0 ]->Add(InBuffer[ i ], "", clRed );
}
```



Compile and run the application. The result should be similar to this one:

Here are the OpenWire connections in this application:



Congratulations! You have just learned how to use data generated by SignalLab inside your code.

Manual data pumping

SignalLab offers a set of generators such as TSLSignalGen and TSLRandomGen. They all have the capability to automatically pump data with a user defined data rate, however there are cases when you may need to control the data generation from inside you code.

All of the generators included in SignalLab have also the option to have code controlled data pumping.

Now we will create a simple manual pumping application using the TSLSignalGen. From the Delphi/C++Builder menu select | File | New | Application |.

70	De	lphi 7			
	ile	Edit Search View Project	Ru	n Component Da	stabase OpenWire Tools Window Help Knone> 💌 🕙 🚳
		New >	۲D	Application	ianal Lab Video Lab Audio Lab Additional Win32 Svstem Data Access Data Controls dbExpress DataSnap BDE ADO 💶
Ť	2	Open	()	CLX Application	II 🖁 A 🔤 🗷 🗵 🛛 🔜 🗃 🚟 🚥 🗋 🚍 🔛
ч	2	Open Project Ctrl+F11		Data Module	
		Reopen >		Form	
		Save Ctrl+S		Frame	
		Save As	j,	Unit	
	1	Save Project As	*	Other	
		Save All Shift+Ctrl+S	-		
		Close			
		Close All			
	Ъ	Use Unit Alt+F11			
	6	Print			
	n.	Exit			

An empty form will appear on the screen.

From the	"Component	Palette"	select the	"Standard"	tab:
----------	------------	----------	------------	------------	------

Proje	ct1										
earch	View	Project	Run	Component	Database	OpenWire	Tools Wind	ow Help	<none></none>	 ■ 	5 S.
] 🍯) 🚑	🖄 🖻	0	Standard	Sianal Lab	Video Lab	Audio Lab	Additional	Win32 Svste	m 🗍 Data Access	Data Controls dt
)	• '	3 🍞			δ A 🗔		x •	i i		R

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



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

~			- 1		(Alana)		ลาส					
Compon	ent Database	Openwire	Tools Wind	Jow Help	(None>	<u> </u>	R 2 P					
Star	dard Signal La	Video Lab	Audio Lab	Additional	Win32 Sv	stem Data Acces	s Data Contro	ls dbExpress	DataSnap	BDE A	.DO InterBase	WebServices
	GEN IIR		R FFT IFFT	IFFT GM	nor		MIXIFI	: : 🔊	🕄 🕄 🔿 GEN	GEN 🚲	। কিন্তু কিন্তু চিচ	
13	් කය කය	गढ ठाढ ठा	<u>are are</u>	वाट वाह	2012 2012	- אר אר	ଅନେ ଆହି ଆହି	ાઢ સાઢ સાઢ	र ठाइ ठाइ	ais ais	ाठ ठाठ ठाह	ट यह यह द

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



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

Component Database OpenWire Tools	Window Help Ksearch component)
Standard Signal Lab Audio Lab Video Lab	Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap

select and drop on the form the following component:





Arrange the form as shown on the picture and select the SLSignalGen1:



•	/ 1				
Object Inspector 🛛 🛛 🔀					
SLSignalGen1	TSLSignalGen 💌				
Properties Ev	vents				
Name	SLSignalGen1 🛛 🔥				
Offset	0				
OutputPin	Disconnected)				
Phase	0				
PumpPrioritu	n —				

You should see the Pin Editor:

🅻 Connections - Source Pin : SLSignalGen1.Outp 🔳 🗖 🔀								
Form	: Form1 (Current)		•	(7)				
Sink	pin	🐥 Component	Connected to	OpenWire				
\square	InputPins.Channel0	SLScope1		openwire				
FrequencyPin		SLSignalGen1		🔁 <u>R</u> estore				

Click on the check box to make it look as in the picture, and then click OK.

In the object inspector change the ClockSource to csExternal:

Object Inspector 🛛 🔀					
SLSignalGen1	TSLSignalGen 💌				
Properties Ev	rents				
Amplitude	20000 🔥				
Asymmetry	0				
BufferSize	1024 🗧				
ClockPin	(Disconnected)				
ClockSource	csExternal 💌				
Enabled	True				
EnablePin	(Disconnected)				

On the form select the Button1:

🌈 Form1		
	111	÷
🗀 Button1 🖬 👭	N	•
	K	•
· · • • • • • • • • • • • • • • • • • •	* * *	•

In the Object Inspector change the caption to "Pump":

Object Inspector				
Button1	TButton	•		
Properties Ev	rents			
Action		^		
⊞Anchors	[akLeft,akTop]			
BiDiMode	bdLeftToRight			
Cancel	False			
Caption	Pump			
⊞ Constraints	(TSizeConstraints)			

Double click on Button1:



In the event handler add the following code:

If you are using Delphi:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   SLSignalGen1.Pump();
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    SLSignalGen1->Pump();
}
```

Compile and run the application.

Press the button few times. Each time you press the button, a new data buffer will be

generated and sent to the scope:



Here are the OpenWire connections in this application:



You have just learned how to pump data from inside you code.

Using SignalLab with data acquisition board

SignalLab is universal signal processing library, it is not bound to any particular data acquisition board or vendor, and can be used with pretty much any hardware. However in order to read the data from the hardware and to feed it into SignalLab you will need to write a little bit of code. Because of the large variety of data acquisition boards, and the different libraries the vendors provide it is impossible to cover all of them. Instead we will design an application that will use a software simulated data acquisition board. First we will define a data acquisition board API, and then we will learn how to use such an API with SignalLab.

The modern data acquisition board usually consists of one or more data sources such as Analog to Digital Converter (ADC) or Digital Input (DI), one or more outputs such as Digital to Analog Converter (DAC) or Digital Output (DO), one or more timers (clocks) and control logic. There is a huge variety of functionalities and implementations. On some boards there are only inputs or only outputs. Some of the boards will allow external clocking or direct data exchange with other hardware, but the basic principles are usually the same.

How the typical input of data acquisition board works?

The timer (clock source) will generate data acquisition clock with a certain rate. The ADC will use the clock to sample the data and the result will go into internal board buffer (part of the control logic). The data then will be sent to the PC in the form of buffer. A driver inside the PC will receive the buffer and will deliver it to a user level library

provided by the board vendor. The library will call a callback function of some type registered from inside your code via the library API. From this point you can use the data as it fits your needs.

How the typical output of data acquisition board works?

When you start the board it will send to the PC a request for data. The driver will receive the request and will signal the board's user level library. The user level library will call a callback registered via its API by your code. Your code must provide the buffer. Once the buffer is obtained it will be delivered to the board and the board will store it inside its own hardware queue. The board timer then will be started.

The timer (clock source) will generate data acquisition clock with a certain rate. The DAC will read the data from the internal board buffer queue. When the queue is empty to a certain level, a new request for data will be sent to the PC. The process will continue until the board is stopped.

Our simulated API must provide a way for us to register callback functions for providing and requesting data buffer. The format of the buffer will most likely be somewhat different than TSLCRealBuffer. The board also will allow us to start and stop the acquisition. Although very simple this API resembles in essence exactly what most of the data acquisition board APIs do.

Now we will define the following API commands:

If you are using Delphi:

```
type TAQDataType = ^Real;
type TDAQCallBack = procedure( Data : TAQDataType ); stdcall;
procedure DAQSetDataSupplyCallback( ACallBack : TDAQCallBack );
procedure DAQSetDataRequestCallback( ACallBack : TDAQCallBack );
procedure DAQStartAcquisition();
procedure DAQStopAcquisition();
```

If you are using C++ Builder:

```
typedef __stdcall void ( *TDAQCallBack ) ( double * Data );
void DAQSetDataSupplyCallback( TDAQCallBack ACallBack );
void DAQSetDataRequestCallback( TDAQCallBack ACallBack );
void DAQStartAcquisition();
void DAQStopAcquisition();
```

For simplicity we will assume fixed data acquisition rate defined by the board's hardware.

We will not go into many details on how the hardware simulator works. We will just use a TTimer to generate timer events and they will trigger the DataSupply and DataRequest events by calling our callback procedures.

7	Del	lphi 7			
J	ile	Edit Search View Project	Ru	in Component D	itabase OpenWire Tools Window Help 🛛 🔨 🖉 🦓
		New 🕨	Ę٦	Application	ianal Lab Video Lab Audio Lab Additional Win32 Sustem Data Access Data Controls dbExpress DataSnap BDE ADO 💶
Ť	Į,	Open	()	CLX Application	II 💺 A 🔤 💷 🗵 🛛 🛃 🗮 🚥 🗋 🗮 🔄
4	2	Open Project Ctrl+F11		Data Module	
		Reopen >		Form	
		Save Ctrl+S		Frame	
		Save As		Unit	
		Save Project As	*	Other	
	9	Save All Shift+Ctrl+S	_		
	日 日 日 日 日 日	Close			
		Close All			
	Ъ	Use Unit Alt+F11			
	6	Print			
	n.	Exit			

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

An empty form will appear on the screen.

From the "Component Palette" select the "System" tab:

Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
省 🔯 🗸 🗐 🦪 🚰 😰 😰 🛷 Standard Sional Lab Video Lab Audio Lab Additional Win32 System Data Access Data Contro
19 2 13 🔲 🕨 - II 💈 🍞 😓 🕓 🖉 🌿 🚥 🖓 📲 🖓 🔤

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





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

Object Inspector 🛛 🛛						
Timer1	TTimer 💽					
Properties E	vents					
Enabled	False					
Interval	100					
Name	Timer1					
Tag	n					

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:



```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
 Dialogs, ExtCtrls;
type
 TForm1 = class(TForm)
   Timer1: TTimer;
   procedure Timer1Timer(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
  end;
var
 Form1: TForm1;
type TAQDataType = ^Real;
type TDAQCallBack = procedure( Data : TAQDataType ); stdcall;
procedure DAQSetDataSupplyCallback( ACallBack : TDAQCallBack );
procedure DAQSetDataRequestCallback( ACallBack : TDAQCallBack );
procedure DAQStartAcquisition();
procedure DAQStopAcquisition();
implementation
{$R *.dfm}
var SuppyCallBack : TDAQCallBack;
var RequestCallBack : TDAQCallBack;
var InternalCounter : Integer;
procedure DAQSetDataSupplyCallback( ACallBack : TDAQCallBack );
begin
 SuppyCallBack := ACallBack;
end;
procedure DAQSetDataRequestCallback( ACallBack : TDAQCallBack );
begin
 RequestCallBack := ACallBack;
end;
procedure DAQStartAcquisition();
begin
 Form1.Timer1.Enabled := True;
end;
procedure DAQStopAcquisition();
begin
 Form1.Timer1.Enabled := False;
end;
procedure DAQRequestData();
var
  DataBuffer : array[ 0..1023 ] of Real;
```

```
begin
 if( Assigned( RequestCallBack )) then
   RequestCallBack(@DataBuffer[ 0 ] );
end;
procedure DAQSupplyData();
var
 DataBuffer : array[ 0..1023 ] of Real;
 I : Integer;
begin
 for I := 0 to 1023 do
   begin
   DataBuffer[ I ] := InternalCounter mod 100;
   Inc( InternalCounter );
   end;
 if ( Assigned ( SuppyCallBack )) then
    SuppyCallBack( @DataBuffer[ 0 ] );
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 _DAQRequestData();
  DAQSupplyData();
end;
end.
```

If you are using C++ Builder:

```
//-----
___
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//-----
                        ------
___
#pragma package(smart init)
#pragma resource "*.dfm"
TForm1 *Form1;
                     -----
//-----
___
typedef stdcall void ( *TDAQCallBack ) ( double * Data );
void DAQSetDataSupplyCallback( TDAQCallBack ACallBack );
void DAQSetDataRequestCallback( TDAQCallBack ACallBack );
void DAQStartAcquisition();
void DAQStopAcquisition();
```

```
TDAQCallBack SuppyCallBack;
TDAQCallBack RequestCallBack;
int InternalCounter = 0;
//-----
void DAQSetDataSupplyCallback( TDAQCallBack ACallBack )
{
SuppyCallBack = ACallBack;
//-----
void DAQSetDataRequestCallback( TDAQCallBack ACallBack )
{
RequestCallBack = ACallBack;
               _____
void DAQStartAcquisition()
Form1->Timer1->Enabled = true;
//-----
--void DAQStopAcquisition()
{
Form1->Timer1->Enabled = false;
//-----
--void DAQRequestData()
{
double DataBuffer[ 1024 ];
if( RequestCallBack != NULL )
  RequestCallBack( DataBuffer );
//-----
--void DAQSupplyData()
{
 double DataBuffer[ 1024 ];
 for( int i = 0; i < 1024; i ++ )
  DataBuffer[ i ] = InternalCounter % 100;
  InternalCounter ++;
 if( SuppyCallBack != NULL )
  SuppyCallBack( DataBuffer );
//_____
-- fastcall TForm1::TForm1(TComponent* Owner)
  : TForm(Owner)
```

```
//----
--void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    __DAQRequestData();
    __DAQSupplyData();
}
//------
```

First we will create a signal generator and will use it to supply data for the DAC of the data acquisition board.

From the "Component Palette" select the "Standard" tab:

Project1										
earch Vie	ew Project	Run	Component	Database	OpenWire	Tools Wind	low Help	<none></none>	- P	a,
) 🗊 🕯	🖹 🖄 🖄)	Standard	Sional Lab	Video Lab	Audio Lab	Additional	Win32 Svstem	Data Access D	ata Controls dt
	•	3 🚱		I I I	δ A 🔤		×	:	- E .	

From the tab select and drop on the form two TButton components.

- TButton

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

select and drop on the form the following two components:

- TSLSignalGen

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

		í.
1	Component Database OpenWire Tools Window Help 🛛 😽 (search component) 🔹 🧐 🖓 🥵	Ç
Τ	Standard Signal Lab Audio Lab Video Lab Plot Lab Additional Win32 System Data Access Data Controls dbExpress DataSnap	İ.
-		

select and drop on the form the following component:

ም - TSLScope

Change the Button1 caption to be Start and Button2 caption to be Stop. Arrange the form as it is shown on the form and select the SLSignalGen1:



In the Object Inspector change the ClockSource to csExternal:

Object Inspector 🛛 🛛						
SLSignalGen1 TSLSignalGen						
Properties Events						
BufferSize	1024 🔨					
ClockPin	(Disconnected)					
ClockSource	csExternal 💌 🔄					
Enabled	True					
EnablaDin	(Disconnected)					

In the Object inspector select the OutputPin property and click the **button**.

Object Inspector 🛛 🔀						
SLSignalGen1 TSLSignalGen 💌						
Properties Events						
Name	SLSignalGen1 🛛 🔥					
Offset	0					
OutputPin	Disconnected)					
Phase	0					
D	0					

In the Pin Editor check the following pins and click OK.

🥻 Connections - Source Pin : SLSignalGen1.Outp 🔳 🗖 🔀								
Form	: Form1 (Current)	6						
Sink	pin	🐥 Component	Connected to	OpenWire				
	InputPin	SLGenericReal1		opennie				
	InputPins.Channel0	SLScope1		S Restore				
FrequencyPin		SLSignalGen1						
				<mark>}⊸{</mark> Link to all				

Switch to the code editor.

Add a data buffer into the public section of your form. In Delphi the buffer will be encapsulated in Variant, in C++ Builder we will use TSLCRealBuffer buffer.

If you are using Delphi:

```
type
TForm1 = class(TForm)
Timer1: TTimer;
SLSignalGen1: TSLSignalGen;
SLGenericReal1: TSLGenericReal;
SLScope1: TSLScope;
Button1: TButton;
Button2: TButton;
procedure Timer1Timer(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
GeneratedBuffer : ISLRealBuffer;
end;
```

If you are using C++ Builder, switch to the header file and add TSLCRealBuffer GeneratedBuffer to the form class:

```
class TForm1 : public TForm
{
 published: // IDE-managed Components
       TTimer *Timer1;
       TSLSignalGen *SLSignalGen1;
       TSLGenericReal *SLGenericReal1;
       TSLScope *SLScope1;
       TButton *Button1;
       TButton *Button2;
       void fastcall Timer1Timer(TObject *Sender);
private: // User declarations
public:
 TSLCRealBuffer GeneratedBuffer;
public:
                 // User declarations
        fastcall TForm1(TComponent* Owner);
```

On the form double click on the SLGenericReal1:



The Delphi/C++ Builder will generate an event handler. Change the code for the handler as shown here:

If you are using Delphi:

```
procedure TForm1.SLGenericReal1ProcessData(Sender: TObject;
    InBuffer: ISLRealBuffer; var OutBuffer: ISLRealBuffer;
    var SendOutputData: Boolean);
begin
```

```
GeneratedBuffer := TSLRealBuffer.CreateCopy( InBuffer );
end;
```

If you are using C++ Builder:

Add the following callback function to your code:

If you are using Delphi:

```
procedure DataRequestCallback( Data : TAQDataType ); stdcall;
begin
    Form1.SLSignalGen1.Pump();
    Move(Form1.GeneratedBuffer.Read()^, Data^, 1024 * SizeOf( Real ) );
end;
```

If you are using C++ Builder:

```
void __stdcall DataRequestCallback( double * Data )
{
    Form1->SLSignalGen1->Pump();
    Form1->GeneratedBuffer.ToDouble( Data );
}
```

On the form double click on Button1. The Delphi/C++ Builder will generate an event handler. Change the code for the handler as shown here:

If you are using Delphi:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   DAQSetDataRequestCallback( DataRequestCallback );
   DAQStartAcquisition();
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    DAQSetDataRequestCallback( DataRequestCallback );
    DAQStartAcquisition();
}
```

On the form double click on Button2. The Delphi/C++ Builder will generate an event handler. Change the code for the handler as shown here:

If you are using Delphi:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
DAQStopAcquisition();
end;
```

If you are using C++ Builder:



Compile and run the application. Click on the start button. You will see a sine wave shown on the scope:



Here are the OpenWire connections in this application:



When the simulated board needs data it will call our callback function. The callback will call the Pump method of the SLSignalGen1 to generate a single sine wave buffer. The buffer will be sent to Channel0 of the scope as well as to the SLGenericReal1 filter. The SLGenericReal1 filter then will store the buffer into our GeneratedBuffer and the callback function will copy the buffer into the buffer supplied by the data acquisition API.

Now we will expand the application to display the data supplied by our simulated data acquisition board.

Stop the running application.

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



select and drop on the form the following component:

- TSLGenericReal

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



select and drop on the form the following component:





Arrange the form as shown on the picture and select the SLGenericReal2:

In the Object inspector select the OutputPin property and click the 🛄 button.

Object Inspector 🛛 🔀						
SLGenericReal2 TSLGenericRea						
Properties Events						
Enabled	True					
EnablePin	(Disconnected)					
InputPin	(Disconnected)					
Name	SLGenericReal2					
OutputPin	(Disconnected)					
SynchronizeT stNone						

In the Pin Editor check the following pin and click OK.

🥻 Connections - Source Pin : SLGenericReal2.0u 🔳 🗖 🔀							
Form	: Form1 (Current)	6)					
Sink	pin	🐥 Component	Connected to	OpenWire			
	InputPin	SLGenericReal1	SLSignalGe	openwire			
	InputPins.Channel0	SLScope1	SLSignalGe	A Restore			
	InputPins.Channel0	SLScope2					
ClockPin FrequencyPin		SLSignalGen1		<mark>}⊸{</mark> Link to all			
		SLSignalGen1					
				💓 <u>U</u> nlink all			

In the code change the event handlers Button1Click and Button2Click as shown here:

If you are using Delphi:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
DAQSetDataRequestCallback( DataRequestCallback );
DAQSetDataSupplyCallback( DataSupplyCallback );
DAQStartAcquisition();
SLGenericReal2.SendStartCommand( 1024 * 10 );
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
DAQStopAcquisition();
SLGenericReal2.SendStopCommand();
end;
```

If you are using C++ Builder:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    DAQSetDataRequestCallback( DataRequestCallback );
    DAQSetDataSupplyCallback( DataSupplyCallback );
    DAQStartAcquisition();
    SLGenericReal2->SendStartCommand( 1024 * 10 );
}
//-----
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    DAQStopAcquisition();
    SLGenericReal2->SendStopCommand();
}
```

Add the following callback function to your code before Button1Click:

If you are using Delphi:

```
procedure DataSupplyCallback( Data : TAQDataType ); stdcall;
var
SupplyBuffer : ISLRealBuffer;
begin
SupplyBuffer := TSLRealBuffer.CreateData( PReal(Data), 1024 );
Form1.SLGenericReal2.SendData( SupplyBuffer );
end;
```

If you are using C++ Builder:

```
void __stdcall DataSupplyCallback( double * Data )
{
   TSLCRealBuffer SupplyBuffer( Data, 1024 );
   Form1->SLGenericReal2->SendData( SupplyBuffer );
}
```



Compile and run the application. Click on the start button. You will see a sine wave shown on the first scope and a ramp displayed on second scope:

Here are the OpenWire connections in this application:



We already explained the signal generating part of the application. Here is how the data acquisition part works. The simulated board will call our callback function. In the callback function we will create a buffer, and will fill it with the data received from the board API. Then we will send the buffer via the SLGenericReal2 to the Channel0 of the second scope, where the signal will be displayed.

Here is the full source code of the application:

If you are using Delphi:

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
 Dialogs, ExtCtrls, SLCommonFilter, SLGenericReal, SLCommonGen,
 SLSignalGen, StdCtrls, SLScope;
type
  TForm1 = class(TForm)
   Timer1: TTimer;
   Button1: TButton;
   Button2: TButton;
   SLSignalGen1: TSLSignalGen;
    SLGenericReal1: TSLGenericReal;
    SLScope1: TSLScope;
    SLGenericReal2: TSLGenericReal;
   SLScope2: TSLScope;
   procedure Timer1Timer(Sender: TObject);
   procedure SLGenericReal1ProcessData(Sender: TObject;
      InBuffer: ISLRealBuffer; var OutBuffer: ISLRealBuffer;
      var SendOutputData: Boolean);
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    GeneratedBuffer : ISLRealBuffer;
  end;
var
 Form1: TForm1;
type TAQDataType = ^Real;
type TDAQCallBack = procedure( Data : TAQDataType ); stdcall;
procedure DAQSetDataSupplyCallback( ACallBack : TDAQCallBack );
procedure DAQSetDataRequestCallback( ACallBack : TDAQCallBack );
procedure DAQStartAcquisition();
procedure DAQStopAcquisition();
```

```
implementation
{$R *.dfm}
var SuppyCallBack : TDAQCallBack;
var RequestCallBack : TDAQCallBack;
var InternalCounter : Integer;
procedure DAQSetDataSupplyCallback( ACallBack : TDAQCallBack );
begin
 SuppyCallBack := ACallBack;
end;
procedure DAQSetDataRequestCallback( ACallBack : TDAQCallBack );
begin
  RequestCallBack := ACallBack;
end;
procedure DAQStartAcquisition();
begin
  Form1.Timer1.Enabled := True;
end;
procedure DAQStopAcquisition();
begin
 Form1.Timer1.Enabled := False;
end;
procedure DAQRequestData();
var
  DataBuffer : array[ 0..1023 ] of Real;
begin
 if( Assigned( RequestCallBack )) then
    RequestCallBack(@DataBuffer[ 0 ] );
end;
procedure _DAQSupplyData();
var
  DataBuffer : array[ 0..1023 ] of Real;
 I : Integer;
begin
 for I := 0 to 1023 do
   begin
    DataBuffer[ I ] := InternalCounter mod 100;
    Inc( InternalCounter );
   end;
  if (Assigned (SuppyCallBack)) then
    SuppyCallBack( @DataBuffer[ 0 ] );
end;
```

```
begin
  Form1.SLSignalGen1.Pump();
 Move(Form1.GeneratedBuffer.Read()^, Data^, 1024 * SizeOf( Real ) );
end;
procedure DataSupplyCallback( Data : TAQDataType ); stdcall;
var
  SupplyBuffer : ISLRealBuffer;
begin
  SupplyBuffer := TSLRealBuffer.CreateData( PReal(Data), 1024 );
  Form1.SLGenericReal2.SendData( SupplyBuffer );
end:
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  DAQRequestData();
  DAQSupplyData();
end;
procedure TForm1.SLGenericReal1ProcessData(Sender: TObject;
 InBuffer: ISLRealBuffer; var OutBuffer: ISLRealBuffer;
 var SendOutputData: Boolean);
begin
 GeneratedBuffer := TSLRealBuffer.CreateCopy( InBuffer );
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
 DAQSetDataRequestCallback( DataRequestCallback );
 DAQSetDataSupplyCallback( DataSupplyCallback );
 DAQStartAcquisition();
 SLGenericReal2.SendStartCommand( 1024 * 10 );
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
 DAQStopAcquisition();
  SLGenericReal2.SendStopCommand();
end;
end.
```

If you are using C++ Builder, here is how your header file will look like:

```
#ifndef Unit1H
#define Unit1H
//-----
--
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "SLCommonFilter.hpp"
#include "SLCommonGen.hpp"
#include "SLGenericReal.h"
```

```
#include "SLScope.hpp"
#include "SLSignalGen.hpp"
#include <ExtCtrls.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
      TTimer *Timer1;
      TSLSignalGen *SLSignalGen1;
      TSLGenericReal *SLGenericReal1;
      TSLScope *SLScope1;
      TButton *Button1;
      TButton *Button2;
      TSLScope *SLScope2;
      TSLGenericReal *SLGenericReal2;
      void fastcall Timer1Timer(TObject *Sender);
      void fastcall Button1Click(TObject *Sender);
      void fastcall SLGenericReal1FilterData(TObject *Sender,
        TSLCRealBuffer InBuffer, TSLCRealBuffer &OutBuffer,
        bool &SendOutputData);
      void fastcall Button2Click(TObject *Sender);
private: // User declarations
public:
 TSLCRealBuffer GeneratedBuffer;
         // User declarations
public:
       fastcall TForm1(TComponent* Owner);
};
                    _____
//-
___
extern PACKAGE TForm1 *Form1;
//-----
___
#endif
```

If you are using C++ Builder, here is how your source file will look like:

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "Unitl.h"
//------
#pragma package(smart_init)
#pragma link "SLCommonFilter"
#pragma link "SLCommonGen"
#pragma link "SLGenericReal"
#pragma link "SLScope"
#pragma link "SLSignalGen"
#pragma resource "*.dfm"
TForm1 *Form1;
```

```
//------
___
typedef stdcall void ( *TDAQCallBack ) ( double * Data );
void DAQSetDataSupplyCallback( TDAQCallBack ACallBack );
void DAQSetDataRequestCallback( TDAQCallBack ACallBack );
void DAQStartAcquisition();
void DAQStopAcquisition();
TDAQCallBack SuppyCallBack;
TDAQCallBack RequestCallBack;
int InternalCounter = 0;
//-----
___
void DAQSetDataSupplyCallback( TDAQCallBack ACallBack )
{
 SuppyCallBack = ACallBack;
//-----
void DAQSetDataRequestCallback( TDAQCallBack ACallBack )
{
RequestCallBack = ACallBack;
        -----
void DAQStartAcquisition()
{
Form1->Timer1->Enabled = true;
//-----
            -----
void DAQStopAcquisition()
 Form1->Timer1->Enabled = false;
}
//-----
___
void _DAQRequestData()
{
 double DataBuffer[ 1024 ];
if( RequestCallBack != NULL )
  RequestCallBack( DataBuffer );
//-----
void DAQSupplyData()
double DataBuffer[ 1024 ];
 for( int i = 0; i < 1024; i ++ )
```

```
DataBuffer[ i ] = InternalCounter % 100;
   InternalCounter ++;
   }
 if( SuppyCallBack != NULL )
   SuppyCallBack( DataBuffer );
//-
                  _____
                        _____
//--
 fastcall TForm1::TForm1(TComponent* Owner)
   : TForm(Owner)
//------
void fastcall TForm1::Timer1Timer(TObject *Sender)
{
 _DAQRequestData();
 DAQSupplyData();
//-----
void stdcall DataRequestCallback( double * Data )
{
 Form1->SLSignalGen1->Pump();
 Form1->GeneratedBuffer.ToDouble( Data );
//-
void stdcall DataSupplyCallback( double * Data )
{
 TSLCRealBuffer SupplyBuffer( Data, 1024 );
 Form1->SLGenericReal2->SendData( SupplyBuffer );
}
//-----
             _____
___
void __fastcall TForm1::SLGenericReal1FilterData(TObject *Sender,
     TSLCRealBuffer InBuffer, TSLCRealBuffer &OutBuffer,
    bool &SendOutputData)
{
 GeneratedBuffer = InBuffer;
//--
                       _____
void fastcall TForm1::Button1Click(TObject *Sender)
 DAQSetDataRequestCallback( DataRequestCallback );
 DAQSetDataSupplyCallback( DataSupplyCallback );
 DAQStartAcquisition();
 SLGenericReal2->SendStartCommand( 1024 * 10 );
```

This was by far the most complex application we covered in this manual. It consists of a simple data acquisition board simulator as well as all the necessary code for feeding data into the simulated board and receiving the generated simulation data from it. By now you should have a good idea how you can integrate SignalLab with the hardware of your choice.

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.