Spectrum Analyzers and Signal Generators in LabVIEW

User Guide

User's Guide to Using Signal Hound Spectrum Analyzers and Signal Generators in LabVIEW

1502 SE Commerce Ave, Suite 101
Battle Ground, WA
Phone 360-217-0112

Contents

1 Overview

This document outlines the process of interfacing Signal Hound spectrum analyzers and signal generators in National Instruments LabVIEW software.

2 Preparation

2.1 REQUIREMENTS

The following items are required to interface Signal Hound devices in LabVIEW:

- A Signal Hound spectrum analyzer (SM200A/B, BB60A/C, SA44(B), SA124A/B) or signal generator (VSG25A, VSG60A, TG44A, TG124A) with required USB cables (*SA44(B), SA124A/B, and VSG25A cannot be used on Linux)
- National Instruments LabVIEW software
- A Windows 7/8/10 computer meeting minimum requirements to run Spike software (see Spike Manual for details)
  or
- An equivalent 64-bit Linux computer

2.2 INSTALLATION

2.2.1 Software Installation

2.2.1.1 Windows

- Install Spike.
  or
- Install VSG60 Software.

- Install LabVIEW.

2.2.1.2 Linux

- Install API for BB Series, SM Series, or TG Series.
- Install LabVIEW.

2.2.2 Device Connection

2.2.2.1 Windows

Refer to the Spike Manual or the VSG60A Manual to get your Signal Hound device connected and running with your computer. When working properly in Spike or the VSG60A software you are ready to use in LabVIEW.

2.2.2.2 Linux

Refer to the README in the API download to get your Signal Hound device connected and running with your computer.

2.2.3 Signal Hound Instrument Drivers

To install Signal Hound instrument drivers in LabVIEW, simply place the Signal Hound driver folder (eg. **Signal Hound BB Series**) in the LabVIEW instrument drivers directory, **instr.lib**, located in the LabVIEW root directory:

2.2.3.1 Windows with 32-bit LabVIEW

C:\Program Files (x86)\National Instruments\LabVIEW 2016\instr.lib

2.2.3.2 Windows with 64-bit Labview

C:\Program Files\National Instruments\LabVIEW 2016\instr.lib

In the Signal Hound driver folder, you will need to rename the 64-bit DLL and either delete or rename the 32-bit DLL.

For example, for the BB series, rename **bb_api.dll** to **bb_api-32.dll**, and rename **bb_api-64.dll** to **bb_api.dll**.

2.2.3.3 Linux

/usr/local/natinst/LabVIEW-2016-64/instr.lib

3 Usage

3.1 PALETTE NAVIGATION

3.1.1 Accessing the Functions Palette

Block diagrams are built using the functions and VIs on the **Functions** palette (Fig. 1). To bring up the **Functions** palette, select **View >> Functions Palette** or right-click anywhere on the block diagram workspace.

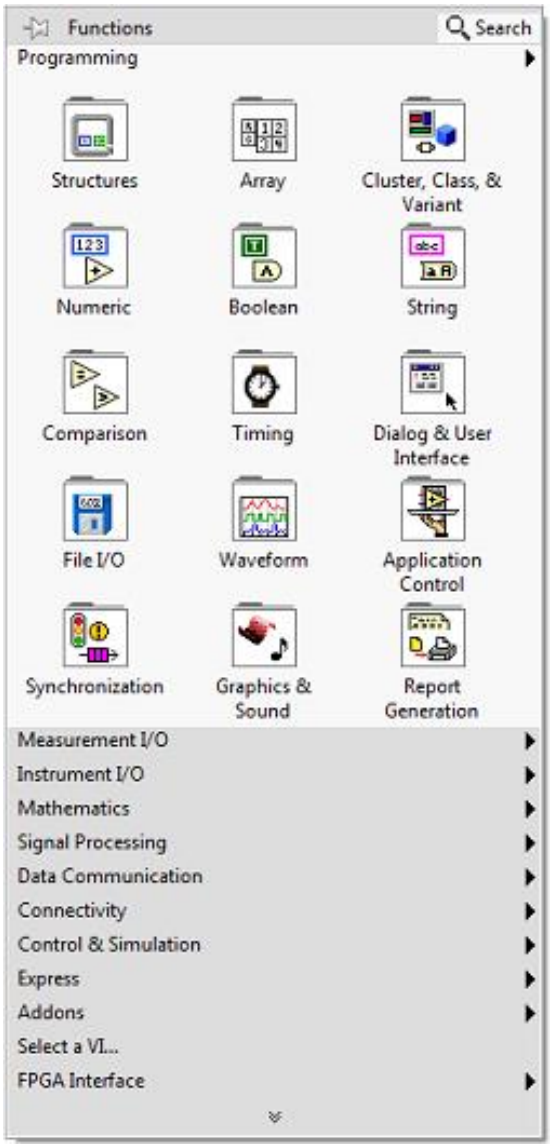

Figure 1. The LabVIEW Functions palette

3.1.2 Accessing the Signal Hound Subpalette

To access the Signal Hound instrument driver subpalette (Fig. 2) from the **Functions** palette, navigate to **Instrument I/O >> Instr Drivers**. Any installed Signal Hound instrument drivers will appear as icons here (eg. **Signal Hound SA Series**).

***Tip: Clicking the pin icon in the upper left corner of the subpalette will cause the subpalette to remain visible on the block diagram workspace.*

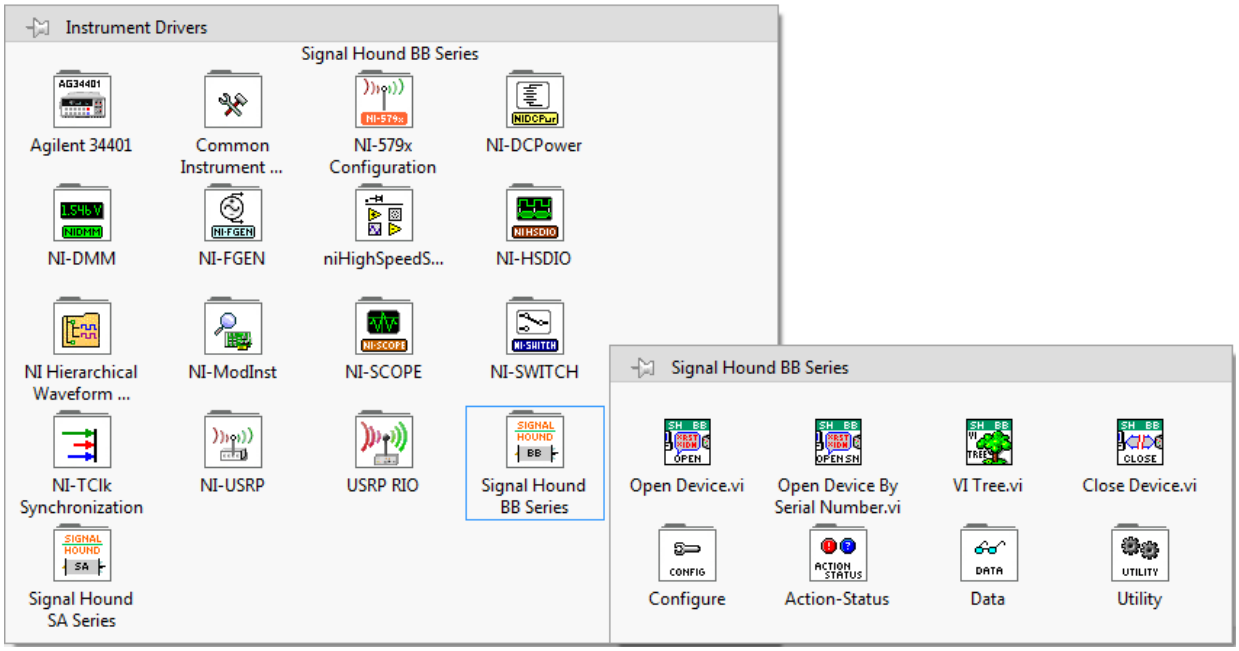

Figure 2. The Signal Hound BB Series subpalette

3.1.3 Signal Hound Subpalette Organization

Signal Hound instrument drivers follow a LabVIEW standard organizational hierarchy. The top level contains primary functions and subpalettes which group VIs into four categories: **Configure**, **Action-Status**, **Data**, and **Utility.**

3.1.3.1 Top Level

The top level of the Signal Hound instrument driver hierarchy contains primary functions to open and close the device, and the **VI Tree**.

The **VI Tree** (Fig. 3) is useful to see an overall view of all VIs in the driver, and where they are located. All VIs can be accessed from the block diagram workspace of the **VI Tree**.

3.1.3.2 Configure

This subpalette contains VIs used to configure the device, including preset routines.

For Signal Hound spectrum analyzers, two convenience VIs are provided which consolidate common configuration routines: **Quick Configure Sweep** and **Quick Configure IQ**. These contain a chain of configuration VIs (eg. **Configure Center Span → Configure Level**), followed by device initiation and a query which returns relevant information. All sub-VIs handle errors and pass the result out.

3.1.3.3 Action-Status

This subpalette contains VIs used to query and return info about the device status, and to execute certain commands.

3.1.3.4 Data

This subpalette contains VIs used to acquire data. It also contains VIs to initiate the device for acquisition using the current configuration, and to abort the current acquisition and configuration.

3.1.3.5 Utility

This subpalette contains VIs used to get critical information about the device, such as diagnostics, firmware version, and serial number. It also may contain convenience functions to perform useful numerical conversions.

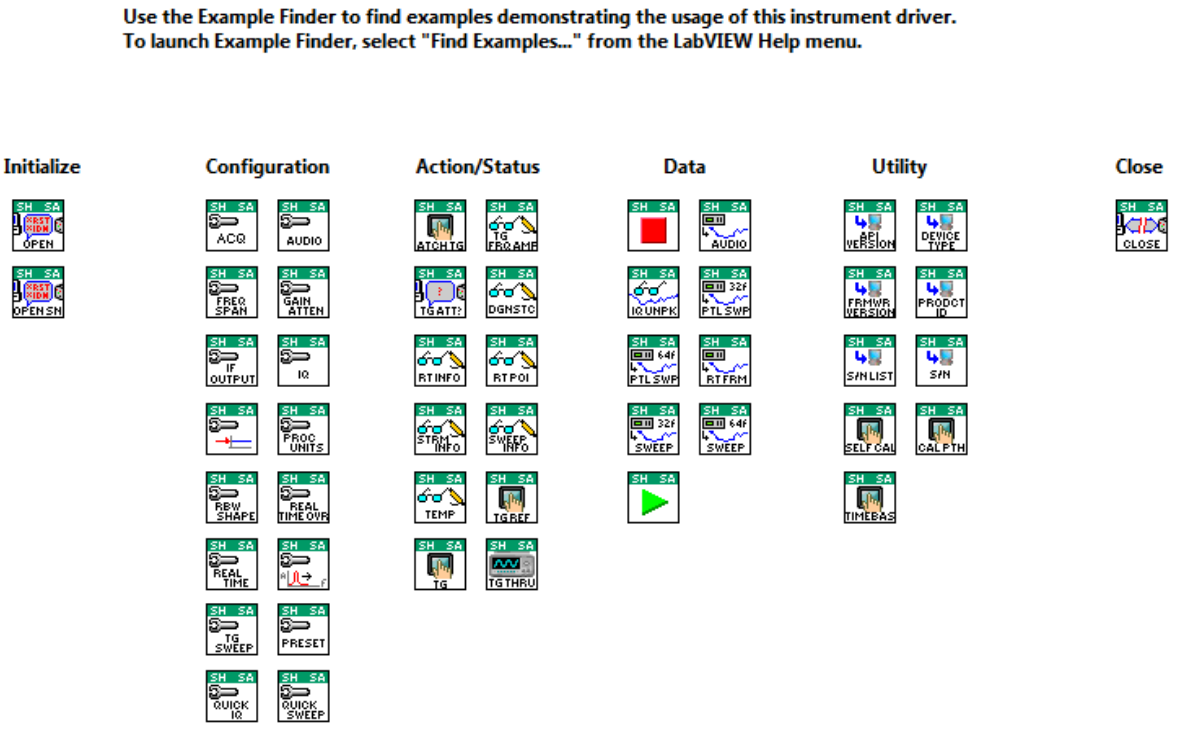

Figure 3. The Signal Hound SA Series VI Tree block diagram workspace

3.2 SESSION IDS

When a device is opened, a new session is started and given an integer handle. This handle is then used to identify the device to all device-specific VIs.

Every device-specific VI inputs and outputs a session ID on its uppermost terminals. These are typically chained together in successive VIs (Fig. 4).

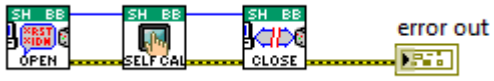

Figure 4. Chaining together the device ID and error line of successive VIs

3.3 ERROR HANDLING

Every VI inputs and outputs an error cluster on its lower terminals. These should be chained together in successive VIs, so that an error event propagates (Fig. 4).

***Tip: Automatically wire together VIs by dragging a VI from the functions palette and, before dropping it on the block diagram workspace, hover it near another VI with compatible terminals until wires appear between them. Drop the VI and the compatible terminals will be connected.*

In a simple approach to error handling, the error line can be wired to a case structure to control the execution path, bypassing blocks in the event of an error (Fig. 5). This is the approach used for Signal Hound sub-VIs.

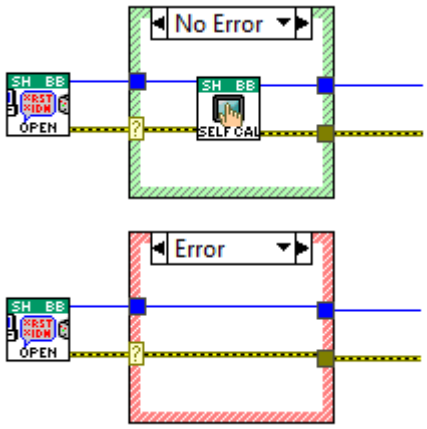

Figure 5. Error handling using a Case structure

3.4 DATA ACQUISITION

3.4.1 Memory Allocation

Most Signal Hound acquisition VIs (eg. **Get Sweep**) take pointers to arrays as parameters. These arrays must be pre-allocated with enough space to hold the data requested.

3.4.1.1 Initialize Array

Array allocation can be accomplished by using the **Initialize Array** VI, located in the **Array** subpalette of the **Functions** palette.

To use **Initialize Array**, wire a constant of the data type the array is to hold (eg. a double-precision floating point number for a C double; a single-precision floating point number for a C float) to the **element** terminal, and the array size needed to the **dimension size** terminal (Fig. 6).

Typically, a return parameter from a query function specifying the array size needed would be wired to **dimension size**. For example, **sweepLength**, returned by **QuerySweepInfo** and **Quick Configure Sweep**.

The output of **Initialize Array** can be connected directly to the data array terminal of the Signal Hound acquisition VI, which will automatically interpret it as an address pointer. Likewise, the address will be dereferenced automatically on output, so that the output of the VI is an array containing the acquired data.

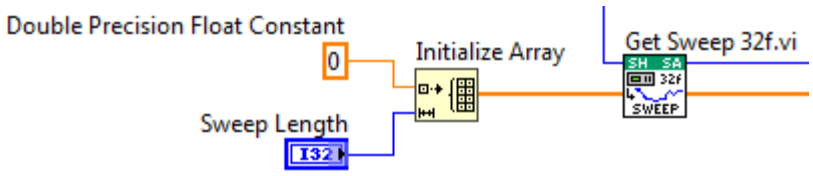

Figure 6. Allocating an array for a sweep

3.4.1.2 Clusters and IQ Data

In handling IQ data Signal Hound APIs use a struct typedef called **IQPacket** to hold associated variables, including a pointer to the data array. Because it can be unwieldy to work with dynamically allocated arrays inside clusters in LabVIEW, Signal Hound provides a VI that uses free parameters—**Get IQ Data Unpacked**.

Get IQ Data Unpacked is always the fastest way to acquire IQ data. On a computer with sufficient processing power sample rates should approach the limits of the device.

3.4.2 Continuous Acquisition

3.4.2.1 Loops

Continuous data acquisition can be accomplished using loop structures. Loop structures are found in the **Structures** subpalette of the **Functions** palette. The most commonly used loop structures are **For** and **While** loops.

For loops perform a set number of iterations, whereas **While** loops iterate as long as some condition is true.

Typically, a **Stop Button** (**Controls Palette >> Boolean**) and the status component of an error line are wired to an **OR** gate (**Functions Palette >> Boolean**), which is in turn wired to the **While** loop's **Conditional Terminal** (Fig. 7). That way loop execution halts when either there is an error in an enclosed VI or the user chooses to stop execution by breaking out of the loop.

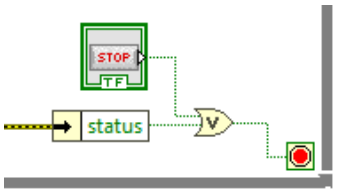

Figure 7. Typical wiring for the Conditional Terminal of a While loop

3.4.2.2 Using Loops in Your Block Diagram

There are several ways to use loops in your block diagram to continuously acquire data; the differentiating factor is which VIs you enclose in the loop.

For fast data acquisition, enclose only the acquisition VI and whatever VIs are being used to process and/or plot the data. Configuration, initiation, and allocation should be done before entering the loop, with variables entering the loop through a **Loop Tunnel**. A **Loop Tunnel** is created automatically when a wire crosses a loop boundary when connecting two terminals.

If interactivity is favored over speed, the loop can enclose configuration and acquisition. In this scenario the device is reconfigured and memory re-allocated at the start of each iteration, which is more resource-intensive and slower.

Alternatively, control structures (such as a **Case** structure) could be set up so that re-configuration only takes place when a setting is changed.

3.5 MULTI-THREADING

To avoid blocking the main GUI thread, all calls to API functions in Signal Hound VIs run "in any thread." This allows user interaction with the GUI to continue during execution of an application that is continuously collecting data and updating the screen.

4 Examples

Each Signal Hound instrument driver provides examples that demonstrate many of the use cases discussed in this manual.

4.1 ACCESSING EXAMPLES

To open Signal Hound example VIs, select **Help >> Find Examples...**, and either use the **Search** tab to search for "SignalHound" or in the **Browse** tab locate the examples under **Analysis, Signal Processing and Mathematics >> Signal Processing** (Signal Hound example VIs begin with "Signal Hound").

5 Information & Support

5.1 INFORMATION

Release and meta information is available in the Readme file located at the top level of the instrument driver folder.

5.2 SUPPORT

The first troubleshooting step is always to power cycle the device and restart LabVIEW.

You may need to install the Visual C++ Redistributable for Visual Studio 2012.

For support contact support@signalhound.com.