

Instrumentation 4.5

Quick Start



www.openwire.org
www.mitov.com

Copyright Boian Mitov 2004 - 2010

Index

Installation	2
C++ Builder installation	2
Set the default search path manually in C++ Builder 5 and 6.....	2
Force C++ Builder 5 and 6 to create the default.bpr file	5
Set the default search path manually in C++ Builder 2006 - 2010	6
Force C++ Builder 2006 - 2010 to create the default.bpr file.....	10
Where is InstrumentLab?.....	11
Using the TSLCRealBuffer in C++ Builder and Visual C++	12
Distributing your application	15

Installation

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

C++ Builder installation

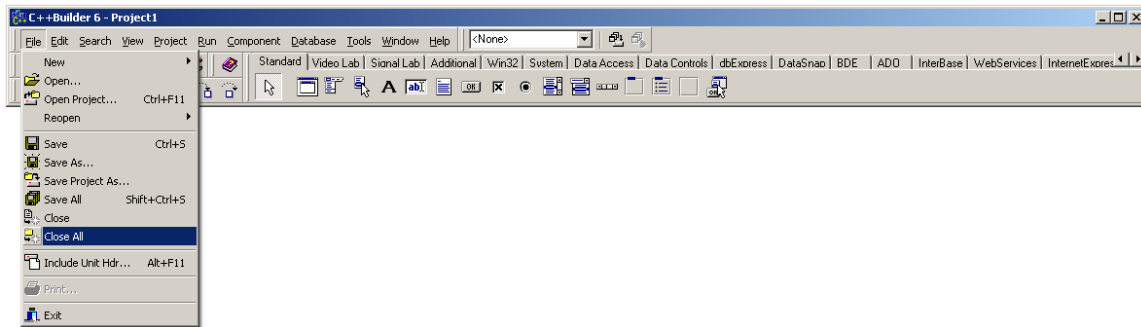
C++ Builder uses a special file named default.bpr to store the default include path for your projects. The file is located into the C++ Builder's bin subdirectory. The file is created after the first time the C++ Builder user saves default project settings. The installation is designed to modify the file in order the new projects to be able to find the necessary include files. If the file does not exist, the installation will show a warning message.

You can set up the default search paths yourself, or you can force C++ Builder to create the default.bpr file, and repeat the SignalLab installation.

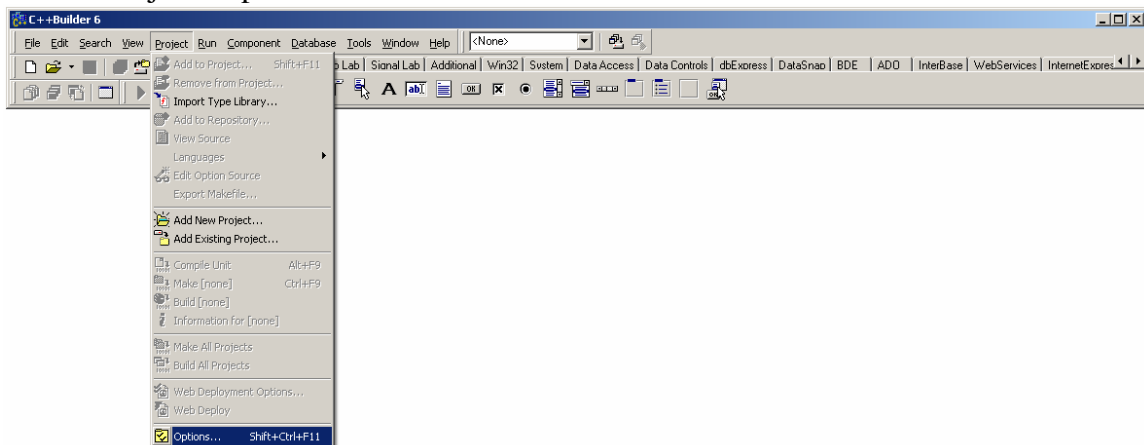
Set the default search path manually in C++ Builder 5 and 6

Start C++ Builder.

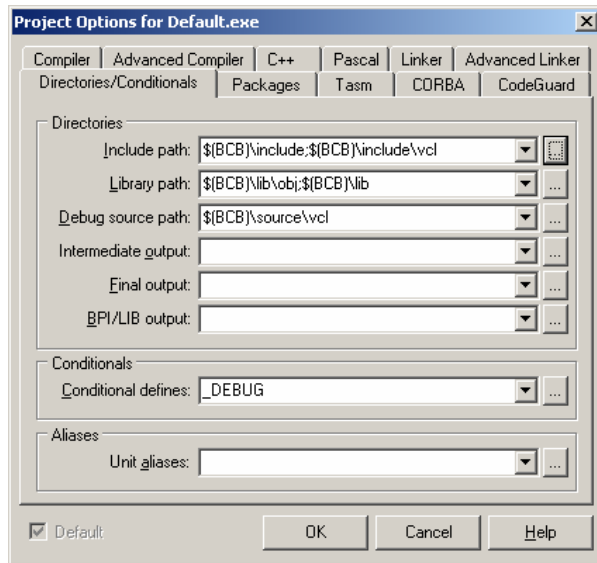
From the menu select | File | Close All |




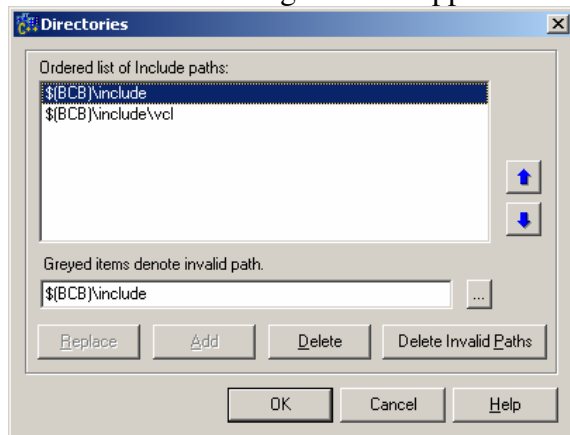
Select | Project | Options... |



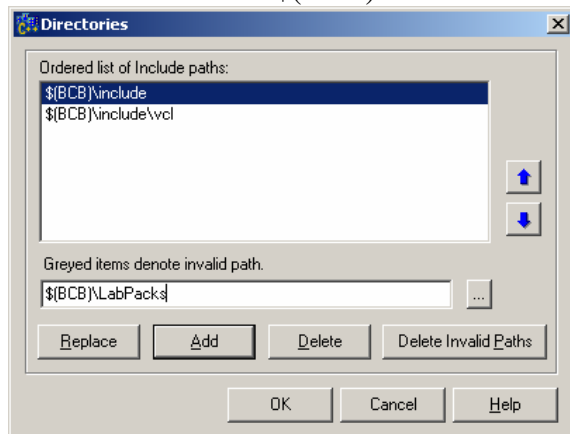
Switch to the | Directories/Conditions | tab:



click on the  button next to the “Include path” text field. The Directories dialog box will appear:

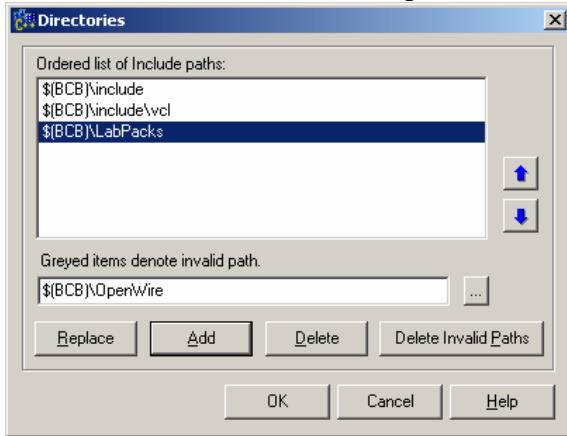


In the text box enter `$(BCB)\LabPacks`:

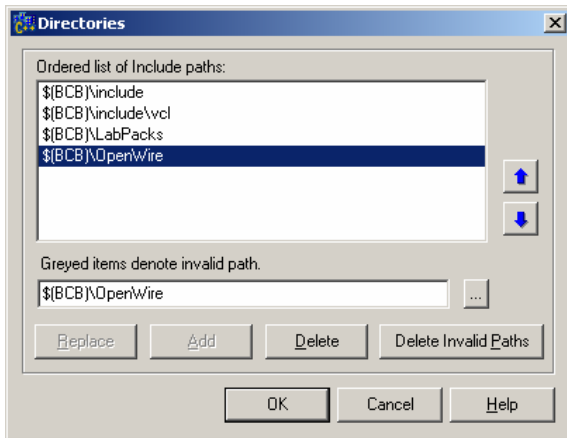


Click the Add button.

In the text box enter \$(BCB)\OpenWire:

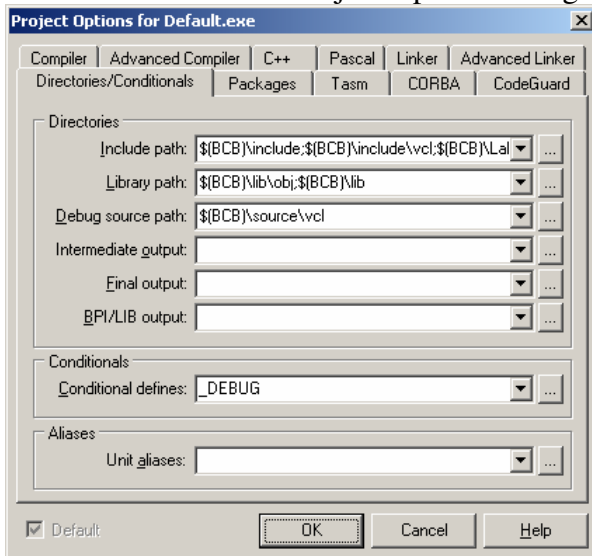


Click the Add button.



Click on the OK button.

You will return to the Project Options dialog:

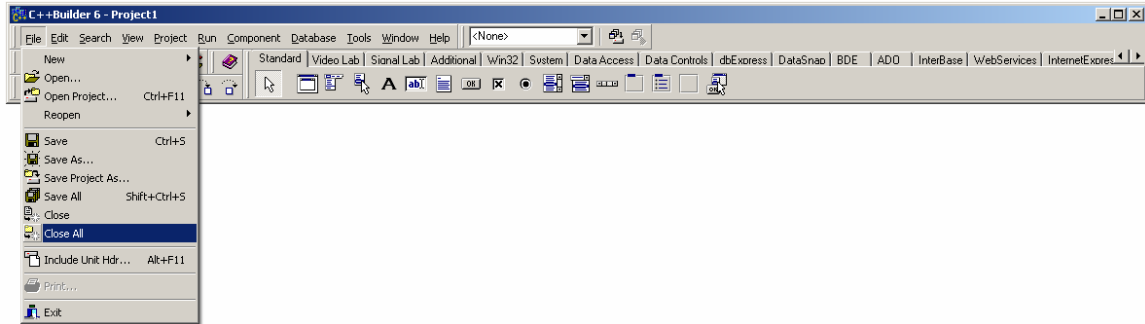


Click on the OK button.

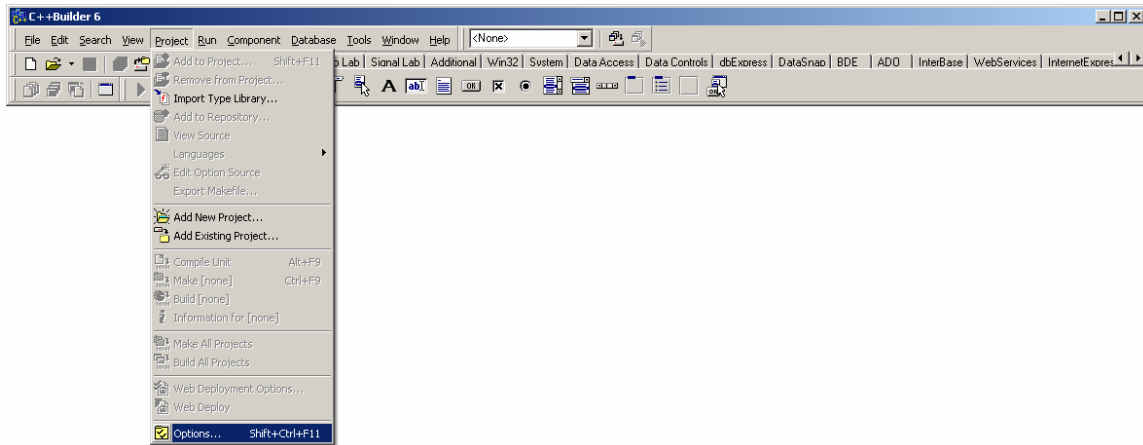
Force C++ Builder 5 and 6 to create the default.bpr file

Start C++ Builder.

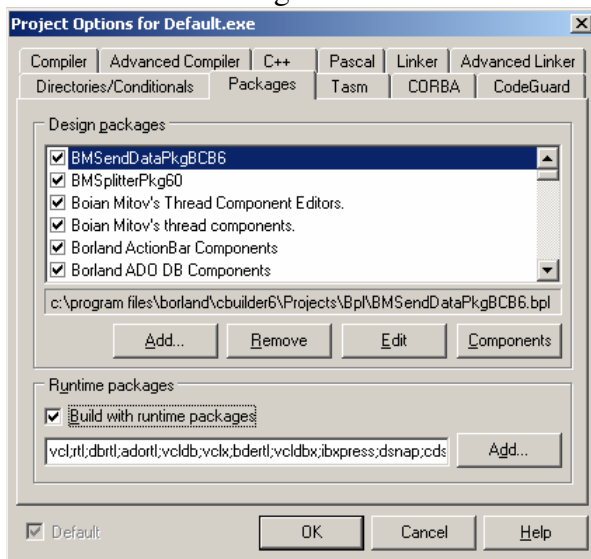
From the menu select | File | Close All |



Select | Project | Options... |



Switch to the | Packages | tab:



Click on the **Build with runtime packages** check box to change the status, and then click OK. Then select | Project | Options... | again.

Click on the **Build with runtime packages** check box to change the status again, and then click OK.

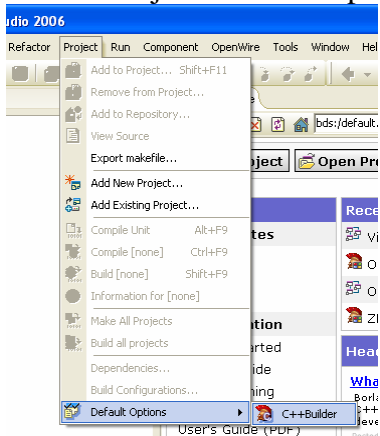
This will create a default.bpr file.

Now you can repeat the installation of SignalLab and the default search paths will be configured properly.

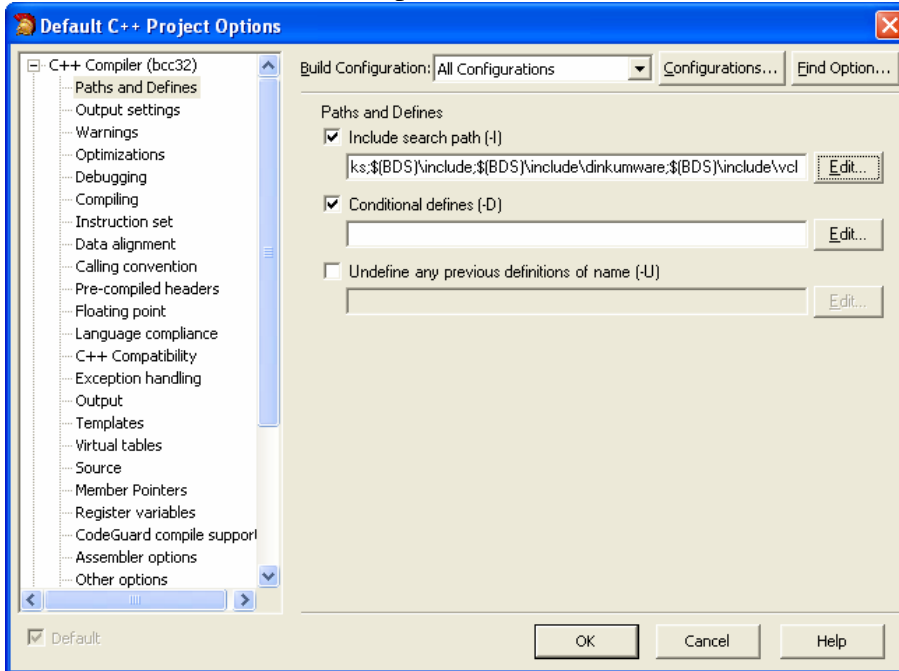
Set the default search path manually in C++ Builder 2006 - 2010

Start C++ Builder 2006 - 2010.

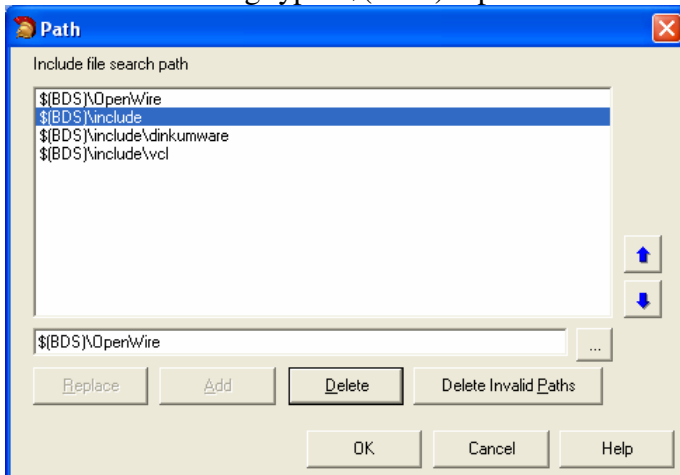
Select | Project | Default Options... |



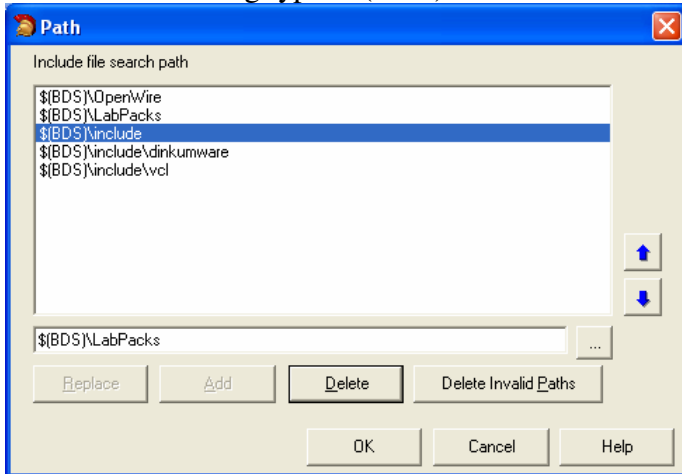
Select the | C++ Compiler (bcc32) | Paths and Defines | options and click on the “Edit...” button for the “Include search path (-I)”:



In the “Path” dialog type “\$(BDS)\OpenWire” and click the Add button:

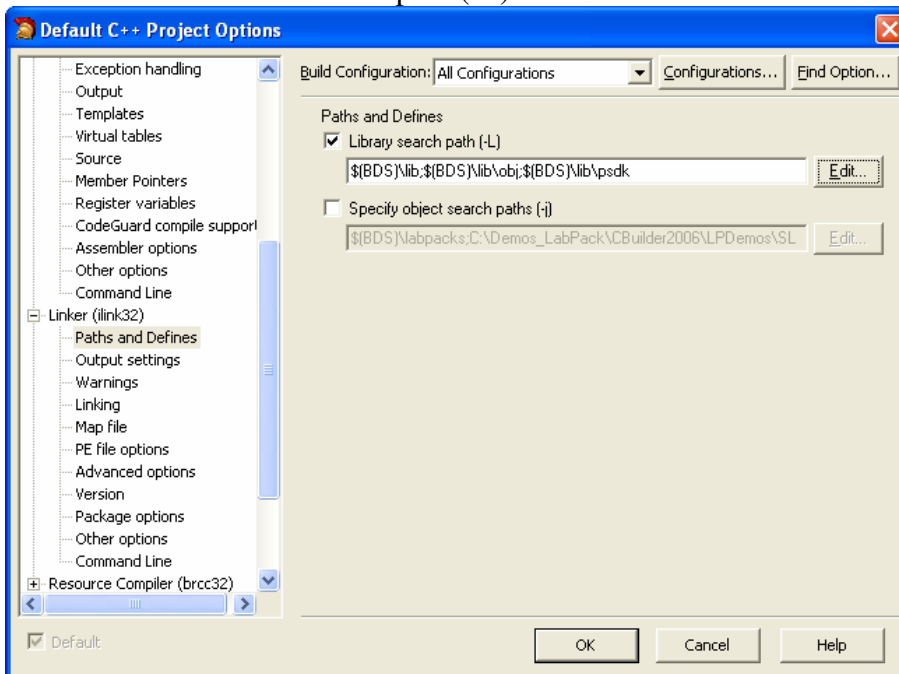


In the “Path” dialog type “\$(BDS)\LabPacks” and click the Add button:

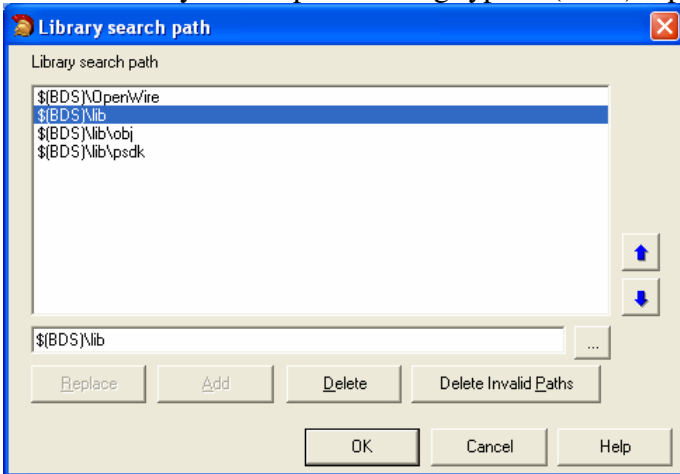


Click OK.

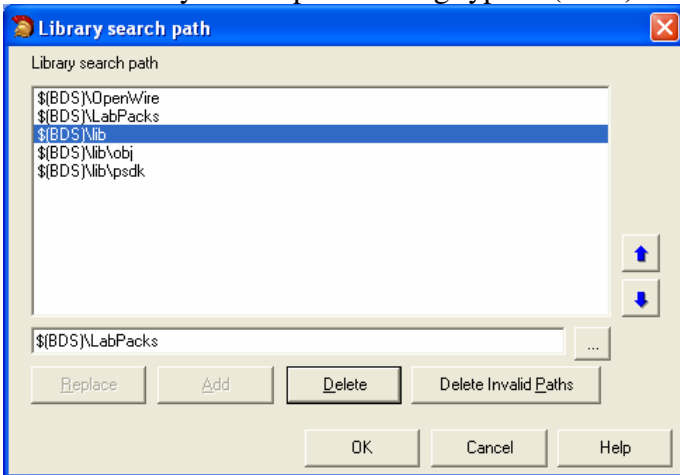
Select the | C++ Compiler (bcc32) | Paths and Defines | options and click on the “Edit...” button for the “Include search path (-L)”:



In the “Library search path” dialog type “\$(BDS)\OpenWire” and click the Add button:

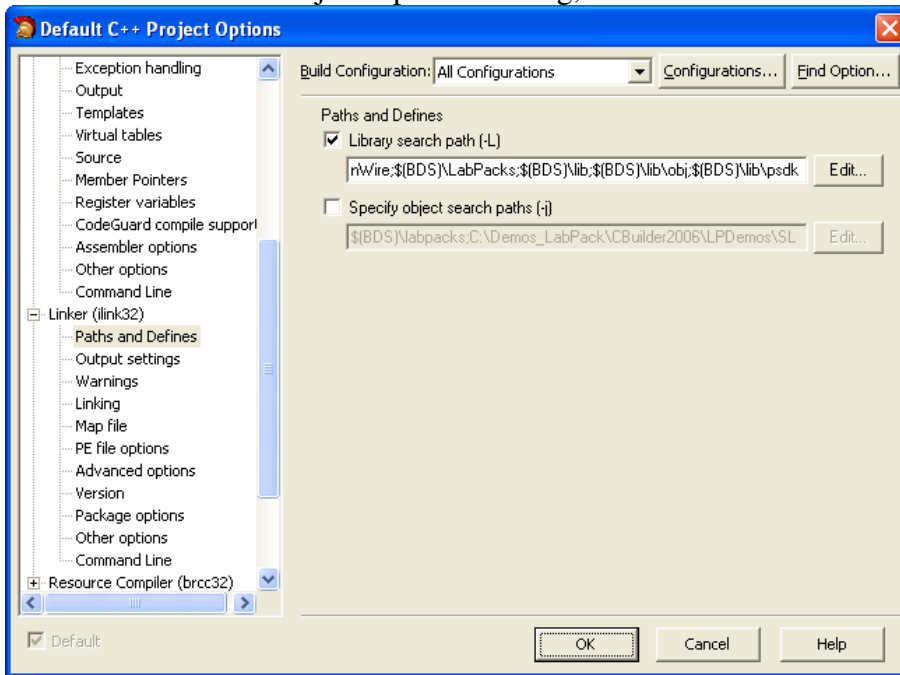


In the “Library search path” dialog type “\$(BDS)\LabPacks” and click the Add button:



Click OK.

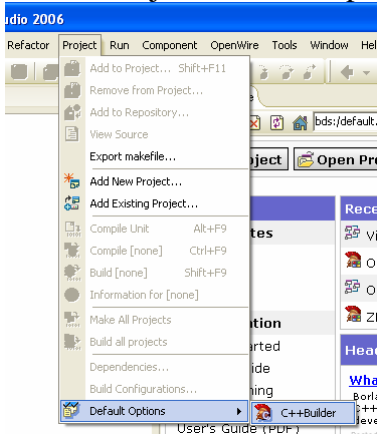
In the “Default C++ Project Options” dialog, click OK:



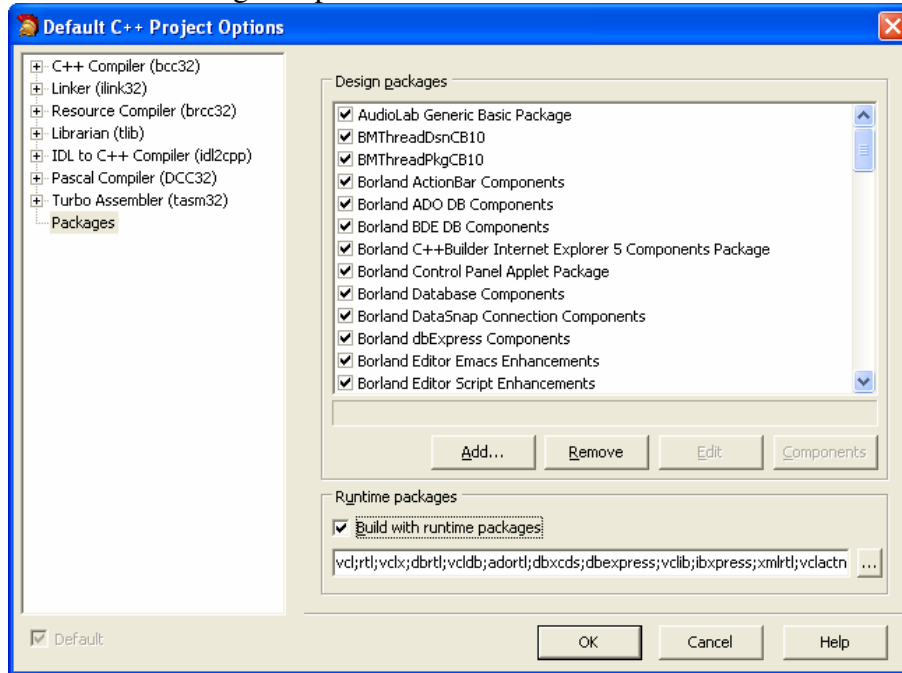
Force C++ Builder 2006 - 2010 to create the default.bpr file

Start C++ Builder 2006 - 2010.

Select | Project | Default Options... |



Select the | Packages | options:



Click on the Build with runtime packages check box to change the status, and then click OK. Then select | Project | Options... | again.

Click on the Build with runtime packages check box to change the status again, and then click OK.

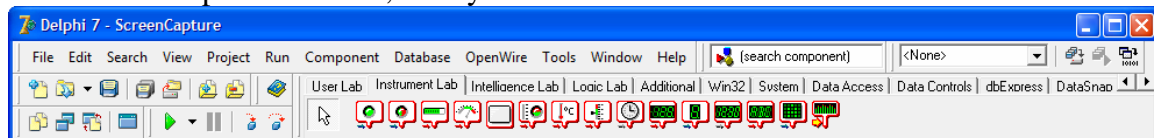
This will create a default.bpr file.

Now you can repeat the installation of SignalLab and the default search paths will be configured properly.

Where is InstrumentLab?

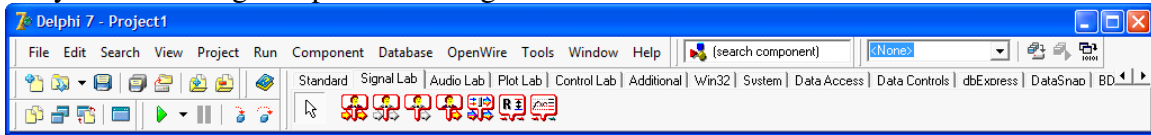
After the installation, start your Delphi or C++ Builder.

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

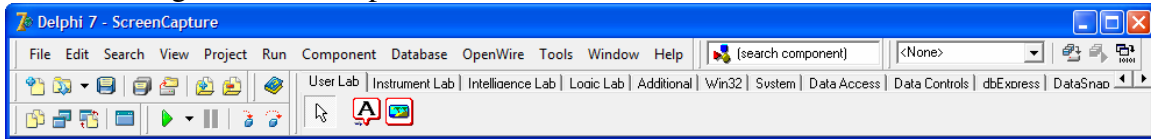


If the installation was successful, they should be named “Signal Lab”, “UserLab”, and “InstrumentLab”. On the SignalLab palette you will have only a subset of the SignalLab components. SignalLab is a separated product, and will not be shipped as full with InstrumentLab.

Only the following components of SignalLab will be available:



The following UserLab components will also be available:



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 InstrumentLab 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 InstrumentLab was installed). 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). 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.