

# AudioLab 7.5

## Quick Start



[www.openwire.org](http://www.openwire.org)  
[www.mitov.com](http://www.mitov.com)

Copyright Boian Mitov 2004 - 2014

# Index

<a href="#">Installation.....</a>	<a href="#">3</a>
<a href="#">Where is AudioLab?.....</a>	<a href="#">3</a>
<a href="#">Why some of the examples don't work?.....</a>	<a href="#">3</a>
<a href="#">Creating a simple audio player using Win32API Components.....</a>	<a href="#">4</a>
<a href="#">Creating a simple audio player using DirectShow components.....</a>	<a href="#">6</a>
<a href="#">Creating a simple Audio Recorder Win32API Components.....</a>	<a href="#">8</a>
<a href="#">Using the TSLCRealBuffer in C++ Builder and Visual C++.....</a>	<a href="#">13</a>
<a href="#">Distributing your application.....</a>	<a href="#">15</a>
<a href="#">Deploying your 32 bit application with the IPP DLLs.....</a>	<a href="#">15</a>
<a href="#">Deploying your 64 bit application.....</a>	<a href="#">16</a>

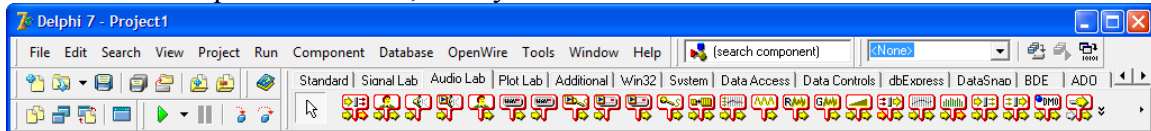
## Installation

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

## Where is AudioLab?

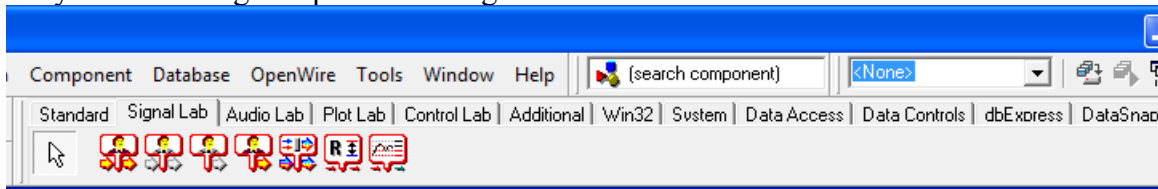
After the installation, start your Delphi or C++ Builder.

Scroll the “Component Palette”, until you see the last three tabs:

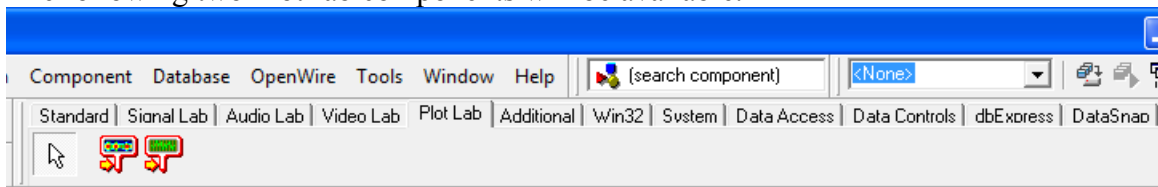


If the installation was successful, they should be named “Signal Lab”, “AudioLab” and “Plot Lab”. On the SignalLab palette you will have only a subset of the SignalLab components. SignalLab is a separated product, and will not be shipped as full with AudioLab.

Only the following components of SignalLab will be available:



The following two PlotLab components will be available.



## Why some of the examples don't work?

Video lab is a unique library that supports both the Win32 API's AVIFile ( VFW ) functions (ACM) and DirectShow. You as a developer have the ultimate choice to use either the Win32 API or DirectShow components or both at the same time.

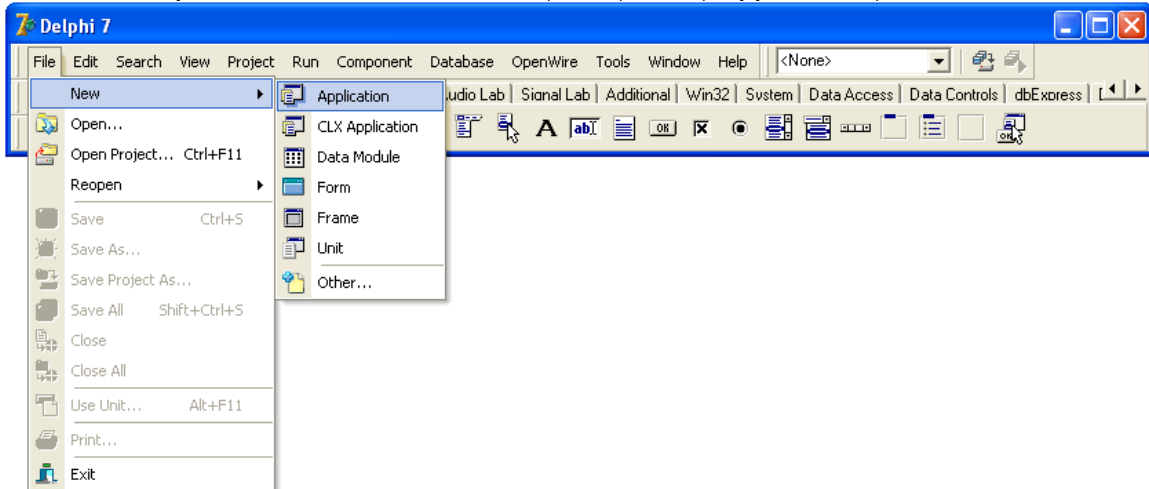
The advantage of the Win32 API components is that they will work on any Windows 95 and up system out of the box, however they are much less capable than the DirectShow components, and should be avoided if not necessary.

The advantage of the DirectShow components is that they will use the latest and greatest capability of DirectShow, the latest video camera devices, and TV Tuners, but they require DirectShow 9.0 or higher to be installed in order to work.

If you don't have DirectX 9.0 or higher installed on your system, you will not be able to use see the DirectShow examples working.

## Creating a simple audio player using Win32API Components

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



An empty form will appear on the screen.

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

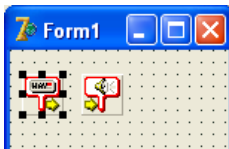


select and drop on the form the following two components:

One  - TALWavePlayer

One  - TALAudioOut

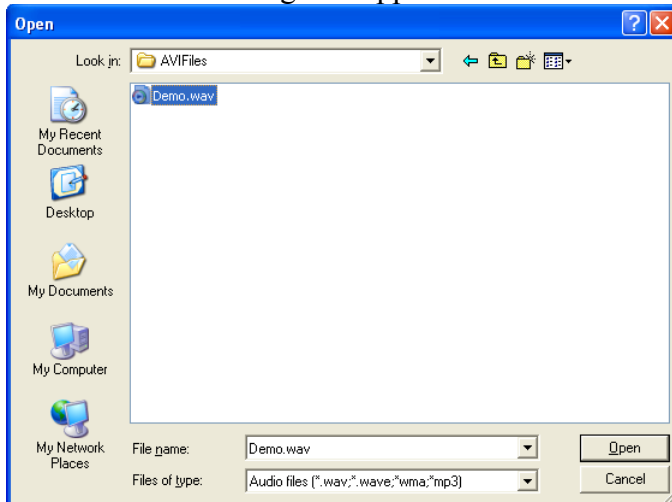
Make the form small and select the ALDSAUDIOOUT1 component:




In the Object Inspector select the FileName property and click the  button.



A File selection dialog will appear:

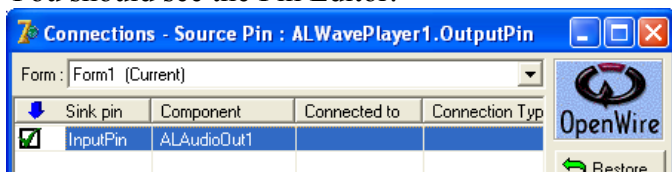


Select a file to play and click “Open”.

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



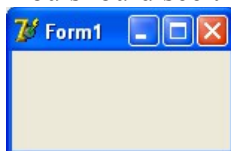
You should see the Pin Editor:



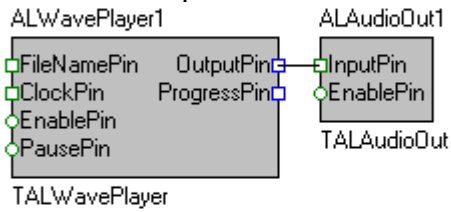
Click on the check box to make it look as in the picture, and then click OK.

Compile and run the application.

You should see the form and hear the audio playing:



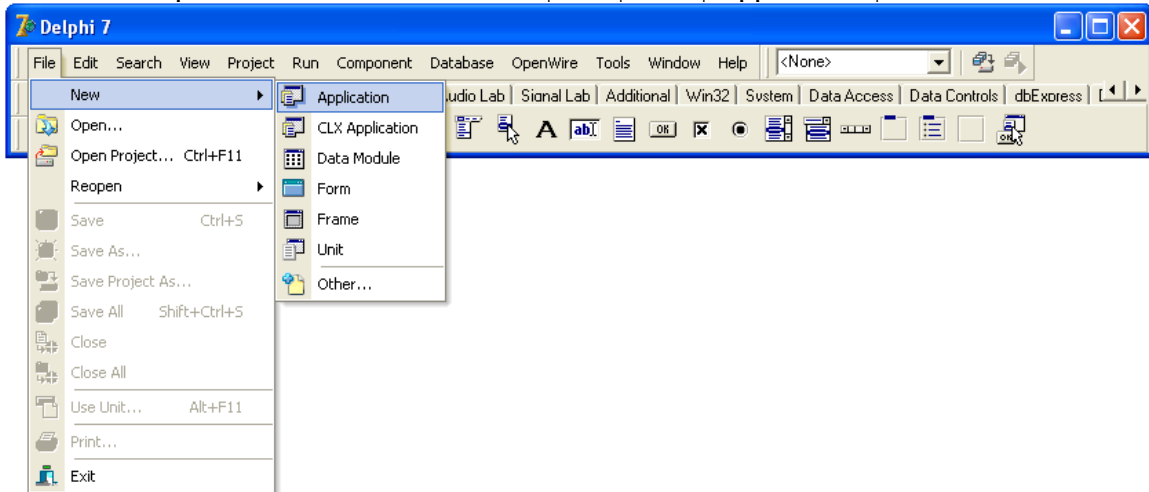
Congratulations! You have just created your first AudioLab application.  
Here are the OpenWire connections in this application:



## Creating a simple audio player using DirectShow components

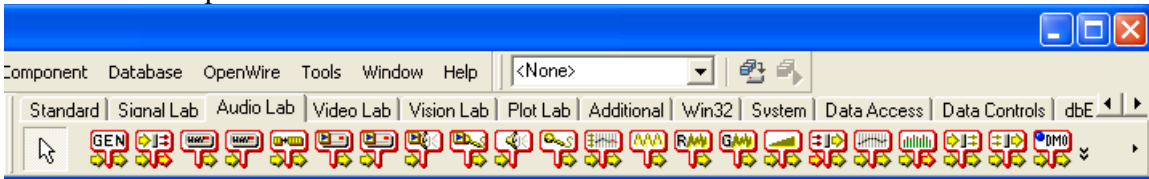
**WARNING:** In order to run the application in this example you must have DirectX 9.0 or higher installed!

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



An empty form will appear on the screen.

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

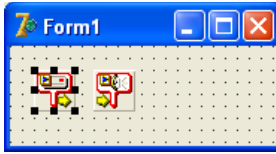


select and drop on the form the following two components:

One  - TALDSAUDIOPLAYER

One  - TALDSAUDIOOUT

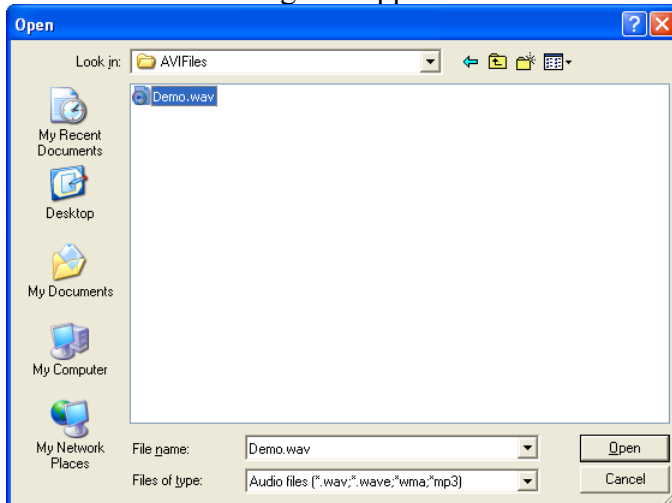
Make the form small and select the ALDSAudioOut1 component:



In the Object Inspector select the FileName property and click the button.



A File selection dialog will appear:

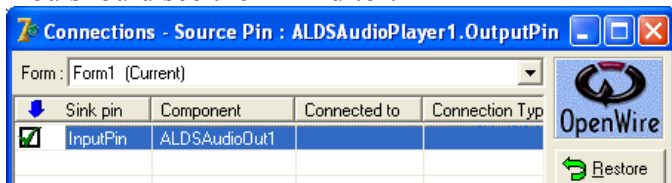


Select a file to play and click “Open”.

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

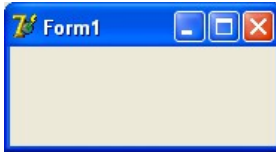


You should see the Pin Editor:

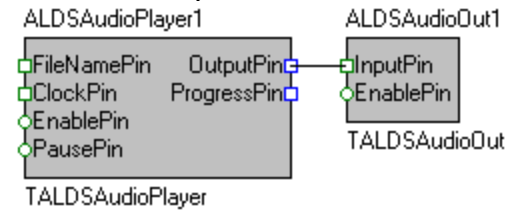


Click on the check box to make it look as in the picture, and then click OK.

Compile and run the application.  
 You should see the form and hear the audio playing:

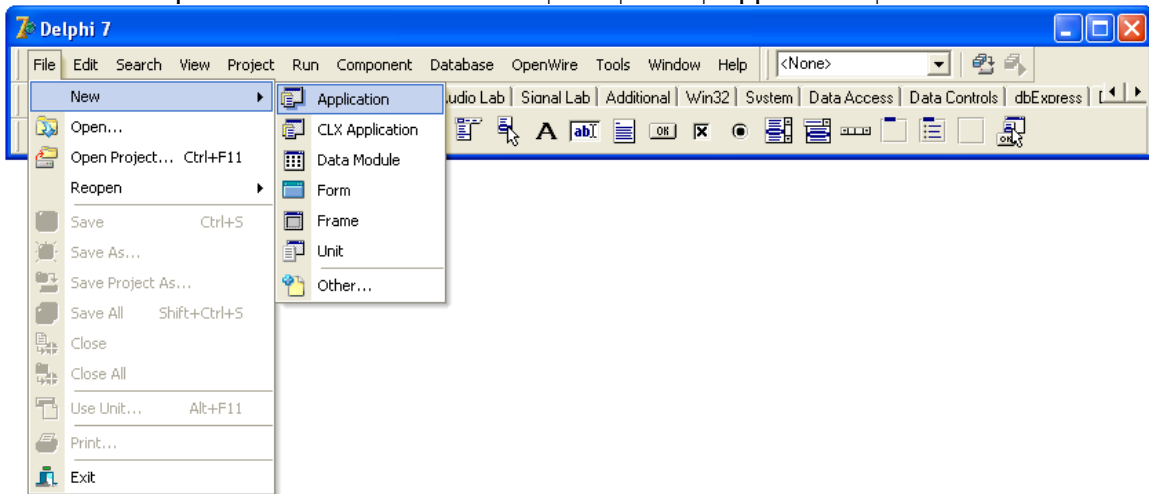


Congratulations! You have just created your first DirectShow AudioLab application.  
 Here are the OpenWire connections in this application:



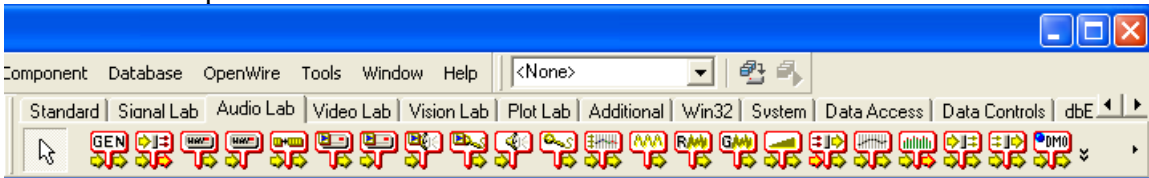
## Creating a simple Audio Recorder Win32API Components

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



An empty form will appear on the screen.

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



select and drop on the form the following three components:

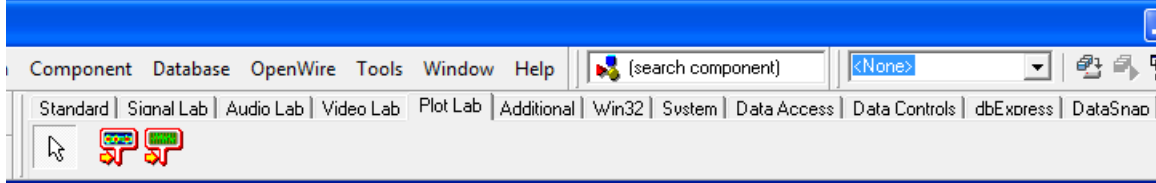
One  - TALAudioIn



One  - TALWaveLogger

One  - TALAudioToReal

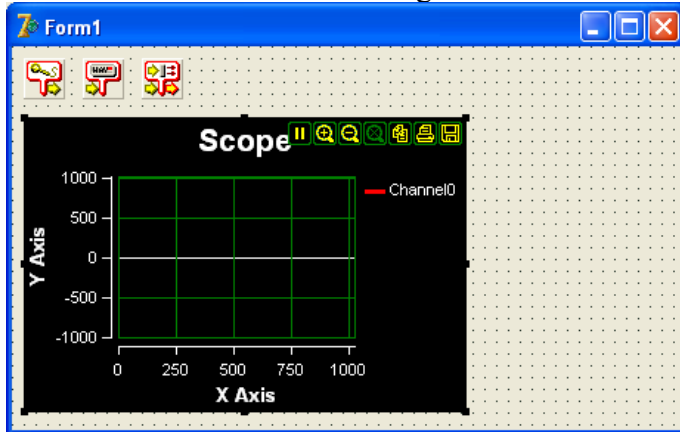
From the “Component Palette” select the “Plot Lab” tab:



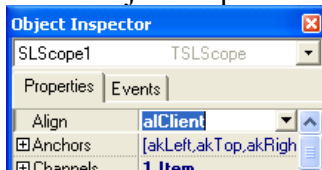
select and drop on the form the following component:

One  - TSLScope

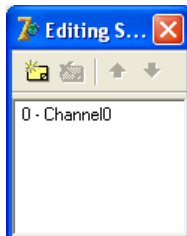
The Form should look something like this. Select the SLScope1 component:



In the Object Inspector set the Align property to alClient:



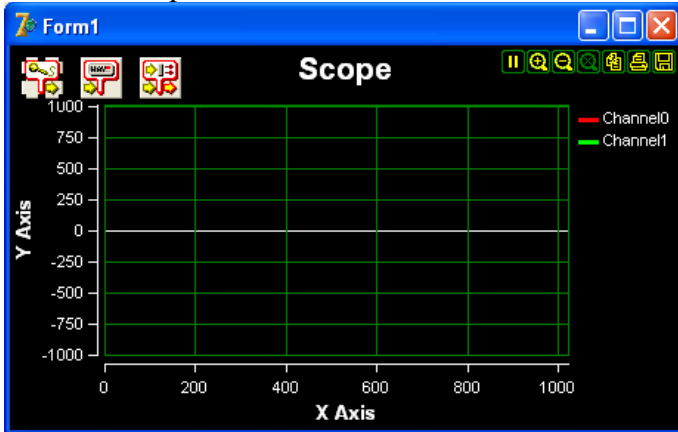
Double click on the SLScope1 component, to open the Channels editor:




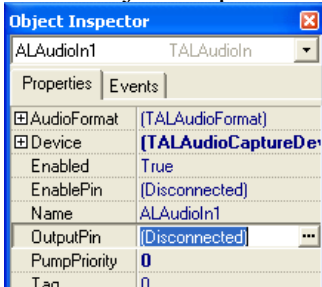
Click on the “Add New”  button to create one more channel:



Now the Scope should look like this. Select the ALAudioIn component:



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

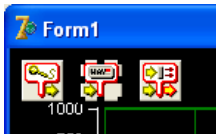


You should see the Pin Editor:



Click on the check boxes as shown in the picture, and then click OK.

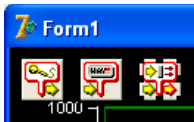
On the form select the ALWaveLogger1 component:



In the Object Inspector select the FileName property and set it to RecordedAudio.wav:



On the form select the ALAudioToReal1 component:



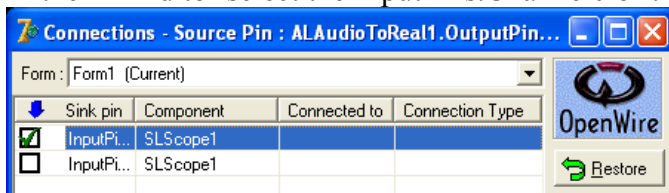
In the Object Inspector click on the [-] button to expand the OutputPins:




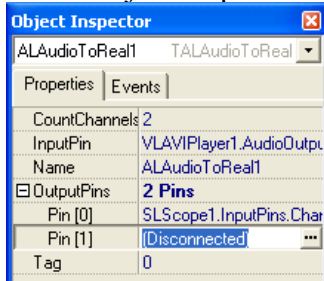
Click the [...] button of the Pin[0] sub property:



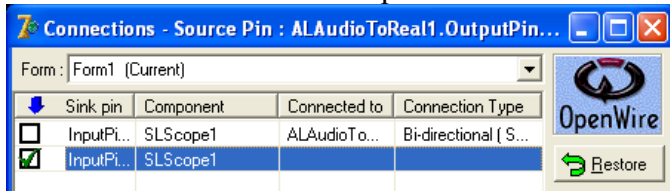
In the Pin Editor select the InputPins.Channel0 of the SLScope1 and click OK:



In the Object Inspector click the  button of the Pin[1] sub property of the OutputPins:

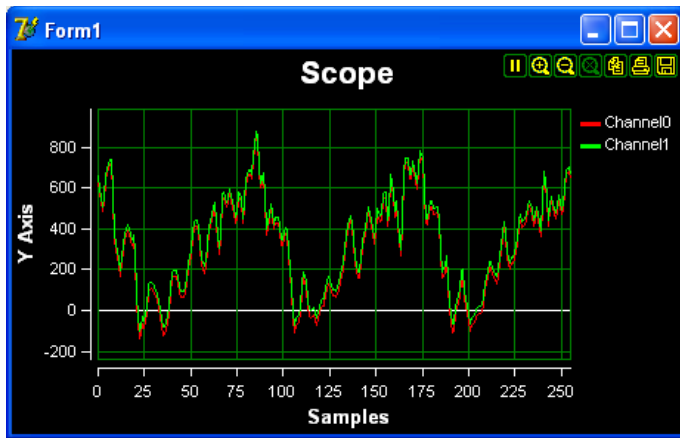


In the Pin Editor select the InputPins.Channel1 of the SLScope1 and click OK:



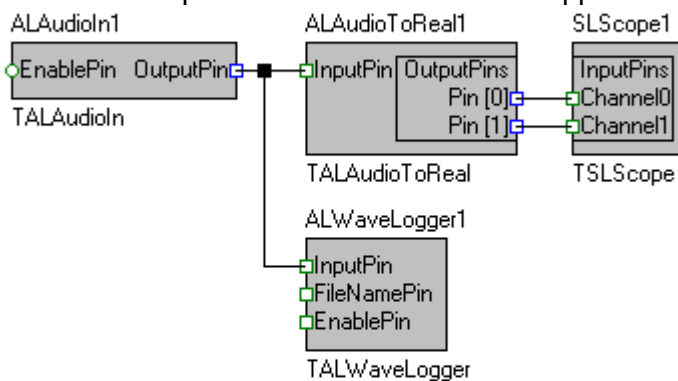
Compile and run the application.

You should see result similar to this one:



A file named RecordedAudio.wav will be created containing the recorded audio.

Here are the OpenWire connections in this application:



You have just learned how to create audio recorder with AudioLab.

## 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 (through 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 element 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 some AudioLab 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 AudioLab was installed).

On 32 bit windows systems, you can distribute them to the [Windows]\System32 directory, or to the distribution directory of your application( [Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS ).

On 64 bit windows systems, you can distribute them to the [Windows]\SysWOW64 directory, or to the distribution directory of your application( [Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS ).

Not all of the components in the library require additional DLLs. Please check if the DLLs are needed by the application before including them in the install.

## 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.