plug & play instruments
oscilloscopes

Cleverscope Ltd
Phone +64 9 524 7456
Fax +64 9 524 7457
Email support@cleverscope.com
28 Ranfurly Rd, Epsom
P.O. Box 26-527
Auckland 1003
New Zealand

V1.2
15 Sep 08

Simple Scope application example

Using the Cleverscope DLL with C or Delphi

Summary

This application example uses the NI Labwindows C code environment to produce a working executable to make a simple Oscilloscope application with Oscilloscope and Signal generator controls. Further examples using NI Labview, Borland Delphi and C++ Builder, and Microsoft Visual Studio C++ and C# are also provided. All of the examples use the same structures, variable names and program flow.

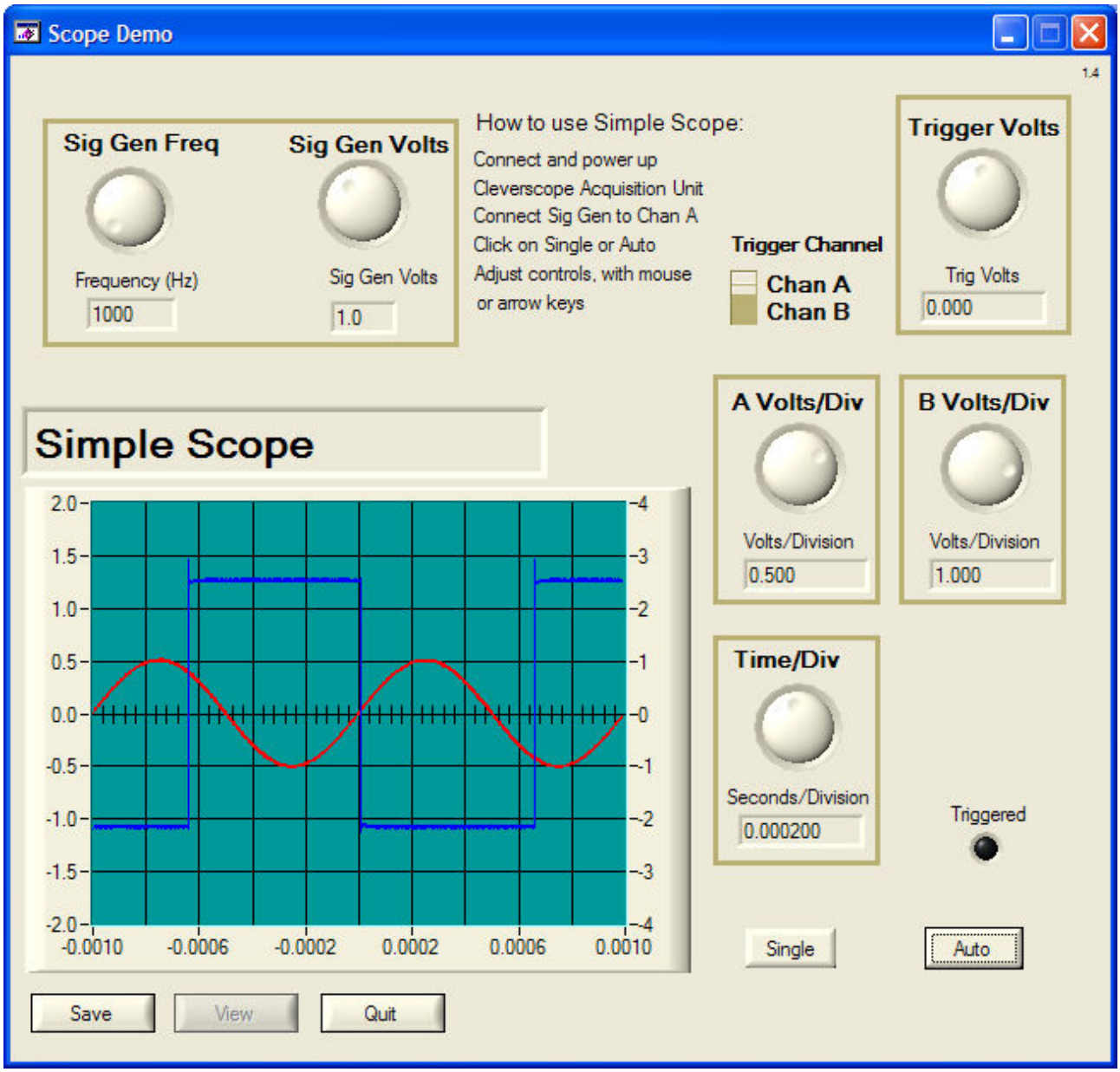
The application may be used as an example of interfacing to the DLL using C code.

Important Information

1. To use the application you will have to have the National InstrumentsLV7.1 Run Time library installed. This happens automatically when you install Cleverscope, so make sure Cleverscope has been installed on the target machine. We can provide an installer if requested.
2. The DLL uses the STD CALL method of parameter passing. Ensure your environment is setup this way.
3. Connect a signal source to the Chan A input (such as the Sig Gen Output) so the application has something to trigger off.

The Simple Scope application

The SimpleScope.exe demonstration application has the following front panel:

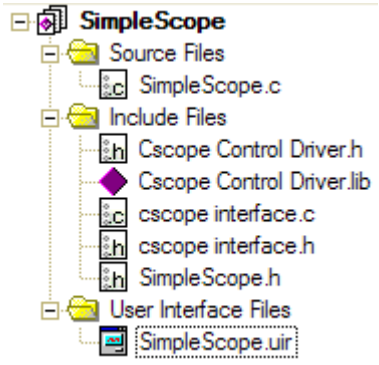


You can see there are controls to set the time base, the A and B channel gains, the trigger channel and amplitude, and the signal generator frequency and amplitude. As such it represents a simple example for use of the major features of the Cleverscope Acquisition Unit.

Note that the application connects immediately when you start it – so connect a signal to the acquisition unit, and have it on, and connected before running the application.

Application Structure

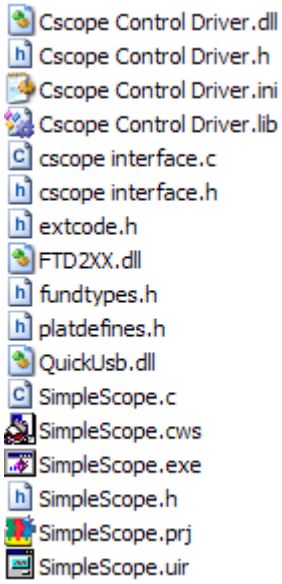
The application structure is shown to the right.



To debug the application you will need Labwindows CVI – the National Instruments C development environment.

The application **SimpleScope.c** communicates with the Cleverscope interface file **cscope interface.c** using the **cscope interface.h** header file. **Cscope interface.c** communicates with the acquisition unit hardware via USB using the **cscope control driver.dll** file, which is accessed using the **cscope control driver.h** header file. You will also need the **ftd2xx.dll** and quickusb.dll files for USB support, and the NI Visa I/O system, if you have a later version of Cleverscope. For other environments you may need other header files or the cscope.lib file to build an application.

The full directory of files is shown here:



- Cscope Control driver.dll – contains the executable code to control the Cleverscope Acquisition Unit
- Cscope Control driver.h – contains a header file with the C code definitions required to call the DLL. The definitions are based on the STD CALL calling protocol.
- Cscope Control Driver.ini – this file is not used, but present for future possibilities.
- Cscope Control driver.lib – a linker library file used by Labwindows CVI
- Cscope interface.c – the C code that is used to talk to the DLL in the example application
- Cscope Interface.h – a header file that allows the main application to call functions in **cscope interface.c**.
- Extcode.h, fundtypes.h, platdefines.h –header files required to define types used by the Labwindows system if you are using an alternative compiler.
- FTD2XX.dll – a DLL called by cscope control driver.dll to communicate via the USB 1.1 link with the Cleverscope Acquisition unit.
- QuickUSB.dll – a DLL called by cscope control driver.dll to communicate via the USB 2.0 link with the Cleverscope Acquisition unit. Supplied for backward compatability. No longer used. We now use the NI-Visa system for USB and Ethernet communications.
- SimpleScope.c – the example application C code source.
- SimpleScope.cws, SimpleScope.prj –Labwindows project definition and build files.
- SimpleScope.exe – the executable generated by the compiler. It runs the examples system.
- SimpleScope.h – an automatically generated header file that allows SimpleScope.c to receive events and call backs from the user interface.
- SimpleScope.uir – the Labwindows user interface definition.

Key Functions in cscope interface

These are the key functions:

```
int scope_init (void);
```
This function initialises the acquisition unit. It returns 0 if no error, a positive integer otherwise.

```
int scope_close (void);
```
This function closes the acquisition unit. It returns 0 if no error, a positive integer otherwise.

```
int scope_config (double a_div, double b_div, double t_div, int
number_of_points, double freq, double sigvolts, double trigvolts, int
trig_chan);
```
This function configures the acquisition unit. It returns 0 if no error, a positive integer otherwise.

```
int   scope_acquire (void);
```
This function starts an acquisition based on the acquire variable setup. It returns 0 if no error, a positive integer otherwise.

```
int scope_read_waveform (double a_waveform[max_samples], double
b_waveform[max_samples], int *num_samples,  double *delta_t, double *t_zero);
```
This function returns the last transferred waveforms for Channels A and B, and sampling details. It returns 0 if no error, a positive integer otherwise.

```
int check_for_samples(void);
```
This function returns 1 if a trigger has occurred and samples have been returned, otherwise 0.

Key variables in scope interface.c

The **acquire** variable holds the current acquisition configuration. The function **cscope config** simply manipulates this variable.

Here is the default setup for acquire:

```
acquire.AcquireMode = 3;                //stopped
acquire.AcquisitionMode = 1;            //peak captured
acquire.Acquirer =  4;                  //cleverscope is the acquirer
acquire.TransferChans = 2;              //transfer both channels
acquire.AMaxScale = 2;                  // Volts range = +/-2
acquire.AMinScale = -2;
acquire.BMaxScale = 2;
acquire.BMinScale = -2;
acquire.AProbe = 0;                     //x1
acquire.BProbe = 0;                     //x1
acquire.ACoupling = 1;                  //DC
acquire.BCoupling = 1;                  //DC
acquire.ABandwidth = 1;                 //100 MHz
acquire.BBandwidth = 1;                 //100 MHz
acquire.TriggerSource = 0;              //A Chan trigger
acquire.TriggerAmplitude = 0;           //Trigger at zero volts
acquire.ATriggerAmplitude = 0;
acquire.BTriggerAmplitude = 0;
acquire.TriggerFilter = 0;              //No trigger filter
acquire.TrigSlope = 0;                  //rising
acquire.TriggerHoldoff = 0;
acquire.DigPatternRqd = 0;              //not used
acquire.DigPattern =  0;                //not used
acquire.ExtTrigThreshold = 0;
acquire.DigInputThreshold = 2;
acquire.StartTime =  -0.005;            //-5 msecs
acquire.StopTime =   0.005;             //5 msecs
acquire.PreTrigTime = 0.005;
acquire.Port = 0;
acquire.NumDivisions = 10;
acquire.NumSeqFrames = 1;
acquire.NumBuffers = 2;
acquire.SigGenFreq = 1000;              //1kHz output
acquire.SigGenAmp = 1;                  //1V amplitude
acquire.SigGenOffset = 0;
acquire.SigGenWaveform = 0;             //sine
acquire.SigGenSweep = 0;
acquire.SigGenFunc = 0;
acquire.SigGenFreq2 = 0;
acquire.SigGenPhase = 0;
acquire.Trig2Function = 0;              //not used
acquire.MinTriggerPeriod = 0.0000001;
acquire.MaxTriggerPeriod = 1;
acquire.TriggerCount = 1;
acquire.Trig2Slope = 0;
acquire.Trig2SourceChan = 0;
acquire.Trig2Level = 0;
acquire.DigPattern2Rqd = 0;
acquire.DigPattern2 = 0;
acquire.Trigger2Source = 0;
acquire.WaveformAverages = 1;
acquire.ValueChanged = 1;
samples_required = 1000;
SamplerResolution = 0;                  //10 bit sampler
```

See the Cscope driver DLL documentation for the full definition of the acquire variable.

Simple Scope Operation

Simple scope operation is determined by SimpleScope.c (or equivalents, or the Form code in other languages). Steps are:

- **Main**, start system. Call **scope_init**. Then call InitializeGUI, below. When GUI finishes, call **scope_close**.

- On opening the form, **InitializeGUI** is called to set each control to the correct value. Key operating values are initialized:
  a. single_acquire - Boolean, means acquire a single shot with triggered capture
  b. auto_acquire – Boolean, means acquire multiple shots with auto capture
  c. waiting_for_trigger – Boolean, if true, waiting for trigger. Otherwise can start acquire.
  d. vals_changed – Boolean, if true, a value has changed, do update or acquire;
  e. trigger_action – how to capture – single (triggered), auto or stop.

- On a user control event: Call the event handler. Each event handler deals with a separate control value. When the value is changed, vals_changed is set true.
  Three events are important:
  a. **SingleAcquire** – start a single capture with trigger. Set single_acquire. Set the trigger_action to either single (if we haven't been waiting for a trigger) or stop (if waiting for a trigger).
  b. **TriggeredAcquire** – start a continuous auto capture. Set auto_acquire. Set the trigger_action to either auto (if we haven't been capturing) or stop (if capturing).
  c. **QuitButton** – or **OnFormClose** – these events cause the SimpeScope program to exit.

- **Event Timer Tick** – this is the central control function in Simple Scope. It fires every 50 or 60 msecs (some environments only support 15msec granularity). Anything less than 50 msec is not effective.

  This function proceeds in 4 stages:
  a. If vals_changed is true, we update the acquire variable using **update_values**. This does not update the acquisition unit itself.
  b. If we are not waiting for a trigger, and either single_acquire or auto_acquire are true, we start an acquisition, using **cscope_acquire**. This function first sends the acquire variable changes to the acquisition unit, and then starts an acquisition.
  c. If we are waiting for a trigger, and **got_samples** is true, we get the samples, and graph them. Note that got_samples will wait up to 40msecs for a trigger. If there is no trigger, got_samples returns after 40msecs, and returns false. While waiting, got_samples sets the thread to sleep, and returns control to the operating system.
  d. If vals_changed is still true (implying that we did not get a trigger, or the user made a control change while acquisition was not complete, or there is no acquisition in progress), we use **update_values** again to update the acquire variable, and then use **scope_config** to update the acquisition unit directly.