GaGe

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

User's Guide

CompuScope Driver Version 4.20+

SDK Version 4.20+

P/N: 0045031
Reorder #: MKT-SWM-SDK17
0807

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: 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. If you do not feel comfortable with C or C#, it is highly recommended that you consult the manuals supplied to you by the vendor of your compiler before starting any program development.

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 CompuScope C/C# SDK supports all GaGe CompuScope cards – PCI and CompactPCI/PXI. Specific hardware features that are available in the SDK sample programs, however, may not be supported by your CompuScope model. Please refer to the CompuScope Hardware Manual for information specific to your CompuScope card in order to determine the capabilities of your CompuScope model.

	C	C#	VB .Net	LabVIEW	MATLAB	CVI	Delphi
Simple	X	X	X	X	X	X	X
Acquire	X	X	X	X	X	X	X
Coerce	X	X		X	X	X	
ComplexTrigger	X	X		X	X	X	
DeepAcquisition	X	X		X	X	X	
MultipleRecords	X	X		X	X	X	
MultipleSystems	X	X		X	X	X	
AdvMulRec	X					X	
AsTransfer	X	X				X	
Average	X	X	X	X	X	X	
MulRecAverage	X					X	
GageFFTe	X	X	X	X	X	X	
Callback	X					X	
CsPrf	X	X				X	
Events	X	X				X	
FIR	X	X		X	X	X	
MinMaxDTC	X					X	

Chapter 1: 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:\Program Files\Gage\CompuScope\CompuScope C_C# SDK. It is recommended that you use the default installation location.

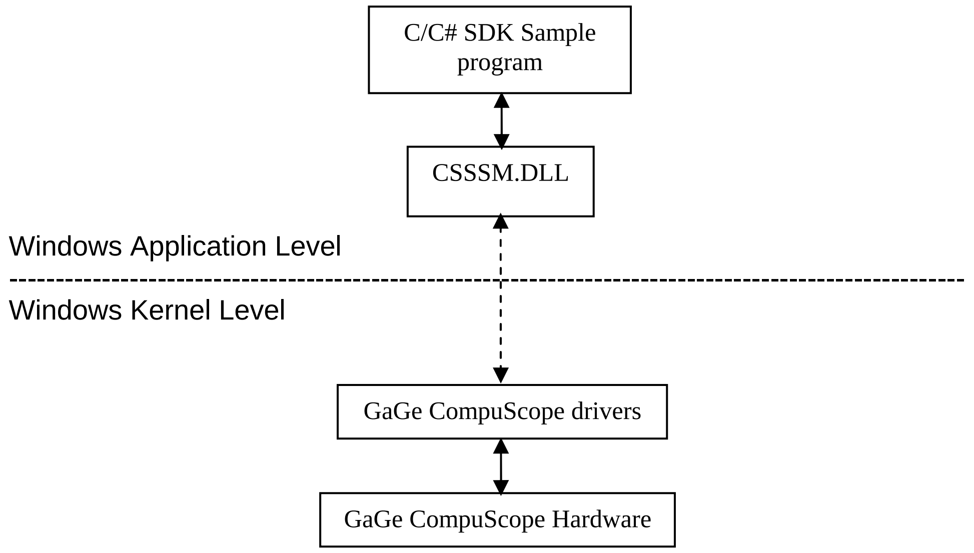
More detailed installation instructions are available in the GaGe CompuScope Startup Guide, which was shipped with your order. On a 64-bit Windows O/S, the CompuScope C/C# SDK provides both 32-bit and 64-bit LIB files, so it may be used to build full 64-bit Windows applications from C, C# or VB.NET.

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 common files) are installed.

Chapter 2: 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 PCI 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

Complete projects files for all C/C# SDK sample programs are provided for Microsoft Visual C/C++ version 6.0 or higher. Upon opening the workspace file for any C sample program, all supporting files, path specifications and build settings are automatically loaded. Compiled executable version of all C and C# sample programs are provided with the SDK for convenience.

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 is required even if newer versions are installed.

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 files.
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 hard drive file according to the settings and file format provided in the local INI file.

A list of all CompuScope sample programs is given below with a brief description of their functionalities.

GageSimple: A simple program used to verify correct operation of the CompuScope hardware and driver. No adjustments are possible and default CompuScope settings are used (except for a shortened trigger time-out value).

GageAcquire: The simplest acquisition program with full configuration control of CompuScope settings that illustrates usage of CompuScope Single Record mode. An error message is displayed upon entry of invalid settings in the local INI file.

GageCoerce: A program that is just like GageAcquire, except that invalid settings are coerced to available values. A message is displayed that indicates the settings that were coerced.

GageMultipleRecord: A program that performs a CompuScope acquisition in Multiple Record mode.

GageDeepAcquisition: A program that performs a large CompuScope acquisition and illustrates management of the large resultant data set.

GageComplexTrigger: A program that illustrates usage of complex triggering using multiple on-board trigger engines, if available.

GageMultipleSystem: A program that operates multiple CompuScope systems, each with their own local INI files and output data files.

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 not a 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.

Configuration Setting INI files

All C and C# sample programs except GageSimple 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 a document called INI_FILE_DESCRIPTION.pdf that is installed with the SDK. 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.

Sample Program DAT Data Files

All C and C# sample programs acquire data from CompuScope hardware and store the results in ASCII data files with file names of the form XXX_N.DAT, where N is the channel number in the CompuScope system. These files may easily be imported into database programs like Microsoft Excel for analysis and display.

The beginning of the DAT file is a header containing information about the acquisition that is enclosed by two lines of minus signs ("-----------------"). Programs that read DAT 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 DAT file header in future but the entire header will always be enclosed between the minus sign delimiter lines.

After the header, all DAT 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 are 1 to 4 Bytes per sample and the ASCII line occupies up to 20 Bytes.

Users may 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.

Chapter 3: 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 very powerful, however, it makes coding quite complex since it must generically account for any CompuScope configuration. Consequently, usage of this function is only recommended where it is deemed to be absolutely necessary.

C Sample Program Structure

All C sample programs are Windows console applications whose output messages are printed within a Windows Console Box, which resembles the Microsoft DOS environment. 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. All support functions that are required for multiple sample programs are defined within CsAppSupport.dll.

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. GageSimple illustrates the direct assignment and implementation of CompuScope configuration settings.

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, ConfigureSystem() uses the CsSet() API method to set all configuration variables in the driver.

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 exit 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 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 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.

An important variable returned by CsTransfer() is the OutData structure. Contained in this structure are the ActualStart and ActualLength variables. Values of these variables may be different from the requested TransferStart and TransferLength values that are provided in the InData structure. The slight difference results from a combination of CompuScope hardware memory architecture and driver buffer alignment requirements and may vary for different CompuScope models. It is very important that code which acts on the transferred data uses the actual values and not the requested values.

Data are stored in the appropriate number of DAT 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

All C sample programs are also provided as C# projects with identical functionality. 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 compatible with NET Framework 2.0 and is also CLR compliant. 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.

Sample Program Descriptions

GageSimple

GageSimple is not intended as a starting point for customer applications but is a simple test program to confirm correct operation of the CompuScope hardware, the Windows drivers, and the C/C# SDK.

GageSimple grabs the handle to the first CompuScope system and does a single acquisition using the driver defaults for all configuration parameters. The driver defaults are a set of default configuration parameters that are tailored to each CompuScope hardware model. The only non-default parameter used is a short trigger time-out value (rather than the default trigger time-out value of infinity) so that a signal is acquired even if the trigger conditions are not met. This program operates only on a single CompuScope card. In the event of a Master/Slave CompuScope system, the program only addresses the first card in the first system.

Again, GageSimple should only be used to test hardware and software integrity and should not be developed by the customer into a custom program, since there are better sample programs for this purpose. While the code for GageSimple is provided for completeness, users should only execute the compiled version in order to verify that no errors occur and that the output DAT files contain correct data.

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. Users who prefer the program to continue operating upon invalid setting entry should use GageCoerce.

GageCoerce

GageCoerce is identical to GageAcquire, except for its handling of invalid configuration setting entry. In this event, the program will not stop with an error message. Rather than stopping, the program will coerce the settings to values that are available on the current CompuScope system. Any setting coercions are displayed, using the ReportAcquisitionChanges() support function.

Coercion is actually executed by CsDo() by using the ACTION_COMMIT_COERCE modifier. The coercion procedure varies for the different types of configuration settings. For the internal sampling rate, for instance, the coercion procedure chooses the closest available sampling rate. For example, if the user enters 6600000 as an internal sampling rate and if only 50 MS/s and 100 MS/s are available, the coercion procedure will choose 50 MS/s.

Except for the coercion, operation of GageCoerce is identical to that for GageAcquire. All sample programs other than GageCoerce will stop with an error upon invalid setting entry, as does GageAcquire. These programs use the ACTION_COMMIT modifier for CsDo() and not ACTION_COMMIT_COERCE. The user can easily modify other programs to coerce settings simply by following the logic illustrated in GageCoerce.

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 32 MegaSamples of on-board memory may acquire up to 16,000 records of 2000 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 100,000 triggers per second and more.

The Acquisitions 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 SegmentCount INI file Application variable 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 Acquisitions variable called SegmentSize within the INI file is used to control the size of each Multiple Record segment. For some CompuScope models, SegmentSize may be set larger than the Depth so that pre-trigger data may be acquired. Older CompuScope hardware does not support pre-trigger data in Multiple Record mode and so SegmentSize will be internally coerced so that it is equal to Depth. Refer to your CompuScope Hardware Manual for information specific to the capabilities of your CompuScope model.

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 ASCII DAT file.

Newer CompuScope models like the CS14200, CS14105 and CS12400 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 if the CompuScope model supports the feature. The counter values are converted into dimensions of time and each time-stamp value is stored in the header of the associated DAT file.

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. Usage of the swap file causes the hard drive to spin and slows down execution.

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 *pages*. By downloading only a page of data at a time, the program may manage very large acquisitions without having to use the swap file. The default page size is 32 k.

The program does initialization, configuration setting and acquisition as usual. After the acquisition, however, only a single data page at a time from the acquisition is downloaded and appended to the DAT file. Successive data pages are downloaded and are appended to the DAT 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 data page 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. Some CompuScope models are equipped with two on-board trigger engines that can be used for complex triggering. On these models, 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, the second engine is disabled.

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.

Digital Input CompuScope cards

All SDK programs can be used to perform acquisitions from digital input CompuScope cards, such as the CS3200 and CS3200C. Parameters specific to digital input cards may be set by using CsSet() with the appropriate modifiers.

The CS3200 and CS3200C allow three digital input sample width modes for optimal memory usage: 8-bit, 16-bit and 32-bit. These modes can be selected using CsSet() with the CS_ACQUISITION modifier using the following u32Mode element values in the CSACQUISITIONCONFIG structure:

CS_MODE_SINGLE - to activate 8-bit digital input mode
CS_MODE_DUAL - to activate 16-bit digital input mode
CS_MODE_QUAD - to activate 32-bit digital input mode

The CS3200 and CS3200C provide different digital input signal level protocols, depending on the model. Different digital input protocols can be selected using CsSet() with the CS_CHANNEL modifier and using the following u32InputRange element values in the : CSCHANNELCONFIG structure:

CS_GAIN_CMOS - to activate CMOS digital voltage levels
CS_GAIN_TTL - to activate TTL digital voltage levels
CS_GAIN_ECL - to activate ECL digital voltage levels

CS_GAIN_PECL - to activate PECL digital voltage levels
CS_GAIN_LVDS - to activate LVDS digital voltage levels

The CS3200 and CS3200C allow inversion of the sampling clock so that digital samples are acquired on the falling edge of the clock signal rather than on the rising edge. In order to select the falling edge, simply perform a Boolean OR on the u32Mode element values in the CSACQUISITIONCONFIG structure with the constant CS_MODE_CS3200_CLK_INVERT before calling CsSet() with the CS_ACQUISITION modifier.

Again, using the above special instructions for digital input CompuScope cards, all C/C# SDK sample programs may be used to operate a digital CompuScope card.

Advanced Sample programs

The C/C# SDK may contain "Advanced" sample programs in addition to the documented sample programs. Usage of some of these files may require special CompuScope hardware options, on-board firmware processing images or special driver versions. These sample programs are provided as-is with limited documentation in the form of an accompanying explanatory text file.

Chapter 4: 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, LabWindows/CVI and Delphi environments.

The C/C# includes a version of GageAcquire and GageSimple that 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 and GageSimple have been ported to the Delphi environment. These examples are located in the Delphi folder of the C/C# SDK. The user may use these examples 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 DAT 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. 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

All C/C# SDK sample programs are configured to save DAT file data that have been scaled so that stored sample values are in Volts. The user may bypass the voltage conversion step in order to achieve the best repetitive capture performance. Voltage conversion may then be done at leisure in post-processing.

Raw ADC code data may be converted to voltage data for all CompuScope cards 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 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.

For instance, for the CompuScope 82G, Offset=127 and Resolution=128. Let us assume that the user has selected the +/-1 Volt Input Range, for a Full Scale Input Range of 2 Volts. Let us further assume that the user has applied a 200 mV DC offset. For this example, therefore, the voltage conversion equation becomes:

$$Voltage = \frac{127 - ADC\_Code}{128} \times 1\ \text{Volts} + 0.2\ \text{Volts}$$

Depth and Segment Size in Multiple Record Mode

On all CompuScope models, on-board acquisition memory is arranged as a circular buffer for Single Record acquisitions. This means that when the CompuScope is storing acquired data and it reaches the end of the memory, the memory counter rolls over. Data storage wraps around the memory and starts digitizing into the beginning of memory. The end of an acquisition is always initiated by the trigger event, after which the requested number of post-trigger data points is acquired and then the acquisition terminates. This circular memory architecture allows the CompuScope to accumulate an amount of pre-trigger data of to up to the amount of CompuScope acquisition memory per channel less the amount of post-trigger data.

Newer CompuScope models, such as the CS82G, CS12400, CS14200 and CS14105, allow for the accumulation of pre-trigger data in Multiple Record mode. On these models, memory is divided into multiple circular buffers, the number of which is equal to the number of records specified. The Segment Size control allows the user to specify the size of each circular buffer. This way, the user may select the amount of memory reserved for pre-trigger data accumulation. For instance, if the user selects a post-trigger Depth of 2048 and a Multiple Record Segment Size of 8192, then up to 6144 Samples of pre-trigger data may be acquired. Increasing the Segment Size beyond the Depth will accordingly reduce the maximum possible number of Multiple Records. In fact, on newer CompuScope models, Segment Size has the same effect in Single Record mode, which is, architecturally speaking, just a special case of Multiple Record Mode with a Number of Records equal to 1.

Only newer CompuScope models such as the CS82G, CS14200, CS14105 and CS12400, support pre-trigger Multiple Record mode. On older CompuScope models without pre-trigger data in Multiple Record mode, the Segment Size setting will have no effect. The value is ignored and is forced to be equal to the Depth by the drivers. If queried, Segment Size will be returned with a value equal to the Depth. In Single Record mode on older CompuScope models, the effective Segment Size is the total available channel acquisition memory.

Trigger Holdoff

Trigger Holdoff is a feature that is useful for ensuring the accumulation of a specified amount of pre-trigger data. 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.

Without Trigger Holdoff, there is no guarantee that a given number of pre-trigger samples will be acquired, since a trigger event may occur immediately after acquisition, leading to a very small number of pre-trigger points. By ignoring trigger events for a time equal to the specified Trigger Holdoff, the accumulation of a number of pre-trigger points equal to the Holdoff setting is guaranteed.

The downside of ensuring pre-trigger data is that triggers are ignored, so that important trigger events may be missed. For instance, in lightning monitoring applications, researchers usually want to acquire pre-trigger data. These data give information about events immediately preceding the lightning strike which triggers the CompuScope hardware. Lightning strikes may occur very close together in time, however, and missing a lighting pulse is much worse than missing pre-trigger data. Consequently, lightning testers should not use the Trigger Holdoff but should simply accept acquisition of only the amount of pre-trigger data that naturally occur between triggers. The user must decide whether to use Trigger Holdoff, based on the application.

Trigger Delay

New-generation PCI CompuScope models such as the CS14200, CS14105 and CS12400, support a feature called Trigger Delay. This feature is useful for situations where the signal portion of interest occurs long after the trigger event. 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. The SegmentSize may be set equal to the depth so that the CompuScope hardware need not waste memory by storing data that are not of interest.

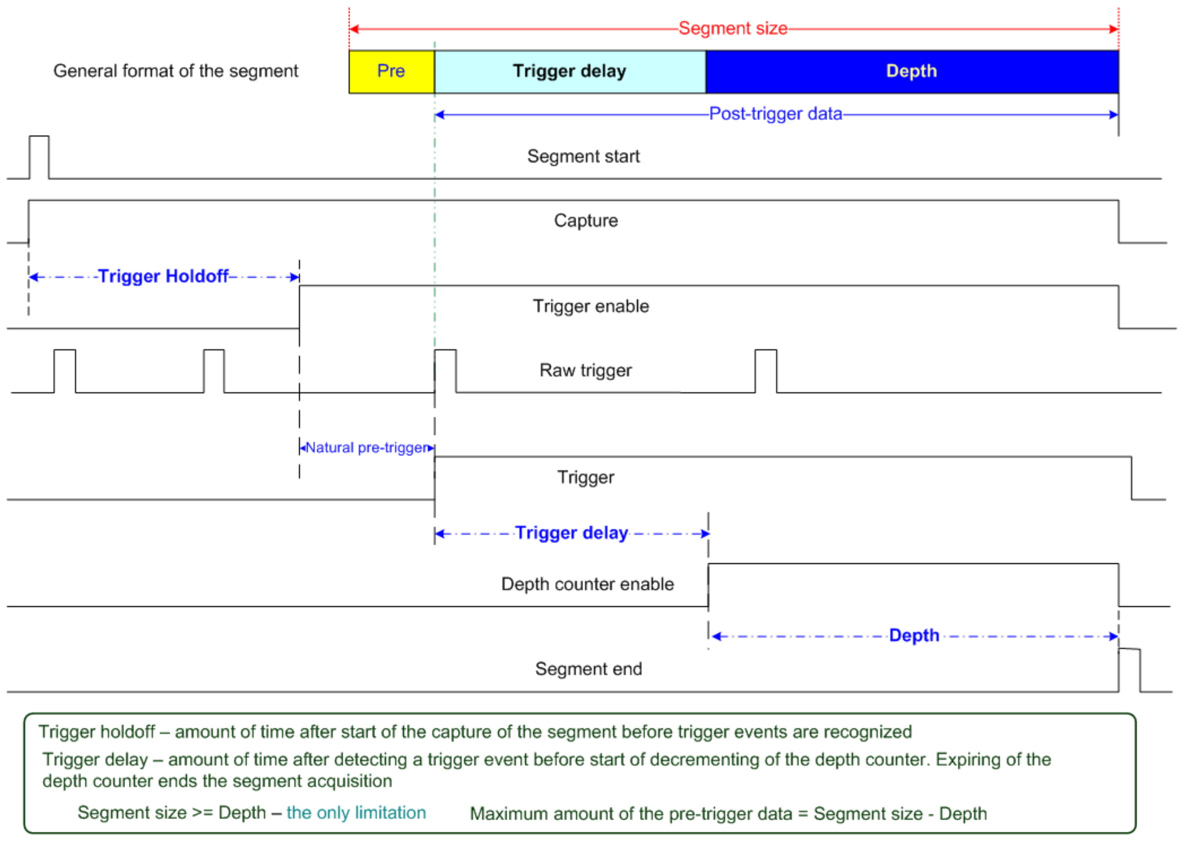
As an example, consider a signal where the feature of interest is 20,000 samples long but begins 100,000 samples after the trigger event. In this case, the SegmentSize and Depth should be set to 20,000. The Trigger Delay value should be set to 100,000. Without the Trigger Delay feature, the user would be forced to set the Depth to 120,000 Samples and waste 100,000 Samples of memory, even if 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.

A CompuScope system always acquires from the beginning of the Segment Start pulse and continues until the Depth counter's count-down has expired. The memory allotted to the acquisition is equal to the Segment Size and is arranged in a circular fashion. Notice that raw trigger events are ignored until the Trigger Holdoff time has elapsed.

Note also that, although pre-trigger data are acquired throughout the time between Segment Start and the Trigger Event, most of the acquired pre-trigger data have been overwritten in the diagram below and only a small fraction are available for download.



Trigger holdoff – amount of time after start of the capture of the segment before trigger events are recognized

Trigger delay – amount of time after detecting a trigger event before start of decrementing of the depth counter. Expiring of the depth counter ends the segment acquisition

Segment size >= Depth – the only limitation   Maximum amount of the pre-trigger data = Segment size - Depth

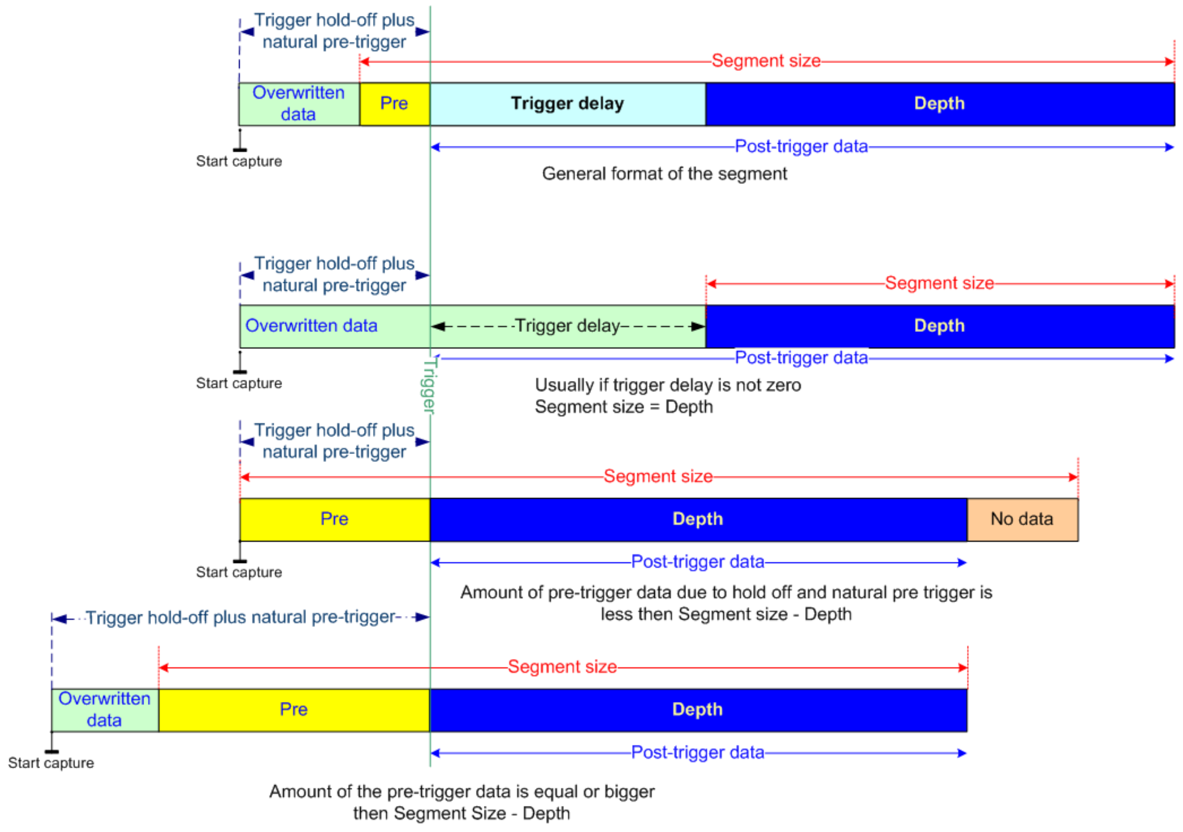
Representative Acquisition Sequences

The diagram below illustrates important representative acquisition sequences that are presented for key acquisition requirements. The first sequence shows a generalized acquisition with data collected during all acquisitions phases illustrated: overwritten pre-trigger data, available pre-trigger data, data acquired during the Trigger Delay and post-trigger data.

The second sequence shows an acquisition using a trigger delay where the Segment Size is made equal to the Depth. This acquisition sequence is useful for applications where the signal region of interest occurs long after the trigger event, as is often the case for Time Domain Reflectometry methods such as ultrasonics, radar and lidar.

The third sequence shows the acquisition in the case of a rapidly occurring trigger. In this case, the Segment size has been set bigger than the Depth. However, the trigger event occurred so rapidly after the start of the acquisitions that there was insufficient time to fill the memory allotted for pre-trigger data, which contains (Segment Size – Depth) samples. Consequently, there is less preceding valid trigger data than the maximum amount of pre-trigger data available. If transfer of invalid data were requested, the driver would return an actual start address that is equal to the address of the first valid pre-trigger point. More pre-trigger data could have been acquired using Trigger Holdoff at the expense ignoring and so possibly missing triggers.

The final sequence shows acquisition in the case of a trigger that occurs long after Segment Start so that all memory allotted for pre-trigger data has been filled up. In fact, the sequence shows that still more pre-trigger data were acquired but were overwritten by post-trigger data.



Trigger hold-off plus natural pre-trigger
Segment size
Overwritten data
Pre
Trigger delay
Depth
Start capture
Post-trigger data
General format of the segment
Trigger hold-off plus natural pre-trigger
Segment size
Overwritten data
Trigger delay
Depth
Start capture
Post-trigger data
Usually if trigger delay is not zero
Segment size = Depth
Trigger
Trigger hold-off plus natural pre-trigger
Segment size
Pre
Depth
No data
Start capture
Post-trigger data
Amount of pre-trigger data due to hold off and natural pre trigger is less then Segment size - Depth
Trigger hold-off plus natural pre-trigger
Segment size
Overwritten data
Pre
Depth
Start capture
Post-trigger data
Amount of the pre-trigger data is equal or bigger then Segment Size - Depth

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:

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

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 (i.e. CD-ROM, joystick, network card, etc.)

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

GaGe Products

For ordering information, see Gage's product catalog, or visit our web site at http://www.gage-applied.com

CompactPCI Bus Products	CompuScope 82GC	8 bit, 2 GS/s Analog Input Card
	CompuScope 14100C	14 bit, 100 MS/s Analog Input Card
	CompuScope 1610C	16 bit, 10 MS/s Analog Input Card
	CompuScope 3200C	32 bit, 100 MHz Digital Input for CompactPCI Bus
PCI Bus Products	CompuScope 1610	16 bit, 10 MS/s Analog Input Card
	CompuScope 1602	16 bit, 2.5 MS/s Analog Input Card
	CompuScope 14200	14 bit, 200 MS/s Analog Input Card
	CompuScope 14105	14 bit, 105 MS/s Analog Input Card
	Octopus multi-channel digitizer family	Up to 8 channels on a single-slot PCI card, 12 or 14-bit resolution, 10 to 125 MS/s
	CompuScope 12400	12 bit, 400 MS/s Analog Input Card
	CompuScope 1220	12 bit, 20 MS/s Analog Input Card
	Cobra CompuScope family	8-bit, up to 2 GS/s Analog Input Card
	BASE-8 CompuScope	8-bit, up to 500 MS/s Analog Input Card
	CompuScope 3200	32 bit, 100 MHz Digital Input for PCI Bus
CompuGen PCI	CompuGen 4300/4302	12 bit, 4-channel, 300 MHz Analog Output Card
	CompuGen 8150/8152	12 bit, 8-channel, 150 MHz Analog Output Card
	CompuGen 11G/11G2	12 bit, 1 GHz Analog Output Card
CompuGen ISA	CompuGen 1100	12 bit, 80 MS/s Analog Output Card
	CompuGen 3250	32 bit, 50 MHz Digital Output Card
Application Software	GageScope	World's Most Powerful Oscilloscope Software
	GageBit	Digital Input/Output Software
	CompuGen for Windows	Arbitrary Waveform Generator Software for Windows
Software Development Kits	CompuScope SDK for C/C#	
	CompuScope LabVIEW SDK	
	CompuScope MATLAB SDK	
	CompuGen SDK for C/C++	
	CompuGen SDK for LabVIEW	
	CompuGen SDK for MATLAB	
Instrument Mainframes	LapScope-1	1-slot PCI Expansion Chassis for CompuScope and CompuGen cards
	LapScope-2	2-slot PCI Expansion Chassis for CompuScope and CompuGen cards
	Instrument Mainframe 7500	4-slot Portable Instrument Mainframe for CompuScope and CompuGen cards
	Instrument Mainframe 2020E	18 PCI slot and 1 ISA slot Instrument Mainframe for CompuScope and CompuGen cards
	Instrument Mainframe 4000	5 PCI-X and 1 PCI slot Instrument Mainframe for CompuScope and CompuGen cards
	Instrument Mainframe 8000C	7 cPCI/PXI slot Instrument Mainframe for CompuScope CompactPCI/PXI cards