

WIND RIVER

Wind River® StethoScope® for VxWorks®

USER'S GUIDE

7.8

Windows Version

Copyright © 2005 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, the Wind River logo, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

<http://www.windriver.com/company/terms/trademark.html>

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation at the following location:
installDir\product_name\3rd_party_licensor_notice.pdf.

Wind River may refer to third-party documentation by listing publications or providing links to third-party Web sites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

Corporate Headquarters

Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

<http://www.windriver.com>

For information on how to contact Customer Support, please visit the following URL:

<http://www.windriver.com/support>

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Overview	2
1.3	Features	2
1.4	Architecture	3
1.4.1	Host	4
1.4.2	Target	4
1.5	Reader's Guide	5
2	Getting Started	7
2.1	Introduction	7
2.2	Installation	8
2.3	License Manager	8
2.4	Getting Help	8
2.5	Starting StethoScope	9
2.5.1	Initializing the Target Server	9

2.5.2	Starting StethoScope On Your Host	9
	Starting Automatically from the Workbench IDE Toolbar	9
	Starting Manually from the Command Line	10
2.5.3	Concepts of Use	12
	Signals Definitions	13
2.6	Data-Display Windows	14
2.6.1	Plot Window	14
2.6.2	Plot XY Window	14
2.6.3	Dump Plot Window	15
2.6.4	Monitor Window	15
2.6.5	Signals Bar	15
2.6.6	Mini-Dump Window	16
2.6.7	Mini-Monitor Window	16
2.6.8	Common Window Elements	17
2.7	Running the Demonstration Program	18
2.7.1	VxWorks Target-Based Demonstration	19
	Viewing the Signals	22
2.7.2	What Does the Demo Do?	22
	Automatic Signal Management from the VxWorks Shell	23
3	StethoScope Features	25
3.1	Introduction	25
3.2	File Menu	26
3.2.1	Connect to Target	26
3.2.2	Load Snapshot	27
3.2.3	Save Snapshot	27
3.2.4	Load Config	27
3.2.5	Save Config	28

3.2.6	Plots	30
	Dump Plot	31
	Plot	31
	Monitor	31
	Plot XY	31
3.2.7	Signal Manager	32
3.2.8	Triggering	33
3.2.9	XY Signals	34
3.2.10	Derived Signals	35
3.2.11	Trace Log Window	36
3.2.12	Preferences	36
	General View	37
	Colors View	38
	Comm Plug-ins View	39
	Plot Plug-ins View	40
	Dump Plot View	41
	Plot View	41
	Monitor View	42
	Plot XY View	43
3.2.13	Close Window	43
3.2.14	Exit StethoScope	43
3.3	Other Menus	44
3.3.1	Plot Menu	44
3.3.2	View Menu	45
3.3.3	Window Menu	47
3.3.4	Help Menu	48
3.4	Toolbars	48
3.4.1	Main Toolbar	48
3.4.2	Plot Toolbar	48
3.4.3	Plot Window Toolbar	50
3.5	Status Bar	52

3.6	Pop-up Menus	52
3.6.1	On-Grid Pop-up Menu	52
	Zooming	54
	Adding and Removing Markers	54
	Adding Annotations	55
	Panning	56
	Taking and Removing On-grid Measurements	56
3.6.2	Trace Pop-up Menu	57
3.6.3	Signals Tree Pop-up Menu	57
3.6.4	Legend Pop-up Menu	58
3.7	StethoScope's Initialization Sequence	60
4	Using the Signal Manager	61
4.1	Introduction	61
4.2	The Signal Manager Window	62
4.3	Working With Signal Trees	62
4.4	Signal Installation	64
4.5	Disconnecting from the Target	64
5	Triggering	65
5.1	Introduction	65
5.2	Configuring a Trigger	66
5.3	Triggering Dialog Box	66
5.3.1	Target	67
5.3.2	Start Condition	68
5.3.3	Trigger Status	69
5.3.4	Stop Condition	69
5.3.5	Options	70

5.3.6	Buttons	71
5.4	Setting a trigger	71
5.5	Understanding and Preventing Overflows	73
5.5.1	Avoiding Overflows	73
5.5.2	Overflow Behavior	74
5.6	Notes and Hints	74
6	Derived Signals	77
6.1	Introduction	77
6.2	Creating Derived Signals	78
6.3	Mathematical Operations	81
6.4	Troubleshooting Derived Signals	83
7	The Plot Window	85
7.1	Introduction	85
7.2	Plot Window Tour	86
7.2.1	Displaying Signal Values in a Plot Window	87
7.3	Menu Bar	87
7.3.1	Plot Menu	88
7.3.2	View Menu	89
7.4	Toolbar	91
7.5	Signals Bar	92
7.5.1	Signals Tab View	92
7.5.2	Legend Tab View	94
7.5.3	Properties Tab View	95

7.6	Signal Properties Dialog Box	98
7.7	Pop-up Menus	103
7.8	Strip Chart Feature	103
7.9	Displaying Events	103
7.9.1	Events Collected as Signals	104
7.9.2	Events Collected as Markers	104
7.9.3	Events Collected as Messages	105
7.10	Setting Preferences for a New Plot Window	106
8	The Plot XY Window	111
8.1	Introduction	111
8.2	Creating XY Signal Pairs	112
8.2.1	Creating a Signal Pair	112
8.2.2	Deleting a Signal Pair	113
8.2.3	Modifying a Signal Pair	113
8.3	Plot XY Window Tour	113
8.3.1	Displaying Signal Values in a Plot XY Window	114
8.4	Menu Bar	115
8.4.1	Plot Menu Commands	115
8.4.2	View Menu Commands	116
8.5	Toolbar	117
8.6	Signals Bar	118
8.6.1	Signals Tab View	119
8.6.2	Legend Tab View	120
8.6.3	Properties Tab View	121

8.7	Signal Properties Dialog Box	123
8.8	Pop-up Menus	127
8.9	Setting Preferences for a New Plot XY Window	128
9	The Dump Plot Window	131
9.1	Introduction	131
9.2	Dump Plot Window Tour	132
9.2.1	Displaying Signal Values in a Dump Plot Window	132
9.3	Menu Bar	133
9.3.1	Plot Menu Commands	133
9.3.2	View Menu Commands	134
9.4	Toolbars	135
9.5	Signals Bar	136
9.5.1	Signals Tab View	136
9.5.2	Properties Tab View	137
9.6	Setting Preferences for a New Dump Plot Window	138
10	The Monitor Window	141
10.1	Introduction	141
10.2	Monitor Window Tour	142
10.2.1	Displaying Signal Values in a Monitor Window	143
10.3	Menu Bar	143
10.3.1	Plot Menu Commands	143
10.3.2	View Menu Commands	144
10.4	Toolbar	145

10.5	Signals Bar	146
10.5.1	Signals Tab View	146
10.5.2	Properties Tab View	147
10.6	Writing Data to the Target	149
10.6.1	Writing Data to the Target for a Selected Signal	149
10.7	Setting Preferences for a New Monitor Window	149
11	Working with Snapshots	151
11.1	Introduction	151
11.2	Taking Snapshots	152
11.2.1	What Happens When You Take a Snapshot?	152
11.3	Saving Snapshots	153
11.3.1	Snapshot	155
11.3.2	Data	155
11.3.3	Output	156
	How Output Filenames Are Built	157
11.4	Loading Snapshots	157
11.5	Exporting Snapshot Data to MATLAB and MATRIXX	158
11.5.1	Creating Notes	158
11.5.2	Creating Variables	159
11.5.3	MATLAB Script Example	160
11.6	Deleting Snapshots	161
12	Using a VxWorks Target	163
12.1	ScopeProbe Requirements	163
12.2	VxWorks Targets	164

12.2.1	Building	164
12.2.2	Automatic Loading and Running	165
	Loading and Starting Automatically	165
	Verifying Target Initialization	168
	Verifying Target Connection	168
12.2.3	Manual Target Loading and Running	169
	Loading the Wind River Utilities Library	169
	Loading the ProfileScope Library	170
	Loading the Demo Library	170
	Starting the Sampler Task	171
12.2.4	Example Target Script	171
12.2.5	Starting the ProfileScope GUI Manually	172
12.3	StethoScopeTroubleshooting	172
12.3.1	Load Errors	172
12.3.2	Connection Failure	172
12.3.3	No Response from Target	173
	Multiple Connections	173
	Starvation	173
	Network Configuration	173
	Version Mismatch	173
	None of the Above	174
12.3.4	No Data	174
	Sampling	174
	Triggering	174
	Starvation	174
	None of the Above	175
13	Signal Installation	177
13.1	Installing Signals for StethoScope	177
13.2	Automatic Signal Installation	179
13.2.1	Requirements	179
13.2.2	Signal Installation Dialog Box	179

13.2.3	Batch Signal Installation	183
	Controls	184
13.3	Manual Installation	184
13.4	Using StethoScope API	185
13.5	Code Instrumentation Alternative	185
13.6	Process Notes	185
13.6.1	Variable Expressions vs. Signal Names	186
13.6.2	Hierarchical Signal Names	187
13.6.3	Classes and Structures	187
13.6.4	Libraries	188
14	API Introduction	189
14.1	Introduction	189
14.2	Using StethoScope API with Your Program	190
14.3	Initializing the Server	190
14.3.1	Scope Index	190
14.3.2	Target Buffers	191
14.4	Registering and Activating Signals	191
14.4.1	Installing Signals	192
14.4.2	Hierarchical Naming of Signals	193
14.4.3	Pointers to Signals	193
14.4.4	Offsets to Signals	194
	Example Registration With Offset	194
	Calculating Offsets	194
14.4.5	Installing Signals	195
14.4.6	Deactivating and Removing Signals	196
14.4.7	Online Documentation	197

14.5	Sampling Signals	197
14.5.1	Asynchronous Sampling	197
14.5.2	Synchronous Sampling	199
14.5.3	Sample Rate	199
14.6	Triggering and Sampling Functions	200
14.7	StethoScope Events API	200
14.7.1	Setting Up the StethoScope Events API	201
14.7.2	Using the StethoScope Events API	201
	Signals vs. Events	203
14.8	scope.ini File	204
A	StethoScope API Reference	207
B	StethoScope Demonstration	237
B.1	Introduction	237
B.2	Source-code Example: vxdemo.c	237
B.2.1	Source Code for vxdemo.c	238
B.2.2	Makefile for vxdemo.c	246
C	MATLAB and MATRIXX Examples	249
C.1	Introduction	249
C.2	MATLAB Example	249
C.3	MATRIXX Example	253
D	Glossary	259
	Index	261

1

Introduction

- 1.1 Introduction 1
- 1.2 Overview 2
- 1.3 Features 2
- 1.4 Architecture 3
- 1.5 Reader's Guide 5

1.1 Introduction

This chapter introduces you to the Wind River StethoScope real-time graphical monitoring tool for VxWorks targets that allows you to analyze your real-time application while it is running.

StethoScope is the user-friendly, real-time graphical-monitoring and data-collection tool from Wind River. It lets you monitor and analyze the values of variables in your real-time application while the application is running. StethoScope is more than an easy-to-use data-collection tool—it is a powerful debugging aid for both hardware and software. You can use its multi-window environment to track down performance problems, “glitches,” and program errors.

1.2 Overview

Anyone who has developed real-time systems knows that getting the code written and compiled is only the first step. You still have to make it work. Where is the noise coming from? How full is the buffer? When did that valve open? What are the best parameters? Why did it do that? Understanding the system is the real challenge.

StethoScope gives you a window into the very heart of your application. It presents a live analysis of your program while preserving real-time performance. You can immediately see the effects of code changes, parameter changes, or external events. StethoScope will quickly become your most valuable diagnostic tool. With StethoScope, you can make your system work.

StethoScope is a licensed product whose licensing is enforced by a license manager. The license-manager software must be running, using a valid license file that contains the proper keys for StethoScope. See the *Wind River Installation and Licensing User's Guide* for details on setting up the license manager and obtaining the necessary keys.

Note that if the **Check Out License** icon is enabled, and you try clicking the icon with no results, or if you have any other licensing problems or issues, please contact your Wind River Technical Support Team.

1.3 Features

Real-time Graphical Display

StethoScope's full-color, real-time graphical displays let you watch your program execute. Multiple windows can be open at the same time, displaying a rich mixture of signals and functionality.

Minimal Intrusion

StethoScope does not impact your real-time system's performance. Collection is very fast; data transfer takes place in the background at low priority.

Modify Data

StethoScope can also modify program variables. Experiment quickly, isolate problems, and run test cases by changing the value of variables and parameters while your program executes.

Dynamic Signal Installation

Install variables by name as your program runs, including **structs**, **classes**, and **unions**, by simply typing in the name of the variable you want to view.

Data Storage

StethoScope exports data in many formats. It is organized (each run is timestamped and labeled with signal names and units), and accompanied by your notes. You can choose which data to save, or have StethoScope save them automatically.

Support for Large Systems

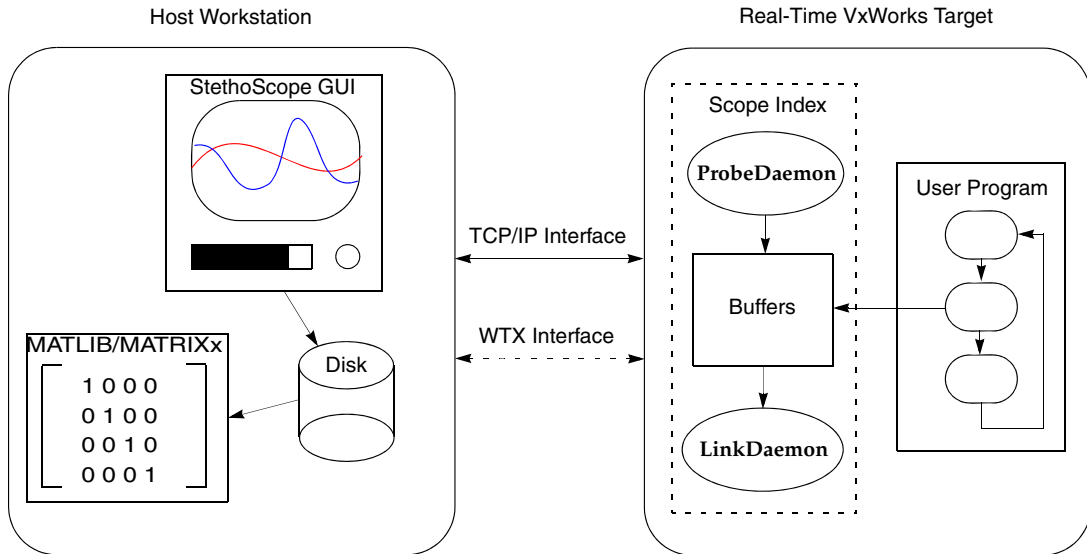
With this program, you can register literally hundreds of variables for monitoring. You can collect any subset of the registered variables, and organize your variables with a powerful hierarchical tree browser.

Type Support

StethoScope supports many data types without loss of precision. This includes support for all common data types—from one-byte char to eight-byte double, support for pointers and structures to make it easier to monitor complex data structures, user-defined buffers, and support for hexadecimal data display.

1.4 Architecture

The figure below shows an overview of StethoScope's run-time architecture for VxWorks.



StethoScope consists of two distinct modules:

- A multi-window graphical user interface (GUI) that runs on the host (PC or workstation), providing dynamic views of your run-time data.
- A real-time data collection module—named **ProbeDaemon**—that is loaded onto the target processor with your application code. **ProbeDaemon** collects and buffers time histories of variables in your program before sending them to the host for display. The target processor could be running a real-time operating system (such as VxWorks) or user operating system (such as Windows or Solaris). It can run on the same processor as the one running the StethoScope GUI or a remote processor with a network connection.

1.4.1 Host

On the host, the StethoScope GUI allows you to view the data interactively as it is received. You can save the data on disk for later off-line analysis. Data can also be exported in a variety of formats.

1.4.2 Target

The application task, running on the target module, is typically a user program, but it also can be a system process, especially with a real-time operating system. This task collects the data.

StethoScope supports the TCP/IP and WTX modes of data transfer. In both modes, two additional low-priority tasks (threads), are in charge of transferring data to the host:

ProbeDaemon

The **ProbeDaemon** receives and processes commands from the host user interface. Through the **ProbeDaemon** interface, the target application can:

- Install variables to monitor.
- Change sample rates.
- Set triggers.
- Collect data.

LinkDaemon

This task transfers data to the host. During program execution, data is collected and placed in a local buffer. This data collection is very fast and is the *only* action that takes place at high priority (that is, at the priority of your application or an asynchronous sampling task). The **LinkDaemon** task, running at very low priority, then takes the data from the buffer and sends them to the host.

The **LinkDaemon** stores the list of signals and the collected data samples on the target (see [14. API Introduction](#) for details). These are sent to the host to be displayed at a later time.

This mechanism is designed for minimal impact on real-time system performance.

1.5 Reader's Guide

This manual is organized into chapters as follows:

- [1. Introduction](#) discusses StethoScope's general structure and capabilities.
- [2. Getting Started](#) provides an overview of the host-side graphical user interface (GUI).
- [3. StethoScope Features](#) introduces you to StethoScope's many features.

- [4. Using the Signal Manager](#) talks about the signal manager and its features.
- [5. Triggering](#) tells you how to set up triggering.
- [6. Derived Signals](#) describes how create and display derived signals.
- [7. The Plot Window](#) provides details on the GUI's **Plot** window, which is the window opened automatically when you run StethoScope. This window plots signals against time.
- [8. The Plot XY Window](#) provides details on the GUI's **Plot XY** window, used for plotting pairs of signals.
- [9. The Dump Plot Window](#) provides details on the GUI's **Dump Plot** window, used to display the history of signal values in a table.
- [10. The Monitor Window](#) provides details on the GUI's **Monitor** window, used to display the current value of selected signals in a table. You can also use this window to modify signal values on the target.
- [11. Working with Snapshots](#) describes how to take and use snapshots of signals.
- [12. Using a VxWorks Target](#) tells how to build and run a VxWorks target application.
- [13. Signal Installation](#) describes how to install signals from a VxWorks target.
- [A. StethoScope API Reference](#) provides detailed reference data on StethoScope's API library.
- [B. StethoScope Demonstration](#) lists the source code for the demonstration program.
- [C. MATLAB and MATRIXX Examples](#) lists example MATLAB and MATRIX_X programs that plot signals saved by StethoScope.
- [D. Glossary](#) gives a list of common terms used in this manual.

2

Getting Started

- 2.1 Introduction 7
- 2.2 Installation 8
- 2.3 License Manager 8
- 2.4 Getting Help 8
- 2.5 Starting StethoScope 9
- 2.6 Data-Display Windows 14
- 2.7 Running the Demonstration Program 18

2.1 Introduction

This chapter describes the main features of Wind River StethoScope's multi-window graphical user interface (GUI). It presents a brief overview of the four distinctive data-display windows: **Plot**, **Plot XY**, **Dump Plot**, and **Monitor**, describing their many unique features and commands, as well as those they have in common. Each of these data-display windows is described in detail in later chapters.

StethoScope is designed to be easy to use. We suggest you quickly skim the first section and then run the demonstration, using the instructions in [2.2 Installation](#), p.8. Once the installation is complete, this manual serves as a reference manual.

2.2 Installation

Detailed instructions for installing StethoScope can be found in the *Wind River Installation and Licensing User's Guide*. If you have any difficulties with either installing or using the StethoScope application please contact your Wind River Technical Support Team.

2.3 License Manager

StethoScope is a licensed product whose licensing is enforced by a license manager. The license-manager software must be running, using a valid license file that contains the proper keys for StethoScope. Please consult the *Wind River Installation and Licensing User's Guide* for details on setting up the license manager and license keys.

2.4 Getting Help

StethoScope comes with the following types of documentation:

- The StethoScope manual in PDF format.

The StethoScope manual is available as a PDF file to view and print from the Adobe Acrobat Reader (<http://www.adobe.com>). The PDF files are located in the directory under which StethoScope is installed.

2.5 Starting StethoScope


This section describes how to begin using StethoScope in a real environment. (2.7 *Running the Demonstration Program*, p.18 shows you how to run the demonstration program.)

2.5.1 Initializing the Target Server

To run StethoScope, you must first initialize the target server. You then need to load the *daemon* libraries and initialize the target by a call to **ScopeInitServer()**. This is followed by installing the signals you want to watch. A complete description of this process is given in 12. *Using a VxWorks Target*.

2.5.2 Starting StethoScope On Your Host


StethoScope can be started in either of two ways:

1. Automatically by clicking  on the Workbench IDE.
2. Manually on the host from a command line window.

Starting Automatically from the Workbench IDE Toolbar

If you installed StethoScope under Workbench, you can simultaneously and automatically load and start StethoScope from the Workbench IDE window by clicking either the **StethoScope** or the **Demo** button. The StethoScope installation places both buttons on the Workbench IDE toolbar, as shown (circled) below.

To launch StethoScope from the Workbench IDE window and connect to your target server:

1. Click the  button to launch StethoScope. The **StethoScope Setup Options** dialog box opens.
1. If StethoScope is already running and you do not want to restart it (but only load libraries), deselect **Start StethoScope GUI** on the host button.

2. Choose **Spawn Sampler Task**, to perform asynchronous signal sampling, or **Load Libraries Only** to only load the target libraries.
3. Enter a value from 0 to 127 for the **Scope Index**, and optionally enter an IP address for the target.
4. Check the **Use WTX** (not TCP/IP) check box if you want to set up a WTX connection only.



NOTE: TCP/IP communication will only work if you use ScopeAPI in an RTP.

5. If you selected **Load Libraries Only**, make any desired modifications to the default sampling parameters. For a detailed description of the remaining parameter settings, see [Loading and Starting Automatically](#), p.165.
6. Click **OK** to execute the selected activity.

The StethoScope button on the Workbench IDE toolbar automatically connects to the currently selected **target@tgtsvrHost** in the drop-down menu on the toolbar. For more details on launching StethoScope, see [12. Using a VxWorks Target](#).

Starting Manually from the Command Line

All the same processes that are run automatically for you when you start StethoScope from the Workbench IDE can be done manually in a **Host Shell** window. The executable file for StethoScope is called **scope.exe**.

Important: Before entering any other commands in the **Host Shell**, type:

```
run wrenv -p vxworks-6.1
```

This will properly set up the environment variables to allow you to start StethoScope using the **scope** command described below.

The full command line options for **scope.exe** are:

```
scope [-target target] [-index n] [-verbosity level]
      [-errorlog filename] [-clicense licenseHost]
      [-save save.ssc] [-load save.ssc]
      [-Version ] [-help ]
      [-tgtsvr targetServer]
      [-wtxMode]
```

where the parameters have the following meanings:

-target target

Connects to the StethoScope API running on the machine named *target*, where *target* can be an IP address or a target name that can be resolved to an IP

address. The **-target** string is optional. If no target is specified at all, the user can choose to connect to a target at a later time.

If the target is a VxWorks target, make sure it is listed in the **HOSTS** file of your host machine. For example, in Windows NT/2000/XP, the file is:

```
c:\winnt\system32\drivers\etc\HOSTS
```

-index *n*

Connects to the target using a specific StethoScope channel. The index may be an integer ranging from 0 to 127. If this option is not specified, StethoScope uses 0. A target name of form *target:n* is equivalent to **-ta target -i n**.

For example: the following are equivalent:

```
C:\scope -ta joshua -i 1  
C:\scope -ta joshua:1
```

-verbosity *level*

Specifies the amount of diagnostic messages printed to the standard-output device. A value of 0 causes only errors to be reported. Increasing the value (in the range of 0 - 3) increases the volume of messages. If this option is not specified, StethoScope uses 0.

-errorlog *filename*

Writes verbosity messages into the file, *filename*, in addition to outputting to the log window.

-license *licenseHost*

Specifies name of a license file or the host name of the machine running the *Wind River License Manager*. If this option is omitted, StethoScope uses the following to obtain a valid license:

- Look for **flex_license.dat** in the directory above **STETHOSCOPEHOME**, typically **c:\rti**.
- Look for host machine specified by the **WR_LICENSE_FILE** environment variable.
- Look for **flex_license.dat** in the directory specified by the **RTIHOME** environment variable.
- Examine each file in the semicolon-separated list specified by the **LM_LICENSE_FILE** environment variable.

-save *save.ssc*

Automatically saves the workspace state in the file, **save.ssc**.

-load *save.ssc*

Reads the state saved in the file, **save.ssc**.

-Version

Prints out the current StethoScope version.

-help

Prints out the information described above.

-tgtsvr *targetServer*

When connected to a VxWorks target, specifies the WTX target-server name that manages the target. This enables the **Signal Installation** window of StethoScope to access the target via WTX as well as the Workbench debugger to install signals automatically.

-wtxMode

Specifies that the data should be collected by the StethoScope GUI using WTX protocol rather than TCP/IP. This option may not be abbreviated.



NOTE: TCP/IP communication will only work if you use ScopeAPI in an RTP.

Most parameters may be abbreviated to a single letter or to as many letters as it takes to make it unique. The exceptions are noted in the descriptions above. For example, the following commands are equivalent:

```
C:\scope-> -target joshua -errorlog err.txt -verbosity 2
C:\scope-> -ta joshua -e err.txt -v 2
```

Important: your **PATH** environment variable must be set up correctly to run tools before attempting to run StethoScope .

After starting StethoScope, the GUI should appear on your screen.

2.5.3 Concepts of Use

There are four basic steps to using StethoScope's GUI on the host to monitor and collect data from your target application.

1. Use the StethoScope API interface on the target to specify which data you want to be able to monitor and collect—these are known as *installed signals*. Only *installed signals* (for example, signals that are registered and activated, see [Signals Definitions](#), p. 13) can be collected and monitored by StethoScope on the host. Signals can be installed manually, or automatically using the mechanism described in [13.1 Installing Signals for StethoScope](#), p.177.

2. Bring up StethoScope's GUI on the host and connect to the target. You can connect to more than one target at a time. Use the **File > Connect to Target** menu command (see [3.2.1 Connect to Target](#), p.26).
3. Use StethoScope's **Signal Manager** (see [4. Using the Signal Manager](#)) to specify which installed signals you want to collect from the target.
4. Use StethoScope's **Plot**, **Plot XY**, **Dump Plot**, and **Monitor** windows to display signals in tables and graphs (see Chapters 7, 8, 9, and 10, respectively). With the **Monitor** window, you can write modified signal values back out to the target. The **Plot** and **Plot XY** windows also allow you to capture and display snapshots.

Signals Definitions

Registered Signals

Initially you must let StethoScope know a signal exists by *registering* it using the API call **ScopeRegisterSignal()**. StethoScope cannot collect data from this signal until you Activate it. Registered signals appear in the **Signal Manager** window in the GUI, where they can be selected for activation.

Activated (or Active) signals

These are registered signals that are set up on the host by the Signal Manager (see [3.2.7 Signal Manager](#), p.32) using the API call **ScopeActivateSignal()**. Active signals then appear in the **Signals Bar** of each data display window (see [2.6.5 Signals Bar](#), p.15). Once activated, they are considered to be **Installed** signals and are automatically collected from the target, but they are not yet displayed in the host GUI until **Selected** in one or more of the four data display windows: **Plot**, **Plot XY**, **Dump Plot**, and **Monitor**.

Installed signals

These are signals that are **registered** and **activated**. (They can be set up using the StethoScope API shortcut **ScopeInstallSignal()**. See [13.1 Installing Signals for StethoScope](#), p.177). A signal must be installed for the StethoScope GUI to "see" it.

Selected signals

Installed signals are not displayed automatically in the GUI. You must select, from the GUI, the installed signals you want to display in the data-display windows—**Plot**, **Plot XY**, **Dump Plot**, and **Monitor**. You can select a different set of signals in each window (see [2.6.5 Signals Bar](#), p.15).

2.6 Data-Display Windows

StethoScope has four unique types of data-display windows you can use to display signal values in graphical and tabular form. By default, StethoScope starts with a **Plot** window displayed, but you can open any of the other window types from the **File** menu or from the **Plots** toolbar (see [3.2 File Menu](#), p.26).

You can have multiple data-display windows of each type open at the same time. Each data-display window is independent of the others, so you can display different sets of signals in each window.

The mini windows, panels, toolbars, and menu bar in these windows are all *dockable*, meaning you can drag them to different locations, on or off the window. You can also cause StethoScope to save and restore their positions on future sessions (see [3.3 Other Menus](#), p.44 and [3.4 Toolbars](#), p.48).

2.6.1 Plot Window

The **Plot** window is the heart of the StethoScope application. You can use this window to select which signals to plot, then see a color-coded plot of your selected signals over time. You can also take snapshots of plots and display them along with real-time plots for easy visual comparisons. The **Plot** window also includes mini versions of the **Dump Plot** and **Monitor** windows (described below). A **Plot** window, as shown in the figure below, is displayed when you first launch StethoScope. See [7. The Plot Window](#), for a detailed description of the **Plot** window.

2.6.2 Plot XY Window

While the **Plot** window graphs each selected signal (on the Y axis) over time (the X axis), the **Plot XY** window, shown below, plots *pairs* of selected signals against each other: one signal on the X axis and the other on the Y axis. More than one pair of signals may be displayed at the same time. You can also take snapshots of XY plots and display them along with real-time plots for easy visual comparisons. See [8. The Plot XY Window](#), for details of the **Plot XY** window.

2.6.3 Dump Plot Window


The **Dump Plot** window, shown in the figure below, is used to monitor the real-time values of selected signals as they are collected from the target. The **Dump Plot** window is a simple read-only table. The first column in the table is a **Timestamp**, followed by a column for each selected signal. See [9. The Dump Plot Window](#) for a detailed description of the **Dump Plot** window.

2.6.4 Monitor Window

The **Monitor** window, shown in the figure below, is used to display the current value of selected signals. While the **Dump Plot** window displays a running history of each selected signal, the **Monitor** window only shows you the last sampled value of each selected signal. You can also use this window to modify the values of signals on the target. See [10. The Monitor Window](#) for a detailed description of the **Monitor** window.

Within each data display window there are other sub-windows that display information for signal selection, and for augmenting your view of what is happening inside your target program. The descriptions of these additional sub-windows follows.


2.6.5 Signals Bar

By default, each of the four data-display window types contains a **Signals Bar** panel. If a **Signals Bar** panel is not displayed, you can open one using the **View, Signals Bar** menu command (). The **Signals Bar** panel has three tabs, **Signals**, **Legend**, and **Properties** as shown below.


- The **Signals** tab is used to select which signals to monitor in the current window. It presents a signals tree, which gives you a tree-like view of the active signals for each connected target and any snapshots you have loaded. Use the check box preceding each signal to select which signals you want to display in the current window. Signals trees are described in [4. Using the Signal Manager](#).

- The **Legend** tab view shows the colors assigned to each selected signal. It also displays the source of each signal (**Live** or **Snapshot**), and the target IP address. See [7.5.2 Legend Tab View](#), p.94 in the *StethoScope User's Manual* for information on using the **Legend**.
- The **Properties** tab allows you to control how the signals are monitored and displayed in the window. Each type of data-display window has different properties, which are described in each window's respective chapter (Chapters 7, 8, 9, and 10). See [7.5 Signals Bar](#), p.92 for information on using the **Signals Bar**.

2.6.6 Mini-Dump Window

Within the **Plot** window only (see [7. The Plot Window](#)), the **Mini-Dump** window, shown in the figure below and opened using the **View > MiniDump** menu command (or the  button), creates a scaled down version of the **Dump Plot** window described in [9. The Dump Plot Window](#). Like the **Dump Plot** window, it lets you see a running history of selected signal values, scrolling through the window with time. This mini-window initially appears at the bottom of the **Plot** window, allowing you to see numeric signal values along side the plotted signals graph. You can drag the window to any other location you wish, and it will remain there until you change its location again.

2.6.7 Mini-Monitor Window

Like the **Mini-Dump** window described above, the **Mini-Monitor** window, shown in the figure below and opened in the **Plot** window only using the **View > MiniMonitor** menu command () , creates a scaled down version of the **Monitor** window described in [10. The Monitor Window](#). This window lets you see, and modify, target data in a static but dynamically updated list format. The modify feature (called **writeback**) is described in detail in [10.6 Writing Data to the Target](#), p.149. Like the **Mini-Dump** window, it also appears as a sub-window within the **Plot** window.

2.6.8 Common Window Elements

Each data-display window has certain items that are common across all the data-display window types. Some of these items are always displayed, while others can have their display toggled on or off. Some items can even be further customized.

The common window elements are:

Title Bar

This bar, on all data-display windows, indicates the name of the tool (StethoScope), its version, and the data-display window name.

Menu Bar

This bar is available for display in each of the data display windows. The **Menu Bars** are described in detail in [3. StethoScope Features](#).

Toolbar

There are three toolbars in each window. These toolbars are described in detail in [3. StethoScope Features](#).

- **Main** toolbar has buttons for many of the **File** menu commands. It has the same buttons on all four data display windows.
- **Plots** toolbar has a button to open each of the data-display windows. It has the same buttons on all four data display windows.
- **Plot Windows** toolbar has different buttons for each unique data-display window. Since the **Plot Windows** toolbar buttons are different for each data display window type, they are described in greater detail separately in each data-display window section (Chapters 7-10).

The three toolbars described above initially appear lined up, left to right, just below the **Menu Bar**, and separated by the docking handles at the left end of each toolbar. You can move each toolbar around to any location in the data display window, including vertical placement, by left-clicking the docking handle and dragging to a new location. It will remain in that location until you change it again.

Status Bar

This bar is common to all four data-display windows, but has minor variations for the different data-display windows. It is therefore described in detail separately in each data-display window section.

2.7 Running the Demonstration Program

A simple demonstration target program that exercises many of StethoScope's features is included in the StethoScope distribution. To quickly and easily become familiar with StethoScope, we strongly encourage you to run the demo program. Additional basic concepts, on which StethoScope is based, will be emphasized in the process of guiding you through the demo program's steps. The demo program will also utilize the StethoScope API to log program behavior.

To start the demo program, do the following:

1. Copy the target binaries from your host, at

installDir/target/arch/target arch

where *installDir* is the directory in which you installed the Wind River ScopeTools, and *target arch* is the directory specific to your target architecture, to the corresponding location on your target.

2. Start the demo program by typing:

Sp ScopeDemo


```

Telnet 10.30.68.144
Linux 2.6.10 (wrsbc8260-1) (1)
wrsbc8260-1 login: root
Password:
Wind River GPP-LE 1.1 Linux 2.6.10 - wrs_sbc8260 RTM
Last login: Tue Jan 6 06:25:04 1970 from 10.30.68.13 on pts/0
root@wrsbc8260-1:~> cd /home/grand/scopetools
root@wrsbc8260-1:scopetools> ls
KAL.ko          libscope78tcp.so    processsampler
MemAgent        libscope78tcpz.a    profileModule.ko
MemScope.so     libutilsip.so       profileModule.ko_no_kgdb
MemScopeModule.ko libutilsipz.a       profiledemo
MemScopeModule.ko_no_kgdb libxmlparsez.a      scopedemo
ProfileAgent    memrun
aledaemon       memscopededemo
root@wrsbc8260-1:scopetools> ./scopedemo

Scope index <0> successfully initialized!
Sample rate is 30.00 Hz

StethoScope demonstration target simulator.
*****


Real-Time Innovations, Inc.

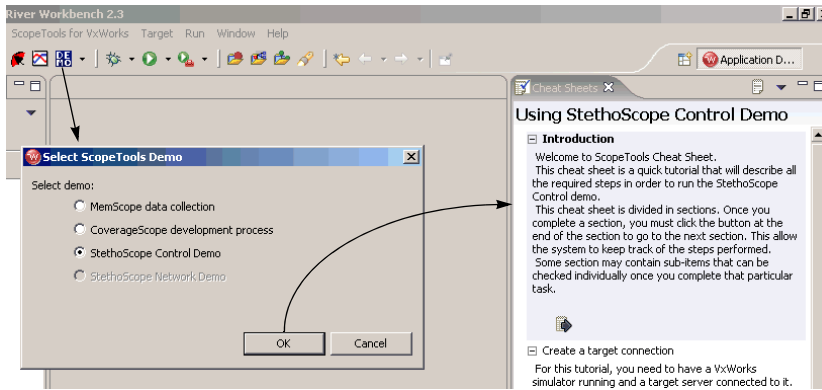
Scope index <0>
-----
h - Print menu
a - Add signals
r - Remove signals
v - Activate signals
w - Deactivate signals
d - Display registered signals
i - Display activated signals
p - Change position step size
l - Change length of step
n - Add noise
R - Change sampling rate
e - Throw event
q - Quit
Scopedemo =? [h]


```

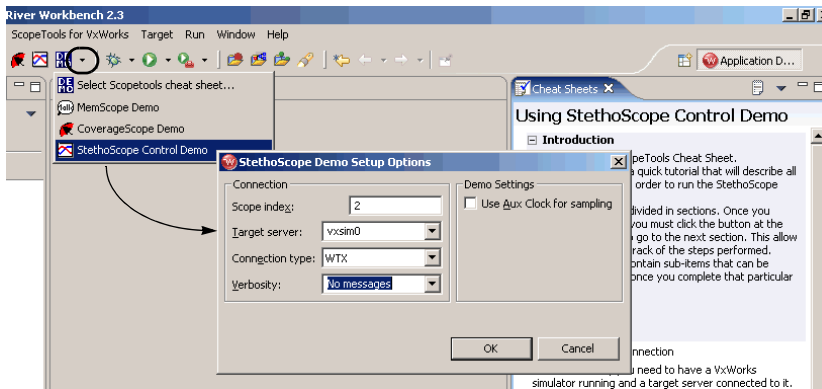
2.7.1 VxWorks Target-Based Demonstration

The VxWorks demonstration program can be started from the Workbench IDE as follows:

1. Connect to the target server for the target on which you wish to run the demonstration program. If you do not have a target server running, you will need to create it first. Refer to *Wind River Workbench User's Guide* for details on how to configure and start a target server.
2. Click Demo  on the Workbench IDE toolbar, shown in the figure below, to open the **Select ScopeTools Demo** dialog box, then choose **StethoScope Control Demo** to open the **StethoScope Control Demo Cheat Sheet** with a tutorial of additional help, as shown.



3. Click the down arrow to the right of the Demo button , as shown in the figure below, select **StethoScope Control Demo** from the menu to open the **StethoScope Demo Setup Options** dialog box, where you can select the desired setup parameters for starting the StethoScope GUI.



The parameters have the following meanings:

Scope index

Use this field to specify the communication channel to use between the target and the StethoScope GUI. Up to 128 different instances may be started on a target. There are 128 valid index numbers available, in the range from 0 to 127.

Target server

The name of server that manages your target.

Connection type

Select the Workbench WTX protocol for getting data from the target.

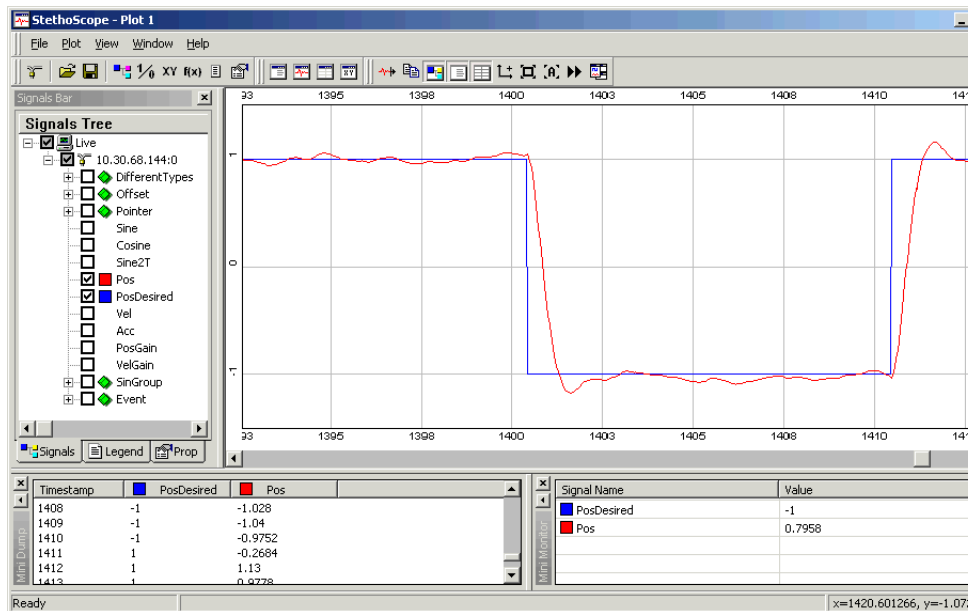
Verbosity

Controls the number and type of messages printed to the standard-output device. A value of 0 causes only errors to be reported. Increasing the value (in the range of 0 - 3) increases the volume of messages. The default is 0.

Use Aux Clock for sampling


Check this box to use the Auxiliary Clock instead of the System Clock.(The default is System Clock).

4. Select **WTX** for the **Connection Type** parameter. The default values for the other initialization parameters should be sufficient for most systems.
5. Click **OK** to load the required libraries and start the StethoScope GUI.
6. A **StethoScope Plot** window should appear on your screen (see the example below). The status bar at the bottom of the window displays the connection status. Under normal conditions, it should display **Ready**.



Viewing the Signals

The VxWorks demonstration program generates several sample signals that can be viewed as follows:

1. If the **Plot** window does not include a **Signals Bar**, open one by using the **View > Signals Bar** command (or the  button).
2. Select the signals you want to plot by using the **Signals Tree** (see [4. Using the Signal Manager](#)).

2.7.2 What Does the Demo Do?

Try each of the following StethoScope features, consulting the referenced manual section if you need help. The demo allows you to:

- Display other signals by clicking on signal entries in the **Signals Bar's Signals Tree** ([2.6.5 Signals Bar](#), p.15).
- Take a **Snapshot** from the **Plot** window, and save it (Sections [11.2 Taking Snapshots](#), p.152 and [11.3 Saving Snapshots](#), p.153).
- Export **Snapshot** data ([11.5 Exporting Snapshot Data to MATLAB and MATRIXX](#), p.158).
- Zoom in and out (**Shift** key + left mouse button in **Plot** screen) ([Zooming](#), p.54).
- Pan the viewing region (click and drag the left mouse button to move) ([Panning](#), p.56).
- Take measurements (**Ctrl** key + left mouse button in **Plot** screen) ([Taking and Removing On-grid Measurements](#), p.56).
- Calculate derived signals (**Derived Signals** from the **File** menu) ([6. Derived Signals](#)).
- Monitor and modify variables. (Select **Pos** and **PosGain** signals in a **Monitor** window with **Writeback** turned on) (Sections [10.2 Monitor Window Tour](#), p.142 and [10.7 Setting Preferences for a New Monitor Window](#), p.149).
- Try X vs. Y plotting (**Plot XY** from the **File > Plots** menu) ([8. The Plot XY Window](#)).
- Display numeric data (**Dump Plot** from the **File > Plots** menu) ([9. The Dump Plot Window](#)).
- Set some triggers (**Triggering** from the **File** menu) ([5. Triggering](#)).

There are additional features you can try. You will observe that the signals produced are simple sine waves, and, in addition, the demo program also produces a simple simulation of a controls system. You can install additional signals automatically from the VxWorks command shell (WindSh or target shell). You can use the following procedure to watch your own signals.

Automatic Signal Management from the VxWorks Shell

A simple command to load a signal is **ScopeInstallSignal()**. The calling syntax is:

```
ScopeInstallSignal(  
    char *name,          /* the string to be displayed by StethoScope */  
    char *units,        /* the units of the signal */  
    void *ptrToVar,     /* a pointer to your variable */  
    void char *type,    /* the type, e.g. "float", "int" */  
    int index)         /* The scope index (defaults to 0) */
```

As an example, the demo program contains a static variable declared as:

```
float Kp = 10;
```

To install this signal from the VxWorks shell, type at the shell:

```
-> ScopeInstallSignal("Kp", "n/a", &Kp, "float", 6)
```

More sophisticated installations of variables referenced by pointers, offsets, etc. also can be done easily.

Some other fun things to play with from the VxWorks shell when running the demo are:

```
-> ScopeRemoveMultipleSignals("sin", 6)  
-> Kp = (float) 100.0
```

Executing the first command should remove all signals that start with "sin". Look at **Pos** and **PosDesired** in the **Plot** window to see the effect of changing the value of **Kp**.

3

StethoScope Features

- 3.1 Introduction 25
- 3.2 File Menu 26
- 3.3 Other Menus 44
- 3.4 Toolbars 48
- 3.5 Status Bar 52
- 3.6 Pop-up Menus 52
- 3.7 StethoScope's Initialization Sequence 60

3.1 Introduction

This chapter describes the major Wind River StethoScope features. These features are available through the **File** menu item, and are common to all four plot windows of StethoScope's multi-window graphical user interface (GUI). Each of these commands is described in detail in the sections that follow.

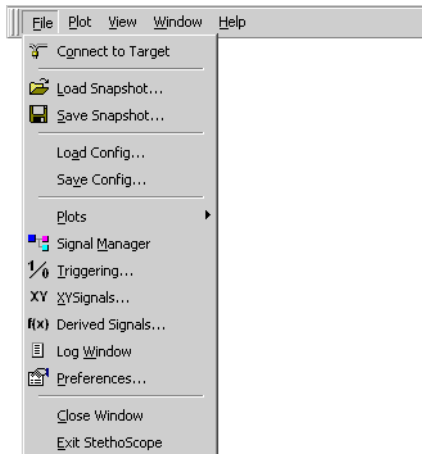
3.2 File Menu

Most of StethoScope's common functionality is accessible specifically through the **File** menu item on the **Menu Bar**. These commands are listed and described in detail in the sub-sections that follow.


Note that commands in the **View** menu are not all available in all data-display window types. These menu commands, as well as some that are available in the various pop-up menus, are described in the chapters on each of the specific data-display windows (see Chapters 7, 8, 9, and 10).

The menu bar is *dockable*, which means you can move it to another location, on or off the window, simply by clicking on the *docking handle* and dragging the menu to the new location.

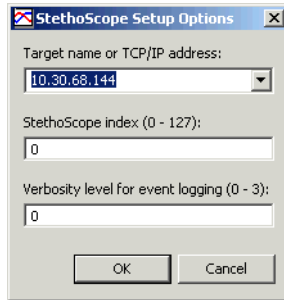
Most of StethoScope's major functions are available through the **File** menu, shown in the figure below. Each of the commands in this menu item are found in, and are active in, all four of the StethoScope data-display windows. A detailed description of each menu command follows.



3.2.1 Connect to Target

After you have successfully started StethoScope's GUI, you need to connect to your target. Select **StethoScope Setup Options** dialog box from the **File** menu (or use the  button) to open a connection to a target. The **StethoScope Setup**


Options dialog box, shown below, opens, with initial values as passed in from the command line or the GUI.





If you have difficulty connecting to the target, check the **Log** window (see [3.2.11 Trace Log Window](#), p.36) for progress messages.

StethoScope can be connected to multiple targets at the same time.

3.2.2 Load Snapshot

Snapshots in general are described in detail in [11. Working with Snapshots](#). Snapshots that have been saved to disk in StethoScope's native format (.ss7 extension) can be reloaded for viewing in any of the plot windows using the **File > Load Snapshot** menu command (or the  button).

3.2.3 Save Snapshot

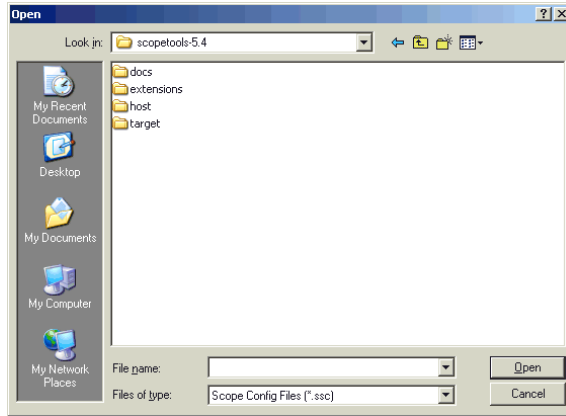
Snapshots are *created* with the **Plot > Take Snapshot** menu command (or the  button). They can be saved to disk using the **File > Save Snapshot** menu command (or the  button) for future reference and reloading. This process is described in [11.3 Saving Snapshots](#), p.153.

3.2.4 Load Config

You can load the configuration parameters set up in a previous StethoScope session, and saved with the **File > Save Config** menu command (described in [3.2.5 Save Config](#), p.28), using the **File > Load Config** menu command.

To load configuration parameters:

1. Select the **File > Load Config** menu command to bring up the **Open** dialog box shown below.



2. Navigate to the pathname containing the file you want to load. The default directory is the same directory where you installed StethoScope.
3. Select or enter the filename in the **Filename** field of the dialog box.
4. Click **Open** to open the file and load the configuration parameters.

If signals that were active when the configuration was saved are not present when it is restored, (that is, they have not been activated or installed via **ScopeInstallSignal()**), then they will not be displayed in the window. However, the window will display them as soon as they become available.

3.2.5 Save Config

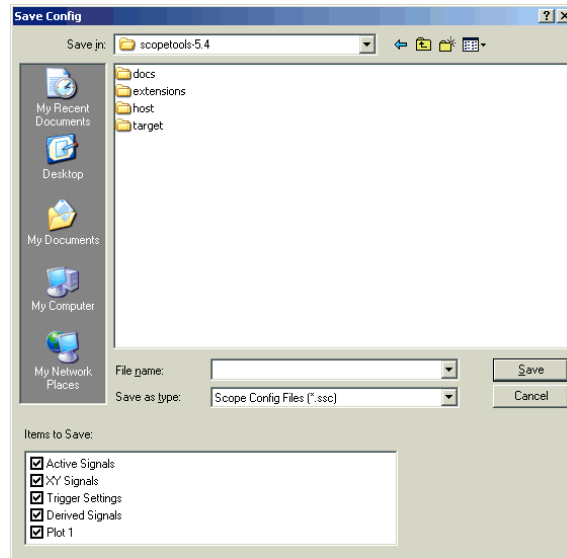
There are several variables you can modify to customize the appearance and behavior of your StethoScope GUI. Whenever you exit StethoScope, the current settings of these configuration variables are saved in a default configuration file, which is then reloaded the next time you start StethoScope. You may find a relatively constant set of configuration parameters that meet your needs, and automatic saving and reloading is satisfactory.

In some cases, however, you may find yourself changing the StethoScope GUI's configuration substantially for different projects or environments. In these cases

you can save the current configuration parameters to a file which can then be reloaded when you work on that project.

To save current configuration parameters at any time:

1. Select the **File > Save Config** menu command to open the **Save Config** dialog, shown in the figure below.



2. Navigate to the pathname where you want to store the file to be saved. The default directory is the same directory where you installed StethoScope.
3. Select or enter a filename in the **Filename** field of the dialog box.
4. The check boxes at the bottom of the **Save Configuration** window allow you to save selected portions of the state of the current StethoScope session. Select from the **Items to save** in the configuration file by checking the corresponding check boxes for the items:

Active Signals

Saves the list of currently activated signals. On reload, if any of the signals in this list have not been registered, they will appear grayed-out in the **Signal Manager** window. See [4. Using the Signal Manager](#) for details on the **Signal Manager**, and [14.4 Registering and Activating Signals](#), p.191 for registering signals.

XY Signals

Saves the list of currently defined **XY Signals**. The **XY Signals** may not appear immediately on reload if any of the component signals are not yet active. They will appear when all their components become available.

Trigger Settings

Saves all the triggering and sampling parameters.

Derived Signals

Saves the list of currently defined derived signals. The derived signals may not appear immediately on reload if any of the component signals are not yet active. They will appear when all their components become available.

Plot *n*, Plot_XY *n*, Monitor *n*, Dump *n*

Saves the state of the named data-display windows. When each of these files is loaded, a *new* window will be opened if none with the saved name is currently open. If a data-display window with that name already exists, then its state will be adjusted to match the saved information. The state of a window includes:

- The list of selected signals being displayed in the window.
- The size and shape of the window.

5. Click **Save** to save the configuration. By default, the filename will have the **extension .ssc**.

This feature allows you to develop a library of saved StethoScope configurations. For example, you might create the following configurations:


- A file named **position.ssc** that contains only the state of one **Plot** window that has been set up exactly the way you like it for displaying your position sensors.
- Another file named **derived.ssc** that contains a useful set of derived signals, such as a scaled variable or the difference of two signals.

3.2.6 Plots


The drop-down menu opened with this command lists the four basic StethoScope data display (**Plot**) windows. Selecting one opens that window (or another occurrence of that window if one is already open). You can have multiple data display windows open at the same time, and each can display different signals or snapshots.

Before using data display windows, it may help to understand how and when data is collected from the target; for this see [5. Triggering](#).


Dump Plot

The **Dump Plot** window, opened with the **File > Plots > Dump Plot** command (or the  button), displays real-time data numerically in a tabular format. A **Dump Plot** window can display both “live” and snapshot data. To learn more about the **Dump Plot** window and how to use it, refer to [9. The Dump Plot Window](#).

Plot


The **Plot** window, opened with the **File > Plots > Plot** command (or the  button), graphically displays real-time signal values, plotted over time. Each signal appears on the plot grid in a different color. A legend shows you each signal’s color. To learn more about using the **Plot** window, see [7. The Plot Window](#).

Monitor

The **Monitor** window, opened using the **File > Plots > Monitor** command (or the  button), lets you watch and modify variables in your program while it is running. It is like the **Dump Plot** window, but differs in that the **Monitor** window only shows you the most recent value of each signal, whereas the **Dump Plot** window displays a running history of signals scrolling down the window. The **Monitor** window can also be used to modify a signal’s value on the target.

To understand how to select data to display in this window, and why you would use this display instead of, or in addition to, the **Dump Plot** window, see [10. The Monitor Window](#).

Plot XY


The **Plot XY** window, opened from the **File > Plots > Plot XY** menu command () , graphs pairs of signals against each other, one signal on the X axis and the other on the Y axis. In visual comparisons, it is very similar to the **Plot** window, the main difference being that *only* XY signal pairs show up in the **Signals Tree**.

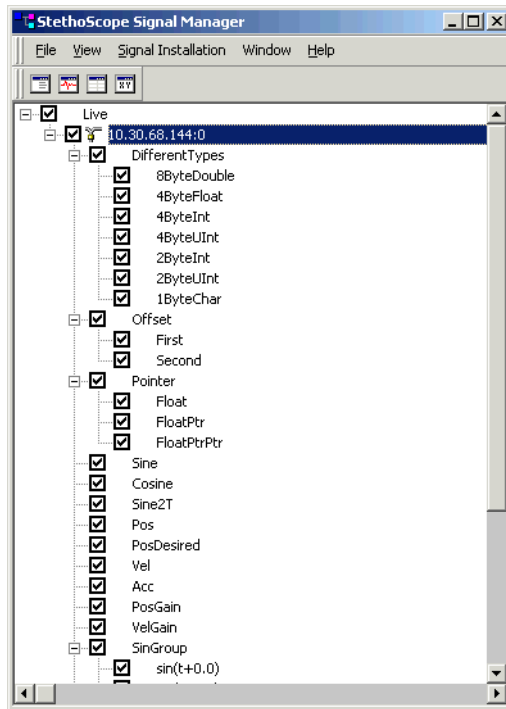
If you open a **Plot XY** window and nothing appears in the **Signals Tree**, it is because you have not yet created any XY signal pairs. Signal pairs must first be

created from separate signals before they appear in the **Dump Plot** window's **Signals Tree**. You do this in the **XYSignals** dialog box, opened with the **File > XYSignals** menu command (or the **XY** button).

More information on the **Plot XY** window, and on the process for creating XY signal pairs for display, can be found in [8. The Plot XY Window](#).

3.2.7 Signal Manager

The **Signal Manager** window, shown below and opened with the **File > Signal Manager** command (or the  button), allows you to control which signals are collected from each target. Only these active signals can be monitored in any of the four types of data-display windows. The **Signal Manager** presents a tree-like view of each target.



The **Signal Manager** can “see” all the signals that were installed (that is, registered and activated—see [14.4 Registering and Activating Signals](#), p.191) on the target. If


your target has many installed signals, the **Signals Trees** in the data-display windows can become very cluttered. You can use the **Signal Manager** to dynamically filter out installed signals that you do not want to collect or have appear in **Signals Trees**.

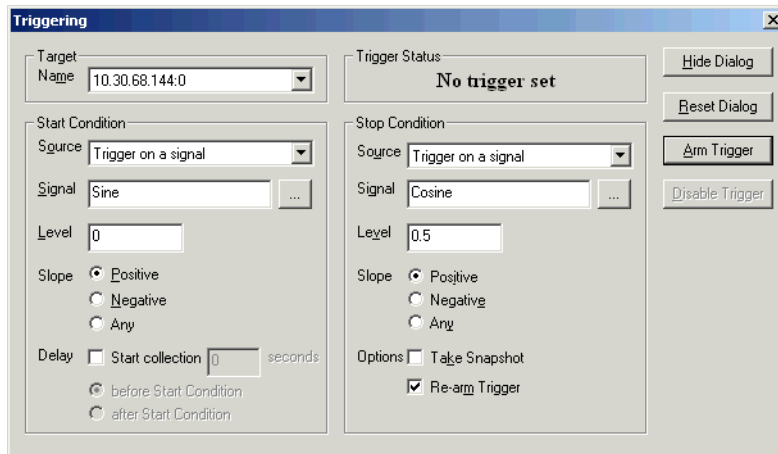
If you do not use the **Signal Manager**, then by default, *all* installed signals on the loaded targets are collected and available for monitoring.

Unlike the other types of StethoScope windows, you only need (and can have only) one **Signal Manager** window, so what you do in this window impacts what you see in all the other types of StethoScope data-display windows. Changes made in the **Signal Manager** are immediately propagated to all data-display windows. For example, if you are monitoring a signal in a **Plot** window and a **Plot XY** window, and you make the signal inactive in the **Signal Manager** window, it will be removed from both your **Plot** and **Plot XY** windows immediately. Similarly, if you activate new signals in the **Signal Manager** window, they will be added to both the **Plot** and **Plot XY** windows' **Signals Tree** (and for any other open data-display windows), where you can choose to select the new signals or not. For more information about using the **Signal Manager**, see [4. Using the Signal Manager](#).

3.2.8 Triggering

The StethoScope API module on the target is responsible for handling periodically collected data, or “samples” (and sporadically collected data, or events). The **Triggering** facility in StethoScope provides control over when and how often the samples are collected.

The StethoScope triggering facility supports a single trigger at any given time. **Triggering** is disabled when the **Triggering** dialog box is not open, or the dialog box is open but the trigger has not yet been armed. In this state, calls to **ScopeCollectSignals()** result in data being collected normally. The **Triggering** dialog box, shown in the figure below, is opened with the **File > Triggering** menu command (or the  button).

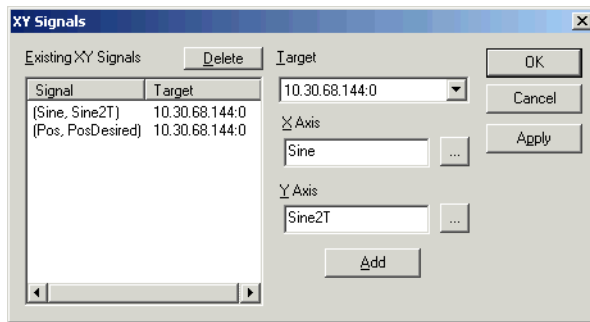


When the **Triggering** dialog has been opened and the trigger is armed with valid start and stop conditions, all calls to **ScopeCollectSignals()** from that point on return without collecting any data. In this “armed” state (that is, the trigger is valid, but has not “fired”), sampled signals will not be plotted in any of the GUI windows, although event data will continue to be plotted as before. Collection and plotting of samples begins when the **Start Condition** is met (for example, the trigger “fires”). Data continues to be plotted on the GUI until the **Stop Condition** occurs. At that point the trigger is then either disabled or rearmed depending on the **Rearm** option specified in the **Triggering** dialog box. To learn more about how triggers are set and used, see [5. Triggering](#).

3.2.9 XY Signals

The **Plot XY** window is used to graph pairs of signals against each other. Signal selection for **Plot XY** windows differs from signal selection for the other types of data-display windows in that each plotted data line is composed of *two* signals. You must create these XY signal pairs using the **XY Signals** dialog box before you will see them in the **Plot XY** window’s **Signals Tree**. They appear in a separate branch of that **Signals** list.

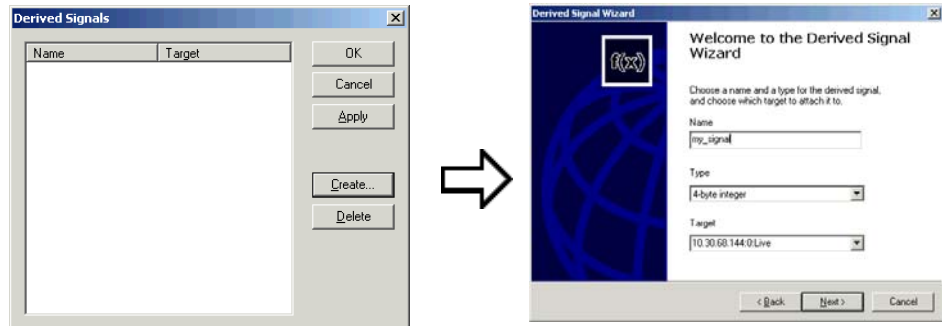
Open the **XY Signal Creation** dialog box, shown in the figure below, using this command (or the **xy** button). For details on how to create signal pairs with the **XY Signals** dialog box, and display those signal pairs using the **Plot XY** window, see [8. The Plot XY Window](#).



3.2.10 Derived Signals


A *derived signal* is a signal whose value is computed by mathematical operations on other signals. Derived signals are calculated by StethoScope on the host in real-time, but not displayed directly from your real-time system. The derived-signals facility provides a simple means of scaling and offsetting signals, plotting differences and ratios of signals, and so forth.

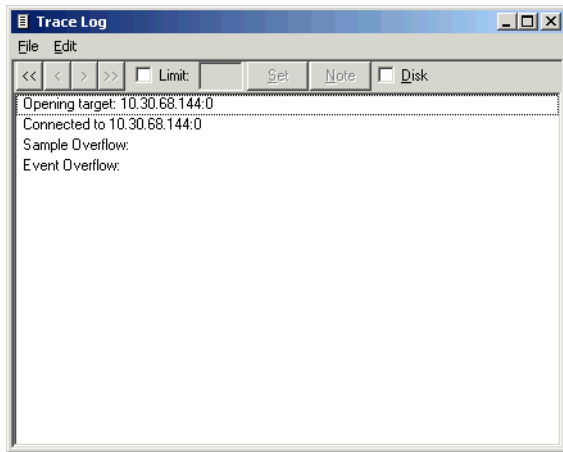
You must create a derived signal using the **Derived Signals** dialog box before you will see it appear in a separate branch in the **Signals Tree** of the **Plot**, **Dump Plot**, and **Monitor** windows. Open the **Derived Signals** dialog box, shown in the figure below, with the **File > Derived Signals** menu command (or the **f(x)** button). This dialog box utilizes a series of **Derived Signal Wizard** dialog boxes that will guide you through the process. To learn details of how to create derived signals, see [6. Derived Signals](#).




3.2.11 Trace Log Window

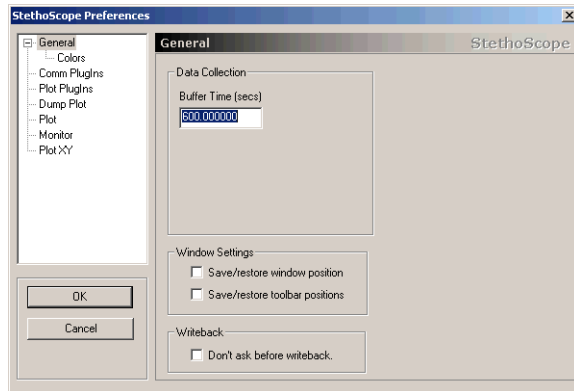
StethoScope records events in a log file, which you can display in the **Trace Log** window, as well as save to disk. The types of events recorded in the log file depend on the **Verbosity** setting selected when StethoScope was started. With each increase in verbosity, more events are included in the log file. **Verbosity** is a command-line option (see [2.5.2 Starting StethoScope On Your Host](#), p.9). **Verbosity** can also be set on a per-target basis with the **Connect to Target** window (see [3.2.1 Connect to Target](#), p.26).

Open the **Trace Log** window, shown in the figure below, with the **File > Log Window** command (or the  button), or by double-clicking anywhere in the error message area of the **Status Bar** (described in chapters 7-10, for each data-display window).



3.2.12 Preferences

The **Preferences** dialog box, shown in the figure below, allows you to set defaults for the StethoScope GUI when it starts and when new data-display windows are created. Open this dialog with the **File > Preferences** command (or the  button).



Preferences are different than properties—preferences are the values used when new data-display windows are created, whereas properties apply only to a specific data-display window that is currently open. In other words, preferences are the *default* properties used for new data-display windows. Then, once a data-display window is open, you can change its properties using the **Properties** menu command (described in the chapters for each data-display window).

The left-most panel displays the various types of preference selections available. The sub-sections that follow describe the options available for each of these preference views.

General View

The **General** view, shown above, allows you to control some aspects of data collection and display, as well as whether window and toolbar positions are maintained when you exit and restart StethoScope.

The parameters are:

Data Collection

Buffer Time (secs)

The buffer time indicates how many seconds of data to show in the plot before refreshing. This is only used when *not* in **Strip Chart** mode, and only in the **Plot** and **Plot XY** windows. This value is seen on the plot as the “width” of the grid (X-axis). The default is 10 seconds.

Window Settings

Save/restore window position

When selected, the position of your StethoScope windows will be saved for use the next time you start StethoScope. When selected, the position of your StethoScope toolbars and window panels will be saved for use the next time you start StethoScope. If you have docked your toolbars and rearranged window panels, and want StethoScope to use the new positions the next time you start up, select this feature. This makes it easy for you to pick up where you left off when you start a new session, all your windows will be restored.

Save/restore toolbar positions

When selected, the default values entered in the **Preference** dialog box for the **Plot View** ([Plot View](#), p.41) and **Plot XY View** ([Plot XY View](#), p.43), for those open plot windows, will be saved and restored.

Writeback

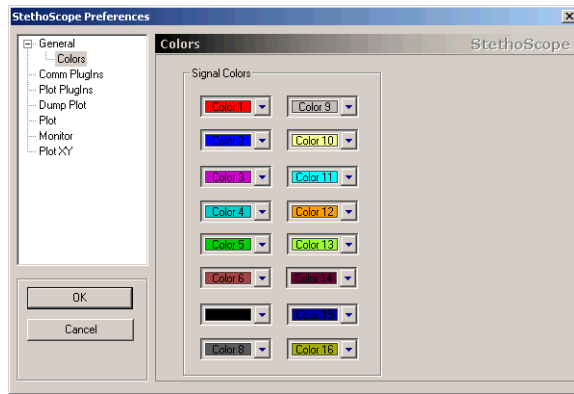
Don't ask before writeback

When selected, a warning message, normally displayed each time a value is written, is not displayed in the GUI.

Click **OK** to apply changes. The settings take affect immediately for the current session and all subsequent sessions, until changed again. Or click **Cancel** to exit the dialog box without saving your changes.

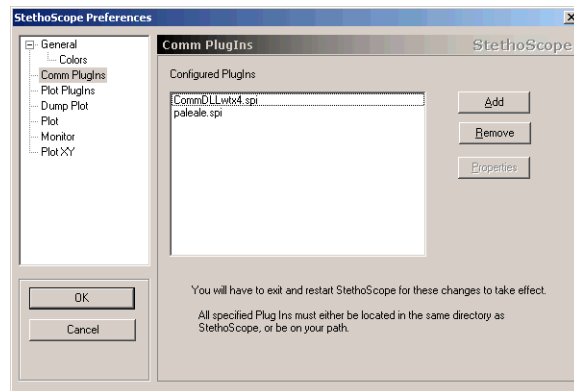
Colors View

The **Colors** view, shown below, allows you to control the trace line color settings only for the **Plot** and **Plot XY** windows, as described in Chapters [7. The Plot Window](#) and [8. The Plot XY Window](#) respectively. **Color Preferences** take effect as soon as you click **OK**.



Comm Plug-ins View

The **Comm Plug-ins** view, shown in the figure below, is used to specify the plug-ins you want to use for communications between the host GUI and the target.



The **Configured Plug-ins** panel displays all the plug-ins currently configured. To delete a plug-in, first select it in this list, then click **Remove** to delete it.

The buttons are:

Add

Click this button to bring up the **Open** dialog box in which you can navigate to, and select, plug-in files.

Remove

Click this button to remove a plug-in selected in the **Configured Plug-ins** panel.

Properties

Opens the **Properties** window where you can configure the selected plug-in.

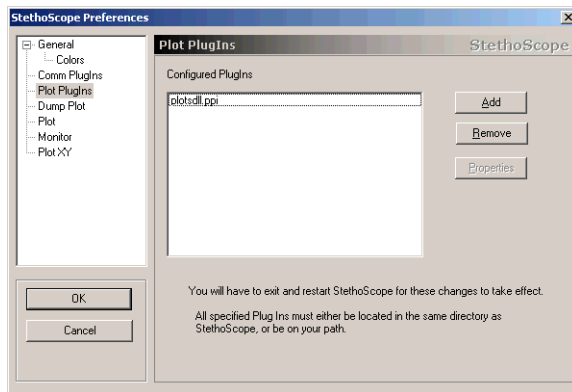
Click **Ok** to apply the changes. The settings take effect immediately for the current session and all subsequent sessions. Click **Cancel** to exit the dialog box without saving your changes.



NOTE: You must exit and restart StethoScope to see the effects of modifying this list.

Plot Plug-ins View

The **Plot Plug-ins** view, shown in the figure below, is used to specify the plug-ins that control which plots are available for use in StethoScope.



The **Configured Plug-ins** panel displays all the plug-ins currently configured. To delete a plug-in, first select it on this list, then click **Remove**.

The buttons are:

Add

Click this button to bring up the **Open** dialog box in which you can navigate to and select plug-in files.

Remove

Click this button to remove a plug-in selected in the **Configured Plug-ins** panel.

Properties

Opens the **Properties** window where you can configure the selected plug-in.

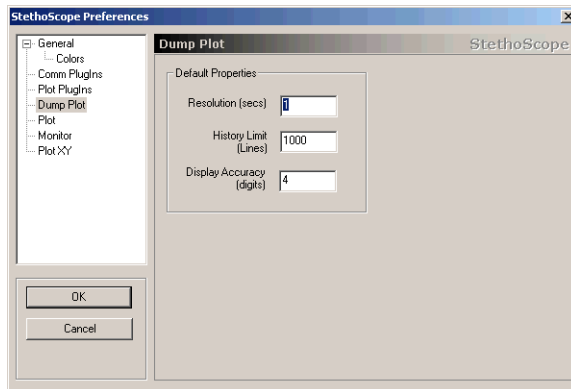
Click **Ok** to apply the changes. The settings take effect immediately for the current session and all subsequent sessions. Click **Cancel** to exit the dialog box without saving your changes.



NOTE: You must exit and restart StethoScope to see the effects of modifying this list.

Dump Plot View

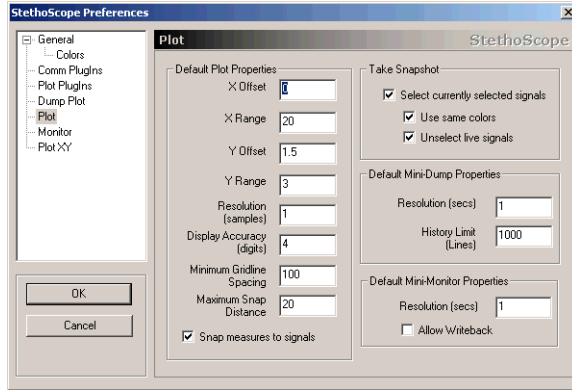
The **Dump Plot** view, shown below, allows you to change the default values used when new **Dump Plot** data-display windows are created. These preferences are described in detail in the **Dump Plot** chapter, [9.6 Setting Preferences for a New Dump Plot Window](#), p.138.



Plot View

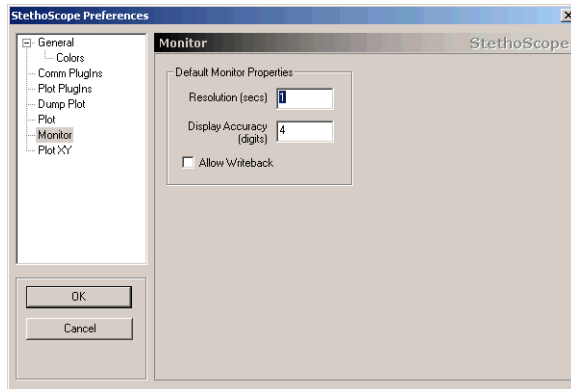
The **Plot** view, shown in the figure below, allows you to change the default values used when new **Plot data-display** windows are created. These preferences are

described in detail in the **Plot** chapter, [7.10 Setting Preferences for a New Plot Window](#), p.106.



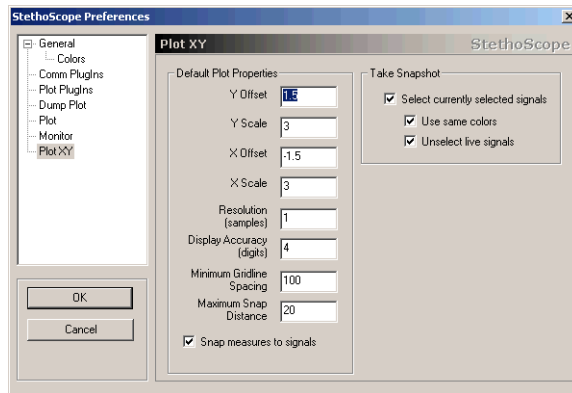
Monitor View

The **Monitor** view, shown in the figure below, allows you to change the default values used when new **Monitor data-display** windows are created. These preferences are described in detail in the **Monitor** chapter, [10.7 Setting Preferences for a New Monitor Window](#), p.149.



Plot XY View

The **Plot XY** view, shown below, allows you to change the default values used when new **Plot XY data-display** windows are created. These preferences are described in detail in the **Plot XY** chapter, [8.9 Setting Preferences for a New Plot XY Window](#), p. 128.



3.2.13 Close Window

Closes only the current window. Any remaining open StethoScope windows are unaffected, except that if this is the only remaining open window, StethoScope will exit.

3.2.14 Exit StethoScope

Quits the StethoScope GUI, but does not stop target daemons. All StethoScope windows are closed.

3.3 Other Menus

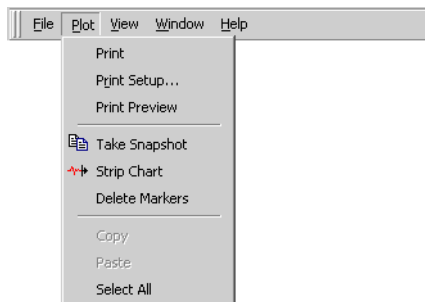
The secondary features and remaining functionality in StethoScope's GUI is accessed through the remaining **Menu Bar** items. They are listed and briefly described in the following sub-sections, but are referenced to, and described in detail in, the various data-display window sections where they are applicable.

The remaining menu items are:

- **Plot**
- **View**
- **Window**
- **Help**

3.3.1 Plot Menu

The **Plot** menu, shown below, contains commands for working with the plots in the **Plot** (7. [The Plot Window](#)) and **Plot XY** (8. [The Plot XY Window](#)) windows.



The **Plot** menu commands are:

Print

Prints the signal traces in the data display window. This command is only available in the **Plot** (7. [The Plot Window](#)) and **Plot XY** (8. [The Plot XY Window](#)) windows.

Print Setup

Allows you to select printer parameters and characteristics before printing. This command is only available in the **Plot** (7. [The Plot Window](#)) and **Plot XY** (8. [The Plot XY Window](#)) windows.

Print Preview

Lets you see, on the screen, what the printed page will look like before actually printing it. This command is only available in the **Plot** (7. [The Plot Window](#)) and **Plot XY** (8. [The Plot XY Window](#)) windows.

Take Snapshot

Saves a copy of all the active signals (not just the selected signals), for all connected targets. For more information, see [11.2 Taking Snapshots](#), p.152.

Strip Chart

Changes the display so that it presents a continuous, scrolling plot. This command is only available in the **Plot** window. For more information, see [7.8 Strip Chart Feature](#), p.103.

Delete Markers

Deletes all markers, measures, and annotations from the grid area. This command is only available in the **Plot** and **Plot XY** windows. For more information, see [Adding and Removing Markers](#), p.54, [Adding Annotations](#), p.55, and [Taking and Removing On-grid Measurements](#), p.56.

Copy

Copies the selected content to the clipboard, and deselects the selected data. This command is only available in the **Plot** window.

Paste

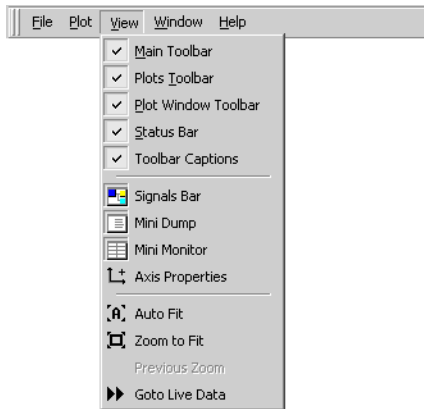
Copy the contents of the clipboard to the window you are in, at the insertion point. This command is only available in the **Plot** window.

Select All

Selects and highlights the entire plotted area up to the instant you selected the option, enabling you to copy it to the clipboard. This command is only available in the **Plot** window.

3.3.2 View Menu

The **View** menu, shown below, contains commands that affect what is presented in the current window, such as toolbars, captions, and so forth. Toolbars mentioned here are dockable, so you can move and resize them. They are described in detail in [3.4 Toolbars](#), p.48.



The **View** menu commands are:

Main Toolbar

Controls whether the toolbar representing a selection of items from the **File** menu is displayed on StethoScope's toolbar. For more information, see [3.4 Toolbars](#), p.48.

Plots Toolbar

Controls whether the toolbar representing a selection of items from the **File > Plot** menu command is displayed on StethoScope's toolbar. For more information, see [3.4 Toolbars](#), p.48.

Plot Window Toolbar

Controls whether the toolbar used within a data-display window is visible. The buttons represent items from the **Plot** and **View Menu**. For more information, see [3.4 Toolbars](#), p.48.

Status Bar

Controls whether the status line along the bottom of the window is visible. For more information, see [3.5 Status Bar](#), p.52.

Toolbar Captions

Controls whether the names of the panels (**Signals Bar**, **Mini Dump**, and **Mini Monitor**) are displayed in the **Plot** or **Plot XY** window. This command is only available in the **Plot** and **Plot XY** windows.

Signals Bar

Controls whether a **Signals Bar** panel appears in the window.

Mini Dump

Creates a **Mini Dump Plot** window within the window. This command is only available in the **Plot** window (see [7.3.2 View Menu](#), p.89).

Mini Monitor

Creates a **Mini Monitor** window within the window. This command is only available in the **Plot** window (see [7.3.2 View Menu](#), p.89).

Axis Properties

Allows you to set certain parameters that affect the appearance and labelling of the X or Y axis of a **Plot** data-display area. This command is only available in the **Plot** window.

Auto Fit

Causes the plotting area to be scaled dynamically and automatically to the extremes of the data in the Y direction being displayed in the window. This command is only available in the **Plot** and **Plot XY** windows.

Zoom to Fit

Changes the scales and offsets so that all the signals fit and take up the entire plot window. This command is only available in the **Plot** and **Plot XY** windows. See also **Previous Zoom**.

Previous Zoom

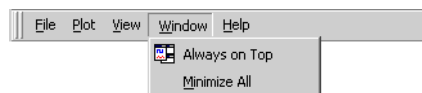
Reverses the action of the last zoom action. This command is only available in the **Plot** and **Plot XY** windows.



NOTE: For more information on **Zoom to Fit** through **Previous Zoom**, (above) see [7. The Plot Window](#) and [8. The Plot XY Window](#) respectively.

3.3.3 Window Menu

The **Window** menu contains items that affect the characteristics of currently open StethoScope windows. It contains the following commands:



Always on Top

Keeps the currently selected window as the topmost display on the desktop.

Minimize All

Minimizes all StethoScope windows.

3.3.4 Help Menu

The **Help** menu provides online help for StethoScope. The **Help** menu currently has only one topic:

About StethoScope

Opens the **About** box displaying version and copyright information.

3.4 Toolbars

The toolbars appearing at the top of each data-display window actually consists of *three* separate toolbars, each offering quick access to commonly used menu commands. Place your mouse pointer over each toolbar button to see a **ToolTip** that shows you the action of each button.

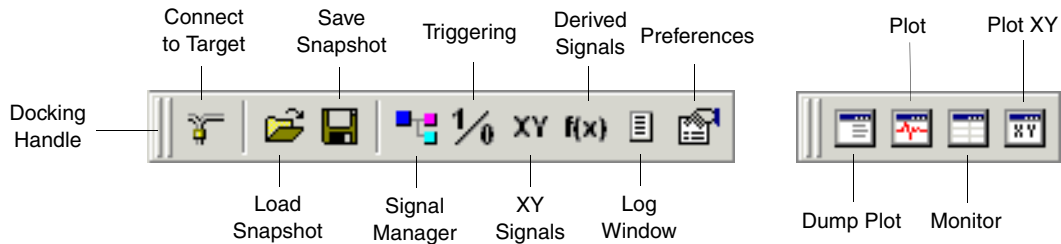
Each of the toolbars are dockable, which means you can move it to another location, on or off the window, simply by dragging it. (The menu bar is also dockable.) Each of the toolbars can be independently displayed or hidden using the **View** menu item. Toolbars that you drag off the screen can be restored again at any time from the **View** menu.

3.4.1 Main Toolbar










The buttons on this toolbar control the various StethoScope functions. This toolbar is the same in all data display windows.

3.4.2 Plot Toolbar


The buttons on this toolbar, shown below, are used to create new data-display windows. This toolbar is also the same in all data display windows.





The following descriptions are for buttons in the **Main** toolbar. The corresponding **menu commands** are in bold.


-  **File > Connect to Target**
Selects a target and connects it to your host GUI, described in [3. StethoScope Features](#).
-  **File > Load Snapshot**
Loads a previously saved snapshot file, described in [11. Working with Snapshots](#).
-  **File > Save Snapshot**
Saves a snapshot to a file, described in [11. Working with Snapshots](#).
-  **File > Signal Manager**
Opens the **Signal Manager** window, described in [4. Using the Signal Manager](#).
-  **File > Triggering**
Opens the **Triggering** window, described in [5. Triggering](#).
-  **File > XY Signals**
Opens the **XY Signals** dialog box, described in [8. The Plot XY Window](#).
-  **File > Derived Signals**
Opens the **Derived Signals** dialog box, described in [3. StethoScope Features](#).
-  **File > Log Window**
Opens the **Trace Log** window, described in [3. StethoScope Features](#).
-  **File > Preferences**
Opens the **Preferences** dialog box, described in [3. StethoScope Features](#).

The following buttons in the **Plots** toolbar appear on each data-display window, and they control the indicated StethoScope functions (the corresponding **menu command** is in bold):

 **File > Plots > Dump Plot**
Creates a new **Dump Plot** window, described in [9. The Dump Plot Window](#).

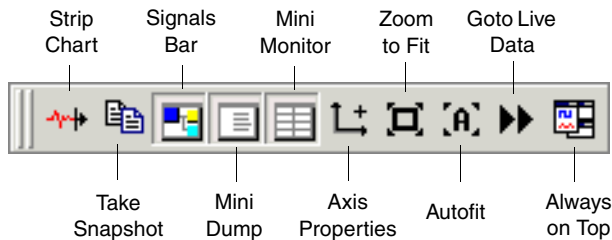
 **File > Plots > Plot**
Creates a new **Plot** window, described in [7. The Plot Window](#).

 **File > Plots > Monitor**
Creates a new **Monitor** window, described in [10. The Monitor Window](#).

 **File > Plots > Plot XY**
Creates a new **Plot XY** window, described in [8. The Plot XY Window](#).

3.4.3 Plot Window Toolbar

The buttons on this toolbar, shown below, vary depending on which of the data-display windows you currently have on-screen. In general, these buttons are used to change something in the current data-display window, whereas the buttons on the **Plots Toolbar** are used to create new windows.



This section briefly describes each toolbar button. The equivalent menu commands are shown in bold. For more details on these commands, refer to [3.2 File Menu](#), p.26.

Different buttons are available in the **Plot** window toolbar depending on the type of data-display window. The following list shows all possible buttons for this toolbar (the corresponding **menu command** is in bold).

Note that on each particular data-display window type (for example, **Plot**, **Plot XY**, **Dump Plot**, and **Monitor**), only the relevant buttons appear from this group.

 **View > Show Snapshots**
Controls what is displayed in the **Signals Tree**. When selected, only snapshots appear in the **Dump Plot** window's **Signals Tree**. When unselected, only the

live data buffer appears. This button (and command) is only available in the **Dump Plot** window.



Plot > Strip Chart

Changes the display so that it presents a continuous, scrolling plot (instead of repainting the plot every 10 seconds). This button (and command) is only available in the **Plot** window.



Plot > Take Snapshot

Saves a copy of all the active signals. You can display the snapshot in the **Plot** and **Plot XY** windows along with real-time data.



View > Signals Bar

Creates a **Signals Bar** panel in the window.



View > Mini Dump

Creates a **Mini Dump Plot** window within the window. This is simply a smaller version of the **Dump Plot** window. It is created as a panel in the **Plot** window, but you can dock it elsewhere on the screen. This button (and command) is only available in the **Plot** window.



View > Mini Monitor

Creates a **Mini Monitor** window within the window. This is simply a smaller version of the **Monitor** window. It is created as a panel in the **Plot** window, but you can dock it elsewhere on the screen. This button (and command) is only available in the **Plot** window.



View > Axis Properties

Opens the **Axis Properties** dialog box, where you can add new axes, and edit the properties for any existing axis.



View > Zoom to Fit

Changes the scales and offsets so that all the signals fit and take up the entire plot window. This button (and command) is only available in the **Plot** and **Plot XY** windows.



View > Auto Fit

The plot automatically zooms to fit when a signal goes off the screen. This button (and command) is only available in the **Plot** and **Plot XY** windows.



Window > Always on Top

Keeps the window as the topmost display on your desktop.

3.5 Status Bar

The **Status Bar** at the bottom of the **Plot** window, hidden or displayed with the **View > Status Bar** menu command, is divided into three panels:

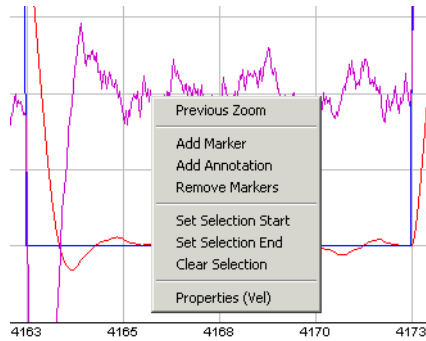
- The left panel provides a description of the command when the mouse pointer is passed over any **Plot** window toolbar button.
- The middle panel displays status messages. Double-click this panel at any time to display the **Log** window and see a history of status messages.
- The right panel displays the XY coordinates of your mouse pointer when it is passed over the plot grid.

3.6 Pop-up Menus

Pop-up menus, available in **Plot** and **Plot XY** windows only, are used to offer options that are unique to specific signals, and in some cases, to where the cursor is within the StethoScope GUI. The following pop-up menus are described:

3.6.1 On-Grid Pop-up Menu

The following commands are available in the pop-up menu, shown below, that appears when you right-click while hovering the cursor anywhere over the grid area of a **Plot** window (see [7.2 Plot Window Tour](#), p.86).



The following menu items are available:

Previous Zoom

Reverses the effect of the previous **Zoom** action (see [Zooming](#), p.54). Note that the **Zoom to Fit** command erases the history of all previous zooms.

Add Marker

Adds text to a specific point on the grid area, showing the coordinates of a desired point (see [Adding and Removing Markers](#), p.54).

Add Annotation

Adds text to a specific point on the grid area, you can type any comment you want (see [Adding Annotations](#), p.55).

Remove Markers

Deletes all markers and measures from the grid area (see [Adding and Removing Markers](#), p.54).

Set Selection Start

Marks the beginning of an area of the graph to select for copying to the clipboard. This mark, a vertical line, is placed at the location of the mouse cursor at the time this option is selected. The selection will be highlighted as soon as you select **Set Selection End** from the pop-up menu.

Set Selection End

Marks the end of the graph area you want to copy to the clipboard. It also is placed at the location of the mouse cursor at the time this option was selected. The graph area between **Set Selection Start** and **Set Selection End** is immediately highlighted and ready to be copied into the clipboard using

either the **Plot > Copy** menu command, or the standard **Ctrl+C** keyboard shortcut. The clipboard can then be copied to most text or bitmap handling applications. The results will vary between text lists or bitmap renderings depending on the application.



NOTE: You can also select the entire graph area thus far using the **Plot > Select All** menu command.

Clear Selection

Deselects the graph area selected using the **Set Selection Start** and **Set Selection End** commands. Note that this does not delete the selected graph data itself.

In addition to the commands in the on-grid menu, you can also:

Pan

Pan to shift the offsets to view a different region of a plot (see [Panning](#), p.56).

Take Measurements


Measure offsets in the horizontal and vertical directions between any two points (see [Taking and Removing On-grid Measurements](#), p.56).

Zooming

You can magnify a region to see details by zooming. The offset and scale of the plot will be adjusted so that the zoomed region fills the **Plot** window.

Zooming in to a Desired Region

1. Press and hold the **Shift** key.
2. Click and drag the left mouse button to select a region of the plot.

The on-grid pop-up menu includes a **Previous Zoom** command, which can be used to return to the previous zoom. Note, however, that using the **Zoom to Fit** menu command (or the  button) erases the history of all zoom actions.

Adding and Removing Markers

A marker shows the coordinates of a point on the grid.

Adding a Marker

- Right-click at the desired point in the grid area. Select **Add Marker** from the pop-up menu.
- Press and hold **Ctrl** and while left-clicking the mouse.

Deleting a Specific Marker

- Drag the marker off the grid area.

Delete All Markers and Measures

- Right-click anywhere in the grid area, then click **Remove Markers** from the pop-up menu.
- Click **Plot > Delete Markers** menu command.

Adding Annotations

An annotation is text that you type in, to mark a point on the grid.

Adding an Annotation

- Right-click at the desired place in the grid area, then click **Add Annotation** in the pop-up menu.

Moving an Annotation

- Drag the annotation to the desired spot on the grid area.

Editing an Existing Annotation

- Right-click the annotation, then click **Edit** from the pop-up menu to open the **Annotation Edit** dialog box.

Deleting a Specific Annotation

- Drag the annotation off the grid area.
- Right-click the annotation, then select **Delete** from the **On-Grid** pop-up menu.

Panning

Panning provides an easy way to change the offset of a plot by allowing you to move the plot directly from within the display window. This is useful especially when you have zoomed in and you want to see an adjacent region.

After zooming in, you can pan in the X and Y directions. When not zoomed in, you can only pan in the Y direction since the entire X range is already in view.

Panning the View Region

- Click and drag the left mouse button to move the viewing region.
- Use the scroll bars.

Taking and Removing On-grid Measurements

You can create a line on the grid that measures the distance between any two points. As you move the mouse pointer over the plot area, the coordinates of the mouse pointer are displayed in the rightmost panel of the status bar at the bottom of the window.

Measuring Offsets Between Any Two Points on the Plot

1. Press and hold the **Ctrl** key.
2. Place the mouse pointer at the first location on the grid.
3. Left-click and drag the mouse pointer to the second location.
4. Release the mouse button and the **Ctrl** key.

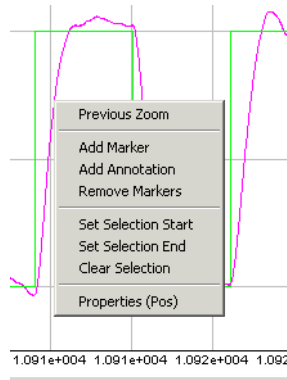
This creates a line between the two points and displays the difference between the two points as an x, y pair.

Deleting a Measurement

- Click an end-point of a measurement and drag the end-point off the plot window.
- Select **Plot > Delete Markers** to remove all markers and measurements.

3.6.2 Trace Pop-up Menu

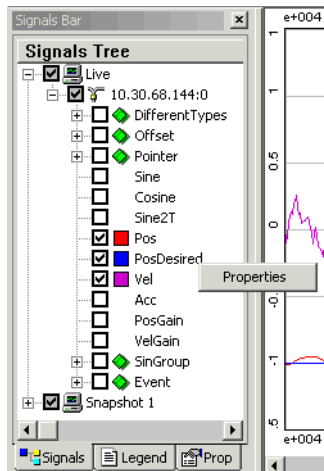
In the **Plot** window only, the pop-up menu that opens when you right-click with the cursor exactly on a trace line is the same as the pop-up menu in [3.6.1 On-Grid Pop-up Menu](#), p.52 above, except there is one additional menu item.



The additional menu item is **Properties**. When you click this item, it opens the **Signal Properties** dialog box where you can configure parameters that affect the appearance and behavior of the specific signal your cursor is on. The **Signal Properties** dialog box is described in detail in [Section 7.6 Signal Properties Dialog Box](#), p.98 for the **Plot** window, or [8.7 Signal Properties Dialog Box](#), p.123 for the **Plot XY** window.

3.6.3 Signals Tree Pop-up Menu

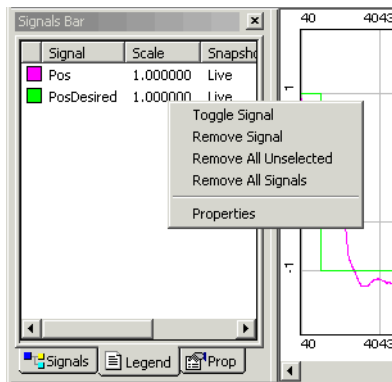
The pop-up menu that opens when you right-click on a signal name in the **Signals Tree** is shown below.



This pop-up menu has only one option, **Properties**. When you click this item, it opens the **Signal Properties** dialog box where you can configure parameters that affect the appearance and behavior of the specific signal your cursor is on. The **Signal Properties** dialog box is described in detail in [7.6 Signal Properties Dialog Box](#), p.98 for the **Plot** window, or [8.7 Signal Properties Dialog Box](#), p.123 for the **Plot XY** window.

3.6.4 Legend Pop-up Menu

A pop-up menu, shown below, opens when you right-click with the cursor on a signal name in the **Legend** tab view of a **Plot** or **Plot XY** window.



This pop-up menu has the following options.

Toggle Signal

Turns the signal trace **on** or **off**. This is the equivalent of going to the **Signals Tree** and alternately checking and deselecting the signal's check box, except that it leaves the signal's name in the **Legend** list when you toggle it **off**.

Remove Signal

Removes the signal from the graph. This is equivalent to toggling the signal **off**, except it removes the signal's name from the **Legend**.

Remove All Unselected

Removes all signals that have been toggled **off**.

Remove All Signals

Removes all signals regardless of their toggled status.

Properties

Opens the **Signal Properties** dialog box (see [7.6 Signal Properties Dialog Box](#), p.98 for a **Plot** window, or [8.7 Signal Properties Dialog Box](#), p.123 for a **Plot XY** window).

3.7 StethoScope's Initialization Sequence

The StethoScope initialization process runs the wrapper shell script for StethoScope. The script looks in its own directory for a sub-directory with its name, and with a **.app** extension. This directory contains plug-ins, resources, and executables used by StethoScope, as well as the actual StethoScope binary.

With the binary now located, the script sets the **LD_LIBRARY_PATH** path to point to the libraries needed by StethoScope to run, and transfers control to the Scope binary. StethoScope then looks for its resource file in:

```
$HOME/.StethoScope/version/StethoScoperc
```

It reads this file to determine any previously set preferences. If this file is not found, it is created.



NOTE: You are strongly discouraged from trying to edit this file.

4


Using the Signal Manager

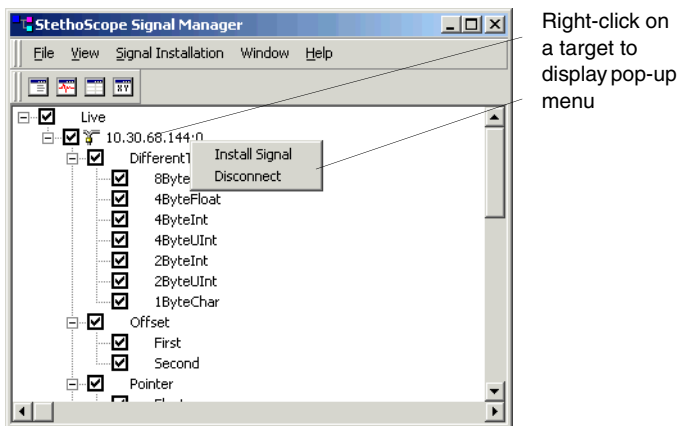
- 4.1 Introduction 61
- 4.2 The Signal Manager Window 62
- 4.3 Working With Signal Trees 62
- 4.4 Signal Installation 64
- 4.5 Disconnecting from the Target 64

4.1 Introduction

S signal from your running target program can be displayed in the StethoScope GUI only if it has been “installed,” meaning that it has been “registered” and “activated” either before or during the StethoScope session. The Signal Manager utility, available from the StethoScope GUI directly, will do this for you. This chapter gives you the information you need to get all the variables you need to look at installed and displayed in the various StethoScope windows.

4.2 The Signal Manager Window

The **Signal Manager** window, shown in the figure below, is opened from the **File > Signal Manager** menu command (or the  button), or by right-clicking on a target in the **Signals Tree** and selecting **Signal Install** in the pop-up menu. It allows you to control which signals are collected from each target. Only these signals, when installed, can be monitored in any of the four types of data-display windows.



4.3 Working With Signal Trees

Installed signals are presented in a tree-like structure in the **Signal Manager** window and in the **Signals Bar** panel of the data-display windows. Use the signals tree in the **Signal Manager** window to choose which signals are active (collected from the target). In the data-display windows (**Plot**, **Plot XY**, **Dump Plot**, and **Monitor**), you use the **Signals Tree** in the **Signals Bar** to select the signals to display in the window's graph or table.

To activate a signal (so that it is collected from the target), simply click a signal entry in the **Signal Manager's Signals Tree**. A check mark appears in the check box to show that it is active. To stop collecting a signal, click it again to clear the check mark.

Signals must be installed before they appear in the **Signals tree**. You can install signals using the **Signal Installation** dialog box (see [4.4 Signal Installation](#), p.64). **Signals Trees** in other windows are identical, except that they only show active signals.

Behavior characteristics of the **Signals tree** are:

- Signals may be organized hierarchically when they are installed. A directory entry—indicated by a or node icon—contains sub-entries that are either signals or other directories.
- A directory entry always begins a new branch on the signals tree. You can expand or collapse these branches by clicking the or icon, respectively.
- Each entry (signal or directory) has a check box. A directory's check box indicates what is selected underneath it. A signal's check box indicates whether or not that signal is selected. Check boxes indicate the following:
 - = No signal in this branch is selected, or, if a signal, this signal is not selected.
 - = All signals in this branch are selected, or, if a signal, this signal is selected.
 - = At a branch node, some, but not all, signals in this branch are selected.
- Selecting the check box at a directory node in a signals tree selects everything underneath it. Similarly, clearing (deselecting) a directory check box clears everything underneath it.
- Click on a signal or branch check box to toggle its selection—click an unselected signal to select it; click it again to deselect it.
- Right-clicking a target in a signals tree displays a pop-up menu with the following commands:

Install Signal

Opens the **Signal Installation** dialog box, from which you can install a signal on your target (see [4.4 Signal Installation](#), p.64).

Disconnect

Disconnects the StethoScope GUI from the selected live target (see [4.5 Disconnecting from the Target](#), p.64).

The best way to become familiar with the signals tree is to experiment while running the demo program. You will quickly see how easy it is to make signals active in the **Signal Manager** window, and to select or clear signals in the data-display windows.



NOTE: If you see an empty **Signals Tree**, either you are not running an application instrumented with the StethoScope API on the target, your target program did not install any signals, or you have only registered but not activated any signals (You can observe this case by opening the **Signal Manger** window and seeing that signals are listed but none are checked).

4.4 Signal Installation

A signal name is the name you assign to a data item that is collected from your target application by StethoScope. An installed signal is one that has been “registered” and “activated,” as described in [14.4.1 Installing Signals](#), p. 192. Signals can be installed by instrumenting target code using the StethoScope API (see [A. StethoScope API Reference](#)), or they can be installed automatically, by variable name, or manually, by address.

To install a signal automatically, select the **Signal Installation** menu item on the menu bar, or right-click on a target name in the **Signal Manager** window to display a pop-up menu and select **Install Signal**. This action opens the **Signal Installation** dialog box, where you can configure all the necessary parameters.

4.5 Disconnecting from the Target

Right-click a target name in the **Signal Manager** window to display a pop-up menu and select **Disconnect**. This action severs the communication link between the StethoScope GUI and the target. This does not, however, stop the collection thread or StethoScope daemons on the target. You may reconnect the GUI to the same, or connect to a different, scope index using the **New Target Connection** dialog box (see [3.2.1 Connect to Target](#), p.26).

5

Triggering

- 5.1 Introduction 65
- 5.2 Configuring a Trigger 66
- 5.3 Triggering Dialog Box 66
- 5.4 Setting a trigger 71
- 5.5 Understanding and Preventing Overflows 73
- 5.6 Notes and Hints 74

5.1 Introduction


As described earlier (see [3.2.8 Triggering](#), p.33), the StethoScope API module on the target is responsible for handling periodically collected data (or samples) and sporadically collected data (or events). The **Triggering** facility in StethoScope provides control over when and how often these samples are collected.

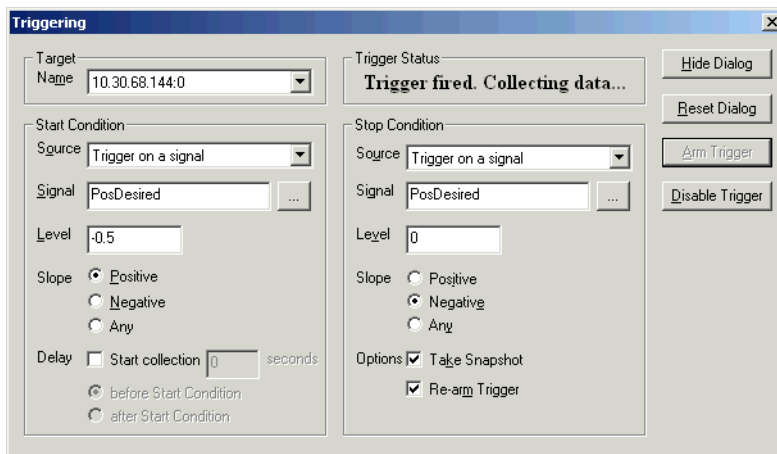
5.2 Configuring a Trigger

The StethoScope triggering facility supports a single trigger at any given time. **Triggering** is disabled when the **Triggering** dialog box is not open, or the dialog box is open but the trigger has not yet been armed, or has been disabled. In this state, calls to **ScopeCollectSignals()** result in data being collected normally.

When the **Triggering** dialog box has been opened and the trigger is armed with valid start and stop conditions, all calls to **ScopeCollectSignals()** from that point on return without collecting any data. In this “armed” state (that is the trigger is valid, but has not yet “fired”), sampled signals will not be plotted in any of the GUI windows, although *event* data will continue to be plotted as before. Collection and plotting of samples begins when the **Start Condition** is met (that is when the trigger “fires”). Data continues to be plotted on the GUI until the **Stop Condition** occurs. At that point the trigger is then either disabled or “rearmed” depending on the **Rearm** option specified in the **Triggering** dialog box.

5.3 Triggering Dialog Box

Use the **File > Triggering** menu command (or the  button) to open the **Triggering** dialog box, shown in the figure below.



The control settings in the **Triggering** dialog box are divided into the following sections:

Target

This dropdown menu lets you select a target to trigger on. Changing the current target will cause the active trigger to be disabled.

Start Condition

The parameters in this section determine when data collection begins on the target.

Trigger Status

Status messages received by the GUI are displayed here.

Stop Condition

The parameters in this section determine when data collection ends on the target.

Buttons

The following buttons are available:

Hide Dialog

Closes the dialog box and disarms any triggers you have set.

Reset Dialog

Reverts all trigger settings to their default values.

Arm Trigger

Sets the selected trigger to “fire” the next time the trigger’s specified start conditions are met.

Disable Trigger

Stops the trigger from “firing” when its start conditions are met, but still maintaining those start conditions so it can be set again later.

5.3.1 **Target**

The **Target** field is a drop-down menu displaying a list of target names and index numbers attached to the StethoScope GUI. Select the target on which you want to configure triggering.

5.3.2 Start Condition

These are the settings that start the trigger. The trigger is fired, and data collection begins, when the trigger conditions you specified in the **Start Condition** panel are met. Data collection stops when the trigger conditions you specified in the **Stop Condition** panel are met.

For both **Start** and **Stop Condition** panels, the following triggering parameters can be set.

Source

Choose the signal or event to be the initial trigger source. The following choices are listed in the dialog box.

Trigger on a signal

Specifies the source to be a periodically sampled signal.

Trigger on an event

Specifies the source to be an event.

Trigger immediately

This will cause triggering to start immediately without waiting on a condition. If this option is chosen, there is no need to fill out the other parameters for **Start Condition**.

Signal

If you selected **Trigger on a signal**, any non-derived signal may be used as the trigger source. This excludes deactivated signals (those not being collected and displayed by StethoScope). Choose a signal from the drop-down list.

If you selected **Trigger on an event**, type in the **event ID**.

Level

Enter the signal value which should set off the trigger. This is relevant only if you selected **Trigger on a signal**.

Slope

Specify the slope for the trigger (applies only if you selected **Trigger on a signal**).

Positive

The trigger fires only if the source signal value is *increasing* when it crosses the trigger level.

Negative

The trigger fires only if the source signal value is *decreasing* when it crosses the trigger level.

Any

The trigger fires when the source signal value crosses the trigger level without respect to direction.

Delay

If checked, it allows you to delay the trigger firing for a specified time.

Start collection

Specifies the time (in seconds) to delay trigger firing.

Before Start Condition

If checked, the delay begins immediately, before the **Start Condition** begins to be evaluated.

After Start Condition

If checked, the delay begins immediately after the **Start Condition** has been met.

5.3.3 Trigger Status

This field displays the status of the StethoScope trigger.

5.3.4 Stop Condition

Triggering is disabled when the trigger **StopConditions** you specify here are met. Note that data collection continues even after triggering is disabled, unless the trigger is **rearmed**. The following trigger stop parameters can be set.

Source

This parameter specifies the specific kind of triggering source for the **Stop Condition**. It can be one of the following.

Trigger on a signal

Specifies the source to be a periodically sampled signal.

Trigger on an event

Specifies the source to be an event.

Trigger after set time period

This will cause triggering to terminate after the number of seconds specified in the **Secs** text field has elapsed.

Signal

If you selected **Trigger on a signal**, any non-derived signal may be used as the trigger stop source. This excludes deactivated signals (those not being collected and displayed by StethoScope). Choose a signal from the drop-down list.

If you selected **Trigger on an event**, type in the **eventID**.

Level or Secs

This field label changes to "**Secs**" if you selected you selected **Trigger after set time period** in Source above; otherwise the label is "**Level**." If "**Level**," enter the signal value which should stop the trigger. If "**Secs**," enter the the number of seconds of triggering after which you want triggering to stop.

Slope

Specify the slope for the trigger (applies only if you selected **Trigger on a signal**):

Positive

The trigger stops data collection only when the source signal value is *increasing* when it crosses the trigger level.

Negative

The trigger stops data collection only when the source signal value is *decreasing* when it crosses the trigger level.

Any

The trigger stops when the source signal value crosses the trigger level without respect to direction.

Options

Optional actions to take after the trigger is stopped.

5.3.5 Options

Optional actions to take after the trigger is stopped.

Take Snapshot

If this box is checked, a snapshot is taken of the signal and events that were collected between the start and stop conditions.

Re-arm Trigger

If this option is chosen, the trigger is automatically rearmed after the **Stop Condition** is met. This is useful in cases where you might want to observe an occurrence that happens over and over again. Choosing the **Take**

Snapshot option along with this option can help you monitor occurrences during an unattended overnight run.

5.3.6 Buttons

The following buttons cause the trigger, with its options set, to take the indicated actions:

Hide Dialog

The trigger will only be active while the dialog box is open. If you want to cancel the action of the trigger, click this button.

Reset Dialog

Resets all options to their default values.

Arm Trigger

Causes the calls to **ScopeCollectSignals()** from that point on to respond to the conditions set for this trigger, until you either **Disable Trigger** or **Hide Dialog** (that is, the trigger will now “fire” when the conditions are met).

Disable Trigger

Causes the trigger to become “disarmed” without closing the dialog box (configured values remain set). Data collection resumes as it was before the trigger was armed.

5.4 Setting a trigger

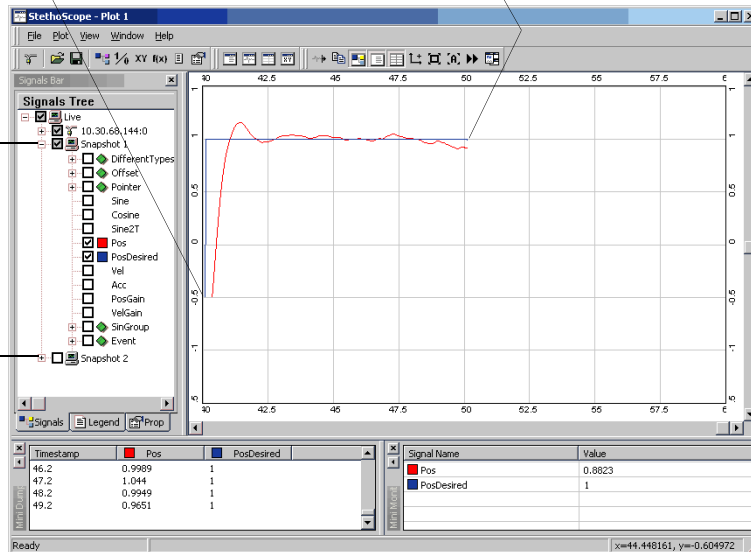
This section walks you through the process of setting an example trigger. We will use the StethoScope demonstration program introduced in [2. Getting Started](#) for this example. The example output from this demonstration is shown and annotated below. Refer to it in the text that follows.

① The trigger “fires” when the value of PosDesired crosses 0.5 on a positive slope

② The trigger is disabled as soon as PosDesired starts on a negative slope

③ A snapshot of the collected data is created and listed in the Signals Tree

④ StethoScope continues to rearm the trigger and create more snapshots



Run the demonstration program, connect the StethoScope GUI to the scope index of the demo program and select a few signals from the **Signals Bar** for viewing. Open the triggering dialog box, choose the target from the target drop-down menu. Specify the **Start Condition** by choosing **Trigger** on a signal for **Source**, **PosDesired** for the **Signal**, **Positive** for **Slope**. Specify the **Stop Condition** by choosing **Trigger** on a signal for **Source** and **Negative** for **Slope**.

Finally, there are optional actions to take after the trigger is stopped, including:

- **Take Snapshot.**
- **Re-arm Trigger.**

Click the **Arm Trigger** button for the trigger to take effect.

The trigger you set will cause periodic collection to be temporarily disabled until the **Start Condition** is met. No signals will be plotted during the above wait period. Signal plotting will resume when the value of **Sine** signal crosses 0.5 with a *positive* slope. Data collection and plotting will continue for 8 seconds before the **Stop Condition** is satisfied. At the end of this period, a snapshot of the data

collected during those eight seconds will be taken and added to the **Signals Tree** in the **Signals Bar** under the name of the target. Check the signal names in the snapshot to view their data.

Since **Re-arm Trigger** was chosen in the UNIX/Linux GUI, the trigger will be automatically rearmed and this will cause the above to be repeated forever. Click **Disable Trigger** to make StethoScope stop triggering. Choose some signals from the **Live** view again (the selected signals have become the snapshot signals since they began appearing) and you should be able to observe data being plotted as usual.

5.5 Understanding and Preventing Overflows

On the target, calls to **ScopeCollectSignals()** add data samples to the data buffer and the **LinkDaemon** task removes samples from the data buffer to send them to the host.

If the **LinkDaemon** task and host cannot keep up with the rate at which the data buffer is being filled, an overflow will occur when the data buffer is full. An overflow is more likely to occur with a small target data buffer and a large number of active signals.

5.5.1 Avoiding Overflows

You can help prevent buffer overflows by implementing one or both of the following:

- Try raising the priority of the **LinkDaemon** task for VxWorks targets.
- Try increasing the *sample* buffer size (see the **ScopeInitServer()** entry, [A. StethoScope API Reference](#)).

5.5.2 Overflow Behavior

When the sample buffer on the target overflows, the host is notified immediately. The host then throws away all the samples in its sample buffer (**Buffer Reset**) and clears its plot windows. The string **Buffer Reset** is displayed in the **Mini-Monitor** window whenever the sample buffer is cleared. If the **Plot** window clears frequently accompanied by the appearance of “Buffer Reset” messages in the **Mini-Monitor** window, it is a clear case of overflows on the target. Try the above recommendations to alleviate frequent overflows of the sample buffer.

5.6 Notes and Hints

Data Collection vs. Buffering

Do not confuse saving snapshots with data collection. Stopping data collection (by calling **ScopeCollectionModeDisable()** on the target) and taking a snapshot will both “freeze” the data displayed in the graphics window. They are, however, quite different actions as explained below.

- If you call **ScopeCollectionModeDisable()**, it prevents further data collection.
- If you take a snapshot, it saves all the data collected in the current cycle, but does not stop data collection. Other data-display windows continue to display the “Live” data.

Programmatic Access

All of the collection control functions can be accessed under program control via calls in the StethoScope API module on the target. See [12. Using a VxWorks Target](#) for details.

If No Data Appears

No data will be collected if collection is not active. That is, **ScopeCollectSignals()** must be called by the target during each sampling period. Try setting a breakpoint at **ScopeCollectSignals()**.

On VxWorks, type the following in the shell:

```
-> b ScopeCollectSignals
```


No data will appear (nor would you see an overflow count indicator in the **Data Collection** window) if the **LinkDaemon** task is starved for CPU time, for example, if some higher-priority process is using *all* of the processor.

Live Buffer

Be sure the **Plot** window is displaying the “Live” buffer rather than a saved buffer from a previous collection cycle.

6

Derived Signals

- 6.1 Introduction 77
- 6.2 Creating Derived Signals 78
- 6.3 Mathematical Operations 81
- 6.4 Troubleshooting Derived Signals 83

6.1 Introduction

A *derived signal* is a signal whose value is computed by mathematical operations on other signals. Derived signals are calculated by Wind River StethoScope on the host, not sampled from your real-time system. The derived-signals facility provides a simple means of scaling and offsetting signals, plotting differences and ratios of signals, and so forth.

Example 6-1 **Valid Derived Signals Include**


```
PosError    = PosDesired - Pos
Pos10       = Pos * 10
Pos10Plus3  = Pos10 + 3
Tan         = Sin / Cos
LogError    = log10 PosError
```

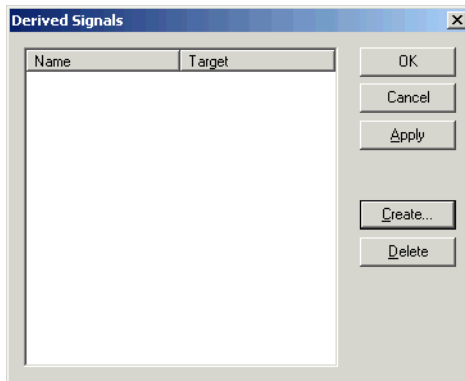
This chapter is organized in the following sections:

- [6.2 Creating Derived Signals](#), p.78 tells you how to create a derived signal.

- [6.3 Mathematical Operations](#), p.81 lists all the mathematical operations you can use in derived signals.
- [6.4 Troubleshooting Derived Signals](#), p.83 helps you to troubleshoot derived signals.

6.2 Creating Derived Signals

You must create a derived signal using the **Derived Signals** dialog box before you can select a derived signal in any of StethoScope's available plot windows. To create a derived signal, first open the **Derived Signals** dialog box, shown below, with the **File > Derived Signals** menu command (or the  button).

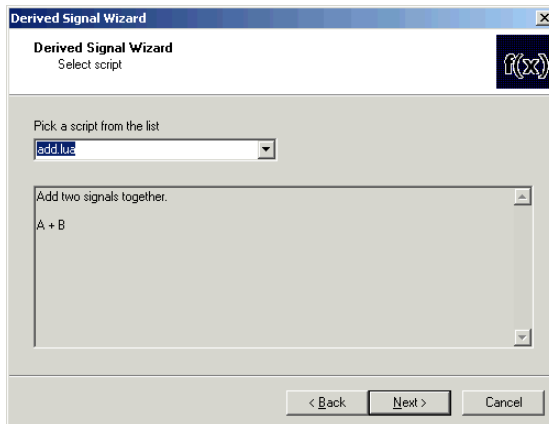


In the **Derived Signals** dialog box:

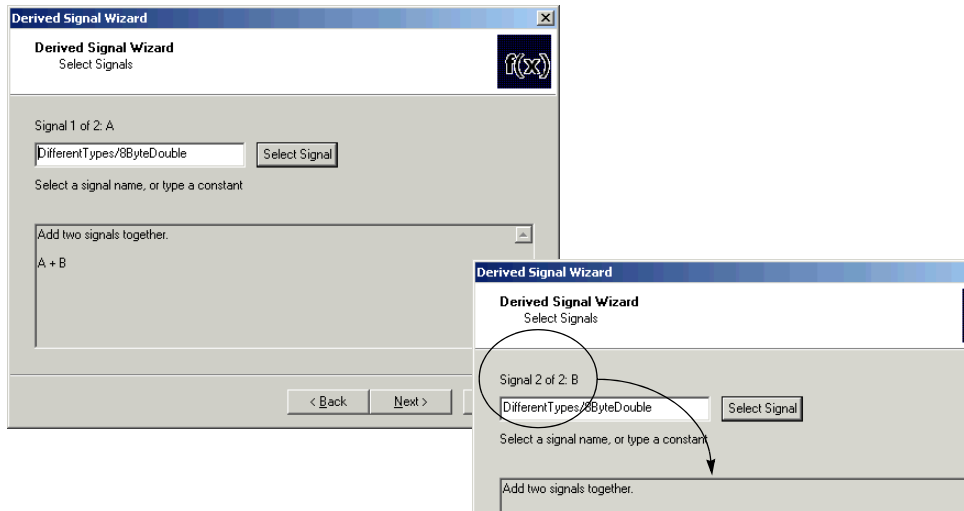
1. Click the **Create** button in the **Derived Signals** dialog box. The first in a series of **Derived Signal Wizard** dialog boxes, shown below, should open.



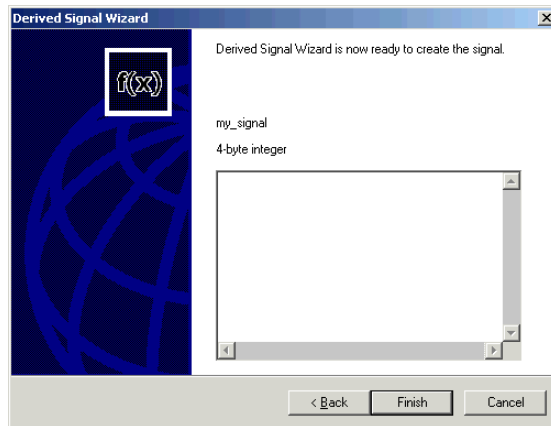
2. Enter the new signal's name in the **Name** text box. Make sure the name is unique.
3. Use the **Type** drop-down menu to select a type for the derived signal, such as a 4-byte integer, char, and so forth.
4. Use the **Target** drop-down menu to select one of your connected targets.
5. Click **Next** to open the next **Derived Signal Wizard** dialog box, shown below.



- Use the drop-down menu in the **Pick a script** from the list field to select a mathematical operation to use in creating your derived signal (you can see a list of the script items in [Table 6-1](#)).
- Click **Next** to open the next (**Select Signals**) **Derived Signal Wizard** dialog box, shown below. Use this dialog box to select a signals to apply to the script you previously selected. If the script you selected requires two or more signals (as in this example), this dialog box will ask you to enter the first signal, and the following dialog boxes will ask for the second and subsequent signals until they have all been selected.



- Click **Next** to open the final **Derived Signal Wizard** dialog box, shown below. There you can inspect the completed derived signal, then click **Finish** to exit the Wizard, or click **Back** to make changes.



9. Click **Finish** to create the derived signal, or click **Back** to make any desired changes. You can also click **Cancel** to exit without saving the new signal

When you click **Finish**, you will arrive back at the **Derived Signals** dialog box, which will display the newly created **Derived Signal** in its list. Click **Apply** to save your new signal if you want the dialog box to remain open to create more **Derived Signals**, or Click **OK** to save the new signal and exit the dialog box. Or you can click **Cancel** to close the dialog box without saving the newly created **Derived Signal**.

To modify a derived signal, you can click the **Edit** button and change it by hand; however, it is recommended that you delete the existing signal and re-enter your changed values. To delete a derived signal, select the signal from the list in the **Derived Signals** dialog box, then click **Delete** or use the **Delete** key.

When computing derived-signal values, StethoScope truncates any infinite values to a value of +/-10,000,000.0. This prevents numeric problems with recursively defined signals.

6.3 Mathematical Operations

The mathematical operations you can use in derived signals are shown in [Table 6-1](#).

Table 6-1 **Derived Signal Operations**

Operation	Meaning
abs	abs(A)
add	Add two signals together (A + B)
arccos	if abs(A)<=1 then arccos(A), else arccos(sign(A)) , result in degrees
arcsin	if abs(A)<=1 then arcsin(A), else arcsin(sign(A)) , result in degrees
arctan	atan(A) , result in degrees
arctan2	if (B==0) then 10000000*sign(A), else atan2(A,B)
avg	Average signal value
cos	cos(A) , where A is in degrees
divide	if (B==0) then 10000000*sign(A), else A/B
exp	exp(A)
exp10	exp10
filter1	A * previous_filter1 + (1-A)*B , where initial_filter1 = B
hypot	sqrt(A*A + B*B)
ln	ln(A)
ln10	ln10(A)
mult	Multiply two signals together (A * B)
saturate	if (A>B) then B, if (A<-B) then -B, else A
sin	sin(A) , where A is in degrees
sqrt	sqrt(abs(A))
sub	Subtract two signals (A - B)
tan	tan(A) , where A is in degrees
add	abs(A)

6.4 Troubleshooting Derived Signals

When a **Derived Signal** is defined, it should appear in the **Signals Tree** of every **Plot**, **Dump Plot**, and **Monitor** window.

There are two reasons derived signals may not appear:

- The derived signal has been toggled off in the **Signal Manager**.
- The signal will only appear when all of its operands are defined. If any of the operand signals do not appear in the buffer being viewed (perhaps caused by a typing mistake), then the derived signal will not be available in that buffer.

7

The Plot Window

- 7.1 Introduction 85
- 7.2 Plot Window Tour 86
- 7.3 Menu Bar 87
- 7.4 Toolbar 91
- 7.5 Signals Bar 92
- 7.6 Signal Properties Dialog Box 98
- 7.7 Pop-up Menus 103
- 7.8 Strip Chart Feature 103
- 7.9 Displaying Events 103
- 7.10 Setting Preferences for a New Plot Window 106


7.1 Introduction

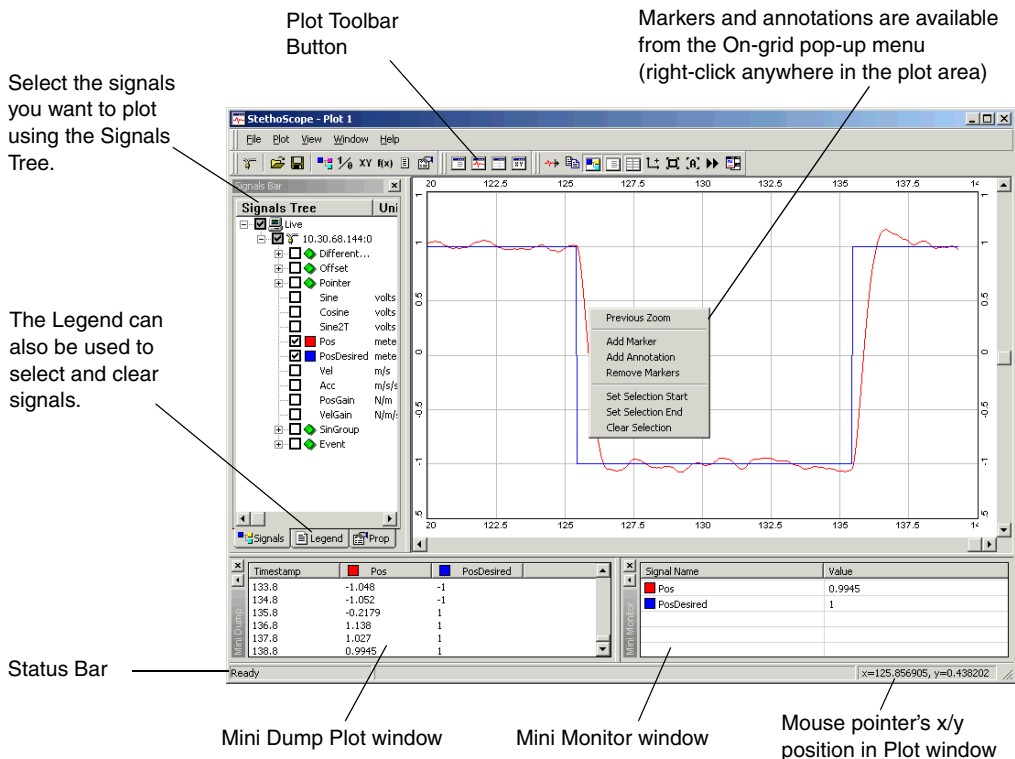
The **Plot** window graphically displays real-time signals plotted over time. Each signal appears on the plot grid in a different color, with a legend showing each signal's color. The figure below depicts Wind River StethoScope's **Plot** window,

connected to the StethoScope demo. You can have multiple **Plot** windows open at the same time, and each with different data.


Before using a **Plot** window, it may help to understand how and when data is collected from the target—please refer to [5.6 Notes and Hints](#), p.74.

7.2 Plot Window Tour

A **Plot** window, an example of which is shown below, opens when you first start StethoScope. To open additional **Plot** windows, use the **File > Plots** menu command (or the  button).



7.2.1 Displaying Signal Values in a Plot Window

1. If you do not already have a **Signals Bar** open in your **Plot** window, open one with the **View > Signals Bar** menu command (or the  button). Make sure the **Signals** tab is selected in the **Signals Bar**, so that a **Signals Tree** is displayed.
2. Use the **Signals Tree** to select which signals you want to display in the graph. The signals in this tree were installed using the **Signal Manager** (see [4. Using the Signal Manager](#)). Selecting a signal from the **Signals Tree** causes the following:
 - The signal data will be plotted in the window, using the next available plot color. You can select a different color using the **Signal Properties** dialog box (see [7.6 Signal Properties Dialog Box](#), p.98), or the **Preferences** dialog box (see [Colors View](#), p.38.)
 - The **Legend** tab view will be updated to include the selected signal. The **Legend** shows you what color is being used for each selected signal. You can also use the **Legend** to select and clear signals. [7.5.2 Legend Tab View](#), p.94 describes the **Legend** tab view .

When you right-click anywhere in the plot area, the **On-grid** pop-up menu, shown in the figure above, opens with several options pertaining specifically to the plot area. These additional options are described in detail in [7.7 Pop-up Menus](#), p.103.

Most of the remaining functionality in the **Plot** window is provided through the toolbars and menus. For common **Plot** window feature descriptions, refer to the following sections for more information:

- File Menu (Section 3.2)
- Toolbars (Section 3.4)
- Pop-up Menus (Section 3.6)

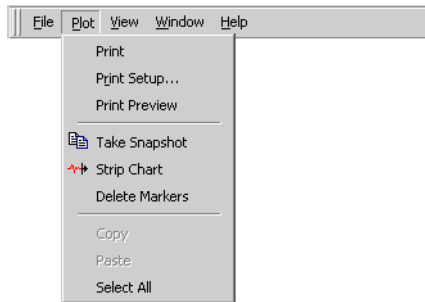
7.3 Menu Bar

Some of the **Menu Bar** items are discussed in detail in [3.2 File Menu](#), p.26, where it is mentioned that the **File**, **Window**, and **Help** menu items are consistently the same across all data display windows. For the **Plot** window, however, the **Plot** and **View** menu items contain some commands that are unique to the **Plot** window.

Even though partially redundant, these menu items are described in detail in the following sections.

7.3.1 Plot Menu

The **Plot** menu item, shown below, contains commands for working with the plots in the **Plot** window.



The **Plot** menu commands are:

Print

Prints the signal traces in the data display window. It opens a **Print** dialog box where you can configure your printer options.

Print Setup

Allows you to select printer parameters and characteristics before printing.

Print Preview

Lets you see, on a print mock-up screen, what the printed page will look like before actually printing it.

Take Snapshot

Saves a copy of all the active signals (not just the selected signals), for all connected targets. For more information, see [11.2 Taking Snapshots](#), p.152.

Strip Chart

Changes the display so that it presents a continuous, scrolling plot (instead of repainting the plot when it fills every 10 seconds). Any snapshots you may be displaying on the grid when you select **Strip Chart** become unselected; that is, snapshots do not appear on the grid while **Strip Chart** is selected. This

command is only available in the **Plot** window. For more information, see [7.8 Strip Chart Feature](#), p.103.

Delete Markers

Deletes all markers, measures, and annotations from the grid area. For more information, see Sections [Adding and Removing Markers](#), p.54, [Taking and Removing On-grid Measurements](#), p.56, and [Adding Annotations](#), p.55 respectively.

Copy

Copies the selected content to the clipboard, and deselects the selected data. Note that this option is greyed out and unavailable unless you have some data in the plot area selected.

Paste

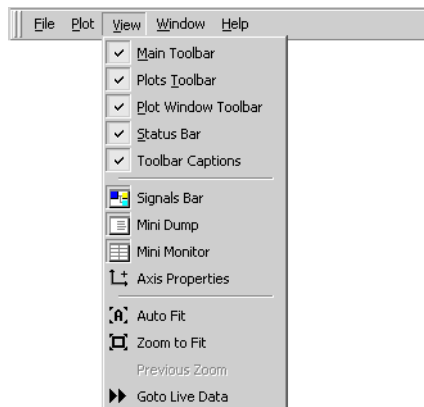
Copy the contents of the clipboard to the window you are in, at the insertion point. Note that this options is greyed out and unavailable until you have copied some data to the clipboard.

Select All

Selects and highlights the entire plotted area up to the instant you selected the option, enabling you to copy it to the clipboard.

7.3.2 View Menu

The **View** menu item, shown below, contains commands for displaying toolbars and auxiliary views in the **Plot** window, and some other **Plot** window-specific commands.



The **View** menu commands for the **Plot** window are:

Main Toolbar

Controls whether the toolbar representing a selection of items from the **File** menu is displayed on StethoScope's toolbar. For more information, see [7.4 Toolbar](#), p.91.

Plots Toolbar

Controls whether the toolbar representing a selection of items from the **File > Plot** menu command is displayed on StethoScope's toolbar. For more information, see [3.4 Toolbars](#), p.48.

Plot Window Toolbar

Controls whether the toolbar used within a specific data-display window is visible. The buttons represent items from the **Plot** and **View** menus for the specified data-display window. For more information, see [3.4 Toolbars](#), p.48.

Status Bar

Controls whether the status line along the bottom of the window is visible. For more information, see [3.5 Status Bar](#), p.52.

Toolbar Captions

Controls whether the names of the panels (**Signals Bar**, **Mini Dump**, and **Mini Monitor**) are displayed in the **Plot** window above their respective invocations.

Signals Bar

Controls whether a **Signals Bar** panel appears in the window. The **Signals Bar** includes tabs for **Signals**, **Legends**, and **Properties**.

Mini Dump

Creates a **Mini Dump Plot** window within the **Plot** window. This is simply a smaller version of the **Dump Plot** window. It lets you see a running history of signal values. This command is only available in the **Plot** window.

Mini Monitor

Creates a **Mini Monitor** window within the window. This is simply a smaller version of the **Monitor** window. It lets you peek at, and poke, data on the target. This command is only available in the **Plot** window.

Axis Properties

Allows you to set certain parameters that affect the appearance and labelling of the X or Y axis of a **Plot** or **Plot XY** plotting area.

Auto Fit

Causes the plotting area to be scaled dynamically and automatically to the extremes of the data in the Y direction being displayed in the window. It thus allows all data points to appear on the plot. It toggles on or off, but when on,

it essentially has the effect of using the **Zoom to Fit** button continuously. This command is only available in the **Plot** and **Plot XY** windows.

Zoom to Fit

Changes the scales and offsets so that all the signals fit and take up the entire **Plot** window. This option is only available in the **Plot** and **Plot XY** windows. See also **Previous Zoom**.

Previous Zoom

Reverses the action of the last zoom action. Note that when you use the **Zoom to Fit** command, the history of all previous zoom actions is lost, therefore this command will have no effect immediately after a **Zoom to Fit** command. This command is only available in the **Plot** and **Plot XY** windows.

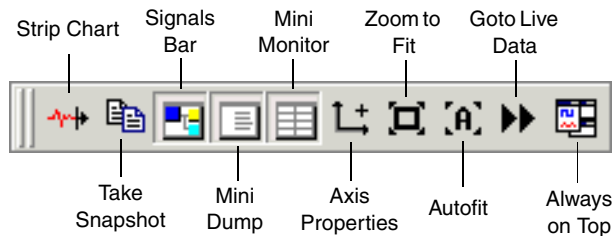


NOTE: For more information on the **Plot** and **Plot XY** windows, mentioned above in **Auto Fit** through **Previous Zoom**, see [7. The Plot Window](#) and [8. The Plot XY Window](#).


7.4 **Toolbar**

In a default **Plot** window, the first two dockable toolbars are identical to the toolbars shown in [3.4.1 Main Toolbar](#), p.48 and [3.4.2 Plot Toolbar](#), p.48, with the menu items described here also. But the right-most dockable toolbar is specific to the **Plot** window. All the toolbars are dockable, which means you can move them to other locations, on or off the window, simply by dragging them. (The menu bar is also dockable.) Each of the toolbars can be independently displayed or hidden using the **View** menu item.

The **Plot** window's toolbar, shown below, is a subset of the one described in detail in [3.4.3 Plot Window Toolbar](#), p.50.



7.5 Signals Bar

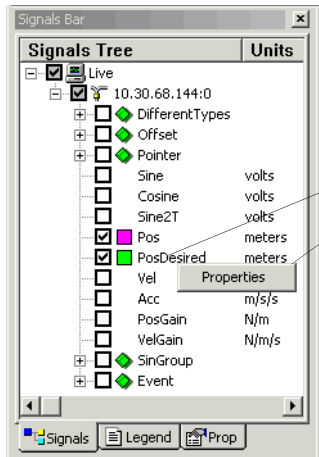
The **Signals Bar**, shown in the figure below, is a sub-window in the StethoScope GUI that allows you to view and configure information that affects what is plotted in the graph area. To open a **Signals Bar** if one is not already displayed, use the **View > Signals Bar** menu command (or the  button).

The **Signals Bar** contains the following tab views:

- **Signals**
- **Legend**
- **Properties**

7.5.1 Signals Tab View

The **Signals Tree** in the **Signals** tab view (see the figure below) displays all the signals available to be plotted. They are shown in an expandable tree structure containing signals that have been installed by the **Signal Manager** (see [4. Using the Signal Manager](#)). Traces on the graph are color-coded to the signals selected in the **Signals Tree**.



Right-click on a signal in the Signals Tree window to open a pop-up menu

The **Signals Tree** is displayed by default when you open a **Plot** window, but it may be re-displayed at any time by clicking the **Signals** tab at the bottom of the sub-window.

The **Signals Tree** allows you to easily locate any signal that has been installed and add it to the graph of signal traces. Each node of the tree, and each signal, is a check box. Clicking an individual signal's check box adds that signal to the graph, or, conversely, clearing the check box removes that signal from the graph. If you click a node check box, all the signal check boxes belonging to that node (but not including nodes below it) are checked at once and their signals are added to the graph. Conversely, clearing a node check box removes all signals below that node from the graph with the single click.

If a node check box is checked, but has a grey fill, it indicates that some, but not all, of the signals belonging to that node are checked and displayed on the graph. You may have to scroll down to see which ones are checked.

The **Signals Tree** is expanded or collapsed using the icon to the left of each node check box as follows:

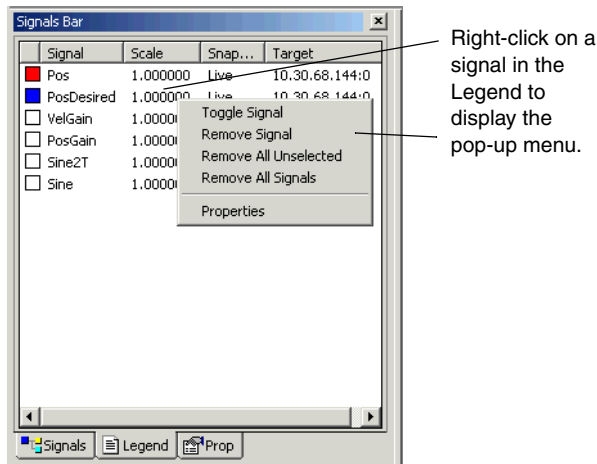
- = collapsed; click to expand down to the next node(s).
- = expanded; click to collapse up to the next node.

The **Units** column in the **Signals** tab view shows the physical measurement units of each signal.

When you right-click a signal in the **Signals** tab view, a pop-up menu opens containing a single menu item as shown in the figure above.

7.5.2 Legend Tab View

The **Legend** tab view shows you what color is assigned to each signal on the plot. It can also be used to select which signals to plot. The figure below shows a **Legend** in the **Signals Bar** sub-window.



The columns in the **Legend** tab view display current parameter settings for each signal, including:

Scale

The scale factor applied to second (Y) component signal's values in relation to the first (X) component signal's values.

Snapshot

Whether the data is live, or from a snapshot.

Target

The name and scope index of the target you are connected to.

When you right-click on a signal in the **Legend** tab view a pop-up menu opens with the following options to manipulate the signal:

Toggle Signal

Turns the signal trace **on** or **off**. This is the equivalent of going to the **Signals Tree** and alternately checking and unchecking the signal's check box, except that it leaves the signal's name in the **Legend** list when you toggle it **off**.

Remove Signal

Removes the signal from the graph. This is equivalent to toggling the signal **off**, except it removes the signal's name from the **Legend**.

Remove All Unselected

Removes all signals that have been toggled **off**.

Remove All Signals

Removes all signals regardless of their toggled status.

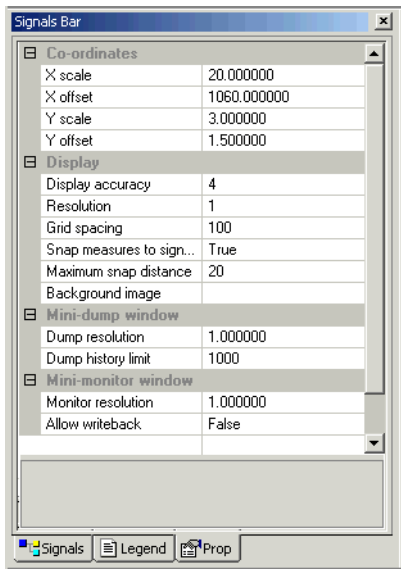
Properties

Opens the **Signal Properties** dialog box (see [7.6 Signal Properties Dialog Box](#), p.98).

You can change the colors used for plot lines by using the **File > Preferences** menu command (see [Colors View](#), p.38), or by using the **Signal Properties** dialog box (see [7.6 Signal Properties Dialog Box](#), p.98).

7.5.3 Properties Tab View

You can view and change the properties for only the **Plot** window you are in by clicking the **Properties** tab in the **Signals Bar**. This tab view, shown in the figure below, accesses properties for the graphic display window environment (for the signals themselves, use the **Signal Properties** dialog box, described in [7.6 Signal Properties Dialog Box](#), p.98).



The graphic display window environment properties shown in this tab view are:

Coordinates

X Scale

Controls the width of the plot grid in units. For example, when first started, X Range defaults to 3 and the plot displays the X-axis from **1.5** to **-1.5**.

X Offset

Controls the unit value for the left-most coordinate on the plot.

Y Scale

Controls the height of the plot grid in units. For example, when first started, Y Range defaults to 3 and the plot displays the Y-axis from **1.5** to **-1.5**.

Y Offset

Controls the unit value for the top coordinate on the plot. When first started, it defaults to **1.5**, and you will see that the default Y value of **1.5** is the top value. This value will change automatically if you use the **Zoom to Fit** command.

Display

Display Accuracy

Controls the number of significant digits displayed in the grid line markers. Set this property to the number of places you want displayed to the right of the decimal point. Enter a value between 0 (for integer numbers) and 6 (default=4).

Resolution

Permits plotting of only a subset of the collected data points. This is useful for increasing rendering speed. A divisor value of n causes only every n th point to be plotted. Thus, if 1000 points are being collected and the **Resolution** is 5, then 200 points will be displayed. Of course, you may not want to reduce resolution if your data can contain “glitches” or high-frequency phenomena (default=1).

Grid Spacing

Specifies the minimum distance (in pixels) allowed between the grid lines. Increasing this number decreases the number of grid lines, decreasing this number increases the number of grid lines. Enter a value between 5 and 500 (default=100).

Snap measures to signals

Controls whether or not measures are “snapped” to the signal lines on the grid. Measures are described in [Taking and Removing On-grid Measurements](#), p.56 (default=True).

Maximum snap distance

Controls how far away (in pixels) you can start a measure and still have it snap to the plot line. Measures are described in [Taking and Removing On-grid Measurements](#), p.56 (default=20).

Background image

Allows you to specify special images to be displayed in the graphing area, with the graph itself superimposed on top (default=none).

Mini-dump window

Dump resolution

Controls how often to refresh the values in the **Mini Dump** window, in seconds (default = 1.000000).

Dump history limit

Controls how many lines of historical data to maintain in the **Mini Dump** window (0 = display all, default = 1000).

Mini-monitor window

Monitor resolution


Controls how often to refresh the values in the **mini Monitor** window (default = 1.000000).

Allow writeback

Select **True** to create a **writeback** column for writing modified signal values back out to the target (default=**False**).

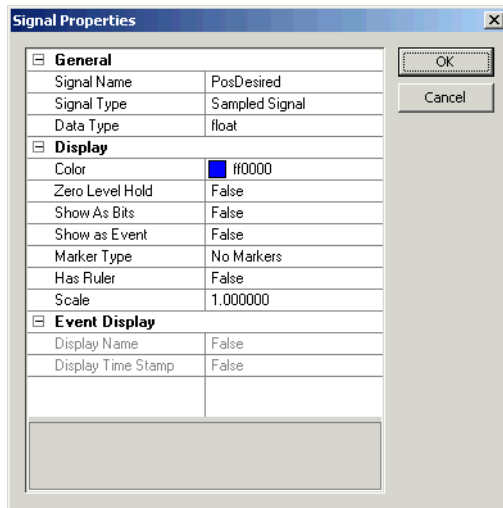
To modify any value, just type directly into the text field.

Any changes you make in this window have no effect on any other open **Plot** windows.

To change the defaults used when new **Plot** windows are created, use the **File > Preferences** menu command (or the  button), described in [3.2.12 Preferences](#), p.36.

7.6 Signal Properties Dialog Box

Characteristics of the line used to plot each signal in this window can be configured using the **Signal Properties** dialog box, shown in the figure below.



This dialog box can be opened from any one of three places:

- Right-click a signal in the **Signals** tab view.
- Right-click a signal in the **Legend** tab view.
- Right-click a signal trace in the data-display area.

In the pop-up menu that opens, select **Properties**.

The **Signal Properties** dialog box opens with the following parameters:

General

Signal Name and Type

These first two lines are the signal's name (as shown in the **Signals Tree**) and its type.

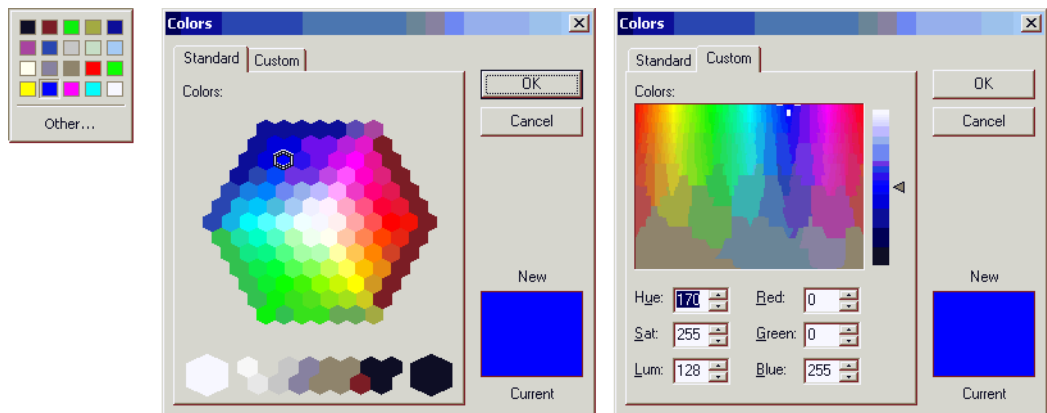
Data Type

This is the C++ data type of this signal (program variable) See [Table 14-1](#) for a list of allowable types.

Display

Color

Clicking on the current value opens a small color palette with a basic color selection from which to choose.



If you would like a color that is not on this color palette, you can click **Other** to open the **Colors** dialog box where you can select a new color from

the larger selection in the **Standard** tab view, or you can create an entirely new color using the **Custom** tab view.

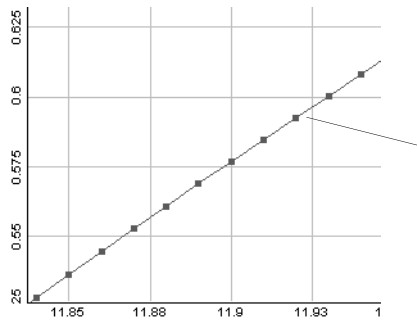


NOTE: An alternative method for changing colors of plotted lines on the **Plot** window graph is given in the discussion of colors preferences in [Colors View](#), p.38.

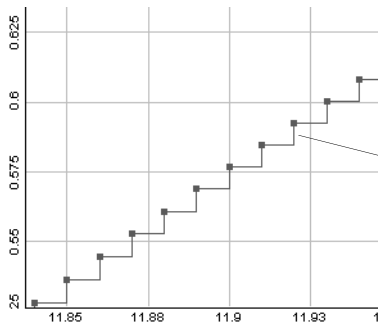
Zero Level Hold

Select **True** from the drop-down menu to “hold” the value of the signal until the next sample arrives.

To demonstrate this feature, consider the sampling of a sine wave at a rate of 100Hz, as shown in the figure below. In the upper view in this figure, with **Zero Level Hold** turned off (= **False**), the data points are connected as usual with straight lines, even though there is no information about the actual values between data points. In the view below, the same data, but with the **Zero Level Hold** feature turned on (= **True**), shows how the line plotted between the same data points is now a straight horizontal line of the same value as the previous data point, until the next data point comes in, at which time the plot line goes vertical up to that value.



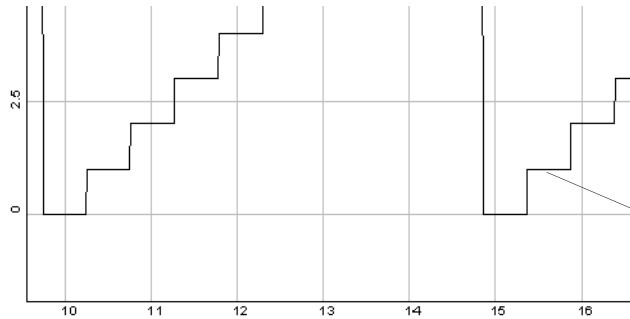
Data plotted with Zero Level Hold turned off



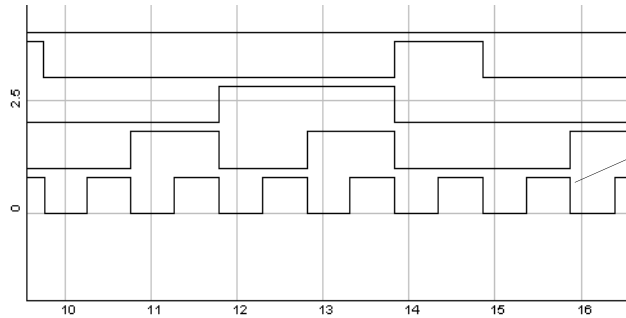
Same data plotted with Zero Level Hold turned on

Show as Bits

Select **True** from the drop-down menu to cause a signal (program variable) being plotted to be represented as a sequence of bits, each bit with its off and on (0 and 1) values indicated. The bits are rendered individually by discrete horizontal lines distributed up the Y axis from bit position 0 starting at the bottom.



Data plotted with Show as bits turned off



Same data plotted with Show as bits turned on

To show this feature, a **Derived Signal**, created using a single 8-bit integer, is incremented through values from 0 to 15 and back to 0 again at the rate of two increments per second, and then repeated continuously until stopped. In the resulting plot the top graph shows the variable's value with **Show as bits turned off** (= **False**), and the bottom graph shows the same data with **Show as bits turned on** (= **True**). You can easily observe the pattern of the individual bits (horizontal lines) making up the 16-bit integer, showing their 0 and 1 positions with respect to time.

This feature works equally well with all signal types, but the resulting bit patterns may be more difficult to decipher depending on the complexity of the signal.

Show as Event


Select **True** from the drop-down menu to cause samples to be marked with vertical lines instead of connecting the individual samples with lines.

Either or both of the selections made in the **Event Properties** section below are displayed along side the vertical marker. For more information on displaying events, see [7.9 Displaying Events](#), p.103.


Marker Type

This drop-down list allows you to select a different symbol (or marker) to be plotted for each sample of this signal. The choices are:


Square

A square () plots each sample.

Plus

A plus sign () plots each sample.

Diamond

A diamond () plots each sample.

Has Ruler

This true/false option displays a separate ruler with colored numbers matching the signal's color on the left (Y) axis. If you have, say, three signals with rulers on, there will be three separate color-coded rulers on the left plot boundary. (The default is no rulers on.)

Scale

Each plotted value is scaled (multiplied) by the value you enter. Default is 1.000000.


When you have finished changing parameters, click **OK** to save your changes and exit the dialog box. Click **Cancel** to exit the dialog box without saving any changes you have made.

7.7 Pop-up Menus

Pop-up menus that open in response to a right-click in the **Plot** window offer options that are unique to specific signals, and in some cases, to where the cursor is within the window. These menus are described in detail in [3.6 Pop-up Menus](#), p.52. They are accessed from different places in the **Plot** window, including:

- **On-grid** (anywhere in the plotting area, but NOT on a trace line)
- **On-trace** (exactly on a trace line in the plotting area)
- **Signals** tab view
- **Legend** tab view

7.8 Strip Chart Feature

This feature, turned on or off using the **Plot > Strip Chart** menu command (or the  button), changes the behavior of the graph area so that it presents a continuous, scrolling plot (instead of repainting the plot every 10 seconds). The overall appearance of the **Plot** window does not change in any way. Any snapshots you may be displaying on the grid when you select **Strip Chart** become unselected; snapshots do not appear on the grid while **Strip Chart** is selected.

7.9 Displaying Events

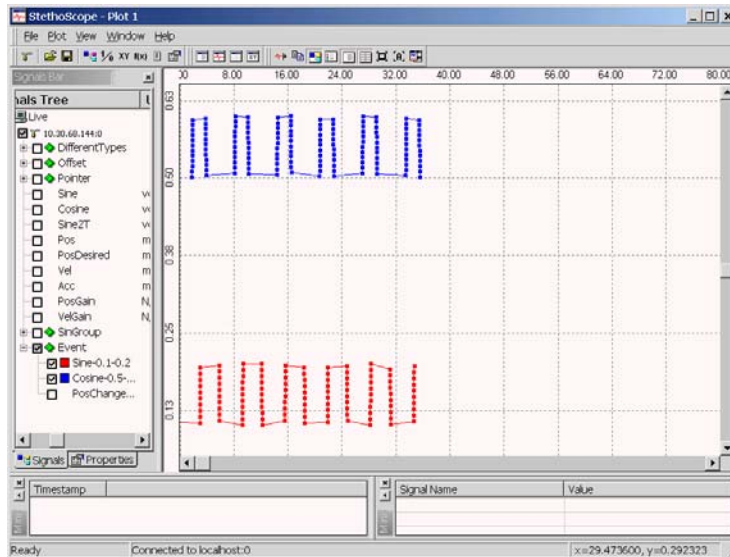
This section describes how events, collected using the StethoScope API, are displayed. Event collection routines include **ScopeEventsCollect()** (collects an event identifier and the value of a variable) and **ScopeEventsMessage()** (collects a string).

Events appear under the **Event** tree branch in the **Signals Bar** for a target. This tree contains a list of all the events that were thrown. By default, events that collect a value (from **ScopeEventsMessage()**) are displayed much like normal signals. The values collected when a certain event is thrown are treated like a sample and the samples are joined by lines to make signals. Alternatively, the **Zero Order Hold**

option can be used to join samples in a step-wise manner (the last value is held until a new sample appears).

7.9.1 Events Collected as Signals

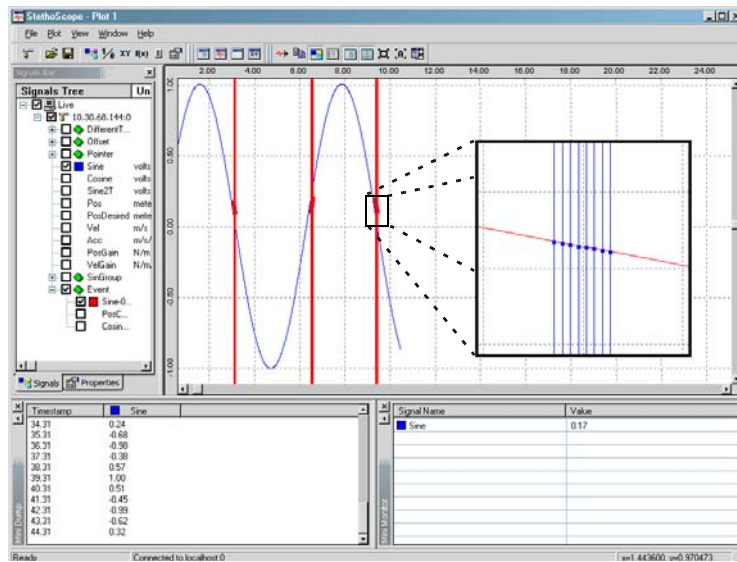
An example of the display of events collected as **signals** is shown in the figure below.



Events that collect a value (from **ScopeEventsCollect()**) can also be displayed as vertical lines, or “markers.” A marker appears at the temporal location on the plot corresponding to the timestamp that was collected with the event. When events are displayed as vertical lines, the name of the event (**event ID**), and the timestamp can be displayed by choosing options **Display Name** and **Display Timestamp** respectively.

7.9.2 Events Collected as Markers

An example of the display of events collected as a value but displayed as **markers** is shown in the figure below.



Event messages, collected using `ScopeEventsMessage()`, appear as "Messages" under the "Event" tree. Turning on event messages causes vertical lines to be displayed, representing the temporal location of the occurrence of the event. The message is also displayed.

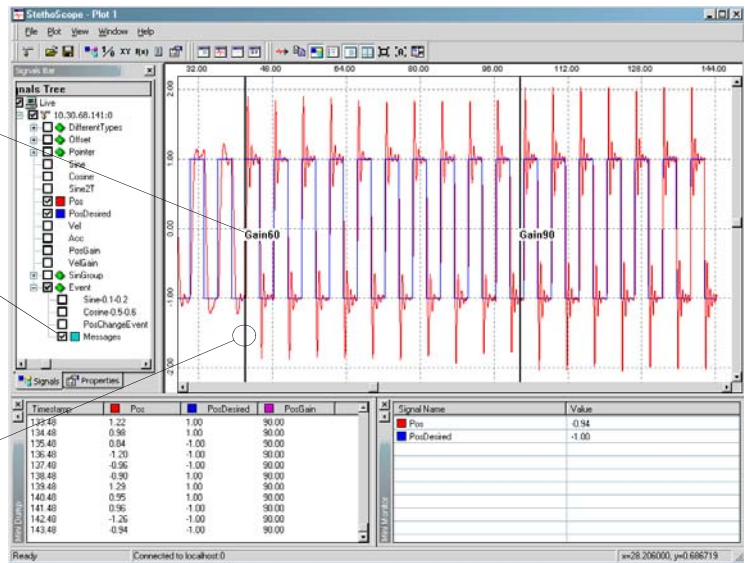
7.9.3 Events Collected as Messages

An example of the display of events collected as **messages** is shown in the figure below.

The message string is printed close to the marker line that indicates when the event was collected

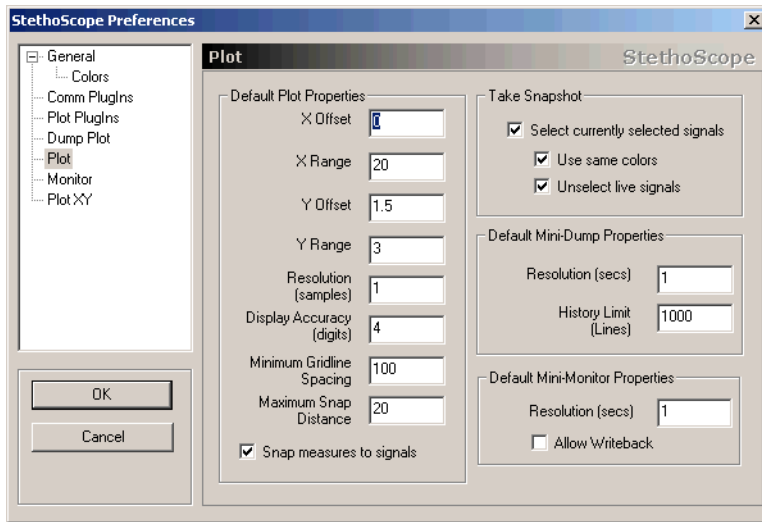
Clicking Event and Messages turns on the display of event messages


The marker is a vertical line from top to bottom of the graph, showing exactly when the event was collected



7.10 Setting Preferences for a New Plot Window

The **Preferences** dialog box allows you to set default parameters used by the StethoScope GUI when it starts, and for new data-display windows when they are created (whereas the “properties” described in this chapter apply only to currently open windows). The **Plot** preferences view, shown in the figure below, allows you to change the default values used when **Plot** windows are created. It also allows you to control some aspects of what happens when a snapshot is taken. One additional preference in particular that affects the **Plot** window is **Colors** ([Colors View](#), p.38).



To modify these preferences, use the **Preferences** dialog box, opened with the **File > Preferences** menu command (or the  button), then click **Plot** in the left panel to display and set the following parameters:

Default Plot Properties

X Offset

Controls the unit value for the left-most coordinate on X axis of the plot. When first started, it is 0. This value will change automatically if you use the **Zoom to Fit** command.

X Range

Controls the width of the plot grid in units. For example, when first started, X Range defaults to 20, and the plot displays the X-axis from 0 to 20.

Y Offset

Controls the unit value for the top coordinate on the plot. When first started, it is 1.5, and you will see that the Y value of 1.5 is the top value. This value will change automatically if you use the **Zoom to Fit** command.

Y Range

Controls the height of the plot grid in units. For example, when first started, Y Range defaults to 3 and the plot displays the Y-axis from 1.5 to -1.5.

Resolution (samples)

Permits plotting of only a subset of the collected data points. This is useful for increasing rendering speed. A divisor value of **n** causes only every *n*th point to be plotted. Thus, if **1000** points are being collected and the **Resolution** is **5**, then **200** points will be displayed. Of course, you may not want to reduce resolution if your data can contain “glitches” or high-frequency phenomena (default=1).

Display Accuracy (digits)

Controls the accuracy of the grid line markers. Set this property to the number of places to the right of the decimal point to use in the grid markers. Enter a value between 0 (for whole numbers) and 6 (default=4).

Minimum Grid Line Spacing

Specifies the minimum distance (in pixels) allowed between the grid lines. Increasing this number decreases the number of grid lines, decreasing this number increases the number of grid lines. Enter a value between 5 and 500 (default=0.25).

Maximum Snap Distance

Controls how far away (in pixels) you can start a measure and still have it “snap” to the plot line. Measures are described in [Taking and Removing On-grid Measurements](#), p.56.

Snap Measure to Signals

Check box controls whether or not measures are “snapped” to the signal lines on the grid. Measures are described in [Taking and Removing On-grid Measurements](#), p.56.

Enable Warnings

Controls whether or not warning messages are sent to the **Console** window.

Take Snapshot

Unlike all other preferences, which only impact data-display windows you may create later, these preferences take effect immediately. See [11. Working with Snapshots](#) for more information on snapshots.

Select currently selected signals

If selected, the same signals currently selected in the live buffer will also be selected in the snapshot.

Use same colors

If selected, the same colors will be assigned to the signals in the snapshot. If you want the snapshot to use different colors for each signal than are used for live data, clear this check box.

Unselect live signals

If selected, the live signals will become unselected when the snapshot is taken.

Default Mini Dump Properties

These properties apply to the **Mini Dump** window inside a new **Plot** window. They have no effect on any full-size **Dump Plot** windows.

Resolutions (secs)

Controls how often (in seconds) to refresh the values in the **Mini Dump** window (default = 1).

History Limit (lines)

Controls how many lines of historical data to maintain in the Mini Dump window (default = 1000).

Default Mini Monitor Properties

These properties apply to the **Mini Monitor** window inside a new **Plot** window. They have no effect on any full-size **Monitor** windows.

Resolution (secs)

Controls how often (in seconds) to refresh the values in the **Mini Monitor** window (default = 1.000000).

Allow Writeback

Check box controls whether or not you can use the **Mini Monitor** window to write modified signal values back out to the target (default=False).

8

The Plot XY Window

- 8.1 Introduction 111
- 8.2 Creating XY Signal Pairs 112
- 8.3 Plot XY Window Tour 113
- 8.4 Menu Bar 115
- 8.5 Toolbar 117
- 8.6 Signals Bar 118
- 8.7 Signal Properties Dialog Box 123
- 8.8 Pop-up Menus 127
- 8.9 Setting Preferences for a New Plot XY Window 128

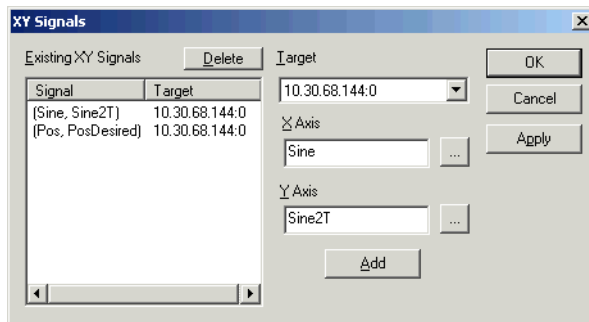
8.1 Introduction

The **Plot XY** window is used to graph pairs of signals against each other. In most aspects, it is very similar to the **Plot** window. The main difference is that only XY signal pairs show up in the **Signals Tree**.

8.2 Creating XY Signal Pairs

You must create XY signal pairs before you can use the **Plot XY** window.

Signal selection for **Plot XY** windows differs from signal selection for the other types of data-display windows, because each plotted line requires *two* signals. These XY signal pairs are created using the **XY Signals** dialog box shown in the figure below, opened with the **File, XY Signals** menu command (or the **XY** button).



8.2.1 Creating a Signal Pair

In the **XY Signals** dialog box:

1. Choose a target from the **Target** drop-down list box.
2. Choose the signal to plot on the X Axis by using the **X Axis** drop-down list box. The list box contains only active signals for the selected target. Double-click the desired signal.
3. Choose the signal to plot on the Y Axis by using the **Y Axis** drop-down list box. The list box contains only active signals for the selected target. Double-click the desired signal.
4. Click **Add**.
5. Repeat the above steps for each **XY Signal** pair you want to be able to see in the **Plot XY** window.
6. When you are done, click **OK**, **Cancel**, or **Apply**.


8.2.2 Deleting a Signal Pair

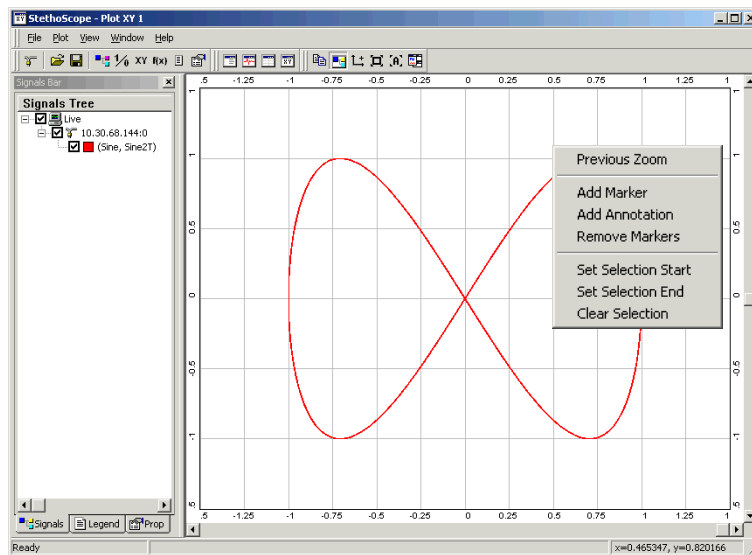
1. In the **XY Signals** dialog box, select the signal pair from the **Existing XY Signals** list.
2. Click **Delete**.

8.2.3 Modifying a Signal Pair



Delete the pair, then add a new one.

8.3 Plot XY Window Tour

Open the **Plot XY** window, shown below, using the **File > Plots > Plot XY** menu command (or the  button).



8.3.1 Displaying Signal Values in a Plot XY Window

1. If you do not already have a **Signals Bar** open in your **Plot XY** window, open one with the **File > Signals Bar** menu command (or the  button).
2. Make sure the **Signals** tab is selected in the **Signals Bar**, so that a **Signals Tree** is displayed.
3. Use the **Signals Tree** to select which signals you want to display in the graph. For details on using **Signals Trees**, see [4. Using the Signal Manager](#). If you do not see any signals in the **Signals Tree**, use the **File > XY Signals** menu command (or the  button) to open the **XY Signals** dialog box and create signal pairs, as described in [8.2 Creating XY Signal Pairs](#), p.112. Selecting a signal from the **Plot XY** window's **Signals Tree** causes the following:
 - The signal data will be plotted in the window, using the next available plot color. You can select a different color using the **Signal Properties** dialog box (see [8.7 Signal Properties Dialog Box](#), p.123), or the **Preferences** dialog box (see [3.2.12 Preferences](#), p.36.)
 - The **Legend** tab view will be updated to include the selected signal. The **Legend** shows you what color is being used for each selected signal. You can also use the **Legend** to select and clear signals. See [7.5 Signals Bar](#), p.92 for information on using the **Legend** window.

When you right-click anywhere in the plot area, the **On-grid** pop-up menu shown in the figure above opens with several options pertaining specifically to the plot area. These additional options are described in detail in [8.8 Pop-up Menus](#), p.127.

Most of the functionality in the **Plot XY** window is identical to that in the **Plot** window, except there is no Strip Chart feature, **Mini Dump** window, or **Mini Monitor** window. For common **Plot XY** window feature descriptions, refer to the following sections for more information:

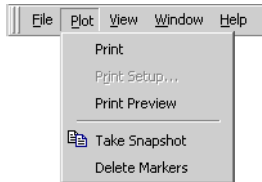
- [3.2 File Menu](#), p.26
- [3.4 Toolbars](#), p.48
- [3.6 Pop-up Menus](#), p.52

8.4 Menu Bar

Some of the **Menu Bar** items are discussed in detail in [3.2 File Menu](#), p.26, where it is mentioned that the **File**, **Window**, and **Help** menu items are consistently the same across all data display windows. For the **Plot XY** window, however, the **Plot** and **View** menu items contain some commands that are unique to the **Plot XY** window. Even though partially redundant, these menu items are described in detail in the following sections.

8.4.1 Plot Menu Commands

The **Plot** menu item contains commands for working with the plots in the **Plot XY** window.



The **Plot** menu commands are:

Print

Prints the signal traces in the data display window. It opens a **Print** dialog box where you can configure your printer options.

Print Setup

Allows you to select printer parameters and characteristics before printing.

Print Preview

Lets you see, on a print mock-up screen, what the printed page will look like before actually printing it.

Take Snapshot

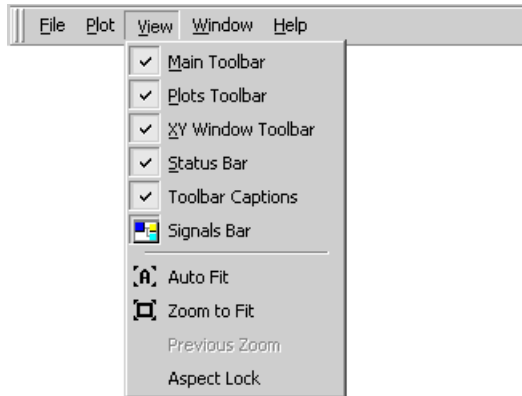
Saves a copy of all the active signals (not just the selected signals), for all connected targets. For more information, see [11.2 Taking Snapshots](#), p.152.

Delete Markers

Deletes all markers, measures, and annotations from the grid area. For more information, see [Adding and Removing Markers](#), p.54.

8.4.2 View Menu Commands

The **View** menu item, see below, contains commands for displaying toolbars and auxiliary views in the **Plot XY** window, as well as some **Plot XY** window sizing commands.



The **View** menu commands are:

Main Toolbar

Controls whether the toolbar representing a selection of items from the **File** menu is displayed on StethoScope's toolbar. For more information, see [8.5 Toolbar](#), p.117.

Plots Toolbar

Controls whether the toolbar representing a selection of items from the **File > Plot** menu command is displayed on StethoScope's toolbar. For more information, see [3.4 Toolbars](#), p.48.

XY Window Toolbar

Controls whether the toolbar used within a specific data-display the **Plot XY** window is visible. The buttons represent items from the **Plot** and **View** menus for the specified data-display window. For more information, see [3.4 Toolbars](#), p.48.

Status Bar

Controls whether the status line along the bottom of the window is visible. For more information, see [3.5 Status Bar](#), p.52.

Toolbar Captions

Controls whether the names of the panels (**Signals Bar**, **Mini Dump**, and **Mini Monitor**) are displayed in the **Plot** window above their respective invocations.

Signals Bar

Controls whether a **Signals Bar** panel appears in the window. The **Signals Bar** includes tabs for **Signals** and **Properties**.

Auto Fit

Causes the plotting area to be scaled dynamically and automatically to the extremes of the data in the Y direction being displayed in the window. It thus allows all data points to appear on the plot. It toggles on or off, but when on, it essentially has the effect of using the **Zoom to Fit** button continuously. This command is only available in the **Plot XY** window.

Zoom to Fit

Changes the scales and offsets so that all the signals fit and take up the entire plot window. The scales and offsets remain at these values until **Zoom to Fit** is selected again. This command is only available in the **Plot XY** windows. See also **Previous Zoom**.

Previous Zoom

Reverses the action of the last zoom action in a **Plot** or **Plot XY** window. Note that when you use the **Zoom to Fit** command, the history of all previous zoom actions is lost, therefore this command will have no effect immediately after a **Zoom to Fit** command.

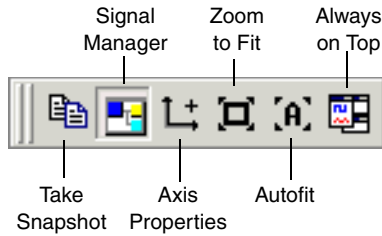
Aspect Lock

When checked, it causes the aspect ratio to be maintained when you zoom in or out.


8.5 Toolbar

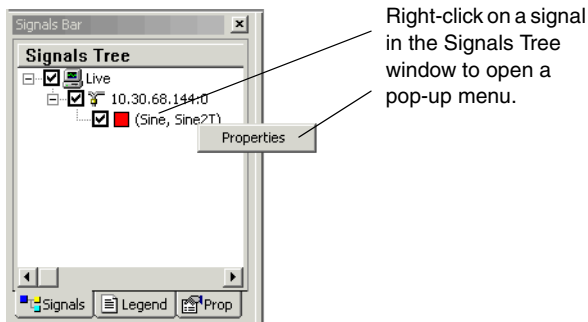
In a default **Plot XY** window, the first two dockable toolbars are shown with the menu items as described in Sections [3.4.1 Main Toolbar](#), p.48 and [3.4.2 Plot Toolbar](#), p.48. But the right-most dockable toolbar is specific to the **Plot XY** window. All the toolbars are dockable, which means you can move them to other locations, on or off the window, simply by dragging them. (The menu bar is also dockable.) Each of the toolbars can be independently displayed or hidden using the **View** menu item.

The **Plot XY** window's toolbar, shown below, is a subset of the one shown and described in detail in [3.4.3 Plot Window Toolbar](#), p.50.



8.6 Signals Bar

The **Signals Bar**, shown in the figure below, is a sub-window in the StethoScope GUI that allows you to view and configure information that affects what is plotted in the graph area. To open a **Signals Bar** if one is not already displayed, use the **View > Signals Bar** menu command (or the  button).



The **Signals Bar** contains the following tab views:

- **Signals**
- **Legend**
- **Properties**

8.6.1 Signals Tab View



The **Signals Tree** in the **Signals** tab view (see the figure above) displays all the signals available to be plotted. They are shown in an expandable tree structure containing signals that have been installed by the **Signal Manager** (see [4. Using the Signal Manager](#)). Traces on the graph are color-coded to the signals selected in the **Signals Tree**.

The **Signals Tree** is displayed by default when you open a **Plot XY** window, but it may be re-displayed at any time by clicking the **Signals** tab at the bottom of the sub-window.

The **Signals Tree** allows you to easily locate any signal that has been installed and add it to the graph of signal traces. Each node of the tree (and hence each signal) has a check box. Clicking an individual signal's check box adds that signal to the graph, or, conversely, clearing a check box removes that signal from the graph. If you click a node check box, all the signals belonging to that node (including their nodes below it) are checked at once and their signals are added to the graph. Conversely, clearing a node check box removes all signals below that node from the graph with the single click.

If a node check box is checked, but has a grey fill, it indicates that some, but not all, of the signals belonging to that node are checked and displayed on the graph. You may have to scroll down to see which ones are checked.

The **Signals Tree** is expanded or collapsed using the icon to the left of each node check box as follows:

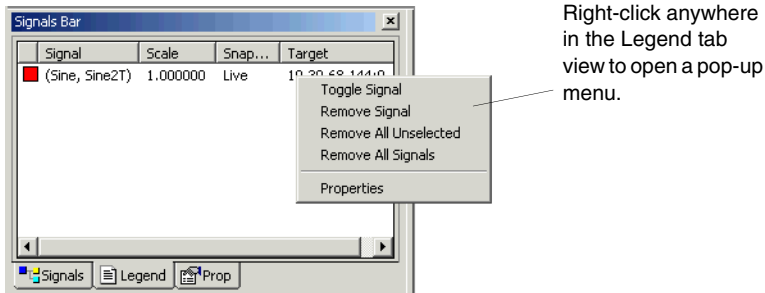
-  = **collapsed**; click to expand down to the next node(s).
-  = **expanded**; click to collapse up to the next node.

The **Units** column in the **Signals** tab view shows the physical measurement units of each component of the **XY Signal** pair.

When you right-click a signal in the **Signals** tab view, a pop-up menu opens, currently containing a single menu item, as shown in the figure above. The **Properties** menu item opens the **Signal Properties** dialog box where you can configure parameters that affect the appearance and behavior of the specific signal your cursor is on. The **Signal Properties** dialog box is described in detail in [8.7 Signal Properties Dialog Box](#), p.123.

8.6.2 Legend Tab View

The **Legend** tab view shows you what color is assigned to each signal on the plot. It can also be used to select which signals to plot. The figure below shows a **Legend** in the left side of the **Plot XY** window.



The columns in the **Legend** tab view display the current parameter settings for each signal, including:

Scale

The scale factor applied to each component signal's values to allow them to be easily viewed on the same graph with other **XY Signal** pairs.

Snapshot

Whether the data is live, or from a snapshot.

Target

The name and scope index of the target to which you are connected.

When you right-click a signal in the **Legend** tab view, a pop-up menu opens with the following options to manipulate the signal:

Toggle Signal

Turns the signal trace **on** or **off**. This is the equivalent of going to the **Signals Tree** and alternately checking and unchecking the signal's check box. It leaves the signal's name in the **Legend** list when you toggle it **off**.

Remove Signal

Removes the signal from the graph. This is equivalent to toggling the signal **off**, except it removes the signal's name from the **Legend**.

Remove All Unselected

Removes all signals that have been toggled **off**.

Remove All Signals

Removes all signals regardless of their toggled status.

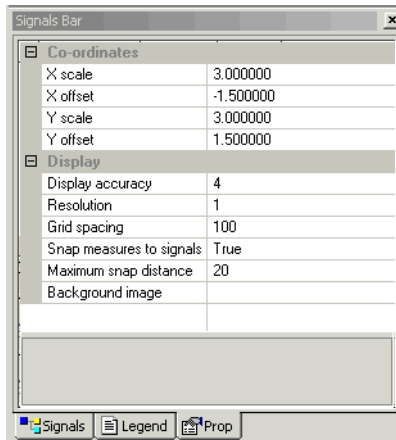
Properties

Opens the **Signal Properties** dialog box (see [8.7 Signal Properties Dialog Box](#), p.123).

You can change the colors used for plot lines by using the **File > Preferences** menu command (see [Colors View](#), p.38), or by using the **Signal Properties** dialog box (see [8.7 Signal Properties Dialog Box](#), p.123).

8.6.3 Properties Tab View

You can view and change the properties for only the **Plot XY** window you are in by clicking the **Properties** tab in the **Signals Bar**. This tab view, shown below, accesses properties for the graphic display window environment (for the signals themselves, use the **Signal Properties** dialog box, described in [8.7 Signal Properties Dialog Box](#), p.123).



The data-display window environment properties shown in this tab view are:

Coordinates

X scale

Controls the width of the plot grid in units. For example, when first started, X scale defaults to 3 and the plot displays the X-axis from 1.5 to -1.5.

X offset

Controls the unit value for the left-most coordinate on the plot.

Y scale

Controls the height of the plot grid in units. For example, when first started, Y Scale defaults to 3 and the plot displays the Y-axis from 1.5 to -1.5.

Y offset

Controls the unit value for the top coordinate on the plot. When first started, it is 1.5, and you will see that the Y value of 1.5 is the top value. This value will change automatically if you use the **Zoom to Fit** command.

Display

Display accuracy

Controls the number of significant digits displayed in the grid line markers. Set this property to the number of places you want displayed to the right of the decimal point. Enter a value between 0 (for integer numbers) and 6 (default=4).

Resolution

Permits plotting of only a subset of the collected data points. This is useful for increasing rendering speed. A divisor value of n causes only every n th point to be plotted. Thus, if 1000 points are being collected and the **Resolution** is 5, then 200 points will be displayed. Of course, you may not want to reduce resolution if your data can contain “glitches” or high-frequency phenomena (default=1).

Grid spacing

Specifies the minimum distance (in pixels) allowed between the grid lines. Increasing this number decreases the number of grid lines, decreasing this number increases the number of grid lines. Enter a value between 5 and 500 (default=100).

Snap measures to signals

Controls whether or not measures are “snapped” to the signal lines on the grid. Measures are described in [Taking and Removing On-grid Measurements](#), p.56 (default=**True**).

Maximum snap distance

Controls how far away (in pixels) you can start a measure and still have it snap to the plot line. Measures are described in [Taking and Removing On-grid Measurements](#), p.56 (default=20).

Background image

Allows you to specify special images to be displayed in the graphing area, with the graph itself superimposed on top (default=none).

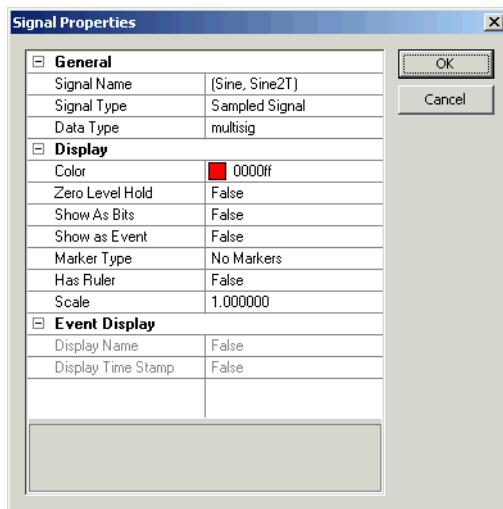
To modify any value, just type directly into the text field. Any changes you make in this window have no effect on any other open **Plot XY** windows.

Any changes you make in this window have no effect on any other open **Plot** windows.

To change the defaults used when new **Plot XY** windows are created, use the **File > Preferences** menu command (), described in [3.2.12 Preferences](#), p.36.

8.7 Signal Properties Dialog Box

Characteristics of the line used to plot each signal in this window can be configured using the **Signal Properties** dialog box, shown in the figure below.



This dialog box can be opened from either of two places:

- Right-click a signal in the **Signals** tab view (see [8.6 Signals Bar](#), p.118)

- Right-click a signal trace in the data-display area

In the pop-up menu that opens, select **Signal Properties**. The **Signal Properties** dialog box opens with the following parameters:

General

Signal Name and Type

These first two lines are the signal's name (as shown in the signals tree) and its type.

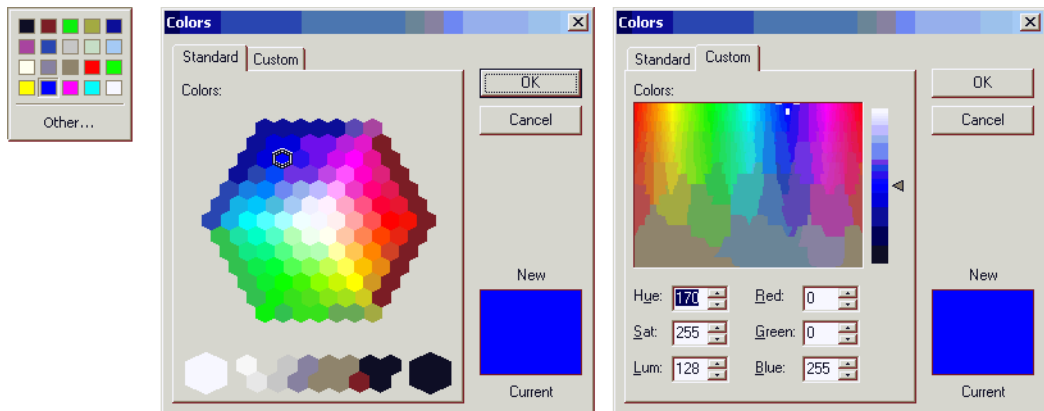
Data Type

This is the C++ data type of this signal (program variable) See [Table 14-1](#) for a list of allowable types.

Display

Color

Clicking on the current value opens a small color palette, below, with a basic color selection from which to choose. If you would like a color that is not on this color palette, you can click **Other** to open the **Colors** dialog box where you can select a new color from the larger selection in the **Standard** tab view, or you can create an entirely new color using the **Custom** tab view.

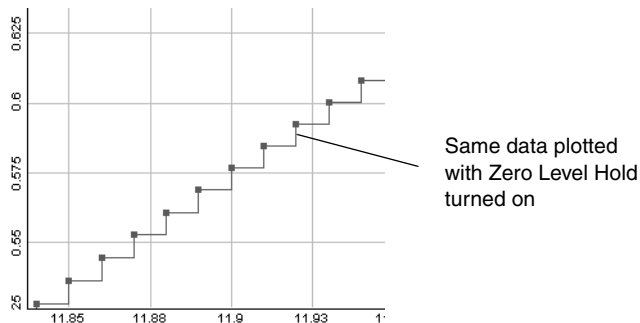
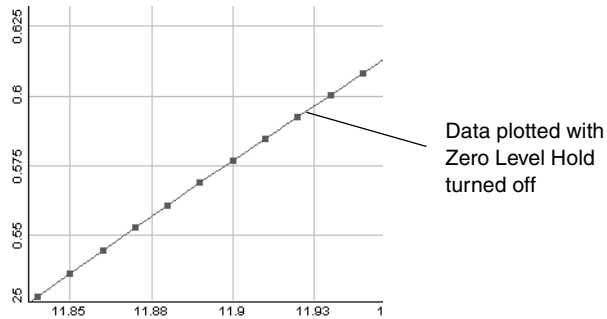


NOTE: An alternative method for changing colors of plotted lines on the graph is given in the discussion of colors preferences in [Colors View](#), p.38.

Zero Level Hold

Select **True** from the drop-down menu to “hold” the value of the signal until the next sample arrives (default=**False**).

To demonstrate this feature, consider the sampling of a sine wave at a rate of 100Hz, as shown in the figure below. In the upper view in this figure, with **Zero Level Hold** turned off (= **False**), the data points are connected as usual with straight lines, even though there is no information about the actual values between data points. In the view below, the same data, but with the **Zero Level Hold** feature turned on (= **True**), shows how the line plotted between the same data points is now a straight horizontal line of the same value as the previous data point, until the next data point comes in, at which time the plot line goes vertical up to that value.



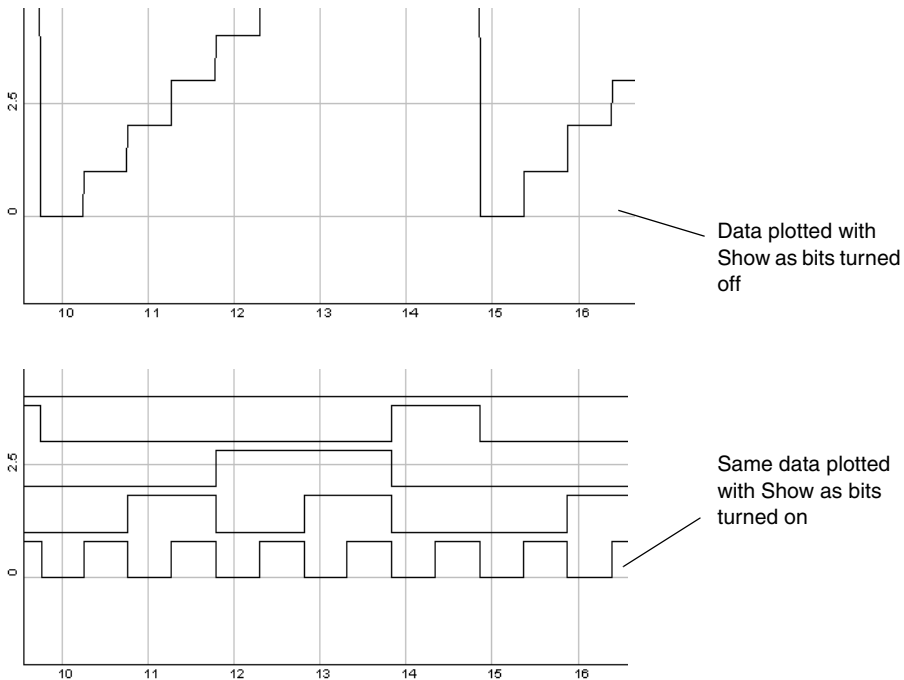
Show As Bits

Select **True** from the drop-down menu to cause a signal (program variable) being plotted to be represented as a sequence of bits, each bit with its off and on (0 and 1) values indicated. The bits are rendered individually by

discrete horizontal lines distributed up the Y axis from bit position 0 starting at the bottom Default=**False**).

To show this feature, a **Derived Signal**, created using a single 8-bit integer, is incremented through values from 0 to 15 and back to 0 again at the rate of two increments per second, and then repeated continuously until stopped. In the resulting plot, shown in the figure below, the top graph shows the variable's value with **Show as bits** turned off (= **False**), and the bottom graph shows the same data with **Show as bits** turned on (= **True**). You can easily observe the pattern of the individual bits (horizontal lines) making up the 16-bit integer, showing their 0 and 1 positions with respect to time.

This feature works equally well with all signal types, but the resulting bit patterns may be more difficult to decipher depending on the complexity of the signal.






Show as Event

Select **True** from the drop-down menu to cause samples to be marked with vertical lines instead of connecting the individual samples with lines (default=**False**). Either or both of the selections made in the **Event Properties** section below are displayed along side the vertical marker. For more information on displaying events, see [8.9 Setting Preferences for a New Plot XY Window](#), p.128.

Marker Type

This drop-down list allows you to select a different symbol (or marker) to be plotted for each sample of this signal. The choices are:

- **No Markers** — No markers are selected.
- **Square** — A square () plots each sample.
- **Plus** — A plus sign () plots each sample.
- **Diamond** — A diamond () plots each sample.
(Default=**No Markers**.)

Has Ruler

This true/false option displays a separate ruler with colored numbers matching the signal's color on the left (Y) axis. If you have, say, 3 signals with rulers on, there will be three separate color-coded rulers on the left plot boundary (default=**False**).

Scale

Each plotted value is scaled (multiplied) by the value you enter (default=**1.000000**).

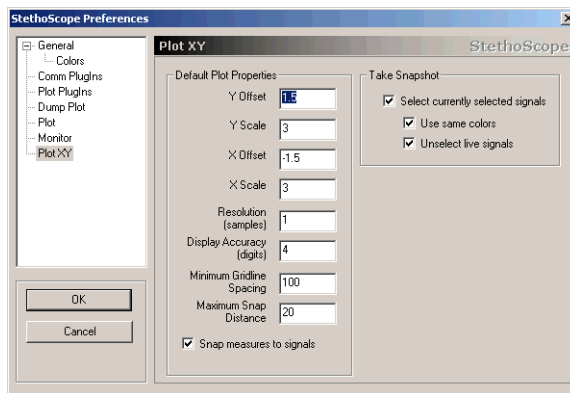
8.8 Pop-up Menus


Pop-up menus that open in response to right-clicking in the **Plot XY** window offer options that are unique to specific signals. These menus are described in detail in [3.6 Pop-up Menus](#), p.52. They are accessed from different places in the **Plot XY** window, including:

- **On-grid** (anywhere in the plotting area)
- **Signals** tab view
- **Legend** tab view

8.9 Setting Preferences for a New Plot XY Window

The **Preferences** dialog box allows you to set default parameters for the StethoScope GUI when it starts and for new data-display windows when they are created (whereas the “properties” described in this chapter apply only to currently open windows). The **Plot XY Preferences** view, shown in the figure below, allows you to change the default values used when **Plot XY** windows are created. It also allows you to control some aspects of what happens when a snapshot is taken.



To modify these preferences, open the **Preferences** dialog box with the **File > Preferences** menu command (), then click **Plot XY** in the left panel to set:

Default Plot Properties

Y Offset

Controls the unit value for the top coordinate on the plot. When first started, it is **1.5**, and you will see that the Y value of **1.5** is the top value. This value will change automatically if you use the **Zoom to Fit** command.

Y Scale

Controls the height of the plot grid in units. For example, when first started, Y Range defaults to **3** and the plot displays the Y-axis from **1.5** to **-1.5**.

X Offset

Controls the unit value for the left-most coordinate on X axis of the plot. When first started, it defaults to **0**. This value will change automatically if you use the **Zoom to Fit** command.

X Scale

Controls the width of the plot grid in units. For example, when first started, X Range defaults to 20, and the plot displays the X-axis from 0 to 20.

Resolution (samples)

Permits plotting of only a subset of the collected data points. This is useful for increasing rendering speed. A divisor value of n causes only every n th point to be plotted. Thus, if 1000 points are being collected and the **Resolution** is 5, then 200 points will be displayed. Of course, you may not want to reduce resolution if your data can contain “glitches” or high-frequency phenomena (default=1).

Display Accuracy (digits)

Controls the accuracy of the grid line markers. Set this property to the number of places to the right of the decimal point to use in the grid markers. Enter a value between 0 (for whole numbers) and 6 (default=4).

Minimum Grid Line Spacing

Specifies the minimum distance (in pixels) allowed between the grid lines. Increasing this number decreases the number of grid lines, decreasing this number increases the number of grid lines. Enter a value between 5 and 500 (default=100).

Minimum Snap Distance

Specifies the minimum distance (in pixels) from a grid line that will cause a plot point to be snapped to the grid line (default=20).

Take Snapshot

Unlike all other preferences, which only impact data-display windows you may create later, these preferences take effect immediately. See [11. Working with Snapshots](#) for more information on snapshots.

The following options can be set by setting their checkboxes:

Select currently selected signals

If selected, the same signals currently selected in the live buffer will also be selected in the snapshot.

Use same colors

If selected, the same colors will be assigned to the signals in the snapshot. If you want the snapshot to use different colors for each signal than are used for live data, clear this check box.

Unselect live signals

If selected, the live signals will become unselected when the snapshot is taken.

9

The Dump Plot Window

- 9.1 Introduction 131
- 9.2 Dump Plot Window Tour 132
- 9.3 Menu Bar 133
- 9.4 Toolbars 135
- 9.5 Signals Bar 136
- 9.6 Setting Preferences for a New Dump Plot Window 138

9.1 Introduction

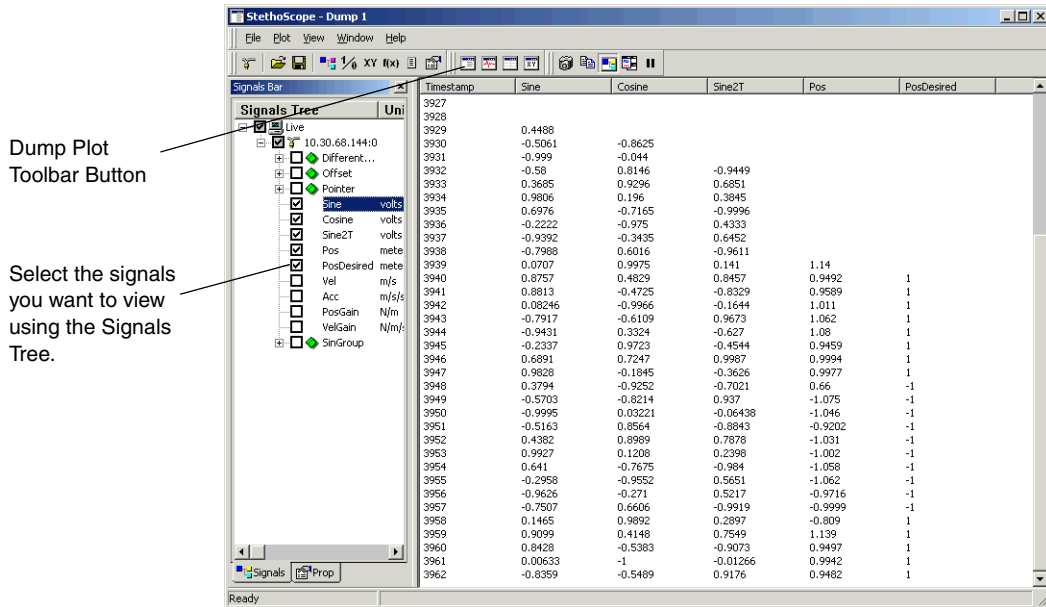
The **Dump Plot** window displays real-time data numerically in a tabular format. Like the **Plot** window, a **Dump Plot** window can display both “live” and snapshot data.

You can have multiple **Dump Plot** windows open at the same time, and each can display different signals or snapshots.

Before using a **Dump Plot** window, it may help to understand how and when data is collected from the target—see [5. Triggering](#).


9.2 Dump Plot Window Tour

To open a **Dump Plot** window, use the **File > Plots > Dump Plot** menu command (or the  button). A typical **Dump Plot** window is shown in the figure below.



The left-most column is always the **TimeStamp**, calculated by multiplying the sample period by each sample's position in the data buffer. **TimeStamp** is reset to zero for each collected data set. The other columns in this table are the values, at the time increments, for each signal selected in the **Signals Tree** to the left. This table is dynamically updated as data collection progresses.

9.2.1 Displaying Signal Values in a Dump Plot Window

1. If you do not already have a **Signals Bar** in your **Dump Plot** window, use the **View > Signals Bar** menu command (or the  button). Make sure the **Signals** tab is selected in the **Signals Bar**, so that a **Signals Tree** is displayed.
2. Use the **Signals Tree** to select which signals you want to display in the table. For details on using **Signals Trees**, see [4. Using the Signal Manager](#).

Each selected signal appears as a column in the table.

Most of the remaining functionality in the **Dump Plot** window is provided through the toolbars and menus. For common data-display window feature descriptions, refer to the following sections for more information:

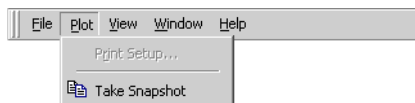
- **Menu Bar** ([3.2 File Menu](#), p.26)
- **Toolbars** ([3.4 Toolbars](#), p.48)

9.3 Menu Bar

Some of the **Menu Bar** items are discussed in detail in [3.2 File Menu](#), p.26, where it is mentioned that the **File**, **Window**, and **Help** menu items are consistently the same across all data display windows. The **Plot** and **View** menu items, however, contain some commands that are unique to the **Dump Plot** window. Even though partially redundant, these menu items are described in detail in the following sections.

9.3.1 Plot Menu Commands

The **Plot** menu item contains commands for working with the plots in the **Dump Plot** window.



The **Plot** menu commands are:

Print Setup

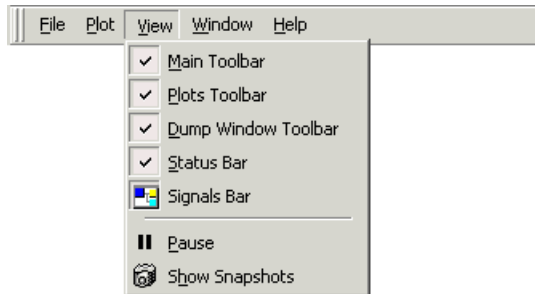
Allows you to select printer parameters and characteristics before printing (not enabled at this time).

Take Snapshot

Saves a copy of all the active signals (not just the selected signals), for all connected targets. For more information about snapshots, see [11.2 Taking Snapshots](#), p.152.

9.3.2 View Menu Commands

The **View** menu item, shown below, contains commands for working with the plots in the **Plot** window.



The **View** menu commands are:

Main Toolbar

Controls whether the toolbar representing a selection of items from the **File** menu is displayed on the **Dump Plot**'s toolbar. For details about the StethoScope **Main** toolbar, see [9.4 Toolbars](#), p.135.

Plots Toolbar

Controls whether the toolbar representing a selection of items from the **File > Plot** menu command is displayed on StethoScope's toolbar. For details about the **Plots** toolbar, see [3.4 Toolbars](#), p.48.

Dump Window Toolbar

Controls whether the toolbar used within the **Dump Plot** data-display window is visible. The buttons represent items from the **View** menu for the **Dump Plot** data-display window. For information about the **Dump Plot** Window toolbar, see [9.4 Toolbars](#), p.135.

Status Bar

Controls whether the status line along the bottom of the window is visible. For information about the **Status Bar**, see [3.5 Status Bar](#), p.52.

Signals Bar

Controls whether a **Signals Bar** panel appears in the window. The **Signals Bar** (see the figure in [9.2 Dump Plot Window Tour](#), p.132) includes tabs for **Signals** and **Properties**. For more information about the **Signals Bar**, see [9.5 Signals Bar](#), p.136.

Pause

Stops updates to the table in the **Dump Plot** window. It does not stop data collection, but merely stops new data from appearing in the **Dump Plot** table. To resume normal display, unselect the button.

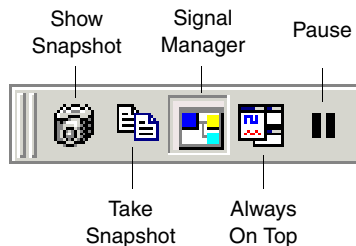
Show Snapshots

Controls what is displayed in the **Signals Tree**. When selected, only snapshots appear in the **Dump Plot** window's **Signals Tree**. When unselected, only the live data buffer appears. For more information about snapshots, see [11.2 Taking Snapshots](#), p.152.


9.4 Toolbars

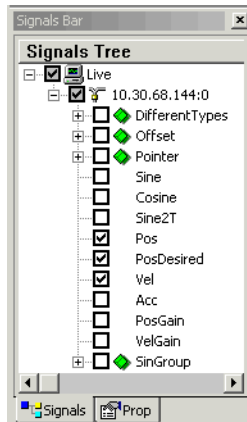
In a default **Dump Plot** window, the first two dockable toolbars are identical to the toolbars shown and with the menu items as described in Sections [3.4.1 Main Toolbar](#), p.48 and [3.4.2 Plot Toolbar](#), p.48. But the right-most dockable toolbar is specific to the **Dump Plot** window. All the toolbars are dockable, which means you can move them to other locations, on or off the window, simply by dragging them. (The menu bar is also dockable.) Each of the toolbars can be independently displayed or hidden using the **View** menu item.

The **Dump Plot** window's toolbar, shown below, is a subset of the one shown and described in detail in [3.4.3 Plot Window Toolbar](#), p.50.



9.5 Signals Bar

The **Signals Bar**, shown in the figure below, is a sub-window in the StethoScope GUI that allows you to view and configure information that affects what signals appear in the data-display table. If a **Signals Bar** panel is not displayed, you can open one using the **View > Signals Bar** menu command (or the  button).



The **Signals Bar** contains these two views:

- **Signals**
- **Properties**

9.5.1 Signals Tab View

The **Signals Tree** in the **Signals** tab view (see the figure above) displays all the signals available to be plotted. They are shown in an expandable tree structure containing signals that have been installed by the **Signal Manager** (see [4. Using the Signal Manager](#)). Signals in the table are color-coded to the signals selected in the **Signals Tree**.

The **Signals Tree** is displayed by default when you open a **Dump Plot** window, but it may be re-displayed at any time by clicking the **Signals** tab at the bottom of the sub-window.

The **Signals Tree** allows you to easily locate any signal that has been installed and add it to the graph of signal traces. Each node of the tree (as well as each signal)

has a check box. Clicking an individual signal's check box adds that signal to the graph, or, conversely, clearing a check box removes that signal from the graph. If you click a node's check box, all the signals belonging to that node (including their nodes below it) are checked at once and those signals are added to the graph. Conversely, clearing a node check box removes all signals below that node from the graph with the single click.

If a node check box is checked, but has a grey fill, it indicates that some, but not all, of the signals belonging to that node are checked and displayed on the graph. You may have to scroll down to see which ones are checked.

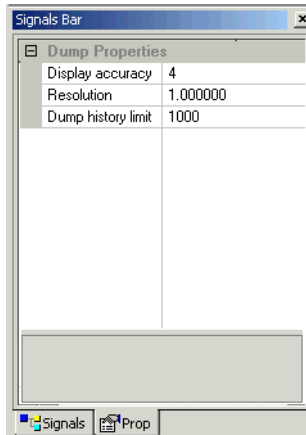
The signals tree is expanded or collapsed using the icon to the left of each node check box as follows:

- = **collapsed**; click to expand down to the next node(s).
- = **expanded**; click to collapse up to the next node.

The **Units** column in the **Signals** tab view shows the physical measurement units of each signal.

9.5.2 Properties Tab View

You can view and change the properties for the currently open **Dump Plot** window by clicking the **Properties** tab, shown in the figure below, in the **Signals Bar**.



The tabular display window environment properties shown in this tab view are:

Display accuracy

Controls the number of significant digits displayed in the table. Set this property to the number of places you want displayed to the right of the decimal point. Enter a value between **0** (for integer numbers) and **6** (default=4).

Resolution

Controls how often (in seconds) to refresh values in the **Dump Plot** table (default = **1.000000**).

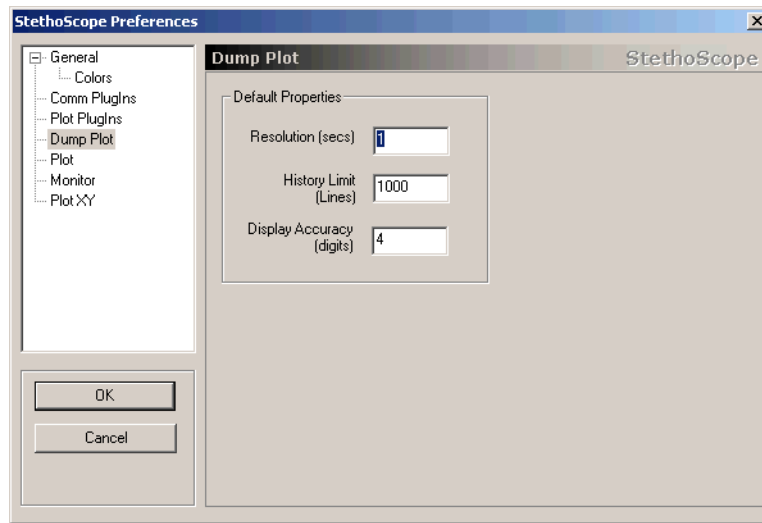
Dump history limit


Controls how many lines of historical data to maintain in the **Dump Plot** table, **0** = display all (default = **1000**).

To modify any value, just type directly into the text field. Any changes you make in this window have no effect on any other open **Dump Plot** windows.

9.6 Setting Preferences for a New Dump Plot Window

The **Preferences** dialog box allows you to set default parameters for new **Dump Plot** windows when they are created (whereas the “properties” described in [9.5.2 Properties Tab View](#), p. 137 apply only to currently open windows). The **Dump Plot Preferences** panel, shown in the figure below, allows you to change these default values. These preferences have no effect on **Mini Dump Plot** windows (in **Plot** windows), or already open **Dump Plot** windows.



To modify these preferences, open the **Preferences** dialog box with the **File > Preferences** menu command (or the  button), then click **Dump Plot** in the left panel to set:

Default Properties

Resolution (secs)

Controls how often, in seconds, to refresh the values in the **Dump Plot** window (default=1).

History Limit (Lines)

Controls how many lines of historical data to maintain in the v window (default=1000).

Display Accuracy (digits)

Controls the accuracy of values displayed in the table. Set this property to the number of places to the right of the decimal point to use. Enter a value between 0 (for whole numbers) and 6 (default=4).

10


The Monitor Window

- 10.1 Introduction 141
- 10.2 Monitor Window Tour 142
- 10.3 Menu Bar 143
- 10.4 Toolbar 145
- 10.5 Signals Bar 146
- 10.6 Writing Data to the Target 149
- 10.7 Setting Preferences for a New Monitor Window 149

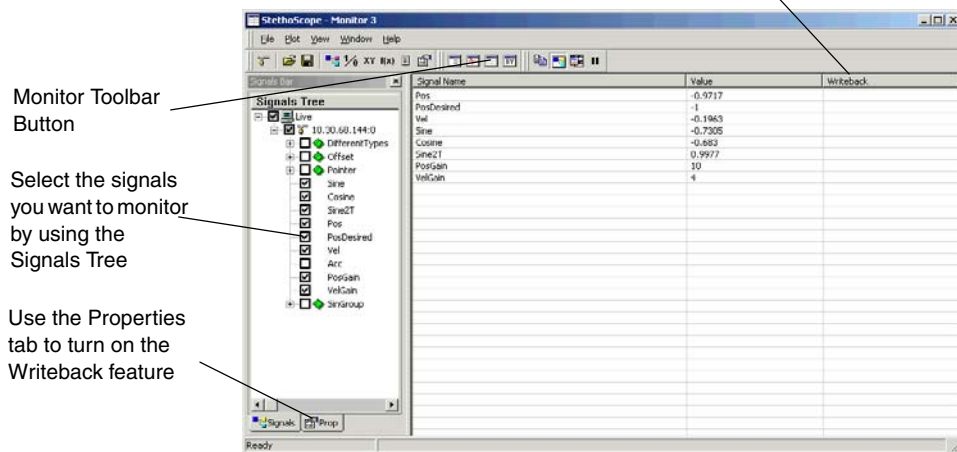
10.1 Introduction

The **Monitor** window differs from the **Dump Plot** window in that the **Monitor** window only shows you the most recent value of each signal, whereas the **Dump Plot** window displays signal values as they change over time. The **Monitor** window can also be used to modify a signal's value on the target.

10.2 Monitor Window Tour

To open a **Monitor** window, use the **File > Plots > Monitor** menu command (or the  button). The figure below shows a typical **Monitor** window session (from the demonstration program).


Click in the Writeback column to enter a new value which will be written to the target



You can select the signals you want to monitor by using the **Signals** tab of the **Signals Bar**. Each selected signal appears as a row in the **Monitor** window's table with the following column descriptions.

- The first column displays the signal names.
- The second column displays the most recent value for each signal.
- A third column is used to write new values back to the target. This **Writeback** column only appears when the **Writeback** property is set to **True** (see [10.6 Writing Data to the Target](#), p.149).

10.2.1 Displaying Signal Values in a Monitor Window

1. If you do not already have a **Signals Bar** in your **Monitor** window, use the **View > Signals Bar** menu command (or the  button). Make sure the **Signals** tab is selected in the **Signals Bar**, so that a **Signals Tree** is displayed.
2. Use the **Signals Tree** to select which signals you want to display in the table. For details on using **Signals Trees**, see [4. Using the Signal Manager](#).

Most of the remaining functionality in the **Monitor** window is provided through the toolbars and menus. For common data-display window feature descriptions, refer to the following sections for more information.

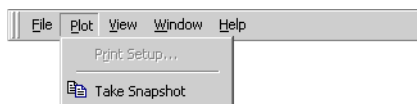
- [3.2 File Menu](#), p.26
- [3.4 Toolbars](#), p.48

10.3 Menu Bar

Some of the **Menu Bar** items are discussed in detail in [3.2 File Menu](#), p.26, where it is mentioned that the **File**, **Window**, and **Help** menu items are consistently the same across all data display windows. For the **Monitor** window, however, the **View** menu item contains some commands that are unique to the **Monitor** window. Even though partially redundant, these menu items are described in detail in the following sections.

10.3.1 Plot Menu Commands

The **Plot** menu item, shown below, contains commands for working with data in the **Monitor** window.



The **Plot** menu commands are:

Print Setup

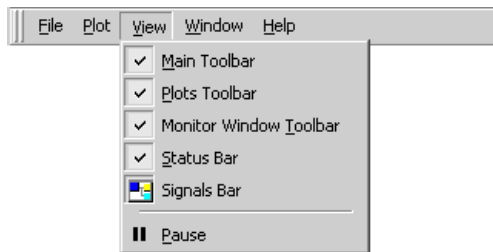
Allows you to select printer parameters and characteristics before printing.

Take Snapshot

Saves a copy of all the active signals (not just the selected signals), for all connected targets. For more information, see [11.2 Taking Snapshots](#), p.152.

10.3.2 View Menu Commands

The **View** menu item, shown below, contains commands primarily for choosing which elements of the **Monitor** window to display.



Main Toolbar

Controls whether the toolbar representing a selection of items from the **File** menu is displayed on StethoScope's toolbar. For more information, see [10.4 Toolbar](#), p.145.

Plots Toolbar

Controls whether the toolbar representing a selection of items from the **File > Plot** menu command is displayed on StethoScope's toolbar. For more information, see [3.4 Toolbars](#), p.48.

Monitor Window Toolbar

Controls whether the toolbar used within a specific data-display window is visible. The buttons represent items from the **Plot** and **View** menus for the specified data-display window. For more information, see [10.4 Toolbar](#), p.145.

Status Bar

Controls whether the status line along the bottom of the window is visible. For more information, see [3.5 Status Bar](#), p.52.

Signals Bar

Controls whether a **Signals Bar** panel appears in the window. The **Signals Bar** (see [10.5.1 Signals Tab View](#), p. 146) includes tabs for **Signals** and **Properties**. For more information about the **Signals Bar**, see [10.5 Signals Bar](#), p. 146.

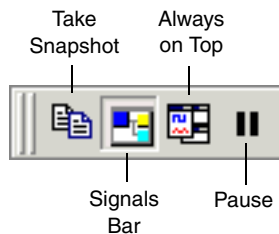
Pause

Stops updates to the table in the **Monitor** window. It does not stop data collection, but merely stops new data from appearing in the table. To resume normal display, unselect the button.


10.4 Toolbar

In a default **Monitor** window, the first two dockable toolbars are identical to the toolbars shown and with the menu items as described in Sections [3.4.1 Main Toolbar](#), p. 48 and [3.4.2 Plot Toolbar](#), p. 48. But the right-most dockable toolbar is specific to the **Monitor** window. All the toolbars are dockable. The menu bar is also dockable.) Each of the toolbars can be independently displayed or hidden using the **View** menu item.

The **Monitor** window's toolbar, shown in the figure below, is a subset of the one shown in [3.4.3 Plot Window Toolbar](#), p. 50, where the icons are described in detail.



10.5 Signals Bar

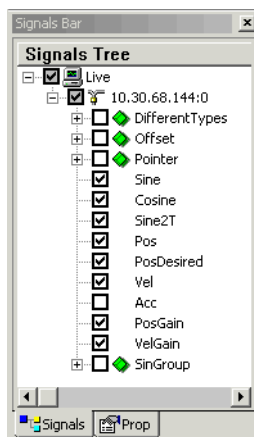
The **Signals Bar**, shown in the figure below, is a sub-window in the StethoScope GUI that allows you to view and configure information that affects what signals appear in the data-display table. If a **Signals Bar** panel is not displayed, you can open one using the **View, Signals Bar** menu command ().

The **Signals Bar** offers the following tab views:

- **Signals**
- **Properties**

10.5.1 Signals Tab View

The **Signals Tree** in the **Signals** tab view, shown in the figure below, displays all the signals available to be plotted. They are shown in an expandable tree structure containing signals that have been installed by the **Signal Manager** (see [4. Using the Signal Manager](#)). Signals in the table are color-coded to the signals selected in the **Signals Tree**.





The **Signals Tree** is displayed by default when you open a **Monitor** window, but it may be re-displayed at any time by clicking the **Signals** tab at the bottom of the sub-window.

The **Signals Tree** allows you to easily locate any signal that has been installed and add it to the graph of signal traces. Each node of the tree (as well as each signal)

has a check box. Clicking an individual signal's check box adds that signal to the graph, or, conversely, clearing a check box removes that signal from the graph. If you click a node's check box, all the signals belonging to that node (including their nodes below it) are checked at once and those signals are added to the graph. Conversely, clearing a node check box removes all signals below that node from the graph with the single click.

If a node check box is checked, but has a grey fill, it indicates that some, but not all, of the signals belonging to that node are checked and displayed on the graph. You may have to scroll down to see which ones are checked.

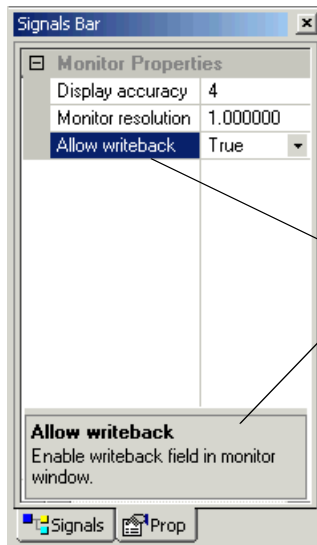
The **Signals Tree** is expanded or collapsed using the icon to the left of each node check box as follows:

-  = **collapsed**; click to expand down to the next node(s).
-  = **expanded**; click to collapse up to the next node.

The **Units** column in the **Signals** tab view shows the physical measurement units of each signal.

10.5.2 Properties Tab View

You can view and change the properties for the currently open **Monitor** window by clicking the **Prop** tab, as shown below, in the **Signals Bar**.



The Properties tab view lets you change parameters affecting the data properties of a currently open Monitor window, including allowing writeback (see [10.6 Writing Data to the Target](#), p. 149)

The tabular display window environment properties shown in this tab view are:

Display accuracy

Controls the number of significant digits displayed in the table. Set this property to the number of places you want displayed to the right of the decimal point. Enter a value between **0** (for integer numbers) and **6** (default=4).

Monitor resolution

Controls how often, in seconds, to refresh the values in the **Monitor** window (default = **1.000000**).

Allow writeback

Select **True** to create a **Writeback** column for writing modified signal values back out to the target (default=**False**).

To modify any value, just type directly into the text field. Any changes you make in this window have no effect on any other open **Monitor** windows.

To change the defaults used when new **Monitor** windows are created, see [10.7 Setting Preferences for a New Monitor Window](#), p. 149.

10.6 Writing Data to the Target

StethoScope provides you with **Writeback**, a very powerful feature with which you can change the values of variables on the target as your program runs.

A message to this effect will be displayed when a value is written. You can disable this message by using the check box in the **Warning** dialog.



WARNING: Modifying values in a running program is powerful, but can be dangerous. Be very careful. You should realize that the wrong value may be written by accident, either by you or by the StethoScope program. This could be caused by a mis-typed entry, an error in determining the variable's correct address or type, or a bug in the StethoScope program itself. **THIS FACILITY SHOULD NOT BE USED TO CONTROL SAFETY-CRITICAL SYSTEMS.** Use this feature at your own risk!

10.6.1 Writing Data to the Target for a Selected Signal

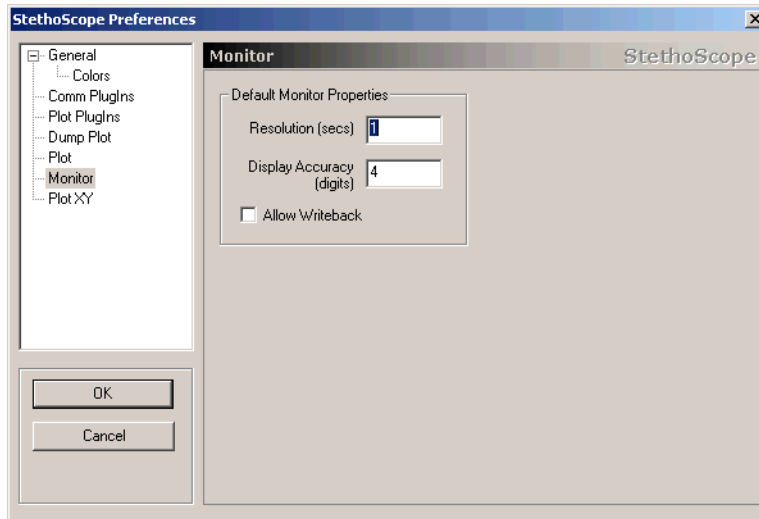
1. Make sure **Writeback** is set to **Yes** in the **Properties** panel. This causes the **Writeback** column to appear in the table.
2. Enter a value in the **Writeback** column for the desired signal at any time. It can be a number or the name of another signal. In the latter case, the last available value of the named signal is used.
3. With the mouse pointer inside the **Writeback** field, press **Enter** to write the value to the target. (A warning prompt will be displayed first.)


Writing a value with the **Monitor** window changes the variable's value in the target's memory. The change occurs asynchronously (when the data arrives).

10.7 Setting Preferences for a New Monitor Window

The **Preferences** dialog box allows you to set default parameters for new **Monitor** windows when they are created (whereas the "properties" described in this chapter apply only to currently open windows). The **Monitor Preferences** panel, as seen in the figure below, allows you to change these default values. These

preferences have no effect on **Mini Monitor** windows within **Plot** windows, or already open **Monitor** windows. (To change these values on **Monitor** windows you have already created, see [10.5.2 Properties Tab View](#), p.147.)



To modify these preferences, open the **Preferences** dialog box with the **File > Preferences** menu command (), then click **Monitor** in the left panel to set:

Default Monitor Properties

Resolution (secs)

Controls how often (in seconds) to refresh the values in the **Monitor** window (default=1).

Display Accuracy (digits)

Controls the accuracy of values displayed in the table. Set this property to the number of places to the right of the decimal point to use. Enter a value between 0 (for whole numbers) and 6 (default=4).

Allow Writeback

Check box controls whether or not you can use the **Monitor** window to write modified signal values back out to the target. Opens a **Writeback** column in the table if checked (default=**unchecked**).

11

Working with Snapshots


- 11.1 Introduction 151
- 11.2 Taking Snapshots 152
- 11.3 Saving Snapshots 153
- 11.4 Loading Snapshots 157
- 11.5 Exporting Snapshot Data to MATLAB and MATRIXX 158
- 11.6 Deleting Snapshots 161

11.1 Introduction

A snapshot saves the data for all active signals, for all connected targets. You can display snapshots in the **Plot** and **Plot XY** windows, along with live data and other snapshots. This makes it easy to compare test runs. You can save snapshots in four different formats, for use in other applications. You can take snapshots, save snapshots to disk, and load snapshots from disk.

11.2 Taking Snapshots

There are multiple ways to take a snapshot:

- Use the **Take Snapshot** () button on the toolbar.
- Use the **Plot > Take Snapshot** menu command.
- Use the **File > Save Snapshot** menu command, with Snapshot type set to **Live** and the **Write** field set to **Immediately, in the middle of this cycle**.
- Use the **Triggering** dialog box to configure and arm a trigger and set automatic **Snapshot**.

11.2.1 What Happens When You Take a Snapshot?

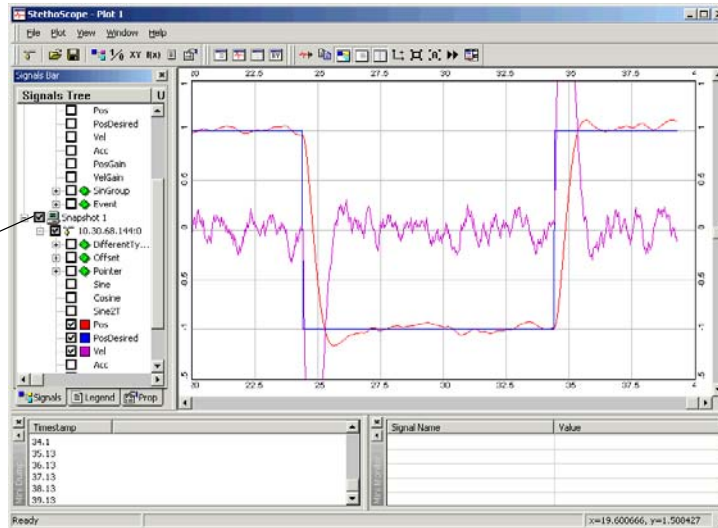
When you connect to a target, StethoScope collects all the active signals and stores them in a buffer known as the **Live** buffer. A snapshot creates a copy of that live buffer. And just as the **Live** buffer includes all active signals (even though only selected signals appear on the plot), the snapshot buffer also includes all active signals.

When you take a snapshot, all of the live data received is copied into a temporary snapshot buffer. No event data other than the event identifiers collected is stored.

What happens next depends on the **Take Snapshot** settings in the **Plot Preferences** panel of the **Preferences** dialog box (see [7.10 Setting Preferences for a New Plot Window](#), p.106), but the following describes the default behavior.

In the **Signals Tree**, the **Live** buffer becomes deselected and the snapshot is added at the bottom of the **Signals Tree** and selected (see the figure below). The plot grid area will switch from displaying the **Live** buffer to displaying the newly saved snapshot. While the **Live** buffer is no longer “selected” in the **Signals Tree**, real-time data is still being collected in the **Live** buffer.

The snapshot is inserted at the bottom of the Signals Tree, and when it is selected, the Live buffer is un-selected



When the snapshot is first displayed on the grid, it uses the same set of selected signals that you were viewing in real-time. However, all active signals are stored in the snapshot, so you can select them in the same manner used for selecting real-time signals.


You can display the live buffer, plus multiple snapshots, all at the same time. This makes it easy to compare previous runs with real-time runs.

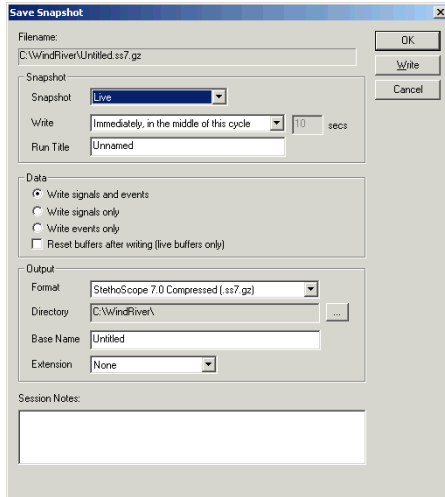
Snapshots are automatically given temporary storage names such as **snapshot1**, **snapshot2**, and so forth. You can change the names when you save the snapshots to disk.

Snapshots are in temporary storage until you save them to disk, as described in [11.3 Saving Snapshots](#), p. 153.

11.3 Saving Snapshots

While the **Take Snapshot** command creates a copy of real-time data in a temporary buffer, the **Save Snapshot** command stores the data on your file system. The

File > Save Snapshot menu command (or the  button) opens the **Save Snapshot** dialog box, as shown in the figure below.



The **Save Snapshot** dialog box contains five panels:

Filename

This read-only text box shows you the filename that will be used when you click **Write**. As you make selections in the **Output** panel, you will see the resulting filename change.

Snapshot

The fields in this panel are used to select which snapshot (or live buffer) to save, when to save it, and what name (**Run Title**) to associate with it. See [11.3.1 Snapshot](#), p.155 for more information.

Data

This panel offers three choices (radio buttons) for saving signals and/or events, as well as a check box to cause the live buffer to be reset after writing.

Output

The fields in this panel specify the format, path, and name of the saved snapshot file. See [11.3.3 Output](#), p.156 for more information.

Session Notes

The notes you enter in this text box can help you keep track of a series of snapshots. Session notes are intended to describe the conditions over a series of collected runs or buffers. Useful session notes, for example, might be:

Session notes:

These runs employ the non-linear friction model.
Both force sensor filters are active, at 20 Hz.

Other useful notes include code fragments and information that clearly identifies the data. Any text on the screen may be pasted into the notes panel via standard “cut and paste” operations.

11.3.1 Snapshot

The **Snapshot** panel controls *when* snapshots are taken, with respect to the data collection cycle. The **Snapshot** panel contains the following fields:

Snapshot

Use this drop-down menu to select which snapshot to save. You can save a snapshot you have already taken, or choose to save the “**Live**” buffer. The list box includes the **Live** buffer, any snapshots already taken, and any snapshots you have loaded from disk.

Write

Use this drop-down menu to choose when data will be written after the **Write** button is clicked. There are two choices:

Immediately, in the middle of this cycle

This is the default setting. This setting causes the data to be written to the output file immediately after the **Write** button is clicked, regardless of the amount of data available. An error occurs if there is no data at all.

Every X seconds, X specified to the right

This setting provides a way to store data periodically. Data is written every X seconds, where X is specified in the field to the right. The write field becomes available for entering the number of seconds between snapshots. The **Write** button is labeled **Stop Writing** after StethoScope starts writing data to the files. Click the **Stop Writing** button to terminate storing data.

Run Title

Use this text box to label the snapshot. The run title is stored with the data. It may be used by data-analysis programs to identify the data set.

11.3.2 Data

By clicking one of the three radio buttons, you can select whether to save signals and events, signals only, or events only, when you take the snapshot. For live

buffers only, you can also click the check box to reset the buffers after taking the snapshot.

11.3.3 Output

The **Save Snapshot** dialog box provides automatic file-naming facilities that make saving multiple data buffers convenient. The resulting pathname and filename will appear in the read-only **Filename** text box at the top. The **Output** panel contains the following fields:

Format

You can save snapshot files in the following formats:

StethoScope

This native format, which is XML, is the default. Files written in this format can be re-loaded into StethoScope for later viewing, for comparison to live data, or for export to other formats. Files in StethoScope format use the extension **.ss7**. The default format is a compressed version of the StethoScope native format, with the extension **.ss7.gz**.

MATLAB

MATLAB is a commercial data-analysis program from The MathWorks, Inc. When data is saved in **MATLAB** format, two files are created: a data file with a **.mat** extension and a script file with a **.m** extension.

ASCII

Human-readable ASCII format is also supported. Files written in this format have a **.txt** extension. ASCII files may be used to import StethoScope data into many popular spreadsheet programs.




WARNING: Files in ASCII format can be very large.

MATRIX_χ

MATRIX_χ is a commercial data-analysis program from the National Instruments Corporation. When data is saved in **MATRIX_χ** format, two files are created: a script file with a **.ms** extension and a data file with a **.xmd** extension.

Directory

Click the browse icon () to select the path for the output file.

Base Name

You can enter a filename in this text box. This is called the **base** filename because you can add automatic extensions, as described below.

Extension

StethoScope can alter automatically the base filename to help you identify your data later. The **Extension** drop-down menu determines how the new base names are produced. The possible settings are:

None

The **Base Name** string is not changed.

Time stamp

The date and time are appended to the **Base Name**.

Cycle 0-9

If the **Base Name** does not end in a digit already, StethoScope appends a **0** to the name. *After* each store operation, the digit is incremented. Once the digit reaches **9**, it is reset to **0**. Using this option, the most recent ten buffers will be saved with unique names.


Increment

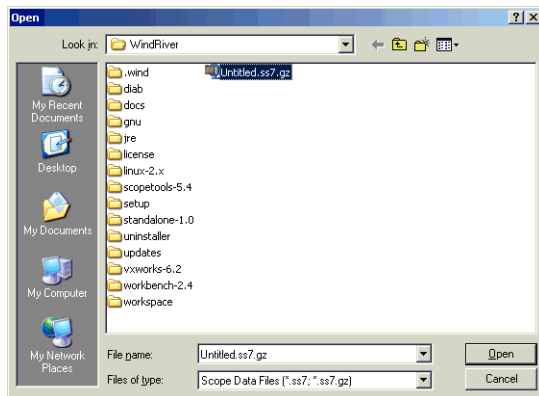
If the **Base Name** does not end in a number, StethoScope appends a **0** to the name. *After* each store operation, the number is incremented, creating a unique name for each save.

How Output Filenames Are Built

Output filenames for snapshots consist of the word **Snapshot** concatenated with an integer number, starting at **1** and increasing by **1** with each new snapshot.

11.4 Loading Snapshots

Snapshots that have been saved to disk in StethoScope's native format (**.ss7** extension) can be reloaded for viewing in the **Plot** or **Plot XY** windows. The **File > Load Snapshot** menu command (or the  button), described in [3.2.2 Load Snapshot](#), p.27, brings up the **Open** dialog box, as shown in the figure below.



11.5 Exporting Snapshot Data to MATLAB and MATRIX_X

When you save a snapshot in **MATLAB** or **MATRIX_X** format (see [11.3.3 Output](#), p.156), two files are created:

- A script file (**.m** extension for **MATLAB**, **.ms** extension for **MATRIX_X**)
- A data file (**.mat** extension for **MATLAB**, **.xmd** extension for **MATRIX_X**)

Running the script file within **MATLAB** or **MATRIX_X** will load the data and create named vectors that correspond to each signal name saved from the buffer. This section describes the notes and variables created by the script.

11.5.1 Creating Notes

The script file contains the following commands that create character-array variables for the run title and session notes, which are both entered on the **Save Snapshot** dialog box (see [11.3 Saving Snapshots](#), p.153):

runtitle

Contains the run title.

notes

Contains the session notes.

This makes it very easy to view your notes from within these analysis programs.

11.5.2 Creating Variables

Running the script file generated when you save a snapshot in **MATLAB** or **MATRIX_X** format creates the following variables:

data

The raw data as a two-dimensional matrix.

signals

The script decomposes **data** into individual arrays that correspond to the signal in the data buffer. The variable names will be the same as the original signal name, except illegal characters are replaced with underscore characters. For example, a signal named, “**xdes[3]**” will be converted to the variable name, “**xdes_3_**” in **MATLAB** or **MATRIX_X**.

numberOfSamples

The length of each signal. This is the number of samples collected in a buffer cycle.

numberOfSignals

The number of signals saved in the data buffer.

time

A vector of time values for each sample.

names

A string array of the signal names.

timestamp

A vector of time values for each sample.

units

A string array of the units for each signal.

timestamp

A string denoting the data-collection date and time.

filename

A string representing the name of the data file, without the filename extension.

runtitle

The run title as displayed in the **Run Title** field.

notes

A character array containing each line of the session notes.

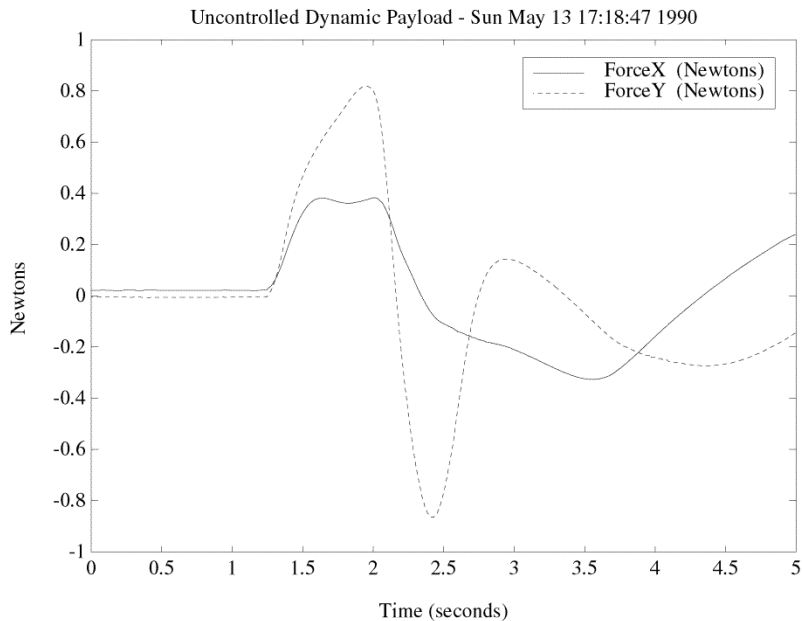


NOTE: Limitations on the size of variable names in **MATLAB** or **MATRIXx** may cause signal names to be truncated. If the truncation results in name clashes, then some signals may not be accessible by name as an individual array. In such cases, you still can extract the signal from the **data** matrix with a statement, such as:

```
ShoulderVel = data(:,2);
```

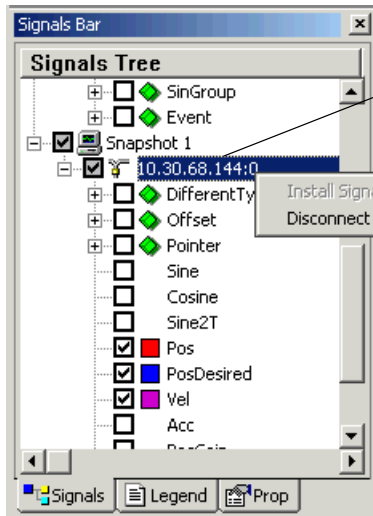
11.5.3 MATLAB Script Example

The **MATLAB** script, **varplot.m**, provides an example of a simple **m-file** program that utilizes stored StethoScope data (see [C. MATLAB and MATRIXX Examples](#)). This script creates a plot, as shown in the figure below.



11.6 Deleting Snapshots

If you want to delete a snapshot from the **Plot** window, you can right-click the snapshot name node in the **Signals Tree** to open a pop-up menu, as shown in the figure below. Click **Disconnect Items** in the pop-up menu to delete the snapshot from the **Signals Tree**. You will not be able to retrieve this snapshot unless you previously saved it, as described in section [11.3 Saving Snapshots](#), p.153.



Right-click the Signal name in the Signals Tree to open a pop-up menu, then select Disconnect Items to delete the snapshot

12

Using a VxWorks Target

[12.1 ScopeProbe Requirements](#) 163

[12.2 VxWorks Targets](#) 164

[12.3 StethoScopeTroubleshooting](#) 172

12.1 ScopeProbe Requirements

In general, to build an application that is instrumented with the ScopeProbe API, you need to:

Add the following include file to your code:

```
#include "scope/scope.h"
```

Add the following **DEFINE** to your makefile or project:

```
-D RTI_VXWORKS
```

Add include paths so the compiler can locate the **scope.h** include file:

```
-I $WIND_SCOPETOOLS/target/include/share/scope
```

Link the libraries:

```
libutilsipz.a  
libxmlparsez.a  
libscope77tcpz.a
```

libutilsipz.lib
libxmlparsez.lib
libscope77tcpz.lib

For TCP/IP:

scopeutils.so (contains **libutilsip.so** and **libxmlparse.so**)
libscope77tcp.so

For WTX:

scopeutils.so (contains **libutilsip.so** and **libxmlparse.so**)
libscope77wtx.so




NOTE: A detailed implementation of some of these steps, as applied to the VxWorks demonstration code **vxdemo.c** and its **makefile**, is shown in [B. StethoScope Demonstration](#).

12.2 VxWorks Targets

The following sections discuss specific requirements that depend on the VxWorks target platform.

12.2.1 Building


The Wind River ScopeTools installation installs the required header files into the ScopeTools installation tree. You do not need to link the ScopeProbe libraries directly to your application. Instead, the ScopeProbe libraries are loaded automatically when you click StethoScope  on the Workbench IDE.

The folder **WIND_SCOPETOOLS\target\src\scopedemo**, contains the source code for the demo program **vxdemo.c** that you can start from Workbench, where **WIND_SCOPETOOLS** is the directory where you installed the ScopeTools. The directory also contains a **makefile** that you can use to compile the code. We suggest you use this makefile as a template for compiling your code instrumented with ScopeProbe API.



NOTE: See [B. StethoScope Demonstration](#) for instructions on how to build **vxdemo.c**.

12.2.2 Automatic Loading and Running

The installation of ScopeTools for Workbench also installs a  button for StethoScope. The StethoScope button appears in the Workbench IDE toolbar.



Clicking the StethoScope button opens the **Setup Options** dialog box that lets you specify initialization parameters for ScopeProbe on the target and StethoScope on the host. After setting any desired parameters and clicking **OK** in this dialog box, the following actions are performed:

- The required VxWorks libraries are loaded onto the target.
- The StethoScope libraries are loaded onto the target.
- ScopeProbe is initialized on the target.
- The StethoScope GUI is started on the host.
- Optionally, the demo program can be loaded and started on the target.


Loading and Starting Automatically

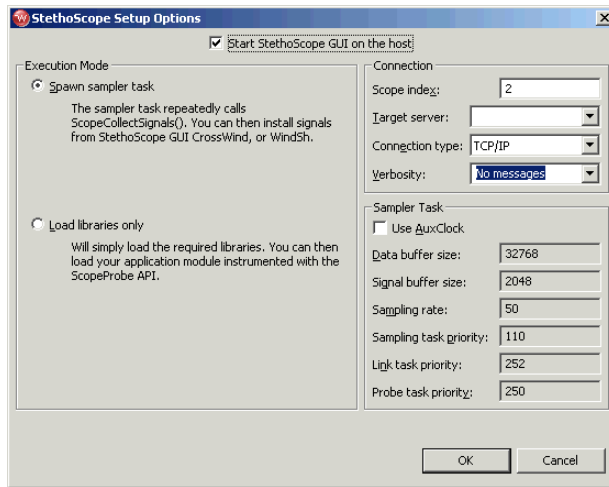
To load and start StethoScope automatically:

1. From the Workbench IDE, select the target server for the target to which you wish to connect StethoScope. If you do not have a target server running, you will need to create it first. Refer to your platform's *User's Guide* for details on how to configure and start a target server.



NOTE: The StethoScope setup script uses the currently selected target-server name, `target@tgtsvrHost`, to determine the host name of the target.

2. From the Workbench IDE, click the StethoScope button (). This opens the **Setup Options** dialog box, allowing you to configure the StethoScope and ScopeProbe initialization parameters.



3. In the **Setup Options** dialog box, specify whether you want to run an asynchronous sampler task, or just load the libraries:

Spawn Sampler Task

Spawns a task on the target that will collect data for any signals that you install (either from the StethoScope GUI on the host, or from the host shell) using the specified **Scope Index**. This task repeatedly calls **ScopeCollectSignals()**, based on its own timing (you can also use the auxiliary clock), so it is asynchronous with the running of your application. This option requires you to provide additional initialization parameters, such as **Sample Buffer Size**, **Signal Buffer Size**, **Sampling Rate**.

Load Libraries Only

Loads the required applications libraries. You can then load your application module instrumented with the ScopeProbe API.

4. For all modes, specify the following settings:

Scope Index

Distinguishes the different instances of ScopeProbe daemons running on the same target. Up to 128 different instances may be started on a target, so the index can range from 0 to 127. The index specified here will be used to initialize the StethoScope GUI and, if selected, the demo program or sampler task.

Target server

Select a target server from the drop-down list of connected target servers.

Connection type

Use the default (TCP/IP), or choose **WTX** type. Use the WTX protocol only if your target does not have TCP/IP support. Specifying WTX protocol when running WTX over a serial line can severely limit data throughput.

Verbosity

Controls the volume of status and information messages written to the log file. **Verbosity** has the following options to choose from:

No messages—Displays only critical messages (most restrictive).

Error messages only—Displays only error messages.

Error and warning messages—Displays both of these message types.

All messages—Displays all system messages (most verbose).

5. If you selected **Spawn Sampler Task**, you must specify the following parameters:

Use AuxClock (check box)

Causes the sampler task to attach a semaphore to the VxWorks auxiliary clock for periodic timing. You must clear this box if you do not have an auxiliary clock, or if you have another application using the auxiliary clock. If this option is not checked, the sampler task calls **taskDelay()** to obtain pseudo-periodic timing.

Data buffer size

Specifies the size (in bytes) of the buffer that will be allocated on the target for collecting data samples. For a description of this buffer, see [14.3.2 Target Buffers](#), p.191.

Signal buffer size

Specifies the size (in bytes) of the buffer that will be allocated on the target to store signal information. For a description of this buffer, see [14.3.2 Target Buffers](#), p.191.

Sampling rate

Specifies the rate (number of times per second) at which the sampler task will call **ScopeCollectSignals()** to collect data for the signals installed to the specified **Scope Index**.

Sampling task priority

Sets the task priority for the sampler task.

Link task priority

Sets the task priority for the link task.

Probe task priority

Sets the task priority for the probe task.

6. Click **OK** to load the appropriate libraries onto the target and initializes the target. If the **Start StethoScope** GUI on the host option is checked, the StethoScope GUI will appear shortly, with an open **Plot** window.

Verifying Target Initialization

To verify that the target has been initialized:

1. From the Workbench IDE, right-click your target, select **Target Tools > Host Shell** to bring up a shell.
2. Type the command **i** in the shell to print a list of running tasks.
3. If you are *not* using WTX mode, the following tasks should be listed:
 - **tProbeDaemon**
 - **tLinkDaemon**
 - **tSamplerTask** (if you chose to run the sampler task)

Verifying Target Connection

To verify that the StethoScope GUI is connected to your target:

1. Verify that the target appears in the **Signals Tree** of the **Plot** window.
2. To open a **Signals Bar** panel if one is not already displayed, see [7.5 Signals Bar](#), p.92.
3. Select the signals you want to display in the **Plot** window (see [4. Using the Signal Manager](#)). If you are not running the demo program, make sure you have installed signals already (see [4. Using the Signal Manager](#) and [14.4 Registering and Activating Signals](#), p.191).
4. Verify that the selected signals are plotted in the **Plot** window.

12.2.3 Manual Target Loading and Running

If you do not use the Workbench IDE, or if automatic loading fails, you will have to determine which libraries to load yourself, load them, and initialize StethoScope manually. Manual loading of StethoScope is more involved than automatic loading. We strongly recommend you create a target-shell script that you “source” from the VxWorks shell (WindSh) to accomplish StethoScope loading and initialization.

To load a library, type in the host window or add to the script file a line using the following syntax.

```
ld 1 < WIND_SCOPETOOLS/target/arch/targetArch/library
```

where **SCOPETOOLSHOME** is the root of the tree where you installed StethoScope, *targetArch* reflects the processor and VxWorks version you are using, and *library* is the library to load. The *targetArch* string format is:

```
cpu os osversion compilerversion
```

Refer to Appendix A in the RTI ScopeTools Installation Guide and Release Notes for a list of supported architectures.

The “1” flag in the “ld” command causes the local symbols to be loaded along with the global symbols; we recommend you always use this flag when debugging or using Wind River ScopeTools with your code.

Table 12-1 lists the libraries needed by the ScopeProbe daemons.

Table 12-1 Target Libraries

Target Configuration	Target Libraries
VxWorks 6.1	scopeutils.so (contains libutilsip.so, libxmlparse.so) libscope77tcp.so or libscope77wtx.so vxdemo.so (for demo only)

The following sections describe in detail how to determine which libraries to load.

Loading the Wind River Utilities Library

StethoScope requires the **ScopeUtils** library.

Load **scopeutils.so** using:

```
-> ld 1 < WIND_SCOPETOOLS/target/arch/targetArch/scopeutils.so
```

Loading the ProfileScope Library

If your target supports TCP/IP, load **libscope77tcp.so** using:

```
WorkbenchVxWorks-> ld 1 <  
WIND_SCOPETOOLS/target/arch/targetArch/libscope77tcp.so
```

If your target does not have TCP/IP enabled, load **libscope77wtx.so** using:

```
WorkbenchVxWorks-> ld 1 <  
WIND_SCOPETOOLS/target/arch/targetArch/libscope77wtx.so
```

Loading the Demo Library

If you wish to load and run the demo, load it using:

```
WorkbenchVxWorks-> ld 1 <  
WIND_SCOPETOOLS/target/arch/targetArch/vxdemo.so
```

To initialize the demo, type the following function call at the VxWorks shell or place it into a target-shell script:

```
sp VxDemo (useAuxClk, scopeIndex, verbosity)
```

where the parameters have the following meanings:

useAuxClk

Set to **0** to use **taskDelay()** for timing, or **1** to use the VxWorks auxiliary clock.

scopeIndex

Specifies the scope index value.

verbosity

Specifies the amount of debug messages printed by the ScopeProbe daemons. These messages will appear in the shell from which you run **VxDemo()**. A value of **0** causes only error messages to be printed. Increasing this value increases the amount of warning and debug messages.

Starting the Sampler Task

If you wish to start the sampler task (rather than run the demo) for asynchronous sampling of signals, type the following function call at the VxWorks shell or place it into a target-shell script:

```
ScopeSamplerTaskCreate(dontUseAuxClk, scopeIndex, sampleBufferSize, signalBufSize,  
samplingRate, verbosity)
```

where the parameters have the following meanings:

dontUseAuxClk

Set to **1** to use **taskDelay()** for timing, or **0** to use the VxWorks auxiliary clock.

scopeIndex

Specifies the scope index value.

sampleBufSize

Specifies the size (in bytes) of the buffer that will be allocated on the target for collecting data samples. For a description of this buffer, see [14.3.2 Target Buffers](#), p.191.

signalBufSize

Specifies the size (in bytes) of the buffer that will be allocated on the target to store signal information. For a description of this buffer, see [14.3.2 Target Buffers](#), p.191.

samplingRate

Specifies the rate (number of times per second) at which the sampler task will call **ScopeCollectSignals()** to collect data for the signals installed to the specified *scopeIndex*.

verbosity

Specifies the amount of debug messages printed by the ScopeProbe daemons. These messages will appear in the shell from which you run **VxDemo()**. A value of **0** specifies that only error messages are printed. Increasing the value increases the amount of messages.

12.2.4 Example Target Script

The following is a complete example of a target-shell script to load and initialize StethoScope and the demo on a VxWorks target. The target in this example supports TCP/IP.

The example script for Workbench 2.3.1 / VxWorks 6.1:

```
ld 1 < WIND_SCOPETOOLS/target/arch/ppcVx6.1gcc3.3.2/scopeutils.so
ld 1 < WIND_SCOPETOOLS/target/arch/ppcVx6.1gcc3.3.2/libscope77tcp.so
ld 1 < WIND_SCOPETOOLS/target/arch/ppcVx6.1gcc3.3.2/vxdemo.so
sp ScopeDemo
```

12.2.5 Starting the ProfileScope GUI Manually

To start the StethoScope GUI manually, see the instructions in [Starting Manually from the Command Line](#), p.10, or consult the reference documentation (paper manual or online manual) for the topic “scope”.

12.3 StethoScope Troubleshooting

12.3.1 Load Errors

If you have trouble loading the object files onto your VxWorks target, check the following:

- You are able to **ping** the target over the network.
- If you are using NFS, the file system is mounted (check with **nfsDevShow**).
- Your target has permission to read the object files from the file server.

12.3.2 Connection Failure

If StethoScope's **Plot** window status message displays, **Target *target* not responding**:

- Make sure your network is configured properly. You must be able to establish a network connection to your target via **ping** or **rlogin** before StethoScope will work.
- On VxWorks targets, if these tests are successful, type **i** on the target processor and make sure the **tProbeDaemon** and **tLinkDaemon** tasks are active.
- Make sure you are using the same scope index on the target and host.

12.3.3 No Response from Target

If the StethoScope **Plot** window status message displays, **Target target not responding**, or StethoScope exits with the message:

```
scope: target is connecting but not responding.
```

then one of the following has occurred:

- Another copy of StethoScope GUI is connected already to your target.
- The **tLinkDaemon** task is being starved for processing time.
- The network is not configured correctly.
- The target is loading an incompatible version of ScopeProbe.

Multiple Connections

The first condition can be verified using the **Windows Task Manager**.

Starvation

Under VxWorks, the second condition can be tested by executing this command on the target:

```
-> taskSpawn("test", 255, 0x1c, 12000, printf, "Not starved.\n")
```

This tries to spawn a low-priority task that only prints a message. If the message, “**Not starved**”, does not print, then starvation is the problem. Starvation can be cured by incrementing the priorities of **tLinkDaemon** and **tProbeDaemon** tasks (via **taskPrioritySet()**) or ensuring that higher-priority processes do not use all the available CPU. This is rarely a problem for Windows targets.

Network Configuration

A **ping** or **rlogin** test suffices to test proper network connection.

Version Mismatch

The ScopeProbe version can be printed via **ScopePrintVersion()**. The version of the StethoScope graphical interface is displayed on the **Help > About StethoScope** dialog box.

None of the Above

Finally, try turning on the verbosity. Call **ScopeInitServer()** (or start the demo or sampler task) with verbosity set to **1**; start the StethoScope GUI with the **-v 1** flag. The output messages may help you pinpoint the problem.

12.3.4 No Data

If the connection appears to be normal, and the **Signals Tree** in the **Signals Bar** on the **Plot** and **Dump** windows show installed signals but no data appears, check the following:

- **ScopeCollectSignals()** is being called to sample data.
- **Triggering** is set up correctly and that trigger conditions are occurring.
- The **tLinkDaemon** task is not being starved for processing time.
- You are viewing an active buffer.

Sampling

Under VxWorks, the first This condition can be tested by placing a breakpoint at **ScopeCollectSignals()** from the VxWorks shell.

Triggering

Disable triggering from the **Triggering** dialog.

Starvation

Starvation can be tested as described in [12.3.3 No Response from Target](#), p.173. Raising the priorities of **tLinkDaemon** and **tProbeDaemon** tasks or insuring that higher-priority processes do not use all the available CPU will alleviate the problem. Again, this is rarely a problem for Windows targets.

None of the Above

Finally, make sure the window you are using is displaying the *Live* buffer and not a stored static buffer (snapshot). Also make sure the range is wide enough to plot data on the screen (try the **Zoom to Fit** button, or from the **View** menu).

13

Signal Installation

- 13.1 Installing Signals for StethoScope 177
- 13.2 Automatic Signal Installation 179
- 13.3 Manual Installation 184
- 13.4 Using StethoScope API 185
- 13.5 Code Instrumentation Alternative 185
- 13.6 Process Notes 185

13.1 Installing Signals for StethoScope

Recall from the discussion in [2.5.3 Concepts of Use](#), p.12 that *installed signals* are the means used by the Wind River StethoScope GUI to specify which data you want to be able to monitor, collect, and display in the GUI. You must first install each signal before it can be collected for analysis and displayed by the GUI. This chapter guides you through that process.

The hierarchal steps that must be executed to create *installed* signals is as follows:

1. **Register a Signal**—Initially you must let StethoScope know a signal exists by *registering* it using the API call `ScopeRegisterSignal()`. StethoScope cannot collect data from this signal until you Activate it. Registered signals appear in

the **Signal Manager** window in the GUI, where they can be selected for activation.

2. **Activate the signal**—This is done to a registered signal that has been set up on the host by the **Signal Manager** (see [3.2.7 Signal Manager](#), p.32) using the API call **ScopeActivateSignal()**. Active signals then appear in the **Signals Bar** of each data display window (see [2.6.5 Signals Bar](#), p.15).

Once activated, the signal is considered to be **Installed** and is automatically collected from the target, but is not yet displayed in the host GUI until **Selected** in one or more of the four data display windows: **Plot**, **Plot XY**, **Dump Plot**, and **Monitor**.

3. **Install the signal** (optional)—This is an operation that registers and activates in a single step using the StethoScope API shortcut **ScopeInstallSignal()** (see [14.4.1 Installing Signals](#), p.192). A signal installed in this manner is also “seen” by the StethoScope GUI.
4. **Selected signals**—Installed signals are not displayed automatically in the GUI. You must use the GUI to select the installed signals you want to display in the data-display windows—**Plot**, **Plot XY**, **Dump Plot**, and **Monitor**. You can select a different set of signals in each window (see [2.6.5 Signals Bar](#), p.15).

The hierarchical steps listed above can be translated into a simple mechanism for signal installation, as implemented on a VxWorks target. StethoScope provides 3 ways to install signals on your target:

Automatically

Signals can be installed using a variable name or a variable expression you enter and letting StethoScope find the variable and determine its address in the executable.



NOTE: For this mode of signal installation you are required to have compiled your code with DWARF2 debugging information.

Manually

You provide the machine address, in hexadecimal, for the variable that you want to install. This method requires that you know the variable's address

Using the StethoScope AP

You have options with this method, but it requires you to recompile your code in any case.

13.2 Automatic Signal Installation

A signal name is the name you assign to a data item that is collected from your target application by StethoScope. Signals can be installed by instrumenting code using the StethoScope API (see [A. StethoScope API Reference](#)). They can also be installed automatically, by variable name, or manually, by address. While installing variables automatically by name is more convenient, it requires that your program be compiled with DWARF2 debugging information.

13.2.1 Requirements

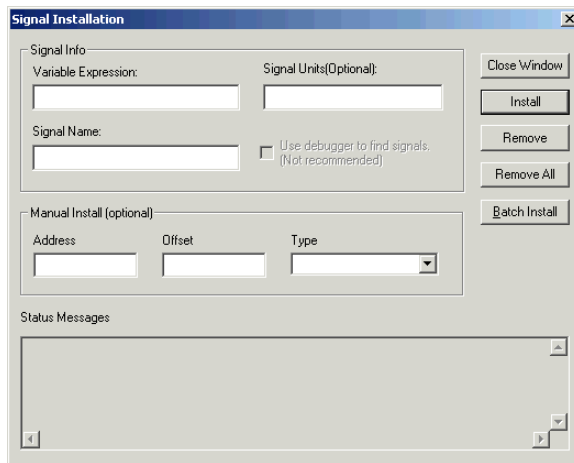
To install signals **automatically**, you need to:

- Compile your code with DWARF2 debugging information enabled (-g option will usually suffice).
- Specify the variable name or variable expression in the **Signal Installation** dialog box.
- Click the **Install** button in the **Signal Installation** dialog box to find the address and data type of the variable, and subsequently install it on the target. VxWorksStethoScope

Internally, StethoScope uses an ELF/DWARF 2.0 parser to extract the type information of all variables in the object modules loaded on the target. During the signal installation, StethoScope uses this type information to find the address of the signal on the target.

13.2.2 Signal Installation Dialog Box

The **Signal Installation** dialog box, shown in the figure below, is opened by selecting **Signal Installation** on the **Signal Manager** window's menu bar, or by selecting **Install Signal** from the pop-up menu when you right-click a target in a **Signal Manager** window (or a **Signals Tree** sub-window).



The **Signal Installation** dialog box allows you to install signals in the following ways:

- Automatically (by variable name)
- Batch install all signals previously installed automatically (see [13.2.3 Batch Signal Installation](#), p.183)
- Manually (by address)
- Remove existing signals

The various text entry fields and buttons in the **Signal Installation** dialog box are described below.

The **Signal Info** panel, for automatically installing a signal by variable name, contains the following text entry fields and controls used in finding and installing a variable:

Variable Expression

Specifies a variable name or variable expression that can be used to locate a variable on the target and install it subsequently. Some examples include **array[3]**, **arm.pos**, and **body->vel**. Variable expression is not required to remove signals.

Signal Units (Optional)

Specifies the units of the signal to be installed.

Signal Name

Specifies the name of the signal to be installed or removed. It does not have to match the **Variable Expression** entry. **Signal Name** is optional for signal installation, but it is required to remove a signal or a group of signals.

During signal installation, if a signal is of a **class/struct/union** type, the top level directory entry in hierarchical signal names (see [13.6.2 Hierarchical Signal Names](#), p.187) corresponds to the **Signal Name** field if specified or **Variable Expression** field if the **Signal Name** field is blank. During signal removal, all the signals matching the **Signal Name** field are removed from the target.

Use debugger to find signals

This check box, if selected, loads and attaches a debugger to your target application and uses it to find the variable. The default setting is not selected.



CAUTION: It is recommended that you **NOT** use this option, due to the slow processing and intrusiveness of using the debugger in your target. Occasionally you may have to use it, however, because searching does not work with some compilers, and on some target processors. The rule of thumb is to only use it only when you have to.



NOTE: This option does not find signals that were installed using the **Manual Install** procedure described in [13.3 Manual Installation](#), p.184.

To save the list of signals you have installed by hand with this dialog box (automatically only), use the **Batch Signal Install** dialog box described in [13.2.3 Batch Signal Installation](#), p.183.

The **Manual Install** (optional) panel, for manually installing signals by address, contains the following text entry fields used to enter the address:

Address

Enters the machine address (in hex) to be monitored. You must know the address of the variable you want to monitor.

Offset

Enters the offset (in hex) of a member within a structure, if the variable expression contains a pointer to a structure (or class). You must know the value of this offset.

Type

A drop-down menu of allowable data types. Select one of the types from the menu.

Status Messages

Displays the status of your connection to the target agent. When connected, it displays the **host name** and **port number** of the target agent you are connected to.

Buttons on the right side and bottom of the dialog box initiate actions required to find and install signals, and to remove existing signals. The buttons are:

Close Window

Closes the **Signal Installation** dialog box.

Install

Searches for the variable expression and, if found, installs it to StethoScope. The variable expression will be used as the signal name if the **Signal Name** field is left blank.

Remove

Removes the signal matching the **Signal Name** field from the StethoScope GUI. This can be a signal installed through the StethoScope GUI on the host or through target code.

Remove All

Removes all signals. This will remove signals installed through the StethoScope GUI *and* any signals installed through target code.

Batch Install

Opens the **Batch Signal Install** dialog box ([13.2.3 Batch Signal Installation](#), p. 183) where you can save, and later install, the list of signals previously installed by variable name.



NOTE: This option can only access the names of signals installed up until you reboot your target. If you must reboot without first saving the signal list, the list is lost.

The **Status Messages** panel displays the results of requested actions, including any error messages.

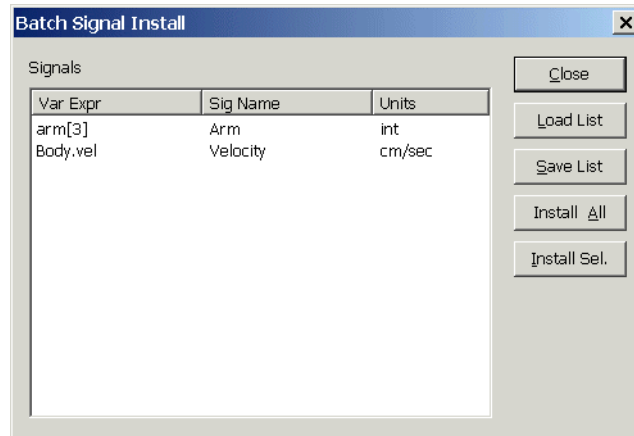
13.2.3 Batch Signal Installation

The **Batch Install** button in the **Signal Installation** dialog box (see below) opens the **Batch Signal Install** dialog box. With this dialog box, you can save the list of signals you previously installed (automatically only) using the **Signal Installation** dialog box. You would want to consider doing this against the possibility of having to reboot your target (in which case these current signals would otherwise be lost and you would have to enter them again by hand). If you have saved a signal list with this option, you can then load it and re-install the signals at any time using this dialog box.

You can also re-install the entire list of signals with one click, or install individual signal(s) you select from the list.



NOTE: The services of this dialog box only work for signals installed automatically (by variable name) using the **Signal Installation** dialog box. Signals installed manually (by address) cannot be saved.



Controls

The various text entry fields and buttons in the **Batch Signal Install** dialog box are described below.

Signals

This list shows the current list of signals you installed (automatically only) using the **Signal Installation** dialog box, or the list of signals you loaded from a file. These are the only signals you can manipulate using this dialog box.

Buttons

Close

This button closes the dialog box.

Load List

This button loads the contents of a file into which you previously saved a list of signals you had installed.

Save List

This button saves all the current signals you have installed by hand (automatically only) prior to the last time you rebooted your target.

Install All

This button re-installs all the files shown in the list in the **Signals** field without having to select any of them.

Install Sel.

This button re-installs only the files you have selected in the list.

13.3 Manual Installation

Like automatic installation, manual signal installation is done while the StethoScope GUI is running. The difference is, however, that you must know the virtual address of the signal in the program's memory.

To install a signal *manually by address*, in the **Signal Installation** dialog box (see [13.2.2 Signal Installation Dialog Box](#), p.179) you need to:

- Specify the address (and offset, if applicable) in the **Manual Install** panel.
- Select a data type from the drop-down menu in the **Manual Install** panel.

- Use the **Install** button to install the signal.

If there are any errors in the process of installing the specified signal, they are reported in the **Status Messages** field.

13.4 Using StethoScope API

This method requires you to instrument your target source code with appropriate calls to routines from the StethoScope API library. The process, and the routines it uses to install and activate signals, are described in detail in [14. API Introduction](#).

13.5 Code Instrumentation Alternative

You may want to take advantage of automatic signal installation, but not always want to instrument your target code with the StethoScope API library. Or perhaps you may not have the source code available for instrumentation. In such cases there is an executable file, **process sampler**, you can run on your target, that is itself fully instrumented with the StethoScope API (see [A. StethoScope API Reference](#)).

This file can be found in the directory where you unpacked the target-side components. Start it as you would any other standalone application. Connecting to your target will find this instrumented program, and you can then use it to automatically install all the signals in your own target code.

13.6 Process Notes

The following sections serve to clarify and elaborate on the concepts used in the process of StethoScope signal installation.

13.6.1 Variable Expressions vs. Signal Names

It is important to understand the distinction between **variable expressions** and **signal names**.

Variable Expression

Is the name of a variable that exists in your code and appears in your programs as a data symbol. Valid variable expression entries include:

- Variables of any primitive type, such as **int**, **float**, etc.
- Pointers to any primitive type, such as **int ***, **float ***, etc.
- Variables of type **enum**, **bool**, or **boolean** will be converted to **int** or **char**, depending on how the compiler implemented them.
- Member variables of structure or class, such as **Body.vel**, **Body->vel**, **Body->Pos->x**.
- Array elements, such as **arm[3]**.
- Instances of **class**, **structure**, or **union**—all member variables (including member variables of **class/struct/union** data types except pointers to the **class/struct/union** types) will be installed as separate signals.
- Array variables—all the array elements are installed.

Signal Name

Is the name you assign to a data item for StethoScope to display in its signals trees and quick-select buttons. This name is also exported to the files you save. It does not have to be the same as the variable expression. A signal name can be any name you choose.

When using the **Signal Installation** dialog box, only global or static variables may be installed. For example, if the variable expression is:

`Body->Pos->x`



then **Body** must be a global or static pointer. Also, **Body** must be pointing to a valid **Pos class/struct/union data pointer** type (and if **x** is also a **class/struct/union data pointer** type, then it in turn should also point to a valid **class/struct/union**) at the time of installing the signal **x** with the above variable expression.

13.6.2 Hierarchical Signal Names

To help you organize your signals better, StethoScope supports hierarchical signal names that use the slash (/) character to separate the levels of hierarchy, much like path names for files. Hierarchical organization is useful for:

- Grouping member variables of a class or structure. Just substitute “/” for “.” or “->” when entering signal names that refer to member variables.
- Creating logical groupings among variables that are not otherwise in a common structure. Just use a common “directory” name in their signal names.

Whenever StethoScope displays signal lists as a tree, the tree consists of signal and directory entries such that:

- A signal entry corresponds to a single registered signal.
- A directory entry—indicated by a  or  node icon—contains sub-entries that are signal entries or other directory entries.
- A directory entry may be expanded or collapsed to show or hide its sub-entries.

Example 13-1 Examples of hierarchical signal names

```
Robot/LeftArm/PosX  
Robot/LeftArm/PosY  
Robot/RightArm/PosX  
Robot/RightArm/PosY
```

13.6.3 Classes and Structures

If the named variable refers to an instance of a class, structure, or union, StethoScope installs all its member variables, grouping them under the same directory (corresponding to the **Signal Name** field or **Variable Expression** field if **Signal Name** field is left blank) using the hierarchical signal-naming capability (see [13.6.2 Hierarchical Signal Names](#), p.187). StethoScope skips a member variable if it is a pointer to another class, struct, or union to prevent a potentially dangerous circular path of variable installations.

Example 13-2 Consider the following type definitions and the variable declaration

```
typedef struct {  
    float PosX;  
    float PosY;  
} Position; /* Position type definition */
```

```
typedef struct {  
    Position RightArm;  
    Position LeftArm;  
} RobotType; /* RobotType type definition */  
  
RobotType Robot;
```

When you specify a variable expression as *Robot* and signal name as *SCARA* in the **Signal Installation** dialog box, *Robot* signal is installed with the following hierarchy:

```
SCARA/LeftArm/PosX  
SCARA/LeftArm/PosY  
SCARA/RightArm/PosX  
SCARA/RightArm/PosY
```

If you specify a variable expression as *Robot.RightArm* with the **Signal Name** field left blank, *RightArm* member variable is installed as:

```
Robot.RightArm/PosX  
Robot.RightArm/PosY
```

Note that the member variables in a **class/struct/union** are installed in an alphabetical order on the target.

13.6.4 Libraries

The **scopeutils.so** library contains the following libraries depending on the communication protocol.

TCP/IP:

Contains **libutilsip.so**, **libxmlparse.so**, and **libscope77tcp.so**.

WTX:

Contains **libutilsip.so**, **libxmlparse.so**, and **libscope77wtx.so**.

14

API Introduction

- 14.1 Introduction 189
- 14.2 Using StethoScope API with Your Program 190
- 14.3 Initializing the Server 190
- 14.4 Registering and Activating Signals 191
- 14.5 Sampling Signals 197
- 14.6 Triggering and Sampling Functions 200
- 14.7 StethoScope Events API 200
- 14.8 scope.ini File 204

14.1 Introduction

The real-time data-collection and signal-management module of Wind River StethoScope that runs on the target platform is also known as StethoScope API. StethoScope API collects the time history of variables in your real-time program. Its architecture is summarized in [1. Introduction](#). If your target is running VxWorks, ScopeProbe libraries need to be loaded onto the target. If your target is running Windows or Solaris, the StethoScope API library needs to be linked with your application code.

This chapter introduces the StethoScope API. See [A. StethoScope API Reference](#) for details of the API. To compile and run a target application that is instrumented with StethoScope API already, see [B. StethoScope Demonstration](#).

14.2 Using StethoScope API with Your Program

StethoScope API is a library of routines linked to your target application. It implements a flexible data-collection utility. StethoScope API saves data from your real-time system in a buffer on the target and transmits them to the StethoScope GUI on the host for display.

To use StethoScope API with your target program, do the following:

1. Initialize the server. ([14.3 Initializing the Server](#), p.190.).
2. Set the sample rate. ([14.5 Sampling Signals](#), p.197.).
3. Register and activate (install) the variables to monitor (signals). ([14.4 Registering and Activating Signals](#), p.191.).
4. Sample the data. ([14.5 Sampling Signals](#), p.197.).
5. Shut down the server when done.

14.3 Initializing the Server

Initializing the target server requires only a single call, `ScopeInitServer()`. You will need to specify the scope index number and buffer sizes. For more details, see [A. StethoScope API Reference](#).

14.3.1 Scope Index

The scope index represents the communications “channel” between an instance of StethoScope API running on the target and a StethoScope GUI running on the host.

You can create up to 128 instances of StethoScope API on a single target machine, each using a different scope index. The index can range from 0 to 127, and it must be specified when you call **ScopeInitServer()**.

14.3.2 Target Buffers

StethoScope API allocates three buffers on the target for each scope index:

Sample buffer

Stores the data samples for the active signals. Data samples for all data types other than **double** are saved as 4-byte values. Data samples for doubles are saved as 8-byte values.

Signal buffer

Stores the information that describes each registered signal, such as name, units and data type. The memory used by a signal is reclaimed when the signal is removed, making it available for additional signals. The information stored for a signal takes up 36 bytes plus the number of bytes it takes to store the signal name and units (including the terminating null characters). Note that a signal registered twice, under different scope indices, counts as two registered signals.

Event buffers

These are optional buffers that can be attached to an initialized scope index by calling **ScopeEventsAttach**. The event collection APIs (**ScopeEventsCollect** and **ScopeEventMessage**) use these buffers to throw events. There can be a maximum of four event buffers per scope index. Since the event collection APIs are not reentrant, we recommend that each task that throws events use a separate buffer for event collection. The above would obviate the need for mutual exclusion among tasks that employ events.

14.4 Registering and Activating Signals

A StethoScope signal can be any variable in your code of any basic data type, such as **float**, **double**, **unsigned int**, or a pointer to any basic type. Refer to [Table 14-1](#) for a complete list of types and abbreviations.

Table 14-1 **Acceptable Data Types**

unsigned char	uchar	char
unsigned short	ushort	short
unsigned int	uint	int
unsigned long	ulong	long
float	double	



NOTE: StethoScope supports both the short form and the long form for unsigned types (for example, **uint** vs. **unsigned int**). It also supports user-defined **classes**, **structures**, and **unions**.

In order for StethoScope to collect samples of a particular signal, you must:

- First **register** a signal
- Then **activate** the signal

Registering a signal provides StethoScope supports both the short form and the long form for unsigned types (for example, **uint** vs. **unsigned int**). It also supports user-defined classes, structures, and unions with relevant information such as its name, type, and memory location. Samples are collected, however, only when the signal is activated. The number of registered signals is limited only by the amount of target memory. The number of active signals for a given scope index is, however, limited to 8192.

14.4.1 Installing Signals

A signal is *installed* when it is both registered and activated. To register and activate a signal:

1. Call **ScopeRegisterSignal()** or **ScopeRegisterSignalWithOffset()** to register the signal. The latter function is discussed in [14.4.4 Offsets to Signals](#), p.194.
2. Call **ScopeActivateSignal()** to activate the signal.

Calling **ScopeInstallSignal()** and **ScopeInstallSignalWithOffset()** accomplishes both these steps in a single call (see [14.4.5 Installing Signals](#), p.195).



NOTE: Installed variables *must* be valid when `ScopeCollectSignals()` is called to sample all active signals. This requirement is met for:

- Static variables (such as, global variables)
- Any variable in allocated memory (that is, created using `malloc()`)
- Automatic (stack) variables, *only if* `ScopeCollectSignals()` is called only within the scope of the variable.

Example 14-1 Signal Installation

The following code registers and activates signals for scope index of 0:

```
#include "scope/scope.h"
static float PosX;
static float PosY;
static unsigned char Type;
void initScope( void )
{
    ScopeRegisterSignal("PosX", "meters", &PosX,
                       "float", 0);
    ScopeRegisterSignal("PosY", "meters", &PosY,
                       "float", 0);
    ScopeRegisterSignal("Type", "meters", &Type,
                       "uchar", 0);
    ScopeActivateSignal("PosX", 0);
    ScopeActivateSignal("PosY", 0);
    ScopeActivateSignal("Type", 0);
}
```

14

14.4.2 Hierarchical Naming of Signals

StethoScope supports hierarchical naming of signals just like a file browser. You can register signals using a name that delimits each level with “/”, such as “**Robot/LeftArm/PosX**”. The **Signal Manager** and **Signal Selection** lists within the StethoScope GUI represent each level as a folder. In order to make it easier to manage a large number of signals, you can expand, collapse, and activate entire directories.

14.4.3 Pointers to Signals

You can register signals by pointers to variables. For these signals, `ScopeCollectSignals()` de-references the pointers at the time of collection. You can register a pointer to a signal by specifying a pointer type when calling

ScopeRegisterSignal(). For example, a pointer to a **float** would have a type of "**float ***". There is no limit to the number of de-references StethoScope can handle, so you can specify a type, such as "**float *******". StethoScope supports pointers to all the basic data types listed in [Table 14-1](#).

Example 14-2 **Pointer to Variable Registration**

```
int *JointPosition = (int *)calloc(1, sizeof(int));  
// Notice that you must pass the address of JointPosition.  
ScopeRegisterSignal("JointPosition", "millimeters", &JointPosition,  
                    "int *", 0);
```

14.4.4 Offsets to Signals

You can register a signal using an offset from an address. **ScopeCollectSignals()** de-references the pointers and applies the offsets at the time of collection. This allows you to collect data that is a member variable of a class or structure.

Signal registration using offsets only makes sense when you do have pointers to classes or structures, where the pointers can point to different instances as the application executes. Otherwise you simply could install the member variables directly. Consequently, StethoScope assumes you are providing a reference to a pointer when you register a signal with offsets.

Example Registration With Offset

To register and activate a signal using an offset:

1. Call **ScopeRegisterSignalWithOffset()**, passing it a reference to a pointer.
2. Call **ScopeActivateSignal()** to activate the signal.

You can combine offsets and pointers. This allows you to collect data from member variables (of structures) that themselves are pointers. See [Calculating Offsets](#), p.194 for an example.

Calculating Offsets

When calling **ScopeRegisterSignalWithOffset()** or **ScopeInstallSignalWithOffset()**, you can use the **offsetof()** macro to calculate automatically the offset of a member variable. The following is an example on how to use **offsetof()**.

Example 14-3 **offsetof() use**

```

typedef struct ArmData_s {
float      PosX;
float      PosY;
unsigned char Type;
double     *Vel; // Just to demonstrate pointers and offsets.
} ArmData_t;

int main ()
{
    ArmData_t *LeftArmData =
        (ArmData_t *)calloc(1, sizeof(ArmData_t));
    LeftArmData->Vel = (double *)calloc(1, sizeof(double));

    /*
    * Notice that you can pass just the address of LeftArmData,
    * not the individual fields.
    * Notice that we are using hierarchical names for the signals.
    * This will be discussed below.
    * The type "float" refers to the type of
    * LeftArmData->PosX.
    */
    ScopeRegisterSignalWithOffset("LeftArm/PosX", "meters",
        &LeftArmData, "float",
        offsetof(ArmData_t, PosX), 0);
    ScopeRegisterSignalWithOffset("LeftArm/PosY", "meters",
        &LeftArmData, "float",
        offsetof(ArmData_t, PosY), 0);
    ScopeRegisterSignalWithOffset("LeftArm/Type", "none",
        &LeftArmData, "uchar",
        offsetof(ArmData_t, Type), 0);

    /*
    * The type "double *" refers to the type of the member Vel.
    */
    ScopeRegisterSignalWithOffset("LeftArm/Vel", "meters/s",
        &LeftArmData, "double *",
        offsetof(ArmData_t, Vel), 0);

    /*
    * Add user code and ScopeCollectSignals(). */
    */
}

```

14.4.5 Installing Signals

StethoScope provides convenience functions, **ScopeInstallSignal()** and **ScopeInstallSignalWithOffset()**, that register and activate a signal in one step.

Example 14-4 **Installing elements of an array**

```
#include "scope/scope.h"
static float yhat[17];

ScopeInstallSignal("EstimatorElbowAngle", "radians",
    &yhat[2], "float", 0);
ScopeInstallSignal("EstimatorElbowRate", "rad/sec",
    &yhat[3], "float", 0);
```

Example 14-5 **Member Variable Installation**

```
#include "scope/scope.h"
typedef struct {
int packetsize;
double desiredSwitchValue;
} *SwitchType;
SwitchType inputSw;

inputSw = (SwitchType) malloc(sizeof(*SwitchType));
ScopeInstallSignal("InputPacketSize", "points",
    &inputSw->packetsize, "int", 0);
ScopeInstallSignal("InputDesiredSwitchValue", "gleebs",
    &inputSw->desiredSwitchValue, "double", 0);
```

14.4.6 Deactivating and Removing Signals

You can deactivate and remove signals using the functions:

ScopeDeactivateSignal()

Deactivates a signal, preventing it from being sampled during **ScopeCollectSignals()** calls.

ScopeDeactivateMultipleSignals()

Deactivates signals that match the specified pattern, preventing them from being sampled during **ScopeCollectSignals()** calls.

ScopeRemoveSignal()

Deactivates and unregisters the signal, removing it from the **Signal Manager** of the StethoScope GUI.

ScopeRemoveMultipleSignals()

Deactivates and unregisters the signals that match the specified pattern, removing them from the **Signal Manager** of the StethoScope GUI.

14.4.7 Online Documentation

Consult the reference pages in [14. API Introduction](#) for signal installation and removal functions:

- `ScopeRegisterSignal()`
- `ScopeRegisterSignalWithOffset()`
- `ScopeInstallSignal()`
- `ScopeInstallSignalWithOffset()`
- `ScopeActivateSignal()`
- `ScopeActivateMultipleSignals()`
- `ScopeDeactivateSignal()`
- `ScopeDeactivateMultipleSignals()`
- `ScopeRemoveSignal()`
- `ScopeRemoveMultipleSignals()`
- `ScopeRegisterArray()`
- `ScopeInstallArray()`
- `ScopeShowSignals()`
- `ScopeShowActiveSignals()`

14.5 Sampling Signals

Signals are sampled by calls to the function, `ScopeCollectSignals()`. One sample of each installed variable is taken each time `ScopeCollectSignals()` is called. Your application can call `ScopeCollectSignals()` asynchronously or synchronously with the generation of data.

14.5.1 Asynchronous Sampling

Asynchronous sampling simply takes a snapshot of the variable values at regular intervals during your program's execution. The intervals are not coordinated with your application's execution. Because the sampling is independent of the application task, the snapshots may be taken at unpredictable points in the application code. For example, consider an application with the following loop:

```
void UserTask()
{
    while (1) {
        semTake(sem); /* Wait for something */
        x++;
        y = x;
    }
}
```

One method of asynchronous sampling is to spawn a task whose only job is to sample:

```
void SampleTask()
{
    while (1) {
        ScopeCollectSignals(0);
        taskDelay(delayTime);
    }
}
```

This may produce the following data:

Sample	x	y
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
...

This result has the following problems:

- Sample 3 appears inconsistent, because the sample was taken between the assignment statements for **x** and **y**.
- Sample 4 is inaccurate because the sampling task missed the fourth loop of the user task altogether.
- Sample 5 is a repeat of Sample 4 because the sampling task ran again before the user task started its next loop.

Thus, asynchronous sampling cannot guarantee:

Data-set consistency

All samples in a set form a consistent view of the application-data set.

Sampling accuracy

Every loop of the application is sampled exactly once.

In spite of the consistency and accuracy issues, asynchronous sampling is often desirable because it is easy to set up and requires no changes to your application

code. Asynchronous sampling also works with programs that are not periodic. In many cases all you really need is a general idea of what all your variables are doing; asynchronous sampling does that job well. In fact, you can load and run the demonstration program along with your application and immediately begin viewing your variables. They will be sampled asynchronously by the demonstration sample loop.

14.5.2 Synchronous Sampling

Because of the issues with asynchronous sampling, many applications require *synchronous sampling*, where the sample times must be coordinated with the application's execution. If your application requires synchronous sampling, call **ScopeCollectSignals()** directly from your application at the instant you wish data to be sampled:

```
void UserTask()
{
    while (1) {
        semTake(sem); /* Wait for something */
        x++;
        y = x;
        ScopeCollectSignals(0);
    }
}
```

This will always produce consistent, accurate sampling:

Sample	x	y
1	1	1
2	2	2
3	3	3
...

A more detailed example of how to set up synchronous sampling in VxWorks is presented in [A. StethoScope API Reference](#).

14.5.3 Sample Rate

The sample rate is defined as the frequency of calls to **ScopeCollectSignals()**, for example, the number of times per second your application calls **ScopeCollectSignals()**. Your application determines the sample rate. **ScopeChangeSampleRate()** simply reports that rate to StethoScope. The sample rate may be changed at any time during execution via another call to **ScopeChangeSampleRate()**.



NOTE: Down-sampling specified by the StethoScope GUI causes every few calls to `ScopeCollectSignals()` to actually store data.

Further details are available under the entries for `ScopeInitServer()` and `ScopeCollectSignals()` in [14.7 StethoScope Events API](#), p.200.

14.6 Triggering and Sampling Functions

The triggering and sampling control functions allow run-time access to the data-collection mode and settings from the real-time program. For instance, one of the best uses of the trigger routines is to collect data when some condition is detected by the code. With pre-triggering in effect, this makes it simple to analyze very difficult-to-find error conditions.

The triggering and sampling control functions are:

- `ScopeTriggerSet()`
- `ScopeTriggerGet()`
- `ScopeChangeSampleRate()`
- `ScopeCollectionModeEnable()`
- `ScopeCollectionModeDisable()`
- `ScopeCollectionModeGet()`

See [14.6 Triggering and Sampling Functions](#), p.200 for details on these functions.

14.7 StethoScope Events API

The StethoScope Events API is a set of low-overhead logging routines that were made part of StethoScope Events API version 7.0. These routines can be useful to monitor real-time systems with minimal effect on the timing behavior.

An “event” is defined simply as a line of code represented by a user-specified string (“`eventID`”). An event is said to be “thrown” when a line of user code

instrumented with StethoScope Events API is executed. Optionally, the value of a variable may also be collected when an event is thrown.

The StethoScope Events API includes the following routines:

- **ScopeEventsAttach()**
- **ScopeEventsMaskSet()**
- **ScopeEventsCollect()**
- **ScopeEventsMessage()**
- **ScopeEventsDetach()**

14.7.1 Setting Up the StethoScope Events API

To implement the StethoScope Events API for data collection at strategic locations in your source code, you must first attach an event buffer to an already initialized **scopeIndex** with a call to **ScopeEventsAttach()**. This routine attaches a buffer of the specified size to that **scopeIndex**. A maximum of 4 buffers can be attached to a single **scopeIndex**.



NOTE: Although it is possible to have multiple tasks share a single buffer, we strongly suggest that you only use 1 task per event buffer. This is recommended because the event collection routines, **ScopeEventsCollect()** and **ScopeEventsMessage()**, are not reentrant.

14.7.2 Using the StethoScope Events API

With a collection buffer established, the key to using the StethoScope Events API is to insert a **ScopeEventsCollect()** statement after each code line you want a message printed for, or that uses a variable you want to collect the value of. **ScopeEventsCollect()** collects, and stores in the buffer, one message, or the value of one global or local variable, per call, without the overhead of using a **printf** statement. The values collected in the events buffer are periodically transmitted to the host by the **LinkDaemon** task running on the target.

A verbosity level (in the range of 1 to 32) can be assigned to each event that is “thrown”. This allows you to assign different verbosity levels to the messages (or variable values) coming from different modules, and, by using events masks, have control over which ones are collected. Event masks can be set using the **ScopeEventsMaskSet()** routine.

When the task is finished, it should call **ScopeEventsDetach()** to detach the events buffer from the **scopeIndex**. If the **scopeIndex** is shut down, all event buffers associated with it will be freed.

Example 14-6 **Include StethoScope Events API in Source Code**

```
#include "scope/scope.h"

#define EventLevel1(0x00000001)
#define EventLevel2(0x00000002)
#define EventLevel3(0x00000003)
#define EventLevel30(0x20000000)
#define EventLevel31(0x50000000)
#define EventLevel32(0x80000000)

void ThrowEvents( int scopeIndex )
{
    int eventsHandle = 0;
    int eventsMask = 0;
    double pi = 3.14159;
    int level = 0;

    /* Initialize scopeIndex by calling ScopeInitServer here. */
    /* No need to do above if scopeIndex is already initialized. */
    /* Attach an event buffer. */
    if((eventsHandle = ScopeEventsAttach(-1, scopeIndex)) == 0) {
        printf("Error attaching event buffer to %d!\n",
scopeIndex);
    }

    /* Set the verbosity mask. */
    /* We want events with verbosity 1, 2, 3, 30, 31, and 32. */
    eventsMask = (EventLevel1 | EventLevel2 | EventLevel3 | EventLevel30
| EventLevel31 | EventLevel32);

    ScopeEventsMaskSet(eventsMask, scopeIndex);

    /* Throw events using ScopeEventsCollect and ScopeEventsMessage
calls. */
    level = 2;
    /* This call will succeed since event level 2 is on. */
    ScopeEventsCollect(eventsHandle, level, "Value of pi:", &pi,
"double");

    level = 5;
    /* This call will not succeed since level 5 is off. *.
    ScopeEventsCollect(eventsHandle, level, "Value of pi:", &pi,
"double");

    level = 32;
    /* This call will succeed since level 32 is on. */
    ScopeEventsMessage(eventsHandle, level, "The quick brown fox jumps
over the lazy dog.");
```



```
level = 25;  
/* This call will not succeed since level 25 is off. */  
ScopeEventsMessage(eventsHandle, level, "The lazy dog sleeps on top  
of the quick brown fox.");  
  
/* Detach the event buffer. */  
ScopeEventsDetach(eventsHandle, scopeIndex);  
}
```

This example includes all the StethoScope Events API routines listed at the beginning of this section. It demonstrates how to set up the StethoScope Events API, how to control the collection of both variable values and messages, and how to properly terminate StethoScope Events API prior to exiting the task.

Signals vs. Events

The following will help clarify the relationship between **signals** and **events**:

Collection

Signals are “collected” when **ScopeCollectSignals()** is invoked. Events are “thrown” when either **ScopeCollectEvent()** or **ScopeCollectMessage()** is invoked.

Frequency of Collection

Usually, signals are “collected” periodically. Variables that are persistent (**global/static**) are good candidates for being “collected” at a certain period.

Variables with limited scope in a program (local variables) can be “collected” using the StethoScope Events API. Events are utilized for sporadic collection of ephemeral variables.

Timing

StethoScope assumes that **ScopeCollectSignals()** was invoked at the sampling rate you set during initialization. The GUI plots each “sample” with an inter-sample distance proportional to the sampling rate.

Since there is no “period” associated with the StethoScope Events API, an absolute timestamp is acquired from the target during the time of collection. The StethoScope GUI then uses this timestamp to place the event at the right temporal location on the **Plot** window.

[7.9 Displaying Events](#), p.103 explains how events are plotted. Though signals and events are different concepts altogether, events can be displayed like signals. StethoScope joins the samples (collected using **ScopeEventsCollect()**) with lines, to make them appear like signals. It should be noted, however, that for this display

method to make sense, the same variable should be collected across multiple collection points.

14.8 scope.ini File

StethoScope maintains some global settings in the **scope.ini** file. Most of these settings are maintained by the **Preferences** dialog box (see [3.2.12 Preferences](#), p.36). All default preferences and settings can be restored by deleting **scope.ini** from the same directory where you installed StethoScope.

Example 14-7 Sample scope.ini File

```
; This section contains some global settings for scope
[defaults]
TotalBufferTime=10
SaveWndPos=0
SaveToolbarPos=0
DontAskWriteback=0
DontAskSave=0

; The following sections save the default window positions
; for the various windows
[SigMan]
wpLeft=1047
wpRight=1461
wpTop=546
wpBottom=938
wpFlags=0
wpShowCmd=1

[LogWnd]
wpLeft=962
wpRight=1518
wpTop=97
wpBottom=595
wpFlags=0
wpShowCmd=1

[PlotWindow]
wpLeft=110
wpRight=1310
wpTop=110
wpBottom=952
wpFlags=0
wpShowCmd=1
```

```
[DumpWindow]
wpLeft=303
wpRight=1503
wpTop=233
wpBottom=1094
wpFlags=0
wpShowCmd=1

[PlotXYWindow]
wpLeft=387
wpRight=1087
wpTop=227
wpBottom=927
wpFlags=0
wpShowCmd=1

[MonitorWindow]
wpLeft=263
wpRight=1463
wpTop=204
wpBottom=1065
wpFlags=0
wpShowCmd=1

; The current colors
; The values are in BGR order
[Colors]
Color0=0x0000ff
Color1=0xff0000
Color2=0xc800c8
Color3=0xc8c800
Color4=0x00c800
Color5=0x4040a0
Color6=0x10bbda4
Color7=0x5a5a5a
Color8=0xc0c0c0
Color9=0x92faf5
Color10=0xfffff00
Color11=0x0099ff
Color12=0x33ff99
Color13=0x330066
Color14=0x990000
Color15=0x08aaa2

; The settings saved for the Preferences Window
[defaults-plot]
YOffset=1.500000
YRange=3.000000
Resolution=1
DispAcc=2
MinGrid=50
MaxSnap=20
SnapToSigs=1
MonRes=1.000000
AllowWrite=1
SelectOnCopy=0
```

```
UseSameColors=1  
UnselectLive=1  
DumpRes=1.000000  
DumpHist=1000
```

```
[defaults-plotxy]  
YOffset=1.500000  
YRange=3.000000  
XOffset=-1.500000  
XRange=3.000000  
Resolution=1  
DispAcc=2  
MinGrid=50  
MaxSnap=20  
SnapToSigs=1  
SelectOnCopy=1  
UseSameColors=1  
UnselectLive=1
```

```
[defaults-dump]  
DispAcc=2  
DumpRes=1.000000  
DumpHist=500
```

```
[defaults-monitor]  
DispAcc=4  
MonRes=1.000000  
AllowWrite=1
```

A

StethoScope API Reference

ScopeProbe	- real-time library for StethoScope	209
ScopeActivateMultipleSignals()	- activate multiple signals.....	210
ScopeActivateSignal()	- activate a signal	210
ScopeChangeSampleRate()	- change the sampling rate	211
ScopeCollectSignals()	- collect a sample from each active signal	212
ScopeCollectionModeDisable()	- disable periodic collection	212
ScopeCollectionModeEnable()	- enable periodic collection	213
ScopeCollectionModeGet()	- return the collection mode	213
ScopeDeactivateMultipleSignals()	- deactivate a group of signals	214
ScopeDeactivateSignal()	- deactivate a signal.....	215
ScopeEventsAttach()	- attach an event buffer to a scope index.....	215
ScopeEventsCollect()	- collect the value of a variable on the spot.....	216
ScopeEventsDetach()	- detach an event buffer from a scope index	217
ScopeEventsMaskSet()	- set the events verbosity mask.....	217
ScopeEventsMessage()	- throw an event with the specified message	218
ScopeInitServer()	- initialize a scope index	219
ScopeInitServerEx()	- initialize a scope index	219
ScopeInstallArray()	- register and activate an array of signals	221
ScopeInstallSignal()	- degister and activate a signal	222
ScopeInstallSignalWithOffset()	- register and activate a signal with an offset	223
ScopePrintVersion()	- print the version number of the StethoScope target library	225
ScopeRegisterArray()	- register an array of signals.....	225
ScopeRegisterSignal()	- register a signal.....	227
ScopeRegisterSignalWithOffset()	- register a signal with an offset	228
ScopeRemoveMultipleSignals()	- remove several similarly named signals	230
ScopeRemoveSignal()	- remove a signal.....	230
ScopeSampleDivisorSet()	- set the sample divisor for sub-sampling.....	231
ScopeShowActiveSignals()	- print all signals that are being collected	231
ScopeShowSignals()	- print all registered signals	232
ScopeShutdown()	- shuts down a scope index	232

ScopeTriggerSet()	- set the triggering parameters	233
ScopeTriggerGet()	- return the current trigger parameters.....	234

NOTE: The *scopeIndex* parameter represents the communications “channel” between an instance of StethoScope API running on the target and a StethoScope GUI running on the host. You can create up to 128 instances of StethoScope API on a single target machine, each using a different scope index. The index can range from 0 to 127, and it must be specified when you call **ScopeInitServer()**.

ScopeProbe

NAME	ScopeProbe – real-time library for StethoScope
ROUTINES	<p>ScopeShutdown() – shut down a scope index</p> <p>ScopeInitServerEx() – initialize a scope index</p> <p>ScopeInitServer() – initialize a scope index</p> <p>ScopePrintVersion() – print the version number of the StethoScope target</p> <p>ScopeEventsCollect() – collect the value of a variable on the spot</p> <p>ScopeEventsMessage() – throw events with the specified message</p> <p>ScopeEventsMaskSet() – set the event verbosity mask</p> <p>ScopeEventsAttach() – attach an event buffer to a scope index</p> <p>ScopeEventsDetach() – detach an event buffer from a scope index</p> <p>ScopeCollectSignals() – collect a sample from each active signal</p> <p>ScopeCollectionModeEnable() – enable periodic collection</p> <p>ScopeCollectionModeDisable() – disable periodic collection</p> <p>ScopeCollectionModeGet() – return the collection mode</p> <p>ScopeChangeSampleRate() – change the sampling rate</p> <p>ScopeSampleDivisorSet() – set the sample divisor for sub-sampling</p> <p>ScopeRegisterSignalWithOffset() – register a signal with an offset</p> <p>ScopeRegisterSignal() – register a signal</p> <p>ScopeRemoveSignal() – remove a signal</p> <p>ScopeRemoveMultipleSignals() – remove several similarly-named signals</p> <p>ScopeActivateSignal() – activate a signal</p> <p>ScopeActivateMultipleSignals() – activate multiple signals</p> <p>ScopeDeactivateSignal() – deactivate a signal</p> <p>ScopeDeactivateMultipleSignals() – deactivate a group of signals</p> <p>ScopeInstallSignalWithOffset() – register and activate a signal with an offset</p> <p>ScopeInstallSignal() – register and activate a signal</p> <p>ScopeShowSignals() – print all registered signals</p> <p>ScopeShowActiveSignals() – print all signals that are being collected</p> <p>ScopeRegisterArray() – register an array of signals</p> <p>ScopeInstallArray() – register and activate an array of signals</p> <p>ScopeTriggerSet() – set the triggering parameters</p> <p>ScopeTriggerGet() – return the current trigger parameters</p>

A

DESCRIPTION This library provides a programmatic interface to StethoScope real-time data collection and signal management, facilitating collection of time-histories of variables in your real-time program.

ScopeActivateMultipleSignals()

NAME ScopeActivateMultipleSignals() – activate multiple signals

SYNOPSIS

```
int ScopeActivateMultipleSignals
(
    const char  .namePrefix,
    int         scopeIndex
)
```

DESCRIPTION This function activates signals that have *namePrefix* as prefix. Activated signals can be selected for viewing from one of the many data display windows (such as **Plot**, **Monitor**, and so forth). **ScopeCollectSignals()** only collects samples of activated signals.

A prefix of "." activates all signals.

RETURNS On success, the number of signals that have been activated.

On failure, it returns 0, indicating one of the following:

- the index *scopeIndex* is invalid or has not been initialized
- there is no signal with the *namePrefix* registered with index *scopeIndex*
- *namePrefix* is NULL or invalid

SEE ALSO ScopeProbe(), ScopeInstallSignal(), ScopeInstallSignalWithOffset(), ScopeRegisterSignal(), ScopeRegisterSignalWithOffset(), ScopeActivateSignal(), ScopeDeactivateSignal(), ScopeRemoveSignal(), ScopeRemoveMultipleSignals()

ScopeActivateSignal()

NAME ScopeActivateSignal() – activate a signal

SYNOPSIS

```
RTIBool ScopeActivateSignal
(
    const  char .name,
    int    scopeIndex
)
```


DESCRIPTION	This function activates a signal. Activated signals can be selected for viewing from one of the many data display windows (such as Plot , Monitor , etc). ScopeCollectSignals() only collects samples of activated signals.
RETURNS	On success, this function returns RTL_TRUE , indicating that the signal was activated. On failure, it returns RTL_FALSE , indicating one of the following: <ul style="list-style-type: none">▪ the index <i>scopeIndex</i> is invalid or has not been initialized▪ there is no signal named <i>name</i> registered with index <i>scopeIndex</i>
SEE ALSO	ScopeProbe() , ScopeInstallSignal() , ScopeInstallSignalWithOffset() , ScopeRegisterSignal() , ScopeRegisterSignalWithOffset() , ScopeActivateSignal() , ScopeDeactivateSignal() , ScopeRemoveSignal() , ScopeRemoveMultipleSignals()

ScopeChangeSampleRate()

NAME **ScopeChangeSampleRate()** – change the sampling rate

SYNOPSIS

```
float ScopeChangeSampleRate
(
    float newSampleRate,
    int   scopeIndex
)
```

DESCRIPTION This routine changes the amount of time that StethoScope assumes passed between calls to **ScopeCollectSignals()**. It does not change the actual sampling rate, that is a responsibility of user code. The rate is used to calculate times between samples. If the rate is incorrect, these calculations will be in error.

The parameter should be the frequency in samples per second of the calls to **ScopeCollectSignals()**.

RETURNS On success, this function returns the old sampling rate.
On failure, this function returns 0.0 indicating one of the following:

- *scopeIndex* is invalid or is not initialized
- *rate* is invalid (≤ 0.0)

SEE ALSO **ScopeProbe()**, **ScopeCollectSignals()**, **ScopeInitServer()**

A

ScopeCollectSignals()

NAME ScopeCollectSignals() – collect a sample from each active signal

SYNOPSIS

```
void ScopeCollectSignals
(
    int scopeIndex
)
```

DESCRIPTION This routine should be called periodically to collect the values of signals. StethoScope will assume that this function is called at the frequency set using **ScopeChangeSampleRate()**. However, this only affects the timing calculations made by StethoScope.

If a "sample divisor" is set and is greater than 1, then this routine will return without collecting data when a sample is to be skipped. For instance, with a sample divisor of 3, this routine will only actually collect the data every third time it is called. The other two times, it will simply return immediately.

Furthermore, the behavior of this function depends on triggering. If the trigger is set and is waiting for the start condition to occur (**ARMED**), this function will return without collecting any data. Refer to **ScopeTriggerSet()** for more information.

VxWorks users: the task that calls this routine should have floating point enabled (**VX_FP_TASK** set).

Example code exists under the *src* directory.

SEE ALSO **ScopeProbe()**, **ScopeChangeSampleRate()**, **ScopeCollectionModeEnable()**, **ScopeCollectionModeDisable()**, **ScopeTriggerSet()**

ScopeCollectionModeDisable()

NAME ScopeCollectionModeDisable() – disable periodic collection

SYNOPSIS

```
RTIBool ScopeCollectionModeDisable
(
    int scopeIndex
)
```

DESCRIPTION Use this function to disable periodic collection of active signals. Calling this function sets a flag that is examined by **ScopeCollectSignals()**. If the flag is set, the function returns without sampling active signals. To turn collection on, use **ScopeCollectionModeEnable()**.

RETURNS On success, this function returns **RTI_TRUE**.

On failure, this function returns `RTI_FALSE` if:

- `scopeIndex` is invalid or is not initialized

SEE ALSO `ScopeProbe()`, `ScopeCollectSignals()`, `ScopeCollectionModeEnable()`,
`ScopeCollectionModeGet()`

ScopeCollectionModeEnable()

NAME `ScopeCollectionModeEnable()` – enable periodic collection

SYNOPSIS

```
RTIBool ScopeCollectionModeEnable  
(  
    int scopeIndex  
)
```

DESCRIPTION Use this function to re-enable periodic collection of active signals if it was turned off earlier using `ScopeCollectionModeDisable()`. By default (after calling `ScopeInitServer()`), the collection mode is enabled. Refer to `ScopeCollectionModeDisable()` for more information.

RETURNS On success, this function returns `RTI_TRUE`.
On failure, this function returns `RTI_FALSE` if:

- `scopeIndex` is invalid or is not initialized

SEE ALSO `ScopeProbe()`, `ScopeCollectSignals()`, `ScopeCollectionModeDisable()`,
`ScopeCollectionModeGet()`

ScopeCollectionModeGet()

NAME `ScopeCollectionModeGet()` – return the collection mode

SYNOPSIS

```
int ScopeCollectionModeGet  
(  
    int scopeIndex  
)
```

DESCRIPTION Use this function to determine the current collection mode.

RETURNS On success, this function returns:

- `SCOPE_MODE_ENABLED` if collection mode is enabled
- `SCOPE_MODE_DISABLED` if collection mode is disabled

On failure, this function returns -1 if:

- `scopeIndex` is invalid or is not initialized

SEE ALSO

`ScopeProbe()`, `ScopeCollectSignals()`, `ScopeCollectionModeEnable()`,
`ScopeCollectionModeDisable()`

ScopeDeactivateMultipleSignals()

NAME

`ScopeDeactivateMultipleSignals()` – deactivate a group of signals

SYNOPSIS

```
int ScopeDeactivateMultipleSignals
(
    const char .namePrefix,
    int scopeIndex
)
```

DESCRIPTION

This function will remove all signals from the list of active signals for the selected `scopeIndex` which start with the specified `namePrefix`. Therefore, data will not be collected for these signals during `ScopeCollectSignals()`. A prefix of "." deactivates all signals.

RETURNS

On success, the number of signals that have been deactivated.

On failure, this function returns 0, indicating one of the following:

- the index `scopeIndex` is invalid or has not been initialized
- there are no active signals with the `namePrefix` registered with index `scopeIndex`
- `namePrefix` is invalid

SEE ALSO

`ScopeProbe()`, `ScopeActivateMultipleSignals()`

ScopeDeactivateSignal()

NAME	ScopeDeactivateSignal() – deactivate a signal
SYNOPSIS	<pre>RTIBool ScopeDeactivateSignal (const char .name, int scopeIndex)</pre>
DESCRIPTION	This function will remove a signal from the list of activate signals. Therefore, data will not be collected for this signal during ScopeCollectSignals() .
RETURNS	On success, this function returns RTI_TRUE , indicating that the signal was deactivated. On failure, this function returns RTI_FALSE , indicating one of the following: <ul style="list-style-type: none">▪ the index <i>scopeIndex</i> is invalid or has not been initialize.▪ there is no active signal named <i>name</i> registered with index <i>scopeIndex</i>
SEE ALSO	ScopeProbe() , ScopeActivateSignal()

ScopeEventsAttach()

NAME	ScopeEventsAttach() – attach an event buffer to a scope index
SYNOPSIS	<pre>int ScopeEventsAttach (int eventBufferSize, int scopeIndex)</pre>
DESCRIPTION	This function attaches an event buffer to a scope index. The eventsHandle returned by this function should be used to <i>throw</i> events. There is a maximum limit of four event buffers that can attach to an index. Throwing events into the same buffer from multiple threads is not recommended.
RETURNS	On success, this function returns a non-zero eventsHandle . On failure, it returns 0 if: <ul style="list-style-type: none">▪ <i>scopeIndex</i> is invalid or is not initialized. The maximum number of event buffers are already attached.

SEE ALSO [ScopeProbe\(\)](#), [ScopeInitServer\(\)](#), [ScopeEventsDetach\(\)](#)

ScopeEventsCollect()

NAME `ScopeEventsCollect()` – collect the value of a variable on the spot

SYNOPSIS

```
void ScopeEventsCollect
(
    int          eventsHandle,
    int          level,
    const char.  eventId,
    void         ptrToVar,
    RTIAAtomicTypeId typeId
)
```

PARAMETERS

eventsHandle
The handle returned by [ScopeEventsAttach\(\)](#).

level
The verbosity level of this event. Range: [1 - 32].

eventId
A *string* that describes this event.

ptrToVar
Pointer to the variable to collect.

typeId
Type identifier for the variable. Use:

- `RTI_INT8ID` for char, unsigned char,
- `RTI_INT16ID` for short, unsigned short,
- `RTI_INT32ID` for int, unsigned int, long, unsigned long,
- `RTI_FLOAT32ID` for float, and
- `RTI_DOUBLE64ID` for double.

DESCRIPTION This function collects the value of one variable on the spot. The variable does not have to be static/global as required by [ScopeCollectSignals\(\)](#) and there is no need to perform registration/activation. This function is a low-overhead equivalent of `printf()` for use in debugging realtime systems.

`ScopeEventsCollect()` is actually a macro that calls the collection function `ScopeEventsCollectInternal()` only if the verbosity level of this event is turned on. Thus, there is no overhead of a function call if the verbosity level of this event is turned off. The verbosity mask can be set using [ScopeEventsMaskSet\(\)](#).

NOTE: Calling this function from multiple threads with the same *eventsHandle* is not recommended. Therefore, use a separate *eventsHandle* (returned by **ScopeEventsAttach()**) for each thread, which calls this function.

SEE ALSO [ScopeProbe\(\)](#), [ScopeEventsMessage\(\)](#), [ScopeEventsAttach\(\)](#), [ScopeEventsMaskSet\(\)](#)

ScopeEventsDetach()

NAME **ScopeEventsDetach()** – detach an event buffer from a scope index

SYNOPSIS

```
RTIBool ScopeEventsDetach
(
    int eventsHandle
)
```

DESCRIPTION *eventsHandle* is the handle returned by **ScopeEventsAttach()**. This function detaches an event buffer from an index.

RETURNS On success, this function returns **RTI_TRUE**.

On failure, it returns **RTI_FALSE** if:

- *eventsHandle* is invalid

SEE ALSO [ScopeProbe\(\)](#), [ScopeInitServer\(\)](#), [ScopeEventsAttach\(\)](#)

ScopeEventsMaskSet()

NAME **ScopeEventsMaskSet()** – set the events verbosity mask

SYNOPSIS

```
RTIBool ScopeEventsMaskSet
(
    int eventsMask,
    int scopeIndex
)
```

DESCRIPTION StethoScope Events API allows for 32 levels of verbosity. This function sets the mask for the specified scope index. Each bit in mask corresponds to one verbosity level. Setting the mask to 0x00000001 turns off all levels except level 1 messages. The mask can be set anytime

during the executing of the program and take effect immediately. By default, all levels are turned on for a scope index.

- RETURNS** On success, this function returns **RTI_TRUE**.
On failure, this function returns **RTI_FALSE** if:
- *scopeIndex* is invalid or is not initialized

SEE ALSO **ScopeProbe()**, **ScopeEventsCollect()**, **ScopeEventsMessage()**, **ScopeEventsAttach()**

ScopeEventsMessage()

NAME **ScopeEventsMessage()** – throw an event with the specified message

SYNOPSIS

```
void ScopeEventsMessage
(
    int eventsHandle,
    int level,
    const char message
)
```

PARAMETERS *eventsHandle*
The handle returned by **ScopeEventsAttach()**.

level
The verbosity level of this event. Range: [1 - 32].

message
Message string.

DESCRIPTION This function throws an event with the specified message string. This function is a low-overhead equivalent of **printf()** for use in debugging real-time systems.

ScopeEventsMessage() is actually a macro that calls the collection function **ScopeEventsMessageInternal()** only if the verbosity level of this event is turned on. Thus, there is no overhead of a function call if the verbosity level of this event is turned off. The verbosity mask can be set using **ScopeEventsMaskSet()**.

NOTE: Calling this function from multiple threads with the same *eventsHandle* is not recommended. Therefore, use a separate *eventsHandle* (returned by **ScopeEventsAttach()**) for each thread, which calls this function.

SEE ALSO **ScopeProbe()**, **ScopeEventsCollect()**, **ScopeEventsAttach()**, **ScopeEventsMaskSet()**

ScopeInitServer()

NAME	ScopeInitServer() – initialize a scope index
SYNOPSIS	<pre>int ScopeInitServer (int sampleBufferSize, int signalBufferSize, int debugLevel, int scopeIndex)</pre>
DESCRIPTION	Calls ScopeInitServerEx() with default priorities for scopeprobe and scopelink daemon threads. Refer to ScopeInitServerEx() for details.
RETURNS	On success, this function returns the initialized scope index. On failure, this function returns a negative number if: <ul style="list-style-type: none">▪ <i>scopeIndex</i> is out of range▪ all indices are occupied▪ memory allocation failed: it failed to create daemon threads (which will happen if the threads failed to bind to TCP ports, or if the priorities specified are not valid on the target operating system)
SEE ALSO	ScopeProbe() , ScopeInitServerEx()

A

ScopeInitServerEx()

NAME	ScopeInitServerEx() – initialize a scope index
SYNOPSIS	<pre>int ScopeInitServerEx (int sampleBufferSize, int signalBufferSize, int debugLevel, int probeDaemonPriority, int linkDaemonPriority, int scopeIndex)</pre>
DESCRIPTION	Initializes the scope index <i>scopeIndex</i> . If <i>scopeIndex</i> is -1, then this function initializes the next uninitialized scope index. This should be called once for each index (on VxWorks, normally at boot time). Calling it multiple times for the same index is harmless though.

The *sampleBufferSize* parameter specifies the size of the target data buffer in bytes. This buffer is used to store the data samples for the active signals. Data samples for all types other than "double" are saved as 4-byte values. Data samples for doubles are saved as 8-byte values. Thus, the maximum number of samples that can fit in the buffer will range between $((\text{sampleBufferSize} / 8) - 1) / \text{numberOfSignals}$ and $((\text{sampleBufferSize} / 4) - 1) / \text{numberOfSignals}$, rounded down to the nearest integer.

If *sampleBufferSize* ≤ 0 , it defaults to (32.1024). Otherwise, if it is less than 1024, a value of 1024 is used instead.

The *signalBufSize* parameter specifies the size of the target signal buffer in bytes. This buffer is used to store the information specific to each signal (such as the name, units and type). The space taken by a signal gets reclaimed (for registering other signals) when the signal is removed. The information stored for a signal takes up 28 bytes plus the number of bytes (including the terminating null character) it takes to store the signal name and units. Note that a signal registered twice, under different scope indices, counts as two registered signals.

If *signalBufferSize* ≤ 0 , it defaults to (32.1024). If *signalBufferSize* > 0 and < 1024 it defaults to 1024.

The *scopeprobeDaemonPriority* and *scopelinkDaemonPriority* parameters specify the scheduling priority levels for the **scopeprobe** and **scopelink** daemon threads that are spawned by this function. Refer to the documentation on threads and scheduling for the operating system that you are using, for valid priority levels. If you do not want to specify a priority level, you can pass a nonpositive value for this parameter. In that case, suitable priority levels will automatically be chosen.

Normally, for each scope index, a target spawns real-time daemons (**scopeprobe** and **scopelink**) which communicate to the host over the network, using TCP/IP. If you are using Tornado, and wish to use the Tornado WTX protocol instead, you must load the StethoScope WTX target library (**libscopewtx.so**). In that case, the daemons will use the WTX protocol to communicate even if IP is available. Note that the protocol is called "WTX", even though the target is talking to the WDB daemon.

RETURNS

On success, this function returns the initialized scope index.

On failure, this function returns a negative number if:

- *scopeIndex* is out of range
- all indices are occupied
- memory allocation failed: it failed to create daemon threads (which will happen if the threads failed to bind to TCP ports, or if the priorities specified are not valid on the target operating system)

SEE ALSO

ScopeProbe(), **ScopeShutdown()**, **ScopeEventsAttach()**, **ScopeEventsDetach()**

ScopeInstallArray()

NAME `ScopeInstallArray()` – register and activate an array of signals

SYNOPSIS

```
int ScopeInstallArray
(
    const char .name,
    int elementsInArray,
    const char .units,
    void .ptrToStaticArray,
    const char .type,
    int scopeIndex
)
```

PARAMETERS

name

A unique name to identify the array.

elementsInArray

The number of elements to install.

units

User specified unit of measurement, for identification.

ptrToStaticArray

Must be a pointer to a static (or global) storage location.

type

Must be one of the accepted types such as:

- *double*
- *float*
- *int*, etc.

See **ScopeProbe()** for details on types.

DESCRIPTION

This function calls **ScopeRegisterArray()** followed by **ScopeActivateMultipleSignals()**. It exists purely for convenience. Users should refer to **ScopeRegisterArray()** and **ScopeActivateMultipleSignals()** for details.

EXAMPLES

```
float global_array[100];
int main( )
{
    static double static_array[50];

    // Initialize ScopeIndex here. See ScopeInitServer( ).

    ScopeInstallArray("my_global_array1", 100, "none", &global_array,
        "float", ScopeIndex);
    ScopeInstallArray("my_static_array1", 50, "none", &static_array,
        "double", ScopeIndex);
```

```
while ( ) {  
    ScopeCollectSignals( );  
}  
// Shutdown ScopeIndex. See ScopeShutdown( ).  
}
```

RETURNS On success, this function returns the numbers of array elements successfully installed.

On failure, it returns zero, indicating one of the following:

- the index (*scopeIndex*) is invalid or has not been initialized
- the type specified (*type*) is not recognized
- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO [ScopeProbe\(\)](#), [ScopeRegisterArray\(\)](#), [ScopeActivateMultipleSignals\(\)](#),
[ScopeDeactivateMultipleSignals\(\)](#), [ScopeInstallSignal\(\)](#)

ScopeInstallSignal()

NAME [ScopeInstallSignal\(\)](#) – register and activate a signal

SYNOPSIS

```
RTIBool ScopeInstallSignal  
(  
    const char .name,  
    const char .units,  
    void .ptrToStaticVar,  
    const char .type,  
    int scopeIndex  
)
```

DESCRIPTION This function calls [ScopeRegisterSignal\(\)](#) followed by [ScopeActivateSignal\(\)](#). It exists purely for backwards compatibility and convenience. Users should refer to [ScopeRegisterSignal\(\)](#) and [ScopeActivateSignal\(\)](#) for details.

EXAMPLES

```
float global_var;  
float .global_ptr;  
  
int main( )  
{  
    static double static_var;  
    static double .static_ptr;  
  
    // Initialize ScopeIndex here. See ScopeInitServer( ).  
    ScopeInstallSignal("my_global_var1", "none", &global_var, "float",  
                      ScopeIndex);  
    ScopeInstallSignal("my_static_var1", "none", &static_var,  
                      "double", ScopeIndex);
```

```
global_ptr = (float .) calloc(1, sizeof(global_ptr));
static_ptr = (double .) calloc(1, sizeof(static_ptr));
ScopeInstallSignal("my_global_var2", "none", global_ptr, "double",
                  ScopeIndex);
ScopeInstallSignal("my_static_var2", "none", static_ptr, "double",
                  ScopeIndex);
while ( ) {
    ScopeCollectSignals( );
}
// Shutdown ScopeIndex. See ScopeShutdown( ).
}
```

RETURNS On success, this function returns `RTL_TRUE`, indicating that the signal was installed.

On failure, it returns `RTL_FALSE`, indicating one of the following:

- the index (*scopeIndex*) is invalid or has not been initialized
- the type specified (*type*) is not recognized
- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO `ScopeProbe()`, `ScopeRegisterSignal()`, `ScopeActivateSignal()`, `ScopeDeactivateSignal()`, `ScopeRemoveSignal()`, `ScopeRemoveMultipleSignals()`

ScopeInstallSignalWithOffset()

A

NAME `ScopeInstallSignalWithOffset()` – register and activate a signal with an offset

SYNOPSIS

```
RTIBool ScopeInstallSignalWithOffset
(
    const char .name,
    const char
    .units, void .ptrToStaticVar,
    const char .type, int offset,
    int scopeIndex
)
```

DESCRIPTION Register and activate a signal with an offset.

PARAMETERS

name
A unique name to identify the signal.

units
User specified unit of measurement, for identification.

ptrToStaticVar
Must point to a static (or global) storage location.

type

Refers to the type of the variable at the given *offset*.

offset

Number of bytes from *ptrToStaticVar* to collect data from.

DESCRIPTION This function calls **ScopeRegisterSignalWithOffset()** followed by **ScopeActivateSignal()**. It exists purely for convenience. Users should refer to **ScopeRegisterSignalWithOffset()** and **ScopeActivateSignal()** for details.

EXAMPLES The following illustrates how to use **ScopeInstallSignalWithOffset()**:

```
typedef struct ArmData_s {
    float PosX;
    float PosY;
    unsigned char Type;
    double .Vel; // Just to demonstrate pointers and offsets.
} ArmData_t;

int main ( )
{
    ArmData_t LeftArmData = (ArmData_t .)calloc(1, sizeof( ));
    LeftArmData->Vel = (double .)calloc(1, sizeof( ));

    // Initialize ScopeIndex here. See ScopeInitServer( ).
    // Notice that you can pass just the address of LeftArmData,
    // not the individual fields.
    // The type "float" refers to the type of LeftArmData->PosX.
    ScopeInstallSignalWithOffset("LeftArm/PosX", "meters",
                                &LeftArmData, "float",
                                offsetof(ArmData_t, PosX), 0);

    ScopeInstallSignalWithOffset("LeftArm/PosY", "meters",
                                &LeftArmData, "float",
                                offsetof(ArmData_t, PosY), 0);

    ScopeInstallSignalWithOffset("LeftArm/Type", "meters",
                                &LeftArmData, "uchar",
                                offsetof(ArmData_t, Type), 0);

    // The type "double ." refers to the type of the member Vel.
    ScopeInstallSignalWithOffset("LeftArm/Vel", "meters", &LeftArmData,
                                "double .", offsetof(ArmData_t, Vel), 0);

    while ( ) {
        // Set the data in the member LeftArmData.
        ScopeCollectSignals( );
    }

    // Shutdown ScopeIndex. See ScopeShutdown( ).
}
```

RETURNS On success, this function returns **RTI_TRUE**, indicating that the signal was installed.

On failure, it returns `RTL_FALSE`, indicating one of the following:

- the index (*scopeIndex*) is invalid or has not been initialized
- the type specified (*type*) is not recognized
- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO `ScopeProbe()`, `ScopeRegisterSignal()`, `ScopeActivateSignal()`, `ScopeDeactivateSignal()`, `ScopeRemoveSignal()`, `ScopeRemoveMultipleSignals()`

ScopePrintVersion()

NAME `ScopePrintVersion()` – print the version number of the StethoScope target library

SYNOPSIS `void ScopePrintVersion(void)`

DESCRIPTION Print the version number of the StethoScope target library.

SEE ALSO `ScopeProbe()`, `ScopeProbe()`

A

ScopeRegisterArray()

NAME `ScopeRegisterArray()` – register an array of signals

SYNOPSIS

```
int ScopeRegisterArray
(
    const char .name,
    int elementsInArray,
    const char .units,
    void .ptrToStaticArray,
    const char .type,
    int scopeIndex
)
```

DESCRIPTION This function registers an array (one dimensional) of signals by calling `ScopeRegisterSignal()` for each element in the array. An array modifier “[]” is appended to the signal name followed by the index of the element.

A valid signal name should not contain asterisks or blank spaces. Invalid characters in signal name (*name*) would be automatically replaced with underscores.

It is the caller's responsibility to ensure that the address passed as *ptrToStaticArray* points to a valid memory location. In particular, a bad address may cause a **Bus Error** or **Segmentation Fault** to occur within **ScopeCollectSignals()**.

EXAMPLES

```
float global_array[100];

int main( )
{
    static double static_array[50];

    // Initialize ScopeIndex here. See ScopeInitServer( ).
    ScopeRegisterArray("my_global_array1", 100, "none", &global_array,
                      "float", ScopeIndex);
    ScopeRegisterArray("my_static_array1", 50, "none", &static_array,
                      "double", ScopeIndex);

    ScopeActivateMultipleSignals("my_global_array1", ScopeIndex);
    ScopeActivateMultipleSignals("my_static_array1", ScopeIndex);

    while ( ) {
        ScopeCollectSignals( );
    }
    // Shutdown ScopeIndex. See ScopeShutdown( ).
}
```

RETURNS

On success, this function returns the numbers of array elements successfully registered.

On failure, it returns zero, indicating one of the following:

- the index *scopeIndex* is invalid or has not been initialized
- the type specified *type* is not recognized
- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO

ScopeProbe(), **ScopeRegisterSignal()**, **ScopeInstallArray()**,
ScopeActivateMultipleSignals(), **ScopeDeactivateMultipleSignals()**

ScopeRegisterSignal()

NAME `ScopeRegisterSignal()` – register a signal

SYNOPSIS

```
RTIBool ScopeRegisterSignal
(
    const char .name,
    const char .units,
    void .ptrToStaticVar,
    const char .type,
    int scopeIndex
)
```

DESCRIPTION This function registers a *signal*. A registered signal is one that is made known to scope’s signal manager, but not to any other window, for example, the **Plot** window. It has the possibility of becoming *activated* through the signal manager or through the target function **ScopeActivateSignal()**. Activated signals can be selected for display from one of the many data display windows (for example, **Plot**, **Monitor**, and so forth).

A signal is any variable in the program, with the caveat that it must have a valid value at the instant that **ScopeCollectSignals()** is called. Thus, most anything can be installed as a signal, with the exception of automatic stack variables whose scope does not include the **ScopeCollectSignals()** call.

A valid signal name should not contain asterisks or blank spaces. Invalid characters in signal name *name* would be automatically replaced with underscores.

It is the caller’s responsibility to ensure that the address passed *ptrToStaticVar* points to a valid memory location. In particular, a bad address may cause a **Bus Error** or **Segmentation Fault** to occur within **ScopeCollectSignals()**.

EXAMPLES

```
float global_var;
float .global_ptr;

int main( )
{
    static double static_var;
    static double .static_ptr;

    // Initialize ScopeIndex here. See ScopeInitServer( ).

    ScopeRegisterSignal("my_global_var1", "none", &global_var,
        "float", ScopeIndex);
    ScopeRegisterSignal("my_static_var1", "none", &static_var,
        "double", ScopeIndex);

    global_ptr = (float .) calloc(1, sizeof(.global_ptr));
    static_ptr = (double .) calloc(1, sizeof(.static_ptr));
```

A

```
ScopeRegisterSignal("my_global_var2", "none", global_ptr,  
                    "double", ScopeIndex);  
ScopeRegisterSignal("my_static_var2", "none", static_ptr,  
                    "double", ScopeIndex);  
  
// Activate the registered signals here. See ScopeActivateSignal( ).  
  
while ( ) {  
    ScopeCollectSignals( );  
}  
  
// Shutdown ScopeIndex. See ScopeShutdown( ).  
}
```

RETURNS On success, this function returns **RTI_TRUE**, indicating that the signal was registered.

On failure, it returns **RTI_FALSE**, indicating one of the following:

- the index *scopeIndex* is invalid or has not been initialized
- the type specified *type* is not recognized
- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO [ScopeProbe\(\)](#), [ScopeActivateSignal\(\)](#), [ScopeDeactivateSignal\(\)](#), [ScopeInstallSignal\(\)](#), [ScopeRegisterSignalWithOffset\(\)](#)

ScopeRegisterSignalWithOffset()

NAME [ScopeRegisterSignalWithOffset\(\)](#) – register a signal with an offset

SYNOPSIS

```
RTIBool ScopeRegisterSignalWithOffset  
(  
    const char .name,  
    const char .units,  
    void .ptrToStaticVar,  
    const char .type,  
    int offset,  
    int scopeIndex  
)
```

DESCRIPTION This function has the same functionality as [ScopeRegisterSignal\(\)](#), except callers can specify an offset from *ptrToStaticVar*. This offset refers to a variable in a structure that a caller wants to sample. However, the variable's address may not be known at register time, hence the offset.

A valid signal name should not contain asterisks or blank spaces. Invalid characters in signal name *name* would be automatically replaced with underscores.

It is the caller's responsibility to ensure that the address and offset parameters point to a valid memory location. In particular, a bad address may cause a **Bus Error** or **Segmentation Fault** during **ScopeCollectSignals()**.

Ideally, this function would be used to sample variables in an array of structures. The caller would register a structure pointer with an offset to a variable they want sampled. Therefore, by changing the pointer callers can sample the same variable at different locations in the array, see below for examples.

EXAMPLES

```
// An example structure.
typedef struct TestData_s {
    float field1;
    double field2;
} TestData_t;

// Here is an array that is read in.
#define MAXLENGTH 100
TestData_t TestDataArray[MAXLENGTH];
TestData_t .TestDataPtr = &TestDataArray[0];

int main( )
{
    int i = 0;

    // Initialize ScopeIndex here. See ScopeInitServer( ).

    // Notice we are using the new hierarchical naming feature.
    ScopeRegisterSignalWithOffset("test/field1", "volts",
                                &TestDataPtr, "float",
                                offsetof(TestData_t, field1),
                                ScopeIndex);
    ScopeRegisterSignalWithOffset("test/field2", "volts",
                                &TestDataPtr, "double",
                                offsetof(TestData_t, field2),
                                ScopeIndex);

    // Activate the registered signals here. See ScopeActivateSignal( ).

    for (i = 0; i < MAXLENGTH; i++) {
        // Read in TestDataArray[i].
        // Set TestDataPtr to the new data.
        TestDataPtr = &TestDataArray[i];
        ScopeCollectSignals( );
    }

    // Shutdown ScopeIndex. See ScopeShutdown( ).
}
```

RETURNS

On success, this function returns **RTI_TRUE**, indicating that the signal was registered.

On failure, it returns **RTI_FALSE**, indicating one of the following:

- the index *scopeIndex* is invalid or has not been initialized
- the type specified *type* is not recognized

- the parameters passed are invalid
- there is no more space in the signal buffer

SEE ALSO `ScopeProbe()`, `ScopeRegisterSignal()`, `ScopeActivateSignal()`, `ScopeInstallSignal()`, `ScopeInstallSignalWithOffset()`, `ScopeRemoveSignal()`, `ScopeRemoveMultipleSignals()`, `ScopeDeactivateSignal()`

ScopeRemoveMultipleSignals()

NAME `ScopeRemoveMultipleSignals()` – remove several similarly named signals

SYNOPSIS

```
int ScopeRemoveMultipleSignals
(
    const char .namePrefix,
    int scopeIndex
)
```

DESCRIPTION This function removes a set of installed signals that have *namePrefix* as prefix. If *namePrefix* is ".", all signals will be removed.

RETURNS On success, returns the number of signals removed.

On failure, this function returns 0, indicating one of the following:

- *namePrefix* is NULL
- there is no signal named with prefix *namePrefix* registered with index *scopeIndex*
- the index *scopeIndex* is out of range or if index is not initialized

SEE ALSO `ScopeProbe()`, `ScopeRemoveSignal()`

ScopeRemoveSignal()

NAME `ScopeRemoveSignal()` – remove a signal

SYNOPSIS

```
RTIBool ScopeRemoveSignal
(
    const char .name,
    int scopeIndex
)
```

DESCRIPTION	This function deactivates and unregisters a signal. This will invalidate any partially filled collection buffer. After this call users can not activate this signal.
RETURNS	On success, this function returns <code>RTL_TRUE</code> , indicating that the installed signal was removed. On failure, it returns <code>RTL_FALSE</code> , indicating one of the following: <ul style="list-style-type: none">▪ the index <i>scopeIndex</i> is invalid or has not been initialized▪ there is no signal named <i>name</i> registered with index <i>scopeIndex</i>
SEE ALSO	<code>ScopeProbe()</code> , <code>ScopeInstallSignal()</code> , <code>ScopeInstallSignalWithOffset()</code> , <code>ScopeRegisterSignal()</code>

ScopeSampleDivisorSet()

NAME	<code>ScopeSampleDivisorSet()</code> – set the sample divisor for sub-sampling
SYNOPSIS	<pre>int ScopeSampleDivisorSet (int newSampleDivisor, int scopeIndex)</pre>
DESCRIPTION	This function sets the sample divisor for this index. From then on, <code>ScopeCollectSignals()</code> will succeed only once in <i>newSampleDivisor</i> number of times it is invoked.
RETURNS	On success, this function returns the old sample divisor. On failure, this function returns 0 indicating one of the following: <ul style="list-style-type: none">▪ <i>scopeIndex</i> is invalid or is not initialized▪ <i>newSampleDivisor</i> is invalid (Range: 1 - 10,000)
SEE ALSO	<code>ScopeProbe()</code> , <code>ScopeCollectSignals()</code> , <code>ScopeInitServer()</code>

ScopeShowActiveSignals()

NAME	<code>ScopeShowActiveSignals()</code> – print all signals that are being collected
SYNOPSIS	<pre>void ScopeShowActiveSignals(int scopeIndex)</pre>

- DESCRIPTION** This function prints a formatted list of all currently installed signals to the standard output. The current signal value is also printed.
- SEE ALSO** `ScopeProbe()`, `ScopeInstallSignal()`, `ScopeInstallSignalWithOffset()`, `ScopeRegisterSignal()`, `ScopeRegisterSignalWithOffset()`

ScopeShowSignals()

- NAME** `ScopeShowSignals()` – print all registered signals
- SYNOPSIS**

```
void ScopeShowSignals( int scopeIndex )
```
- DESCRIPTION** This function prints a formatted list of all currently registered signals to the standard output. The current signal value is also printed.
- SEE ALSO** `ScopeProbe()`, `ScopeInstallSignal()`, `ScopeInstallSignalWithOffset()`, `ScopeRegisterSignal()`, `ScopeRegisterSignalWithOffset()`

ScopeShutdown()

- NAME** `ScopeShutdown()` – shuts down a scope index
- SYNOPSIS**

```
RTIBool ScopeShutdown  
(  
    int scopeIndex  
)
```
- DESCRIPTION** This function shuts down the scope index *scopeIndex*. Event buffers, if any, associated with this index are also shutdown.
- RETURNS** On success, this function returns `RTI_TRUE`.
On failure, this function returns `RTI_FALSE` if:
- *scopeIndex* is invalid or not initialized
- SEE ALSO** `ScopeProbe()`, `ScopeInitServer()`, `ScopeInitServerEx()`, `ScopeEventsAttach()`, `ScopeEventsDetach()`

ScopeTriggerSet()

NAME `ScopeTriggerSet()` – set the triggering parameters

SYNOPSIS

```
RTIBool ScopeTriggerSet
(
    ScopeTriggerInfoPtr pTriggerStart,
    ScopeTriggerInfoPtr pTriggerStop,
    RTIBool triggerRearm,
    int scopeIndex
)
```

DESCRIPTION This function sets up a trigger. Triggering is a way to control when and for how long periodic data (collected using `ScopeCollectSignals()`) is collected. A trigger consists of a start condition and a stop condition. After the trigger is armed using `ScopeTriggerSet()`, all calls to `ScopeCollectSignals()` will return without collecting data. Data collection will resume only after the specified start condition is met. The start condition can be based on a signal or an event or a trigger can be set with no start condition (trigger immediately option). The stop condition can be based on a signal or an event. Stop condition for a trigger based on a set time period is available only from the StethoScope GUI.

The trigger will expire as soon as the stop condition is met. Data collection will continue as before after the trigger expires. If the *triggerRearm* flag is set, the trigger will again be set with the values passed.

To set a trigger, fill out the following fields in the `ScopeTriggerInfo()` structure:

source

The source of the condition. Can be one of:

- `SCOPE_TRIG_SRC_SIGNAL` (triggers on a signal)
- `SCOPE_TRIG_SRC_EVENT` (triggers on an event)
- `SCOPE_TRIG_SRC_NOW` (triggers immediately)

slope

The slope of the signal as it passes the level in order to trigger:

- `SCOPE_TRIG_SLOPE_POS` (positive slope)
- `SCOPE_TRIG_SLOPE_NEG` (negative slope)
- `SCOPE_TRIG_SLOPE_ANY` (any slope)

This parameter is relevant only if the source is a signal.

level

A value that the signal must reach in order to trigger. This parameter is relevant only if the source is a signal.

signal

The name of the signal to trigger on.

eventId

The event to trigger on.

To set a trigger based on a signal, fill out *source*, *slope*, *level*, and *signal*. To set a trigger based on an event, fill out the *eventId*. To set the trigger to start immediately, fill out *source* (set it to `SCOPE_TRIG_SRC_NOW`).

For all the trigger modes listed above, pass in a non-NULL value to parameters that are not used in that mode.

To disable the trigger, pass NULL for `pTriggerStart` and `pTriggerStop`.

RETURNS On success, this function returns `RTI_TRUE`, indicating that the trigger was set.

On failure, this function returns `RTI_FALSE` if:

- *scopeIndex* is invalid or is not initialized

SEE ALSO `ScopeProbe()`, `ScopeTriggerGet()`, `ScopeCollectSignals()`

ScopeTriggerGet()

NAME `ScopeTriggerGet()` – return the current trigger parameters

SYNOPSIS

```
RTIBool ScopeTriggerGet
(
    int .pTriggerMode,
    int .pTriggerRearm,
    ScopeTriggerInfoPtr pTriggerStart,
    ScopeTriggerInfoPtr pTriggerStop,
    int scopeIndex
)
```

DESCRIPTION This function returns the current trigger parameters.

If `pTriggerMode` returns:

- `SCOPE_TRIG_MODE_DISABLED`, then there is no trigger set for that index.
- `SCOPE_TRIG_MODE_ARMED`, there is a trigger set for that index and is waiting for the start condition to occur. This also means that no data is being collected by `ScopeCollectSignals()`.
- `SCOPE_TRIG_MODE_TRIGGERING`, there is a trigger set for that index and is waiting for the stop condition to occur.

RETURNS On success, this function returns `RTI_TRUE`.

On failure, this function returns `RTI_FALSE` if:

- `scopeIndex` is invalid or has not been initialized

SEE ALSO `ScopeProbe()`, `ScopeTriggerSet()`, `ScopeCollectSignals()`

B

StethoScope Demonstration

[B.1 Introduction](#) 237

[B.2 Source-code Example: vxdemo.c](#) 237

B.1 Introduction

This appendix contains a listing of the **vxdemo.c** demonstration source code file(s) provided with the Wind River StethoScope distribution. This includes example source code for an application program that installs signals to StethoScope. The file and its makefile are located at:

```
WIND_SCOPETOOLS\target\src\scopedemo
```

where **WIND_SCOPETOOLS** is the directory where you installed the Wind River ScopeTools.

B.2 Source-code Example: vxdemo.c

The file, **vxdemo.c**, is an example of a simple application that uses StethoScope. The example application plots a number of sinusoids and contains a simple control

system with a simulated physical system that is noisy. It is meant only to illustrate some of StethoScope's features.

The `\target\src\scopedemo` directory contains the `vxdemo.c` demo code file (see [B.2.1 Source Code for vxdemo.c](#), p.238), and a makefile (see [B.2.2 Makefile for vxdemo.c](#), p.246), to help you recompile the demonstration program.



NOTE: You will need to edit the **makefile** to have it reflect your system's file structure.

B.2.1 Source Code for vxdemo.c

```
\- A demo for StethoScope. */

/*
modification history
-----
5.4a,25sep00,ss Modified ln. 165 to properly support random numbers
in VxWorks versions < 5.1
5.4a,18sep00,gah added support for protection domain testing
5.4a,17jun00,vwc Better cleanup of memory by doing semDelete().
Also use flag to cause main task to terminate rather
than taskDelete()
5.4a,12jun00,vwc Add hook routine to be run in sample loop. Install
Scope's debug signals by default.
5.3c,30mar00,vwc Added param to ScopeDemo to set data-buf sz
5.3a,09jun99,laf Changed buffer sizes - we have 17 variables, not 16,
don't need quite as much memory for either buffer.
If ScopeInitServer fails (usually due to lack of
target memory), don't spawn the other tasks.
5.1f,17may99,laf Double data buffer size
5.1e,15apr99,sda Changed ordering of statements in Shutdown to safer.
5.1e,02apr99,laf Provide Shutdown functionality to clean up.
5.1a,26oct98,nm ScopeInitServer() now takes two buffer sizes.
5.1a,16oct98,nm Set WtxOverride to 1, not to true.
5.1a,12aug98,nm Updated for scope 5.1 release.
RTI,20dec92,sas Added ScopeIndex. Removed support for VxWorks 4.x
RTI,25nov92,sas Ported to VxWorks 5.1
RTI,06may92,sas Added rebootHookAdd to insure reboot success.
Made VxWorks 5.0 the default.
RTI,03apr92,sas Converted to mangeln format.
RTI,07nov91,sas Added plant simulation.
RTI,01sep89,sas written.
*/
```

/*

DESCRIPTION:

This file contains code to start a simple synchronous sampler, and install and de-install a few demo signals. It also contains a simple double-integrator plant simulator.

To run the demo (this assumes you have already loaded StethoScope):

```
rlogin target
cd "scopedir/lib/m68kVx5.1"
ld 1 < vxdemo.lo
ScopeDemo
```

To recompile:

```
cc68k -I/local/VxWorks/h -Iscopedir/include -c vxdemo.c
```

Useful things to play with:

```
Omega = (float) 80;
ScopeRemoveMultipleSignals("sin");
InstallFakeSignals(40)
ScopeDemoSetRate(20.0)
sis("phase[3]")
sis("Time", "none", &Time, "float");
```

A makefile is provided; it can be used to compile this code. You should first edit it to reflect your system's file structure.

This is a complete VxWorks application. To utilize StethoScope in your system, all you need is a call to ScopeInitServer in your startup script, a call to ScopeCollectSignals somewhere in your regular processing cycle, and a few calls to ScopeInstallSignal.

WARNING:

With 64 signals installed, the calculation of the sin functions below (FakeSignals()) can only be done at about 200 Hz (on a 16MHz 68020). Thus, setting the sample rate higher than this will starve both tasks tScopeProbeDaemon and tScopeLinkDaemon; Scope will not be able to display any data. This is intentional---Scope always strives for minimal impact on the real-time system.

NOTES:

This code normally works using taskDelay. It can also work by attaching a semaphore (sampleSemaphore) to the Aux clock interrupt, the recommended means of executing periodic functions in VxWorks. If your processor does have an aux-clock, set the "useAuxClock" parameter to 1.

SEE ALSO:

```
ScopeProbe(2), scope(1), vxdemo2(3)
```

```
----- */
/* $Id: vxdemo.c,v 1.1 2003/03/27 17:52:12 Exp $ */
```

```
/* (c) Copyright Real-Time Innovations, Inc., 1999. All rights reserved. */
```

```
#include <stdio.h> /* Change to stdioLib.h for 5.0 */
#include <math.h>

#include "vxWorks.h"
#include "taskLib.h"
#include "sysLib.h"
#include "semLib.h"
#include "rtilib/vxVersions.h"
#include "rtilib/rti_types.h"
#include "rtilib/rti_endian.h"
#include "scope/scope.h"

#define MAXPHASES(128)
#define PHASESHIFT(0.1)
#define SQUARECOUNT(20)

float square= 1.0;
struct ExampleStruct {
    float sint;
    float sin2t;
    float sin3t;
    float cost;
} *singroup;
float phases[MAXPHASES];
float Omega = 1.0;

int NumberOfPhases = 0;

float Pos = 0.0;
float Vel = 0.0;
float Acc = 0.0;

float Posdes = 1.0;
float Veldes = 0.0;
float Kp = 10;
float Kv = 4;

float NoiseMag = 0.1;

float Time = 0.0;
float Dt = 0.005;

SEM_ID SampleSemaphore = NULL;
int ScopeIndex = -1;
int ScopeHandle = -1;

int tid = 0;
static int ScopeDemoShutdownRequest = 0;
static int ScopeNoServer = 0;

void (*SampleHookFunc)(void *) = NULL;
void *SampleHookParam = NULL;
```

```
void SampleHookSet( void (*func)(void *), void *param )
{
    SampleHookFunc = func;
    SampleHookParam = param;
}

static
/* Calculate a set of simple wave signals, paying absolutely no attention to
   efficiency. */
void SineWaves(float t)
{
    register int i;
    static int squareCount = SQUARECOUNT;
    register float wt = t * Omega;

    float sint, cost;

    if(--squareCount <= 0) {
        squareCount = SQUARECOUNT;
        square = 1.0 - square;
    }
    singroup->sint = sin(wt);
    singroup->cost = cos(wt);
    singroup->sin2t = sin(2 * wt);
    singroup->sin3t = sin(3 * wt);
    for(i=0; i<NumberOfPhases; i++) {
        phases[i] = sin(wt + PHASESHIFT*i);
    }

    sint = singroup->sint;
    cost = singroup->cost;

    /* Throw an event for when Sin and Cos are calculated. */
    if((singroup->sint > 0.1) && (singroup->sint <= 0.2)) {
        ScopeEventsCollect(ScopeHandle, 1, "Sine-0.1-0.2", (void *) &sint,
                           RTI_FLOAT32ID);
    }

    if((singroup->cost > 0.5) && (singroup->cost <= 0.6)) {
        ScopeEventsCollect(ScopeHandle, 1, "Cosine-0.5-0.6", (void *) &cost,
                           RTI_FLOAT32ID);
    }
}

static
/* Return a uniform random variable between a and b. */
float Noise(float a, float b)
{
    float result;

    result = (((float) rand() / RAND_MAX) - 0.5);
    return (result * (b - a) + (a + b) / 2.);
}
```

```
static
/* This "plant" is an Euler double integrator. */
void Plant(float t)
{
    Vel += Acc * Dt;
    Vel += Noise(-NoiseMag, NoiseMag);
    Pos += Vel * Dt;
}

static
void Control(float t)
{
    static int stepCount = 300;
    if(stepCount-- <= 0) {
        stepCount = 300;
        Posdes = -Posdes;

        /* Throw an event whenever position desired changes. */
        ScopeEventsCollect(ScopeHandle, 2, "PosChangeEvent", (void *) &Posdes,
            RTI_FLOAT32ID);
    }

    Acc = Kp*(Posdes - Pos) + Kv*(Veldes - Vel);
}

static
void Sample(float t, int noCollection)
{
    SineWaves(t);
    Plant(t);
    Control(t);
    if (noCollection == 0) { ScopeCollectSignals(ScopeIndex); }
    if (SampleHookFunc != NULL) {
        (*SampleHookFunc)(SampleHookParam);
    }
}

static
void InstallFakeSignals(int num)
{
    register int i;
    char str[132];

    if(num > MAXPHASES) {num = MAXPHASES;}
    NumberOfPhases = num;

    /* Install signals for debugging */

    ScopeInstallSignal("Square", "volts", &square, "float", ScopeIndex);
    singroup = (struct ExampleStruct *) calloc(1, sizeof(struct
ExampleStruct));
    singroup->sint = 0.0;
    singroup->sin2t = 0.0;
    singroup->sin3t = 0.0;
    singroup->cost = 1.0;
    ScopeInstallSignal("Sine", "volts", &(singroup->sint), "float",
```



```
ScopeIndex);
    ScopeInstallSignal("Cosine", "volts", &(singroup->cost), "float",
        ScopeIndex);
    ScopeInstallSignal("Sine2T", "volts", &(singroup->sin2t), "float",
        ScopeIndex);
    ScopeInstallSignal("Sine3T", "volts", &(singroup->sin3t), "float",
        ScopeIndex);

    ScopeInstallSignal("Pos", "meters", &Pos, "float", ScopeIndex);
    ScopeInstallSignal("PosDesired", "meters", &Posdes, "float", ScopeIndex);
    ScopeInstallSignal("Vel", "m/s", &Vel, "float", ScopeIndex);
    ScopeInstallSignal("Acc", "m/s/s", &Acc, "float", ScopeIndex);

    for(i=0; i<num; i++) {
        sprintf(str, "sin/sin(t+%.1f)", PHASESHIFT*i);
        ScopeInstallSignal(str, "volts", &phases[i], "float", ScopeIndex);
    }
}

static
/* This routine simply lets the sampler run. An alternative, asynchronous
   sampling strategy is to simply call ScopeCollectSignals() from this
   routine
   (at interrupt level). However, this scheme is probably more indicative of
   the way most users will implement Scope. */
int ScopeDemoTimerInterrupt(void)
{
    semGive(SampleSemaphore);
    return(0);/* sysAuxClkConnect should take VOIDFUNCPTR */
}

static
/* Called by main loop to clean up mem usage */
void ScopeDemoShutdownInternal(void)
{
    if (tid != 0) {
        if (SampleSemaphore != NULL) {
            /* Must be aux clk, so clean up. */
            sysAuxClkDisable();
            sysAuxClkConnect(NULL, 0);
            semDelete(SampleSemaphore);
            SampleSemaphore = NULL;
        }
        if (ScopeNoServer == 0) {
            ScopeShutdown(ScopeIndex);
        } else {
            #if VXWORKS_AE_VERSION_1_0_OR_BETTER
                ScopeRemoveMultipleSignals("", ScopeIndex);
            #else
                ScopeRemoveMultipleSignals("", ScopeIndex);
            #endif
            ScopeNoServer = 0;
        }
        ScopeIndex = -1;
        free(singroup);
        tid = 0;
    }
}
```

```
    }
    /* Reset flag */
    ScopeDemoShutdownRequest = 0;
}

static
/* This routine generates the signals, and does the sampling.
 * It attaches a semaphore to the Aux clock iff useAuxClock is non-zero.
 */
int ScopeDemoSampler(int requestedSR, int useAuxClock, int noCollection)
{
    int nTicksBwSamples = 0;
    float actualSR;

    if (useAuxClock) {
        SampleSemaphore = semBCreate(SEM_Q_PRIORITY, SEM_EMPTY);
    }

    /* Connect the timer to the auxillary clock interrupt. */
    if (useAuxClock &&
        (sysAuxClkConnect(ScopeDemoTimerInterrupt, 0) == OK)) {
        sysAuxClkEnable();
        sysAuxClkRateSet(requestedSR);
        /* hardware may not be exact */
        actualSR = sysAuxClkRateGet();
    } else {
        actualSR = sysClkRateGet();
        if (actualSR < requestedSR) {
            /* Requested rate (requestedSR) is higher than
             * maximum achievable rate.
             */
            nTicksBwSamples = 1;
        } else {
            nTicksBwSamples = actualSR / requestedSR;
            actualSR = requestedSR;
        }
    }

    /* Tell scope the actual sample rate (hardware may not be exact). */
    ScopeChangeSampleRateInt((int) actualSR, ScopeIndex);

    if((actualSR/requestedSR > 1.1) || (actualSR/requestedSR < 0.9)) {
        printf("ScopeDemo: requested rate (%f Hz) not \n"
            "achievable, actual sampling rate is %f Hz\n",
            (float) requestedSR, actualSR);
    }

    Dt = 1./actualSR;
    while(!ScopeDemoShutdownRequest) {
        if (useAuxClock) {
            semTake(SampleSemaphore, WAIT_FOREVER);
        } else {
            taskDelay(nTicksBwSamples);
        }
        Time += Dt;
        Sample(Time, noCollection);
    }
}
```

```
    }
    ScopeDemoShutdownInternal();
    return(0);
}

/*
ARGUMENTS
    If useAuxClock is TRUE, use the Aux clock. Otherwise (or if there is no
    Aux clock, use taskDelay.
    scopeIndex is the StethoScope index to use. It must be coordinated with
    the GUI.
    verbosity controls the amount of warning and debug messages. 0 means
    only errors will be printed. Higher values causes more output.
    If reqDataBufSize is non-zero, use that value rather than the default
    (51k)
    If noServer is true, then this demo will not start a Scope
    server. Assumes
        one is already started.
    If noCollection is non-zero then this demo will not call
    ScopeCollectSignals
        since it is assumed that another process is collecting.
*/
void ScopeDemo( int useAuxClock, int scopeIndex, int verbosity,
                int reqDataBufSize, int noServer, int noCollection )
{
    int samplingRate = 60;      /* Hz */
    int dataBufSize, signalBufSize, eventBufSize;

    /* Initialize */

    if (ScopeIndex > 0) {
        return;
    }

    if (reqDataBufSize == 0) {
        dataBufSize = 17*8*384; /* 51k data buffer (ints/floats) */;
    } else {
        dataBufSize = reqDataBufSize;
    }
    signalBufSize = 40960;
    eventBufSize = -1;
    if (noServer == 0) {

        /* Initialize an index. */
        ScopeIndex = ScopeInitServer(dataBufSize, signalBufSize, verbosity,
                                     scopeIndex);
        if (ScopeIndex < 0) {
            printf("ScopeInitServer failed, return code = %d\n", ScopeIndex);
            return;
        }
    }
}
```

```
/* Attach an event buffer to that index. */
ScopeHandle = ScopeEventsAttach(eventBufSize, ScopeIndex);
if(ScopeHandle == 0) {
    printf("ScopeEventsAttach failed, exiting!\n");
    return;
}

} else {
ScopeIndex = scopeIndex;
ScopeNoServer = 1;
}

ScopeIndex = scopeIndex;

InstallFakeSignals(8);

tid = taskSpawn("ScopeDemo", 100, VX_FP_TASK|VX_STDIO, 0x4000,
    (int (*)()) ScopeDemoSampler,
    samplingRate, useAuxClock, noCollection,0,0,0,0,0,0);
}

void ScopeDemoShutdown(void)
{
    if (tid != 0) {
        ScopeDemoShutdownRequest = 1;
    }
}
```

B.2.2 Makefile for vxdemo.c

Use this makefile as a template for compiling your code (or the **vxdemo.c** program) instrumented with ScopeProbe API.

```
#####
#
#           StethoScope demonstration makefile.
#
# This makefile is provided as an example only! It compiles the
# VxWorks version 6.0 objects using the "ccpentium" gnu cross-
# compiler for solaris2. Users will of course have to modify
# this for their installations.
#
# We recommend you copy the current demo source directory before
# compiling. For instance:
#   mkdir mydir
#   cp -r ./src/scopedemo mydir/src
#   cd mydir/src
#   make
#
```

```
# The makefile will create a "mydir/lib" directory to hold your
# objects.
#
#####

# This makefile should work even if you have not properly installed your
# cross-compiler environment. However, the following variables must be set
# to reflect your configuration:

WIND_BASE = /local/VxWorks/VDT.3.0
WIND_HOST_TYPE = sun4-solaris2
ARCH = pentium
CPU = PENTIUM3

# The following script should automatically be able to compile StethoScope
# demonstration files.
#
# Note 1: This makefile generates ".so" object files.
# Note 2: This makefile creates the USEROBJDIR directory, if it doesn't
# exist.

GCCHOME = $(WIND_BASE)/gnu/3.3.2-vxworks60/$(WIND_HOST_TYPE)
CC = $(GCCHOME)/bin/cc$(ARCH)

USEROBJDIR = lib/$(CPU)Vx6.0gcc3.3.2
SCOPE_INCLUDES = ../../include/share

# makefile rules

all:    $(USEROBJDIR) $(USEROBJDIR)/scopedemo.so

$(USEROBJDIR):
    mkdir -p $@

$(USEROBJDIR)/%.o: %.c
    $(CC) -Wall -O -c -DCPU=$(CPU) \
        -I$(WIND_BASE)/vxworks-6.0/target/h -I$(GCCHOME)/include \
        -I$(SCOPE_INCLUDES) -DRTI_VXWORKS $< -o $@

$(USEROBJDIR)/%.so: $(USEROBJDIR)/%.o
    $(GCCHOME)/bin/ld$(ARCH) -r $< -o $@
```

B

C

MATLAB and MATRIX_x Examples

- C.1 Introduction 249
- C.2 MATLAB Example 249
- C.3 MATRIX_x Example 253

C.1 Introduction

This appendix presents example script files that can be used in **MATLAB** and **MATRIX_x** to plot signals saved by StethoScope.

C.2 MATLAB Example

This section shows a **MATLAB** script file, **varplot.m**, that plots signals. It also shows a sample **MATLAB** session that loads a script file saved by StethoScope, then uses the **varplot** script to produce a PostScript file.


```

        if((length(s) == 3))
            if((s == 'end'))
                more = 0;
                break;
            end;
        end;
    else
        this_Curve = this_Curve + 1;
        if(Num_Curves == this_Curve),
            more = 0;
        end;
        s = varplot_Names(this_Curve,:);
    end;

    index = nameindex(names,s);
    if(index == 0)
        disp(s);
        disp(': Variable not found');
    else
        plotindex = [plotindex index];
    end
end;

% Build the arrays to be plotted.
for i=plotindex
    plotdata = [plotdata data(:,i)];
    plotnames = [plotnames ; names(i,:)];
    unitnames = [unitnames ; units(i,:)];
end
plotnames          % Display the plot names.
plot(time,plotdata);

xlabel('Time (seconds)');
ylabel(units(index,:));

if(exist('runtitle'))
    titlestring = [runtitle ' - ' timestamp];
else
    disp('warning - no runttitle');
    titlestring = [timestamp];
end
title(titlestring);

subnum = 0;
if(exist('subplotactive')),
    subnum = subplotactive;
end;

legend(plotnames, unitnames, 1, subnum, 1);
Example MATLAB Session

```

This **MATLAB** session loads the data saved as **dynamicPayload.mat** by running the **dynamicPayload.m** script, both created by StethoScope when saving data in **MATLAB** format. The session then uses **varplot** to plot two variables. The resulting

plot is saved in encapsulated PostScript format. The figure produced by this example appears in the manual as:

```
>> dynamicPayload
```

```
notes =
```

```
Notes:
```

```
This is a slew with the uncontrolled dynamic payload.  
Both force sensor filters active, at 20Hz.  
No friction compensation active.
```

```
controller =
```

```
ComputedTorque
```

```
PDGains0 =
```

```
    4.0000    1.5000    0
```

```
PDGains1 =
```

```
    1.0000    0.5000    0
```

```
>> who
```

```
Your variables are:
```

DesForceX	ForceYUnfiltered	ans
DesForceY	PDGains0	controller
DesiredAccX	PDGains1	data
DesiredAccY	PosX	filename
DesiredPosX	PosY	names
DesiredPosY	RawForceProbeX	notes
DesiredVelX	RawForceProbeY	numberOfSamples
DesiredVely	SampleDivisor	numberOfSignals
ElbowPos	SampleRate	runtitle
ElbowTorque	ShoulderPos	time
ElbowVel	ShoulderTorque	timestamp
FILE_EXISTS	ShoulderVel	units
ForceX	TERM	
ForceXUnfiltered	VelX	
ForceY	Vely	

```
>> varplot
```

```
Plot which signal? [end] ForceX
```

```
Plot which signal? [end] ForceY
```

```
Plot which signal? [end] end
```

```
plotnames =
```

```
ForceX
ForceY

>> meta
>> !gpp metatmp -deps -fSaveMatlabFigure.ps
```

C.3 MATRIX_x Example

This section shows a **MATRIX_x** script file, **varplot.ms**, that plots signals. It also shows a sample **MATRIX_x** session that loads a script file saved by StethoScope, then uses the **varplot** script to produce a PostScript file.

Example C-2 Varplot ms-File

```
# Name:
#   varplot.ms   variable plot program for scoped MATRIXx data
#
# Usage:
#   <load your scope data>
#   [define a global variable varplot_Names]
#   execute file="varplot"
# Parameters:
#   varplot_Names: a matrix of signal names to plot.
#                   All rows must have the same number of characters.
#                   Example: ["ForceUp  ";
#                             "ForceDown"]
#                   varplot_Names should be in order curves are plotted
#
# Description:
#   If varplot_Names is defined, "varplot" will print the named
#   signals.
#   Otherwise, "varplot" will prompt you for signals to be
#   plotted, then plot
#   them, with labels, etc.
#
# See Also:
#   plot
#
# Language:  MathScript                               Version 6.0
#
```

```
plotindex = [];  
plotdata = [];  
plotnames = [];  
unitnames = [];  
s = " ";  
If (exist(varplot_Names)),  
    # find the number of curves  
    Num_Curves=size(varplot_Names); Num_Curves=Num_Curves(1,1);  
    this_Curve = 0;  
endIf;  
  
numsignals = length(data(1,:));  
  
# If varplot_Names doesn't exist, prompt the user for signal names.  
more = 1;  
While(more == 1)  
    If (!exist(varplot_Names)),  
        s = getLine("Plot which signal? [end]");  
        If(length(s) == 3)  
            If(s == "end")  
                more = 0;  
                exit 0;  
            endIf;  
        endIf;  
    else  
        this_Curve = this_Curve + 1;  
        If(Num_Curves == this_Curve),  
            more = 0;  
        endIf;  
        s = varplot_Names(this_Curve,:);  
    endIf;  
  
    # find index of s within names  
    [rows, sz] = size(names);  
    sz = length(s); found = 0;  
    For i = 1:rows  
        If (sz == 0), exit 1; endIf;  
        If (stringex(names(i,1),1,sz) == s)  
            If (index(names(i,1)," ") == index(s," ") ...  
                | index(names(i,1)," ") == sz+1)  
                found = 1;  
                exit 1;  
            endIf;  
        endIf;  
    endFor  
  
    If (found == 0)  
        display(s + ": Variable not found");  
    else  
        plotindex = [plotindex, i];  
    endIf;  
endWhile;
```

```

# Build the arrays to be plotted.
For i=plotindex
    plotdata = [plotdata, data(:,i)];
    plotnames = [plotnames ; names(i,:)];
    unitnames = [unitnames ; units(i,:)];
endFor

display(plotnames)    # Display plot names

If (plotindex == [])
    return;
endIf;

If(exist(runtitle))
    titlestring = runttitle + " - " + timestamp;
else
    display("warning - no runttitle");
    titlestring = timestamp;
endIf

plot(time, plotdata, {title=titlestring, xlab="Time (seconds)",
    legend=plotnames + unitnames});

```

Example C-3 **MATRIX_x Session**

This **MATRIX_x** session loads the data saved as **dynamicPayload.xml** by running the **dynamicPayload.ms** script, both created by StethoScope when saving data in **MATRIX_x** format. The session then uses **varplot** to plot two variables. The resulting plot is saved in encapsulated PostScript format.



NOTE: In an actual **MATRIX_x** session, the input and output appear in different text areas. The listing below “merges” the two to give a sense of cause and effect, where the output of a command is shown following the command itself.

```

>> execute file="dynamicPayload"

notes (a column vector of strings) =

Session notes:
This is a slew with the uncontrolled dynamic payload.
Both force sensor filters active, at 20Hz.
No friction compensation active.

controller =

ComputedTorque

PDGains0 =

    4.0000    1.5000    0

```

```
PDGains1 =
    1.0000    0.5000    0

>> who

main:
    data -- 1000x38
    numberOfSamples -- 1x1
    numberOfSignals -- 1x1
    filename -- 1x1
    runtitle -- 1x1
    notes -- 2x1
    buffernotes -- 1x1
    SampleRate -- 1x1
    SampleDivisor -- 1x1
    DifferentTypes_8ByteDouble -- 1000x1
    DifferentTypes_4ByteFloat -- 1000x1
    DifferentTypes_4ByteLong -- 1000x1
    DifferentTypes_4ByteULong -- 1000x1
    DifferentTypes_4ByteInt -- 1000x1
    DifferentTypes_4ByteUInt -- 1000x1
    DifferentTypes_2ByteInt -- 1000x1
    DifferentTypes_2ByteUInt -- 1000x1
    DifferentTypes_1ByteChar -- 1000x1
    Pointer_Float -- 1000x1
    Pointer_Float_ -- 1000x1
    Pointer_Float__ -- 1000x1
    Pointer_Int -- 1000x1
    Pointer_Int_ -- 1000x1
    Pointer_Int__ -- 1000x1
    Pointer_Double -- 1000x1
    Pointer_Double_ -- 1000x1
    Pointer_Double__ -- 1000x1
    Offset_First -- 1000x1
    Offset_Second -- 1000x1
    Pos -- 1000x1
    PosDesired -- 1000x1
    Vel -- 1000x1
    Acc -- 1000x1
    PosGain -- 1000x1
    VelGain -- 1000x1
    Sine -- 1000x1
    Cosine -- 1000x1
    Sine2T -- 1000x1
    Square -- 1000x1
    SinGroup_sin_t_0_0_ -- 1000x1
    SinGroup_sin_t_0_1_ -- 1000x1
    SinGroup_sin_t_0_2_ -- 1000x1
    SinGroup_sin_t_0_3_ -- 1000x1
    SinGroup_sin_t_0_4_ -- 1000x1
    SinGroup_sin_t_0_5_ -- 1000x1
    SinGroup_sin_t_0_6_ -- 1000x1
    SinGroup_sin_t_0_7_ -- 1000x1
    time -- 1x1000
```

```
names -- 38x1  
units -- 38x1  
timestamp -- 1x1  
gains -- 1x2
```

```
>> execute file="varplot"
```

```
Plot which signal? [end] ForceX
```

```
Plot which signal? [end] ForceY
```

```
Plot which signal? [end] end
```

```
ForceX
```

```
ForceY
```

```
>> hardcopy file="SaveMatlabFigure.ps", {ps}
```


D

Glossary

ellipsis (“...”)

On all menu items, an ellipsis indicates that the option will open a new window that requires further response or interaction.

Graphical User Interface (GUI)

The collection of computer programs and the media-oriented screens, windows, dialog boxes, menus, and buttons they produce that provide for enhanced human-computer interactions with no, or minimal, keyboard input.

Host

The computer on which StethoScope is running, which receives and processes the allocation record data collected from the target agent machine.

License

A private encoded character string from Wind River that confers authority, as well as physical capability, to execute and use the StethoScope system software.

routine

A self-contained code module that can accept input, execute, and produce output; used interchangeably with function.

RTP

A VxWorks Real-Time Process, running in a protected environment.

Verbosity

Controls the type and number of messages generated by the target server connection process. Using the default value of 0 generates only error messages. Specifying a larger value (in the range of 1-4) generates an increasingly greater variety and volume of messages.

Index

A

- accuracy
 - Dump Plot Windows 139
 - Monitor Window 150
 - Plot Windows 97, 108, 122, 129, 138, 148
- active signals
 - creating 62
 - defined 13, 178
 - examples of activating 193
 - installing 192
 - setting up 32, 62
- adding markers 54
- Annotation Edit dialog box 55
- annotations
 - adding in Plot Windows 55
- API
 - how to use 207
- architecture 3
- ASCII snapshots 156
- aspect lock, in Plot XY window 117
- asynchronous sampling
 - described 197
 - example 198
 - task 166
- auxiliary clock 167
- Axis Properties dialog box 51

B

- browse button 259
 - snapshot dialog box 156
- buffers
 - snapshots 74
- building
 - header files 163
 - include files 163

C

- calculating offsets 194
- capabilities 1, 7
- changing base file name 157
- Check Out License icon 2
- collecting signals 197
- colors
 - alternative method to change trace lines 100, 124
 - changing color of plot lines 95
 - dialog box 99
 - in Legend tab view 94, 120
 - in plot windows 95, 121
 - in Signal Properties dialog box 99, 124
 - in Signals tab view 92, 119
 - on left axis (Has Ruler option) 102, 127
 - preferences 38

- selecting trace line colors 87, 114
- signal trace lines 31, 85
- table coordinated with Signals Tree 136, 146
- used in snapshots 108, 129
- communications plugins
 - preferences 39
- configuration files
 - loading from disk 27
 - saving to disk 28
- configuring StethoScope 166
- connecting to targets 10, 26
- connection problems 173
- connection status 21
- context sensitive menu 52, 103
- creating
 - derived signals 78
 - XY signal pairs 112
- cycle numbering 157

D

- daemons
 - manually starting 169
 - roll in buffer overflow 73
 - task descriptions 5
- data collection
 - status 67
- data-display windows
 - introduction 14
 - toolbar 48
- deactivating signals 196
- deleting markers 54
- demonstrations
 - running demonstration target program 18
 - VxWorks 19
- derived signals
 - creating 78
 - mathematical operators 81
 - troubleshooting 83
 - Wizard 78–81
- Derived Signals dialog box 35, 49, 78
- Derived Signals wizard 35
- dialog boxes
 - Annotation Edit 55

- Axis Properties 51
- Colors 99
- Derived Signals 35
- derived Signals 49
- New Target Connection 27
- Open 28, 39, 40
- Preferences 36, 49, 87, 107, 114
- Print 88, 115
- Save Config 29
- Signal Properties 57, 58, 87, 95, 98, 114, 119, 121, 123
- StethoScope Setup Options 26
- Triggering 33
- XYSignals 32, 34, 49, 112, 114
- disconnecting targets 63
- display accuracy
 - Dump Plot Windows 139
 - Monitor Window 150
 - Plot Windows 97, 108, 122, 129, 138, 148
- displaying events 103
- docking toolbars 48
- downsampling 69, 73
- Dump Plot Window
 - description 15, 132
 - display accuracy 139
 - history limit 139
 - preferences 41
 - resolution 139
 - table size 139

E

- ellipsis button 259
- environment variables
 - PATH 12
 - RTIHOME 11
- error log 11
- events
 - displayed as markers 104
 - displayed as messages 105
 - displayed in the Signals Bar 103
 - displaying 103
 - ScopeEventsCollect 103, 201
 - ScopeEventsMessage 103, 201

- vs. signals 203
- Events API
 - described 200
 - in the demonstration program 18
 - setting up 201
 - using 201
- examples
 - activation 193
 - asynchronous signal sampling 198
 - events displaying 104
 - installing signals 196
 - MATLAB 249–253
 - MATRIXx 253–??
 - registration 193
 - registration with offset 195
 - ScopeCollectSignals() 198, 199
 - ScopeInstallSignal() 196
 - signal installation 23
 - synchronous signal sampling 199
 - vxdemo.c 237–??
 - Workbench target script 171
- exiting StethoScope 43

F

- feature overview 36
- features 1, 2, 7
 - preferences 36
 - triggering, overview 33
- file
 - name extensions 157
- File menu item 26
- functions
 - See also routines

G

- general preferences 37
- graph coordinates 54
- graphical user interface (GUI)
 - defined 259
 - host-side 61, 65

- gridline spacing 97, 108, 122, 129, 138

H

- header files 163
- hierarchical naming of signals 193
- history limit 139
- host, defined 259
- host-side GUI 61, 65

I

- include files 163
- initialization
 - .stethoscoperc file 60
 - sequence 60
 - StethoScope (UNIX) 60
 - StethoScope (Windows) 12
- installed signals
 - defined 13, 178
 - using all by default 33
- installing signals
 - examples 196
 - first 63
 - from command shell 23
 - in one step 192
 - organizing signal names 187
 - Signal Manager window 64
 - StethoScope API 191
- instrumenting
 - alternative to 185
 - code 185
- introduction 1, 7

K

- keeping window on top 47

L

- Legend tab view 94, 120
- libraries
 - libscope.so 170
 - libscopewtxonly.so 170
 - libutilsip.so 169
 - libutilsnoip.so 169
 - vxdemo.so 170
- license
 - Check Out License icon 2
 - defined 259
 - policy 2
- license file 11
- Link Daemon
 - buffer overflows 73
 - task description 5
- Live buffer 75
- loading
 - automatic 165
 - configuration files 27
 - manual 169
 - snapshots 157
- log file 36
- Log window
 - verbosity 36

M

- markers 54
- MATLAB
 - example 249–253
 - snapshots 156
- MATRIXx
 - example 253–??
 - snapshots 156
- maximum snap distance 97, 108, 122
- measurements 97, 108, 122
- menu
 - File 26
 - Plot 44
 - View 45
 - Window 47
- mini Dump Plot Window

- default properties 109
- description 47
- mini Monitor Window
 - default properties 109
 - description 47
- minimum gridline spacing 97, 108, 122, 129, 138
- modifying signals 149
- Monitor Window
 - description 15, 142
 - display accuracy 150
 - preferences 42, 149
 - resolution 150
 - writing data to target 149

N

- naming of signals 193
- New Target Connection dialog box 27

O

- offsets, calculating 194
- on-grid menu 52, 103
- online documentation reference
 - signals 197
- Open dialog box 28, 39, 40
- opening snapshots 157
- overflow prevention 69, 73

P

- panning in Plot Windows 56
- PATH 12
- Pause 135, 145
- Plot menu 44
- plot plugins, preferences 40
- Plot Window
 - adding annotations 55
 - adding markers 54
 - colors 95
 - description 14, 86

- display accuracy 97, 108, 122, 129, 138, 148
 - displaying signal values 87
 - Legend tab view 94, 120
 - maximum snap distance 97, 108, 122
 - minimum gridline spacing 97, 108, 122, 129, 138
 - panning 56
 - popup menu 52, 103
 - preferences 41, 106, 138, 149
 - previous zoom 47, 53, 91, 117
 - resolution 97, 108, 122, 123, 129, 138
 - selecting a signal 87
 - snap measure to signals 97, 108, 122
 - snapshots 108, 129
 - status bar 52
 - strip chart 103
 - taking on-grid measurements 56
 - toolbar 50
 - X offset 107, 128
 - X range 107, 129
 - Y offset 96, 107, 122, 128
 - Y range 107, 128
 - Y scale 96, 122
 - zooming 54
 - Plot XY Window
 - colors 121
 - description 14, 113
 - preferences 43, 128
 - previous zoom 47, 53, 91, 117
 - plots toolbar 48
 - popup menu
 - deleting snapshots 161
 - description, Legend tab view 58
 - description, on-grid 52
 - description, on-trace 57
 - description, Signals Tree 57
 - Plot window, general reference 103
 - Plot window, Legend tab view 95
 - Plot window, Signal Properties dialog box 99
 - Plot window, Signals tab view 94
 - Plot XY window, general reference 127
 - Plot XY window, Legend tab view 120
 - Plot XY window, on-grid 114
 - Plot XY window, Signal Properties dialog box 124
 - Plot XY window, Signals tab view 119
 - to disconnect GUI from the target 64
 - to open Signal Installation dialog box 64, 179
 - preferences
 - colors 38
 - comm plugins 39
 - dump plot window 41
 - feature overview 36
 - general 37
 - Monitor Window 149
 - monitor window 42
 - plot plugins 40
 - Plot Window 106, 138, 149
 - plot window 41
 - Plot XY Window 43, 128
 - scope.ini file 204
 - Preferences dialog box 36, 49, 87, 107, 114
 - preventing overflows 69, 73
 - previous zoom
 - in Plot Windows 47, 53, 91, 117
 - in Plot XY Windows 47, 53, 91, 117
 - Print dialog box 88, 115
 - Probe Daemon
 - task description 5
 - processampler 185
 - Properties tab 16
- ## R
- registering signals 192
 - registration
 - examples 193
 - examples with offset 195
 - removing markers 54
 - removing multiple signals 196
 - removing signals 196
 - resolution
 - Dump Plot Windows 139
 - Monitor Windows 150
 - Plot Windows 97, 108, 122, 123, 129, 138
 - routines
 - defined 259
 - StethoScope API library overview 190
 - triggering 200

RTIHOME 11
RTP
 defined 259
running
 demonstration target program 18
 StethoScope 9
running with a VxWorks target
 targets
 VxWorks 163

S

sampling
 asynchronous signals 197
 downsampling 69, 73
 functions 200
 rate 199
 ScopeChangeSampleRate() 199
 signals 197
 synchronous signals 199
Save Config dialog box 29
save.ssc 11
saving
 configuration files 28
 snapshots 27
scope index 20
 and event buffers 191
 described 190
 example code 193
 Legend tab view 94, 120
 limits 192
 Signal Manager window 64
 triggering example 72
scope.ini file 204
ScopeActivateMultipleSignal() 197
ScopeActivateSignal() 192, 194, 197
ScopeChangeSampleRate() 69, 73, 199
ScopeCollectSignals() 69, 73, 74, 174, 193, 196, 197,
 198, 199
ScopeDeactivateMultipleSignals() 197
ScopeDeactivateSignal() 196, 197
scopeIndex 209
ScopeInitServer() 174
ScopeInstallArray() 197

ScopeInstallSignal() 192, 197
ScopeInstallSignalWithOffset() 192, 195, 197
ScopePrintVersion() 173
ScopeProbe
 capabilities 4
 description 4
ScopeRegisterArray() 197
ScopeRegisterSignal() 192, 194, 197
ScopeRegisterSignalWithOffset() 192, 194, 197
ScopeRemoveMultipleSignal() 196
ScopeRemoveMultipleSignals() 197
ScopeRemoveSignal() 196, 197
ScopeSamplerTaskCreate() 171
ScopeShowActiveSignals() 197
ScopeShowSignals() 197
selected signals
 defined 13, 178
 in Dump Plot window 15, 133
 in Legend Bar 16, 87, 114
 in Monitor window 15, 142
 in Plot window 14
 in Plot XY window 14
 in Signal Manager 63
 in Snapshot window 45, 88, 108, 115, 129, 133,
 144, 152
 in state info of data display windows 30
 verifying target connection 168
 writeback feature in Monitor window 149
setting up active signals 32, 62
setting up APIs 201
setup options 166
show snapshot 135
signal installation
 organizing signal names 187
 Signal Manager window 64
Signal Manager 32, 62
signal pairs 112
Signal Properties dialog box 57, 58, 87, 95, 98, 114,
 119, 121, 123
signals
 activating 62
 activation 32, 62, 192, 194, 197
 activation examples 193
 collection 69, 73, 196, 197
 deactivate 197

- deactivation 196
 - downsampling 69, 73
 - install array 197
 - installation 63, 191, 197
 - installation with offset 195, 197
 - installing 23, 191
 - multiple deactivation 197
 - multiple activation 197
 - naming 193
 - online documentation reference 197
 - register array 197
 - registration 192, 194, 197
 - registration with offset 192, 194, 197
 - removal 197
 - remove multiple signals 197
 - removing 196
 - removing multiple 196
 - sampling 197
 - show active 197
 - show signals 197
 - vs. events 203
 - writing to target 149
 - Signals Bar
 - description 15
 - menu item 46, 90, 117, 134, 145
 - using 62
 - Signals tab 15
 - signals trees
 - described 15
 - using 62
 - slope trigger parameter 68, 70
 - snap distance 97, 108, 122
 - snapping measures to signals 97, 108, 122
 - snapshots
 - ASCII formatting 156
 - automating 155
 - behavior 108, 129
 - buffering 74
 - building file names 157
 - changing base file name 157
 - colors used 108, 129
 - cycle numbering 157
 - description of process 152
 - file name extensions 27, 157
 - format 156
 - loading 27, 157
 - MATLAB formatting 156
 - MATRIXx formatting 156
 - preferences 108, 129
 - saving to disk 27, 153, 156
 - taking 108, 129, 153
 - time stamping 157
 - triggering 71
 - starvation 173, 174
 - status bar
 - control of viewing 46, 90, 116, 134, 144
 - Plot Windows 52
 - used to open Log window 36
 - StethoScope
 - architecture 3
 - exiting 43
 - features 1, 7
 - host-side GUI 61, 65
 - icons on Workbench 9
 - initialization sequence 60
 - main windows 14
 - reference information 207
 - running 9
 - target application 4
 - StethoScope API reference section 207
 - StethoScope Setup Options dialog box 26
 - strip chart 45, 88, 103
 - synchronous sampling 199
- ## T
- tabs
 - Properties 16
 - Signals 15
 - taking on-grid measurements
 - demonstration program 22
 - in Plot Windows 56
 - on-grid plot commands 54
 - target
 - code, instrumenting 185
 - connecting to 26
 - disconnecting 63
 - names 10

- processsampler - alternative to instrumenting
 - your code 185
 - writing data to 149
- target application 4
- target-shell script 171
- terminology 12
- time stamps
 - feature of data storage 3
 - in definition of "Timing" 203
 - in Dump Plot window 15, 132
 - in MATLAB example 252
 - in MATRIXx example 257
 - in Snapshot window 157, 159
 - in Varplot-mFile example 251
- tLinkDaemon task
 - connection failure 172
 - connection, but no data 174
 - verifying target initialization 168
- toolbar button descriptions 49, 50
- toolbars
 - captions 46, 90, 117
 - data-display windows 48
 - docking 48
 - Main 46, 90, 116, 134, 144
 - Plot Window 46, 50, 90, 116, 134, 144
 - Plots 46, 48, 90, 116, 134, 144
- ToolTips 48
- tProbeDaemon task
 - connection failure 172
 - verifying target initialization 168
- trace log 36
- Triggering dialog box 33
- triggers
 - configuring 66
 - feature, full description 65
 - functions 200
 - overview 33
 - parameters 68, 69
 - rearming 70, 72, 73
 - snapshot 70, 72
 - status 67
 - tips 74
- troubleshooting
 - connection failures 172
 - derived signals 83

- empty signals tree 64
- loading errors 172
- no data appears 74, 174
- no response from target 173
- overflows 69, 73

U

- using
 - Events APIs 201
 - installed signals by default 33
 - log file 36
 - Signals Bar 62
 - signals trees 62

V

- verbosity 11
 - defined 260
 - Log window, effect on 36
- View menu 45
- vxdemo.c example 237-??

W

- Window menu 47
- wizard
 - Derived Signals 35
- Workbench IDE icons 9
- Workbench targets
 - automatic loading and running 165
 - example script 171
 - manual loading and running 169
 - setup options 166
- writeback
 - feature described 149
 - to write data to target 149
- writing to target 149
- WTX
 - protocol 5, 12, 167
 - target server 12

X

- X offset [107, 128](#)
- X range [107, 129](#)
- XY Signals window [112](#)
 - creating XY signal pairs [112](#)
- XYSignals dialog box [32, 34, 49, 112, 114](#)

Y

- Y offset [96, 107, 122, 128](#)
- Y range [107, 128](#)
- Y scale [96, 122](#)

Z

- zoom to fit
 - in Plot Windows [47, 91, 117](#)
 - in Plot XY Windows [47, 91, 117](#)
- zooming in Plot Windows [54](#)