CompuScope Software Development Kit (SDK) for C/C# for Windows

User's Guide

Revision 2.03 – 3/18/2021

Gage Contact Information	
Toll Free:	1-800-567-GAGE
Tel:	1-514-633-7447
Fax:	1-514-633-0770
Sales Email:	sales@gage-applied.com
Support Email:	tech-support@gage-applied.com
Web:	http://www.gage-applied.com

Gage is a Product Brand of:

DynamicSignals LLC
900 North State Street
Lockport, Illinois 60441-2200
USA

Toll Free: 1-800-DATA-NOW
Tel: 1-815-838-005
Fax: 1-815-838-4424

http://www.dynamicsignals.com

Tel: 1-800-567-GAGE or +1-514-633-7447. Fax: 1-800-780-8411 or +1-514-633-0770

How to reach Gage Product Support
Toll-free phone: (800) 567-4243
Toll-free fax: (800) 780-8411

To reach Gage Product Support from outside North America
Tel: +1-514-633-7447
Fax: +1-514-633-0770

E-mail: prodinfo@gage-applied.com
Web site: http://www.gage-applied.com
On-line Support Request Form: http://www.gage-applied.com/support/support-form.php

Table of Contents

Preface

This manual is meant to serve as an aid to engineers using the CompuScope series of high-speed data acquisition cards in the C or C# programming language from within the Microsoft Windows environment.

Throughout this manual, it is assumed that you are familiar with the C or C# programming environment. It is also assumed that you have correctly installed and configured the CompuScope Windows drivers. It is also assumed that you are familiar with the PC and Microsoft Windows.

The table below lists which sample programs are supported for each programming language. Some sample programs for VisualBasic.NET and Delphi are provided within the C/C# SDK. The user may use the C Sample programs as a guide to construct sample programs that are unavailable for VB.NET and Delphi. Usage of some advanced sample programs requires installation of optional eXpert firmware. These programs are indicated by an asterisk (*).

To determine which CompuScope models are supported by which advanced sample programs, please check the online Advanced Functionality Matrix at:
http://www.gage-applied.com/expert-fpga-dsp/index.htm

	C	C#	VB .Net	LabVIEW	MATLAB	CVI	Delphi
GageAcquire	X	X	X	X	X	X	X
GageComplexTrigger	X	X		X	X	X	
GageDeepAcquisition	X	X		X	X	X	
GageMultipleRecord	X	X		X	X	X	
GageMulrecFastTransfer	X	X		X	X	X	
GageMultipleSystems	X	X		X	X	X	
Streaming Sample Programs*	X						
GageSignalAveraging*	X	X		X	X	X	
GageCsPrf	X	X				X	
GageFastAcquire2Disk	X	X		X	X		
GageCallback	X					X	
GageEvents	X	X				X	
GageAsTransfer	X	X				X	

Installation of CompuScope C/C# SDK

If you purchased the CompuScope C/C# SDK you will have been shipped a software key that allows installation of the SDK from the Gage CompuScope CD. Simply select the installation of the CompuScope C/C# SDK from the CompuScope CD and enter the key when prompted.

By default, the CompuScope C/C# SDK will install itself in the O/S system drive under:

\Program Files\Gage\CompuScope\CompuScope C_C# SDK.

It is recommended that you use the default installation location.

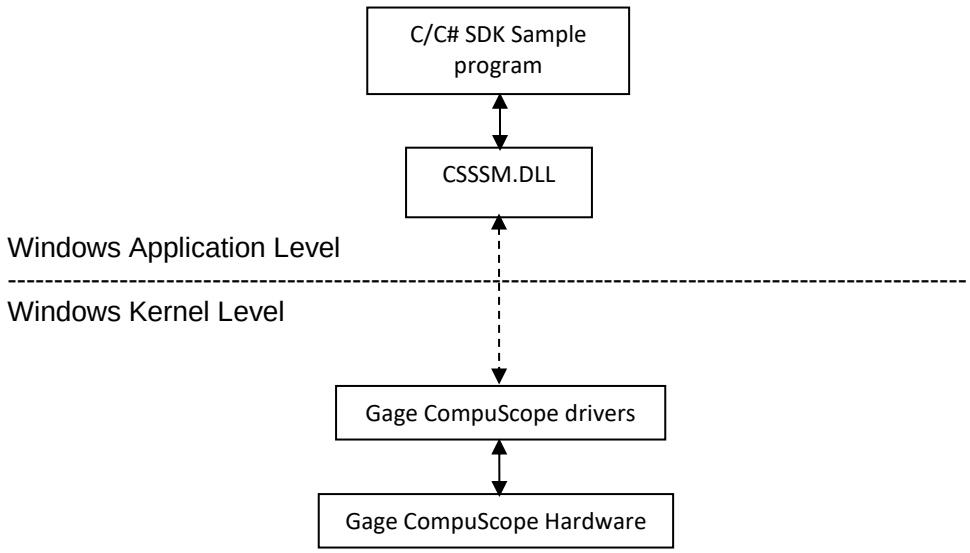
More detailed installation instructions are available in the Gage CompuScope Getting Started, which was shipped with your order. The CompuScope C/C# SDK provides both 32-bit and 64-bit LIB files, so it may be used to build full 64-bit Windows applications from C, C# or VB.NET.

During the installation of the driver, a Windows environment variable called "GageDir" is created. This variable points to a folder that contains CompuScope driver "common files", which are required for compilation of the C/C# SDK sample programs. If this folder is moved without updating the GageDir environment variable, then this will result in an inability to compile sample programs. Do not copy these files into the sample program folder (which was recommended for earlier SDKs), since doing this will cause problems if newer drivers (and their associated newer common files) are installed.

Overview of CompuScope C/C# SDK

Structure of CompuScope C/C# SDK

The overall structure of the CompuScope C/C# SDK and its relation to the Gage CompuScope hardware is best described with reference to the diagram below.



At the lowest level is the CompuScope hardware, which is installed within a slot connected to the host PC's PCIe or PXIe bus. The CompuScope hardware is directly controlled by the CompuScope Windows drivers. The drivers reside at the Windows Kernel level, which allows direct low-level access to CompuScope hardware registers and to physical PC RAM. The drivers communicate with Windows Applications through a Dynamically Linked Library (DLL) called CSSSM.DLL. (Intermediate components between CSSSM.DLL and the drivers have been omitted for clarity).

Communication through CSSSM.DLL is conducted using the CompuScope Applications Programming Interface (API). The CompuScope API is a set of C subroutine calls or *methods* that allows control of all CompuScope functionality. Communication from a C/C# SDK sample program is made directly with CSSSM.DLL through the API with no intermediate software layers. This explains why the best possible CompuScope hardware performance is achievable using the C/C# SDK.

CompuScope C/C# SDK Compiler Requirements

Project files for all C/C# SDK sample programs are provided for Microsoft Visual C/C++ version 2008 or higher. Upon opening the project file for any C sample program, all supporting files, path specifications and build settings are automatically loaded. Compiled 64-bit and 32-bit executable versions of all C and C# sample programs are provided with the SDK for convenience and debugging.

The C sample programs may be operated under compilers other than Visual C/C++. However, complete project files must be assembled by the user. There may be other requirements for operating the C Sample programs under other compilers. For instance, in order to use Borland C Builder, the Gage CSSSM.LIB file must be imported using the IMPLIB procedure that is described in Borland documentation. Please check your compiler manual for other possible requirements on converting Visual C/C++ projects.

C# sample programs with identical functionality to the C sample programs are provided for operating CompuScope hardware in the .NET environment. .NET version 2.0 or newer is required in order to compile and run all C# sample programs.

CompuScope Systems

A CompuScope system is defined as a single CompuScope board or a group of CompuScope boards configured as a Master/Slave multi-card CompuScope system. Since Master/Slave CompuScope systems sample, trigger and reset simultaneously on all channels, it is considered to be one multi-channel CompuScope system. For instance, a Master/Slave system composed of four two-channel CompuScope boards will be considered to be a single CompuScope system with eight available input channels.

By structuring the drivers to consider CompuScope systems, a single PC can be equipped with almost any imaginable combination of CompuScope hardware. For instance, a PC could be equipped with two separate Master/Slave systems of four channels each and then an additional single independent CompuScope board for a total of three CompuScope systems. The CompuScope Manager utility may be used to separately list all CompuScope systems in the PC.

CompuScope systems are addressed from C or C# sample programs by first obtaining a *Handle* for the System. After usage of the system is complete, the user must release the Handle so that it is free for usage by other processes. By obtaining handles for different systems, a single C or C# program may simultaneously operate different CompuScope systems. Alternately, separate C or C# programs may operate independently and simultaneously operate separate CompuScope systems by calling handles for each one. Different programs may even access the same hardware as long as one program frees the system handle before the other program obtains it. This is because different applications may not simultaneously access the same CompuScope system.

While most C or C# sample programs access only a single system, understanding the CompuScope system structure allows users to easily extend these sample programs to multiple CompuScope system operation.

SDK Folder Structure and Content

Within the main C/C# SDK folder are several sub-folders that are listed below:

Executables: This folder contains compiled version of all sample programs. INI files are also included that determine the configuration settings that the sample programs will use. These executable files can be executed with no compilation required.

C Samples: This folder contains all C sample programs. Complete projects for each sample program are contained within a folder that has the same name as the sample program.

C# Samples: This folder contains all C# sample programs. Complete projects for each sample program are contained within a folder that has the same name as the sample program.

C Common: This folder contains C common files that are required for compilation for all sample programs.

Distinct from C common files, *Driver common files* are installed by the CompuScope driver installation in a folder specified by the compiler environment variable called "*GageDir*". Driver common files are also required for sample program compilation. All sample program project files are configured to refer to both the C common files and to the driver common files.

Overview of C and C# Sample Programs

The C and C# sample programs are intended to be convenient starting points around which users can develop their own customized CompuScope software application. They are not intended to be used as complex stand-alone applications with sophisticated analysis functionality or graphical user interfaces.

The basic algorithm for all of the C and C# sample programs is the following:

1. Initialize the CompuScope driver and the CompuScope hardware.
2. Obtain the handle to the first CompuScope system in the PC.
3. Obtain the desired configuration settings from the local Gage INI file.
4. Pass the desired configuration settings to the driver.
5. Commit the configured settings by transferring them from the driver to the CompuScope hardware.
6. Start the CompuScope hardware to digitize data into its on-board acquisition memory and await a trigger event.
7. Continuously check to see if the CompuScope hardware has finished the acquisition.
8. Download the data of interest from the CompuScope hardware to a PC RAM array variable.
9. Store the acquired data to a file according to the settings and file format provided in the local INI file.

Not all possible CompuScope operation configurations are covered by the sample programs. For instance, there is a program that handles deep acquisitions and a program that handles acquisitions from multiple CompuScope systems. However, there is no sample program that handles deep acquisitions from multiple CompuScope systems. In order to use both of these functionalities, the user must study GageDeepAcquisition and GageMultipleSystem and then intelligently combine them to create a program that meets their requirements.

All CompuScope sample programs and a brief description of their functionalities will be found later in the section C++/C# Sample Programs

Configuration Setting INI files

All C and C# sample programs read Windows INI files in order to obtain CompuScope configuration settings that are used for the program's acquisitions. The format of the INI files is completely described in the section INI File Description in this document. A support function is provided that reads INI files, parses the input setting values and loads the values into the appropriate internal structure variables.

INI files allow configuration settings to be adjusted without altering or re-compiling the sample program. Users need not employ INI files but may modify the sample programs so that configuration settings are obtained elsewhere, for instance from a graphical user interface. INI files may be used to test the CompuScope driver and SDK functionality.

The INI files provide an easy if not optimal way to program CompuScope hardware from a non-supported programming environment, like Visual Basic. Users may simply create a Visual Basic program that creates the required INI file, do a system call to the appropriate executable sample program and then read resultant ASCII files created by the sample program. For better performance, a user may call CompuScope API functions directly from the non-supported programming environment.

Output .TXT Data Files

When configuring from INI file to save data as ASCII files, all C and C# sample programs acquire data from CompuScope hardware then store the results in ASCII data files with file names of the form XXX_N.TXT, where N is the channel number of the CompuScope system. These files may easily be imported into database programs like Microsoft Excel for analysis and display.

The beginning of the TXT file is a header containing information about the acquisition that is enclosed by two lines of minus signs ("----------"). Programs that read TXT files must recognize at least three successive minus signs in order to parse a delimiter line. The minus sign line allows easy removal of the file header during file reading. We may add more information to the TXT file header in future but the entire header will always be enclosed between the minus sign delimiter lines.

After the header, all TXT files contain a single column of ASCII data, where each entry represents a single data sample. The sample format may be selected as voltage values, where the calibration is applied in order to convert raw ADC data into voltage values. For faster throughput, the user may instead elect to store raw ADC code values in either decimal or hexadecimal format. Raw ADC data storage is recommended for customers who are not concerned with the absolute amplitude of their signal but only with the signal shape and so may forgo the voltage conversion step. Also, raw ADC values will occupy less space on the hard disk as they occupy 1 to 4 Bytes per sample and the ASCII line can occupies over 20 Bytes.

Beside the TXT file format, users can also save data in other formats: SIG, BIN … by changing the setting in the INI file. By saving of data as Gage SIG files, users can view and analyzed it later in GageScope

Users may also easily modify the sample programs for data analysis or display, rather than storage, by simply locating the file storage code and replacing it with alternate code.

C/C# SDK Sample Programs

Overview of CompuScope API

All C and C# sample programs control CompuScope hardware using the CompuScope Applications Programming Interface (API), which is a small set of very powerful C subroutines or *methods*. The API is completely described in the CompuScope API Reference Manual, CsAPI.chm, which is installed with the SDK. All CompuScope API methods have names of the form CsXxxx().

Each subroutine API call or method may be called with up to three distinct types of parameters. The first parameter type is the *handle* to the CompuScope system, which is the first parameter of almost all API methods. The CsGetSystem() method is used to search for and obtain the handle of an arbitrary or specific CompuScope system. The handle identifies the CompuScope system that is to be used by other methods.

The second type of method parameter is the CompuScope structure variable. Each structure variable has its own type definition that is contained within the C common file CsStructs.h and is listed in the CompuScope API Reference Manual. When calling API methods, all sample programs measure the size of the type definition in Bytes and pass this value within the structure itself. The size of the structure may increase in future driver versions. The driver, however, checks the size embedded in the structure and only returns this number of Bytes, rather than passing back an increased number that would lead it writing past the buffer's allocated size and causing an error. This way, newer drivers may be installed and operated using older programs with no recompilation required.

The type definition for each structure type that is documented within the CompuScope API Reference Manual lists all variables within the structure, their type, and a short description of the variable and its functionality. Assignments to variables within the structures may be to straight numerical values or to pre-defined constants that are all listed within the CompuScope API Reference Manual.

The third type of method parameter is a simple control parameter, which must be assigned to a CompuScope API pre-defined constant. All constants are listed in the API Reference Manual. All constants have capitalized names. Users must

always use constant names rather than their numerical values, since we reserve the right to change constant values in future.

The CompuScope API provides an API method called CsGetSystemCaps(), which obtains the capabilities of a CompuScope system. This method is provided for users who must write an application that supports an unknown CompuScope system consisting of arbitrary CompuScope models. This method is used extensively in GageScope, for instance, which must support any arbitrary CompuScope system configuration. CsGetSystemCaps() is a very powerful method, however, it makes coding more complex since it must generically account for any CompuScope configuration. Consequently, usage of this function is only recommended where it is deemed to be necessary.

C Sample Program Structure

All C sample programs are Windows console applications whose output messages are printed within a Windows Command Prompt. The main sample program consists of the int _tmain(), which is the standard frame for console applications with Unicode compatibility. None of the sample programs explicitly use Unicode strings. Throughout the programs, however, string variables are always handled in a Unicode-compatible fashion. This way, the customer may easily add Unicode strings, if required.

In addition to API functions, the sample programs use *support functions*. These are not API subroutines but are useful functionality groupings that are provided for convenience. Source code for the Support functions is found within the "c common" folder. Since the support functions are used by multiple sample programs, we recommend that the user copy and rename these functions if required rather than modifying them.

All C sample programs begin with a call to CsInitialize(), which initializes the CompuScope hardware and drivers. Subsequently, a call is made to CsGetSystem(), which obtains a handle to the first available CompuScope system. All subsequent calls to the CompuScope drivers will use this handle. Next, a call is made to CsGetSystemInfo(), which obtains static information about the characteristics of the current CompuScope system.

The next step in all C sample programs is a call to ConfigureSystem(). This support function reads the local INI file, parses the setting values and assigns these values to the correct CompuScope structure variables. If the user wishes to obtain setting values from a source other than an INI file, such as from a GUI interface or some other input source, then they must replace ConfigureSystem() with appropriate CsSet() calls.

Internally, ConfigureSystem() calls the LoadConfiguration() support function, which in turn calls lower-level LoadXxxxxConfiguration() support functions, which actually do the parsing of the INI file data. Finally, LoadXxxxxConfiguration() uses the CsSet() API method to set all configuration variables in the driver to the values requested in the INI file.

The CsSet() method is used to set all configuration settings on the CompuScope hardware, except for a few parameters used for optional eXpert firmware and other advanced functionalities, which are configure by the CsExpertCall() API call. All standard C SDK sample programs use only CsSet() and only some advanced sample programs use CsExpertCall(). CsSet() divides CompuScope configuration settings into three basic groups: Acquisition settings, Channel settings and Trigger settings. These three groups of settings are respectively configured by qualifying the CsSet() with the CS_ACQUISITION, CS_CHANNEL and CS_TRIGGER constants and also passing pointers to structures of types CSACQUISITIONCONFIG , CSCHANNELCONFIG and CSTRIGGERCONFIG, which are filled by the user with requested configuration setting values. The contents of these structures are documented within the API manual. The CsGet() method is the reverse of the CsSet() method and allows the user to obtain current configuration settings.

When the CsSet() method is called with the CS_ACQUISITION modifier, the variable u32Mode in the structure CSACQUISITIONCONFIG must be set properly. This variable is normally used to set the number of active CompuScope channels (by assigning u32Mode to constants CS_MODE_SINGLE for one channel, CS_MODE_DUAL for two channels, CS_MODE_QUAD for four channels and CS_MODE_OCT for eight channels.) Various acquisition features may also be activated by logically ORing the appropriate u32Mode constant with a specific feature constant. For example, in order to operate a CompuScope with eight active channels and using an external 10 MHz Reference signal, assign the u32Mode variable with a value that is the result of logically ORing the CS_MODE_OCT constant with the CS_MODE_REFERENCE_CLK constant.

CsSet(), and therefore ConfigureSystem(), does not communicate with the CompuScope hardware in any way. Settings are only *committed* or passed from the CompuScope structure variables to the CompuScope hardware by a call to the CsDo() API method using the ACTION_COMMIT or ACTION_COMMIT_COERCE constant.

The next step is to call LoadConfiguration() with the APPLICATION_DATA constant, which obtains Application settings. These settings include variables related to the data of interest that is to be downloaded from the CompuScope on-board memory after acquisition. Knowledge of these settings is required in order to allocate the size of target data buffers within the sample program.

If no valid INI file is found, default settings are used by the program and an indicative message is displayed. Now that all CompuScope structures have been configured, a call is made to CsDo(), using the ACTION_COMMIT or ACTION_COMMIT_COERCE constant. This call passes all configuration settings to the CompuScope hardware, thus committing them.

The next step is a call to CsDo() using the ACTION_START constant. This call starts the CompuScope system acquiring data and awaiting a trigger event (or trigger events in the case of Multiple Record mode).

Once the CompuScope system is acquiring, the DataCaptureComplete() support function is called. This function does not return until the CompuScope acquisition is complete. Internally, the function calls CsGetStatus() repeatedly until acquisition is complete.

Unlike in previous drivers, the CompuScope hardware is not actually polled directly by CsGetStatus(). Instead, CsGetStatus() polls an internal driver variable that is updated by the hardware. On newer CompuScope models, this variable is actually updated by a hardware interrupt. Typical Gage user applications are single-threaded and so have nothing to gain by using interrupts. Advanced multi-threaded applications for newer CompuScope models, however, can save wasted polling time through the use of hardware interrupts by using event notification. For more details, refer to the CompuScope API Reference Manual description of the CsGetEventHandle() method.

Next, buffers that will hold the data to be transferred from the CompuScope on-board acquisition memory are allocated using the VirtualAlloc() function. Allocation is done in accordance with the requirements imposed by the Application settings. The sample programs allocate a buffer for the raw data and one for the converted voltage data, if required.

Under Windows, buffer allocation is done both in real physical RAM and in the swap file, which is actually a reserved section of the hard drive. There is no way to know how much memory will be allocated in real physical RAM and how much will be allocated in the swap file. In fact, the allocation fractions may even change with time. For smaller buffer allocations, the buffer may reside completely in PC RAM, while for larger allocations the swap file is also used. Of course, use of the swap file requires hard drive access, which will slow down performance. For acquisitions with a total size of about 16 MB or more, usage of the GageDeepAcquisition sample program is recommended. This program is specifically designed to minimize PC RAM usage, which reduces swapping.

The next step is to transfer data from CompuScope memory for each channel to the raw ADC data buffer using the CsTransfer() method. For GageMultipleRecord, data are transferred for multiple segments. For GageDeepAcquisition, data are transferred in pages, as described below. The data are then converted into Volts, if requested, using the ConvertToVolts() support function. The conversion step may be omitted for best transfer and storage performance.

The user passes CsTransfer() values of i64lStartAddress and i64Length within the structure of type PIN_PARAMS_TRANSFERDATA. These values are respectively the start point within a waveform from which the user requests data download to begin and the number of waveform points to download. CsTransfer() returns a structure of type OUT_PARAMS_TRANSFERDATA, which contains two important variables called i64ActualStart and i64ActualLength. As suggested by their names, the values of these variables respectively indicate the actual start point and transfer length that corresponds to the returned waveform data set. The slight differences between i64lStartAddress and i64ActualStart and between i64Length and i64ActualLength result from a combination of CompuScope hardware memory architecture limitations and driver buffer alignment requirements and may vary for different CompuScope models. It is very important that code that acts on the transferred data uses the Actual values (i64ActualStart and i64ActualLength) and not the requested values (i64lStartAddress and i64Length). Otherwise, artificial waveform jitter and invalid waveform data may occur.

Data are stored in the appropriate number of files, whose format is described earlier in this document. The user can easily modify the storage portion of the code for data display, analysis or transfer to another process. Finally, all allocated buffers and the CompuScope system handle are freed.

C# Sample Programs

Some C sample programs are also provided as C# projects. The C# projects may be operated in the .NET environment. While the .NET environment is good for web-based requirements, it is unclear how this environment adds any real value for CompuScope operation. Furthermore, management of large data buffers and process interoperability is not optimal under .NET. Consequently, it is recommended that users remain in the C Run Time environment, if possible.

The .NET environment requires that code be "managed", which means that it is platform-independent,among other things. Strictly speaking, it is not possible for hardware drivers to be managed. Consequently, we have added a translation layer between the drivers and the C# sample programs that provides interoperability. This translation layer essentially defines a new API – a C# CompuScope API. While structurally and semantically this API is similar to the C CompuScope API, it is documented separately in NETCompuScopeAPI.chm. The C# API is implemented in GageFunctions.dll, which is installed in your .NET GAC. GageFunctions.dll is CLR compliant and is compatible with NET Framework 2.0 and newer. Consequently, GageFunctions.dll provides support not only for C# but also for any other .NET language. GageFunctions.dll was written with performance in mind and has a minimal footprint in memory. Nonetheless, this layer does introduce an additional overhead and a slight decrease in performance, as compared with performance from a C program.

Sample Program Descriptions

GageAcquire

GageAcquire is the main sample program for Single Record capture from a single CompuScope system. In the event of multiple CompuScope systems, the program only operates the first available system. The program accepts configuration settings from the local INI file and uses them to operate the CompuScope system.

If any of the configuration settings are determined by the program to be invalid for the active CompuScope system, the program will stop and a descriptive error string will be displayed.

GageMultipleRecord

GageMultipleRecord is the sample program for Multiple Record acquisition from a single CompuScope system. Multiple Record Mode allows multiple waveforms to be rapidly acquired and stacked in on-board CompuScope memory. For instance, a CompuScope card with 4 GigaSamples of on-board memory may acquire up to 1,000000 records of 4000 samples each in Multiple Record mode. Between successive record acquisitions, the CompuScope acquisition engine is rapidly re-armed in the CompuScope hardware with no CPU interaction required. Consequently, in Multiple Record mode, a CompuScope system is capable of capturing bursts of triggers that repeat at rates of 500,000 triggers per second and more.

The Acquisition variable called SegmentCount within the INI file is used to set the number of records to be acquired. If this exceeds the maximum possible number of records, then an error will occur. The Application variable called NumberOfSegments is used to specify the number of Multiple Records to be downloaded and stored and must be less than or equal to the number of acquired records.

The Acquisition variable called SegmentSize within the INI file is used to control the size of each Multiple Record segment. SegmentSize may be set larger than the Depth so that pre-trigger data may be acquired. In this case, the Trigger Hold off which determines the number of sample points during which triggers are ignored and pre-trigger data accumulates, must be greater than or equal to.

When executed, GageMultipleRecord first initializes and configures the CompuScope systems, as in GageAcquire. Next, a single Multiple Record Acquisition is performed. After the acquisition is finished, the program downloads records sequentially and stores each in its own data file.

All CompuScope models support time-stamping in Multiple Record mode. Time-stamping is achieved using a high-speed on-board counter that may be clocked by an internal fixed oscillator source or by a source that is derived from the sampling clock. When a trigger event occurs, the current counter value is latched, thus stamping the time of occurrence of the trigger event. After a Multiple Record acquisition, GageMultipleRecord downloads all time stamp data. The counter values are converted into dimensions of time and each time-stamp value is stored in the header of the associated DAT file. (not in SIG or BIN files)

GageDeepAcquisition

GageDeepAcquisition is the sample program for large acquisitions from a single CompuScope system. The definition of "large" varies with system configuration but is roughly of order 16 MegaBytes. For small acquisitions, a C program is fully capable of downloading the entire acquisition into a physical PC RAM buffer. For larger acquisitions, however, the host PC will begin to use the swap file which is a section of the hard drive that Windows treats like PC RAM. Accessing the swap file is much slower than accessing PC RAM.

In order to avoid trying to keep large data sets in the program at one time, GageDeepAcquisition manages deep acquisitions by dividing them up into more manageable data chunks. By downloading only a chunk of data at a time, the program may manage very large acquisitions without having to use the swap file. The default chunk size is 32 kSamples.

The program does initialization, configuration setting and acquisition as usual. After the acquisition, however, only a single chunk of data at a time is downloaded and appended to the output file (.TXT or .DAT..). Successive chunks of data are downloaded and are appended to the same file until all acquired data have been stored. In this way, the program is able to manage an arbitrarily large acquisition without ever holding more than one chunk of data in its memory at one time. The logic illustrated in GageDeepAcquisition may be applied to allow other sample programs to similarly manage deep acquisitions and avoid swapping.

GageComplexTrigger

GageComplexTrigger is a sample program that illustrates complex triggering in Single Record mode. All CompuScope boards have two on-board trigger engines per channel that can be used for complex triggering. The two engines can be configured independently and their outputs are Boolean ORed together so that either engine may cause a trigger event. For simple triggering, only one trigger engine should be enabled.

The Trigger group of keys in the INI files allows the separate configuration of each trigger engine. For instance, by setting the two engine sources to Channel 1 and Channel 2, the user may configure the system to trigger on a pulse that occurs on either channel. Alternately, by setting both engine sources to Channel 1 but selecting different levels and slopes for each, the user may configure the system to do windowed triggering, where the system trigger if the input level leaves a specified voltage range.

GageMultipleSystem

GageMultipleSystem is the multi-threaded sample program for acquisition from multiple CompuScope systems. The program begins by obtaining the number of available CompuScope systems. In a loop, the program then sequentially obtains the handle to system number N, reads an INI file called SystemN.ini and then launches a thread for CompuScope system number N that is very similar to GageAcquire. Threads are launched for each system and then the program waits until all threads are finished before freeing all resources and terminating.

GageMultipleSystem acquires waveforms from all the CompuScope systems in a completely asynchronous fashion. Only one CompuScope system is operated at a time but the switching amongst them is indeterminate and is controlled by the Windows process management. The user must keep this in mind in designing the overall experiment. For instance, the program can be used to trigger all CompuScope systems simultaneously. In order to do this, however, the user must ensure that trigger signals are inhibited until all CompuScope systems are armed and awaiting a trigger event. If both CompuScope systems are not so prepared, an earlier trigger may otherwise trigger one system and not the other.

Advanced Sample programs

The C/C# SDK includes an "Advanced" folder that contains sample programs in addition to the documented sample programs. Usage of some of these programs may require special CompuScope hardware options, on-board firmware processing images or special driver versions. All advanced sample programs are documented within the "Advanced Sample Programs for CompuScope SDK" manual.

INI File Description

INI files used by Gage Sample programs are in the Windows INI format and consist of lists of setting variable assignments called keys. Keys are arranged in groups, whose titles must be between square brackets ([ ]). Each key must be under the correct group heading in order to operate correctly. For instance, the "Sample Rate" key must be under the "Acquisition" group heading. Keys under different group headings may have the same name.

An example of an INI file is shown below:

```
[Acquisition]
Mode=Single
SampleRate=10000000
Depth=8160
SegmentSize=8160
SegmentCount=1
TriggerDelay=0
TriggerHoldOff=0
TriggerTimeOut=1000000
ExtClk=0
TimeStampMode=Free
TimeStampClock=Fixed

[Channel1]
Range=2000
Coupling=DC
Impedance=50
DcOffset=0

[Channel2]
Range=2000
Coupling=DC
Impedance=50
DcOffset=0

[Trigger1]
Condition=Rising
Level=0
Source=1
Coupling=DC
Impedance=50

[Application]
StartPosition=0
TransferLength=8192
SegmentStart=1
SegmentCount=5
SaveFileName=Acquire
SaveFileFormat=TYPE_FLOAT
```

There are functions provided within the CompuScope C/C# SDK that read the settings from the INI file and pass these settings to the CompuScope drivers. Note that it is not necessary to use the INI files, although the sample programs do use them. All the configuration settings can be set up manually by using the SDK structures and API calls.

Any superfluous groups (for example, groups for a channel or trigger number that is not available) or superfluous keys will be ignored. Any necessary groups or keys that are not present in the INI file will be given default values. Invalid key settings will cause an error in the sample programs. Key names are always one word in length with no spaces. Names of keys or their assigned values are not case sensitive.

Below are descriptions of all the available INI file groups and keys. Note that any keys that your application does not require need not be in the INI file. If absent, a required key value is set to a reasonable default value.

[Acquisition]

This group sets all the CompuScope system's acquisition parameters.

Mode

Mode sets the operating mode of the CompuScope system. Valid values are:

D or ***Dual***	sets the mode to dual channel
S or ***Single***	sets the mode to single channel
Q or ***Quad***	sets the mode to quad channel
O or ***Octal***	sets the mode to octal channel

In some cases, special modes can be set by Boolean ORing a special value with the mode. In these cases, you must use a number to represent the mode, where:

1	means single channel (or 8-bit mode)
2	means dual channel (or 16-bit mode)
4	means quad channel (or 16-bit mode)
8	means octal channel (or 32-bit mode)

The special option would be ORed with the mode number. For example, to enable the reference clock input (on CompuScope models that have this functionality), the mode must be ORed with 0x400 (1024 decimal). For instance, 0x402, would enable the reference clock input in Dual Channel Mode.

SampleRate

SampleRate sets the sampling rate of the CompuScope system. The value is in Hertz. Check your CompuScope hardware manual for the available sample rates for your board(s). The default value depends on the CompuScope model.

Depth

Depth sets the post-trigger depth of the CompuScope system. The value is in samples.

The default value is 8160 samples.

SegmentSize

SegmentSize sets the size of the segment to capture. A segment is the sum of the pre- and post-trigger samples that are to be acquired.

The default value is 8160 samples.

TriggerDelay

TriggerDelay sets the trigger delay of the CompuScope system, which is the number of samples to acquire between the occurrence of the trigger event and the actual logging of the trigger event. A non-zero value is useful for signals whose region of interest occurs long after the trigger event. Not all CompuScope models support Trigger Delay. Please refer to the section Special Topics for more information about Trigger Delay.

The default value is 0.

SegmentCount

SegmentCount is the number of segments (or records) to acquire in a Multiple Record acquisition. For non-Multiple Record (Single Record) acquisitions, SegmentCount must be set to 1. The default value is 1.

TriggerTimeOut

TriggerTimeOut sets the trigger timeout of the CompuScope system. The value is in 100 nanosecond units of time. For example, a value of 10000000 gives a trigger timeout 1 second. The trigger timeout is the amount of time the driver will wait for a trigger event to occur before forcing a trigger event. A value of -1 will cause the driver to wait indefinitely for a trigger. The default value is -1.

TriggerHoldOff

TriggerHoldOff sets the time, in number of samples, during which trigger events will be ignored after the CompuScope system begins capturing and awaiting a trigger event. The function is useful for ensuring the accumulation of a specified amount of pre-trigger data that is equal to the TriggerHoldOff value. Please refer to the section 'Special Topics' for more information about Trigger Holdoff.

The default value is 0.

ExtClk

ExtClk is a flag to turn on or off external clocking functionality, assuming that the CompuScope hardware has external clock functionality available. A value of 1 enables external clocking and a value of 0 disables it. If external clocking is activated, then the SampleRate key must be set to the external clocking frequency. The default value is 0.

TimeStampMode

The time-stamping counter may be reset upon the start of each acquisition or left free running.

Free means do not reset the counter upon the start of each acquisition

Reset means reset the counter upon the start of each acquisition

TimeStampClock

The time-stamping counter may operate using for a source a fixed on-board oscillator that is independent of the sample rate. Alternately, the counter source may be derived from the sampling clock. Specifically, the source will be equal to the sampling clock frequency divided by an integer value.

Fixed means use the fixed on-board oscillator as the counter source

Sample means use frequency derived from the sampling clock as the counter source

[Channel1]

The Channel1 group sets the configuration settings for channel 1 of the CompuScope system. Similar groups, (i.e. Channel2, Channel3 etc.) are to set the other channels on the CompuScope system.

Range

Range sets the full scale input range for the channel. The values must be in millivolts. For example, a value of 2000 sets the input range to +/- 1 volt. Check your CompuScope hardware manual for the available input ranges for your CompuScope model. The default value is 2000 if the model supports it. Otherwise, the default is system-dependent.

Coupling

Coupling sets the input coupling for the channel. Available values are AC or DC. Alternatively, driver constants can be used, which are 1 for DC and 2 for AC. Consult your CompuScope hardware manual to check which values are available for your CompuScope model. The default is DC, unless the CompuScope system only supports AC coupling.

Impedance

Impedance sets the channel's terminating input impedance. Available values are 50 for 50 Ohms and 1000000 for 1 MOhm. Consult your CompuScope hardware manual to check which values are available for your CompuScope model. The default is 1000000, unless the CompuScope system only supports 50 Ohms.

DiffInput

DiffInput is a flag that turns differential input coupling on or off (if this functionality is available on your CompuScope model). A value of 1 activates differential input coupling and a value of 0 enables single-ended input coupling. Consult your CompuScope hardware manual to check which values are available for your CompuScope model. The default is 0.

DirectADC

DirectADC is a flag that turns on or off Direct-to-ADC input coupling (if it is available for your system). A value of 1 enables Direct-to-ADC input coupling and a value of 0 disables it. Consult your CompuScope hardware manual to check which values are available for your CompuScope model. The default is 0.

Filter

Filter sets the input bandwidth for the channel in kHz. Note that this functionality is currently unavailable.

DcOffset

DcOffset sets the value for channel's DC Offset in millivolts. The minimum and maximum values are dependent on the current input range. For example, on the 2000 mV input range, the maximum value is 1000 and the minimum is -1000. The default value is 0.

[Trigger1]

The Trigger1 group sets the configuration settings for trigger engine number 1 of the CompuScope system. Similar groups (i.e. Trigger2, Trigger3, etc.) are used to set up the other trigger engines of the CompuScope system, if available.

Condition

Condition sets the condition on which the system will trigger. Current valid values are:

Rising	system will trigger on a positive slope.
R	system will trigger on a positive slope.
Positive	system will trigger on a positive slope.
P	system will trigger on a positive slope.
1	system will trigger on a positive slope.
Falling	system will trigger on a negative slope.
F	system will trigger on a negative slope.
Negative	system will trigger on a negative slope.
N	system will trigger on a negative slope.
0	system will trigger on a negative slope.

The default value is Rising.

Level

Level sets the trigger level as a percentage of the input range of the trigger source (half the full scale input range). For example, in the 2000 mV range at 0 DC offset, 50 would be positive 50 % of 1 Volt, or 500 millivolts. The default value is 0.

Source

Source sets the trigger source. Valid values are:

1,2,3,4 …	trigger from channel 1, 2, 3, 4 …
External	trigger from the external trigger input
-1	trigger from the external trigger input
Disable	disable trigger engine
0	disable trigger engine

The default value is 1 (Trigger from the channel 1)

Coupling

Coupling is the value of the input coupling for the external trigger input. If external trigger is not being used, this value is unnecessary and is ignored. Valid values are AC and DC. Alternatively, the driver constants may be used, which are 1 for DC and 2 for AC. Consult your CompuScope hardware manual to check which values are available for your CompuScope model. The default is DC, unless a system only supports AC.

Range

Range sets the input range for the external trigger input, if the external trigger is being used. If external trigger is not being used, this value is unnecessary and is ignored. The values are given in millivolts for the full scale input range of the external trigger input. For example, a value of 10000 represents the +/- 5 Volt external trigger input range. Check your CompuScope hardware manual for the available external trigger ranges for your CompuScope model. The default is 10000 if the CompuScope hardware supports it. Otherwise, the default is system-dependent.

Impedance

Impedance sets the terminating impedance of the external trigger input. If external trigger is not being used, this value is unnecessary and is ignored. Available values are 50 for 50 Ohms and 1000000 for 1 MOhm. Consult your CompuScope hardware manual to check values which are valid for your CompuScope model. The default is 1000000, unless the CompuScope system only supports 50 Ohms.

[Application]

The Application group sets the data transfer parameters, as well as file format settings.

StartPosition

StartPosition is the address from which to start transferring the acquired data from on-board CompuScope memory to the application. The address is specified relative to the trigger address, so that a value of 0 will start transfer at the trigger address. Negative values will transfer pre-trigger data and positive values will start transfer data acquired after the trigger event. The default value is 0. The user must ensure that the start value is set properly so that invalid data are not transferred.

TransferLength

TransferLength sets the number of samples to transfer, starting from StartPosition. The default value is 4096. The user must ensure that the TransferLength is not too large so that invalid data are not transferred.

SegmentStart

SegmentStart sets the number of the first segment (record) to transfer after a Multiple Record acquisition. The default is 1, for the first segment. If Multiple Record is not being used, this setting is ignored and is unnecessary.

SegmentCount

SegmentCount is the number of segments (records) to transfer after a multiple record transfer. The default is 5. The user must ensure that SegmentCount does not exceed the number of acquired segments so that invalid data are not transferred.

PageSize

PageSize may be used for extremely large transfers. Rather than transferring all the data at one time, GageDeepAcquisition.c may be used to transfer the data in chunks with a size of PageSize.

SaveFileName

SaveFileName is the base file name for the files saved in each of the sample programs. The channel number is appended to the base filename. If the acquisition was a Multiple Record mode, the record number is also appended to the file name. For example, a single record dual-channel capture would save channel 1's data to GAGE_FILE_1.DAT and channel 2's data to GAGE_FILE_2.dat.

SaveFileFormat

SaveFileFormat determines how the sample programs save output data files. Data can be saved in either binary or ASCII files. In case of ASCII file, all data are saved as a single-column file, where each entry denotes a single sample point.

The available values for SaveFileFormat are:

TYPE_FLOAT	Output as .TXT files. Data saved as ASCII. Data converted to voltages in floating-point format.
TYPE_DEC	Output as .TXT files. Data saved as ASCII. Raw ADC data in decimal format
TYPE_HEX	Output as .TXT files. Data saved as ASCII. Raw ADC data in hexadecimal format
TYPE_BIN	Output as .DAT files. Data saved as binary.
TYPE_SIG	Output as .SIG files. Data saved as GageScope SIG file format.

The default is TYPE_DEC.

Special Topics

Operating CompuScope cards from Unsupported Programming Environments

There are three CompuScope SDKs, one for C/C#, one for MATLAB and one for LabVIEW. CompuScope cards may be operated from other programming environments, however, using the C/C# SDK as the starting point. Explicit provisions are made within the C/C# for CompuScope operation within the Visual Basic.NET, LabWindows/CVI and Delphi environments.

The C/C# includes a version of GageAcquire operates under Visual Basic.NET. These projects are located within the "VB.NET Samples" folder in the "C# Samples" folder of the C/C# SDK. The user should follow the GageAcquire project as a guide for translating other C sample programs for operation under Visual Basic.NET.

All C sample programs may be compiled using the LabWindows/CVI compiler, all necessary translations are done within the CsCVI.h file. The user can, therefore, access all CompuScope functionality from LabWindows/CVI. No LabWindows/CVI GUI is provided, therefore the user must construct one if required.

For user convenience GageAcquire the sample program has been ported to both the Delphi and the Python environment. This example is located in the Delphi and Pythonfolders of the C/C# SDK. The user may use this example as a guide to port other examples to the Delphi environment.

While no explicit code is provided for programming environments other than C, C#, Visual Basic.NET and LabWindows/CVI, the C/C# may be used as the starting point for CompuScope operation in unsupported programming environments, such as VEE, DASYLab, SoftWIRE, or TestPoint.

The simplest method of controlling a CompuScope card from unsupported programming environments is to make use of the compiled executable version of the C sample programs. From any programming environment, the user can make a system call to a compiled C program executable, which will use configuration settings from the corresponding INI file. Resultant TXT text data files produced by the executable may then be read directly into the programming environment for display, analysis or storage. While it will not provide optimal performance, this method of system calls to an executable combined with file-mediated data transfer has the advantage that it can be implemented very quickly and easily.

For better performance, the user may call the CompuScope driver Dynamically Linked Library (DLL) directly from an unsupported programming environment. For simplicity, the user might continue to read configuration settings from an INI file but then read CompuScope data directly through the CompuScope driver DLL so that rapid repetitive acquisitions may be performed. Alternatively, users may elect to control configuration settings directly from the unsupported programming environment through the DLL.

For such complete CompuScope control from an unsupported environment, the users should use the appropriate C sample program as a guide for API method calling order and then use the CompuScope API manual for detailed information on API method operation. The CompuScope API is uniform for all CompuScope models so that once support has been provided for one CompuScope model within an unsupported programming environment, support is automatically available for all CompuScope models.

Converting from CompuScope ADC Code to Voltages

SDK sample programs are configured to save or display CompuScope waveform data that have been scaled so that the sample values are in Volts. The user may want to bypass the voltage conversion step, however, in order to achieve the best repetitive capture performance. Voltage conversion may then be done at leisure in post-processing or not at all.

Raw ADC code waveform data may be converted to voltage data for all CompuScope models using the following equation:

$$Voltage = \frac{Offset - ADC\_Code}{Resolution} \times \frac{(Full\ Scale\ Input\ Voltage)}{2} + DC\_Offset$$

The *Offset* and *Resolution* for the CompuScope system may be obtained programmatically using various SDK tools. The DC offset settings must be added (Do not confuse "DC offset:" and the "Offset" used in the above equation).

For instance, for the 12-bit CompuScope EON Express, *Offset*=16 and *Resolution* = -32768 Let us assume that the user has selected the ±1 Volt Input Range using the CS12501, for a Full Scale Input Range of 2 Volts. Let us further assume that the user has applied a 200 mV offset. For this example, therefore, the voltage conversion equation becomes:

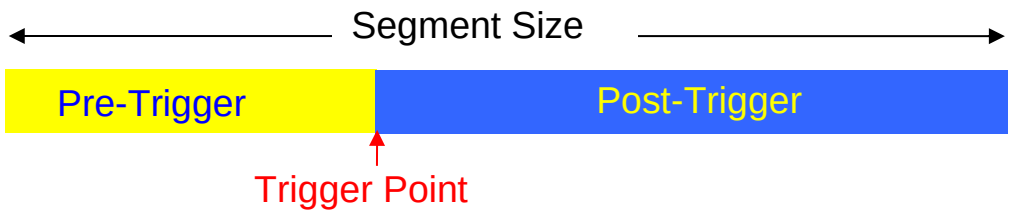
$$Voltage = \frac{16 - ADC\_Code}{-32768} \times \frac{2 \text{ Volts}}{2} + 0.2 \text{ Volts}$$

The Offset and Resolution for the CompuScope system may be obtained using the CsGet() API method using the CS_ACQUISITION Index. The resolution and offset values are returned in the CSACQUISITIONCONFIG structure as the values of i32SampleRes and i32SampleOffset, respectively. If the user has applied a DC Offset voltage to the signal, then this voltage may be obtained by calling CsGet() with the CS_CHANNEL index. The DC offset voltage is returned as the value of i32DcOffset within the CSCHANNELCONFIG structure.

Depth and Segment Size

The end of any CompuScope waveform acquisition is always initiated by the trigger event. After the trigger event occurs, the CompuScope acquires the requested number of post-trigger data points, called the post-trigger *Depth,* is acquired and then the acquisition terminates.

The total amount of CompuScope acquisition memory dedicated to the waveforms called the *Segment Size*. The difference between the Segment size and the Depth is equal to the amount of memory available for the acquisition of pre-trigger data. The image below illustrates the relationship between the Segment Size and the pre- and post-trigger data.



As an example, consider a user who wants to acquire 4 kiloSamples of post-trigger depth and 2 kiloSamples of pre-triggers data. In this case, the user must set the Segment Size to 6k and the Depth to 4k so that 6k-4k = 2k are available for pre-trigger data accumulation.

Trigger HoldOff

Trigger HoldOff is a feature that is useful for ensuring the accumulation of a specified amount of pre-trigger data. The Trigger HoldOff setting specifies the amount of time, in Samples during which the CompuScope hardware will ignore trigger events after acquisition has begun and pre-trigger data are being acquired.

Beginning the consequently, while user must fill the memory available for pre-triggering data, higher values of Trigger Hold Off may be used to delay the interval between successive triggers.

The Trigger HoldOff must be set to be greater than or equal to the amount of acquisition memory that is available for pre-trigger data, which is (Segment Size – Depth).

Trigger Delay

All CompuScopes support the Trigger Delay feature. This feature is useful for situations where the signal portion of interest occurs long after the trigger event. Trigger Delay allows suppression of storage of samples that occur during the Trigger Delay so that memory is not needlessly consumed.

Normally, with a Trigger Delay of zero, the trigger event activates count-down of the post-trigger depth counter, which was preloaded with the post-trigger depth. The counter is decremented by one count for each sample acquired after the trigger event. When the counter value reaches 0, the acquisition is terminated. In this way, the CompuScope acquires a number of samples equal to the depth after the trigger event.

A non-zero Trigger Delay value is used to delay the beginning of the countdown of the post-trigger depth counter. The Trigger Delay value sets the number of samples that the CompuScope hardware will wait after the trigger event occurs before beginning the count-down of the depth counters. When the Trigger Delay value is non-zero, pre-trigger data will not be acquired. Consequently, when the Trigger Delay value is non-zero, the user must set the Segment Size equal to the Depth.

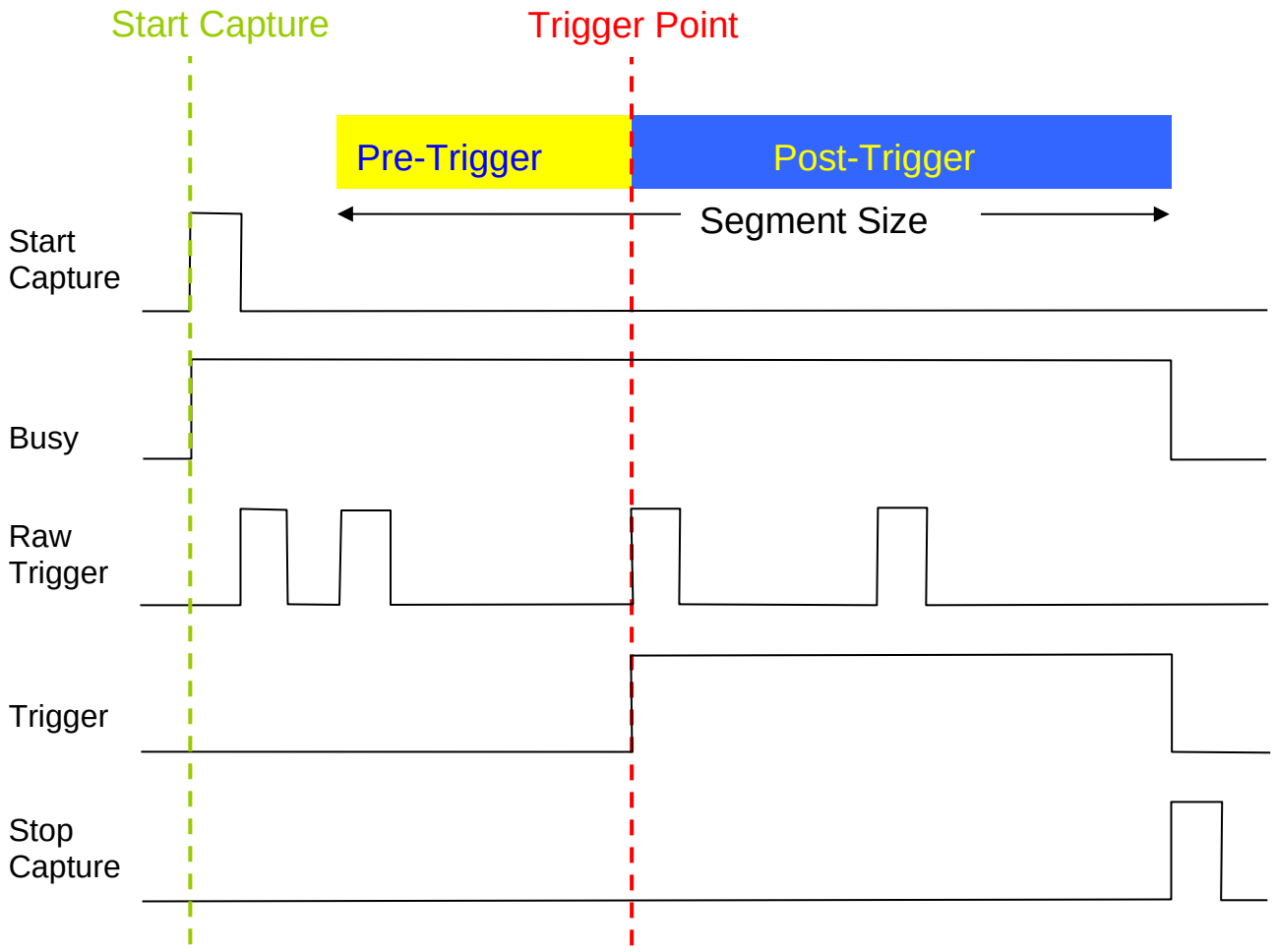
As an example, consider a signal where the feature of interest is 20,000 samples long but begins 100,000 samples after the trigger event. In this case, the SegmentSize and Depth should be set to 20,000. The Trigger Delay value should be set to 100,000. Without the Trigger Delay feature, the user would be forced to set the Depth to 120,000 Samples and waste 100,000 Samples of memory, even though only the last 20,000 were downloaded. Using the Trigger Delay feature, however, only the data of interest are retained in CompuScope memory.

CompuScope Acquisition Timing Diagram

The timing diagram below is provided as an aid for understanding the timing of events during the acquisition of a segment or record. The pseudo-signals are indicated as HIGH when the labeled function is active. In this case, the Trigger HoldOff must be set greater than or equal to (Segment Size – Depth) so that triggers are ignored at least until the memory allocated for pre-trigger data is filled. The Trigger Delay setting is assumed to be 0.

As discussed, the user sets the Segment Size and post-trigger Depth, which leaves (Segment Size – Depth) for the acquisition of pre-trigger data. The Segment is illustrated above with pre- and post-trigger data respectively shown in yellow and blue.

The Start Capture signal starts the CompuScope acquiring pre-trigger data and awaiting a trigger event. When this occurs, the card begins acquiring post-trigger data and terminates when Depth post-trigger points have been acquired. The Busy signal above shows the time during which the board is acquiring data.



The Raw Trigger signal shows that 4 raw trigger pulses occurred during the acquisition. The first two raw triggers were ignored because, when each occurred, the CompuScope had not yet acquired the requested number of pre-trigger points. The fourth raw trigger is ignored because it occurred during the acquisition of post-trigger data that was initiated by the third raw trigger, which is the only raw trigger that registered as a true trigger event. That this third raw trigger is the true trigger event is illustrated above by the Trigger signal below it. The Stop Capture signal occurs when the post-trigger depth counter decrements to zero and terminates the acquisition.

Notice that not all pre-trigger data were retained in memory. The pre-trigger data fills the (Segment Size – Depth) available memory like a FIFO (First In First Out) so that only the most recent (Segment Size – Depth) points are retained. Consequently, the data that were acquired between the vertical green line and the yellow box were overwritten and lost in the above acquisition.

Technical Support

We offer technical support for all our Software Development Kits.

In order to serve you better, we have created a web-based technical support system that is available to you 24 hours a day.

By utilizing the internet to the fullest, we are able to provide you better than ever technical support without increasing our costs, thereby allowing us to provide you the best possible product at the lowest possible price.

To obtain technical support, simply visit:

http://www.gage-applied.com/support/support-form.php

Please complete this form and submit it. Our form processing system will intelligently route your request to the Technical Support Specialist (TSS) most familiar with the intricacies of your product. This TSS will be in contact with you within 24 hours of form submittal.

In the odd case that you have problems submitting the form on our web site, please e-mail us at

tech-support@gage-applied.com

As opposed to automatic routing of technical support requests originating from the Gage web site, support requests received via e-mail or telephone calls are routed manually by our staff. Providing you with high quality support may take an average of 2 to 3 days if you do not use the web-based technical support system.

Please note that Technical Support Requests received

via e-mail or by telephone will take an average of 2 to 3 days to process.

It is faster to use the web site!

When calling for support we ask that you have the following information available:

1. Version and type of your CompuScope SDK and drivers.
   (The version numbers are indicated in the About CD screen of the CompuScope CD. Version numbers can also be obtained by looking in the appropriate README.TXT files)

2. Type, version and memory depth of your CompuScope card.

3. Type and version of your operating system.

4. Type and speed of your computer and bus.

5. If possible, the file saved from the Information tab of the CompuScope Manager utility.

6. Any extra hardware peripherals (example: RAID Controller, DSP board, etc.)

7. Were you able to reproduce the problem with standalone Gage Software (e.g. GageScope, CsTest)?