LMI TECHNOLOGIES

Title: Interfacing Gocator 5.3+ Using LabVIEW .NET Connectivity

Revision: 1.4

Table of Contents

HEADQUARTERS

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com

9200 Glenlyon Parkway
Burnaby, BC V5J 5J8
Canada

Overview

LabVIEW is a comprehensive software package for controlling measurement and control systems. LMI provides a set of example Virtual Instruments (VIs) that can be used to interface LabVIEW with Gocator sensors utilizing .NET connectivity to access ***GoSdkNet.dll*** API. These VIs can be used to control and stream measurement results, intensity and 3D data (as individual profiles or surface scans) into a LabVIEW real-time application for processing and control.

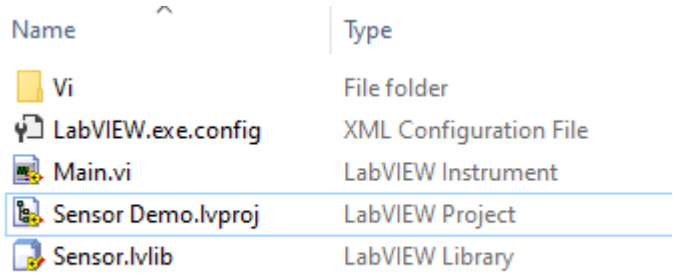
This document assumes that LabVIEW is already installed. You should already be familiar with LabVIEW, and the Gocator must already be set up to generate profile or surface data.

Gocator Firmware Release 5.3 or newer and LabVIEW Version 2018 (32-bit) or higher are required.

Installing the LabVIEW VIs

The LabVIEW example project is included in the Gocator Utilities package (5.3+): *14405-5.3.xx.xx_SOFTWARE_GO_Utilities.zip,* under the *Integration/LabVIEW (5.3+) directory.* Ensure the package is unzipped before continuing.

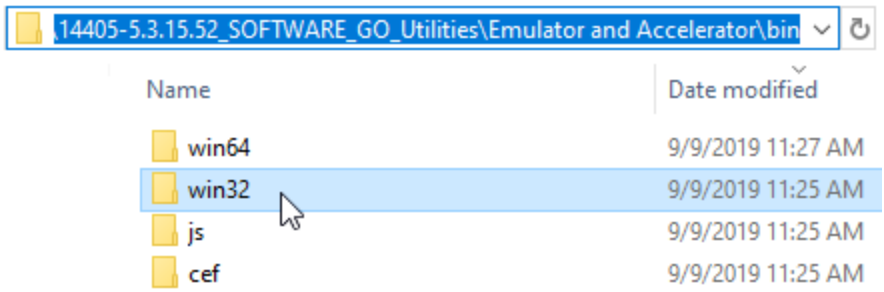
The Gocator Demo project has following directory structure:



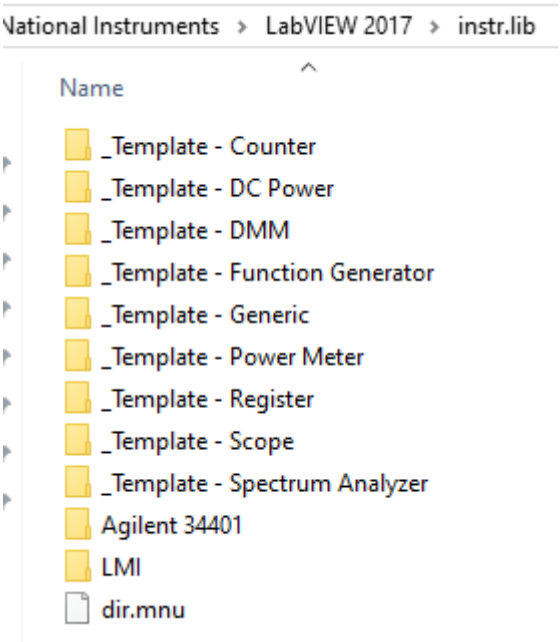
Directory	Contents
\Vi	Contains public Gocator LabVIEW library VIs and controls.

To install the VIs:

1. Copy the win32 folder from the LMI Emulator directory to the LabVIEW instr.lib directory: *14405-5.x.x.x_SOFTWARE_GO_Utilities\Emulator and Accelerator\bin.* If you are using the 64-bit version of LabVIEW, copy the 64-bit folder.



2. Paste the win32 (or win64) directory into the LabVIEW 32-bit (or 64-bit) Instr.lib directory: *C:\Program Files (x86)\National Instruments\LabVIEW 20xx\instr.lib*. In the case of 64 bit, the LabVIEW directory is in *C:\Program Files\National Instruments\LabVIEW 20xx\instr.lib*
3. Rename win32 to LMI, or win64 to LMI.



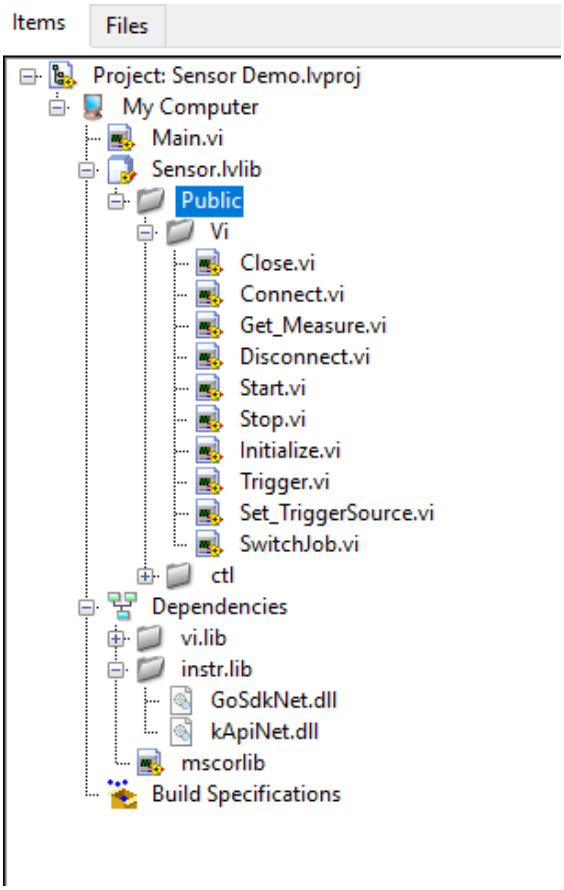
4. Copy the LabVIEW.exe.config file from *14405-5.3.15.52_SOFTWARE_GO_Utilities.zip,* under the *Integration/LabVIEW (5.3+) directory to C:\Program Files (x86)\National Instruments\LabVIEW 2017.*

   For information about why this step is needed, see this article.

5. Close LabVIEW if it's already open.

   A restart of LabVIEW is needed to load the new .NET security settings.

6. Open Gocator Demo.lvproj file in LabVIEW.



7. Open the example VI, *Main_Gocator.vi*, from the project tree.

   Because LabVIEW keeps references of application interfaces (*GoSdkNet.dll* and *kApiNet.dll*), including the path in the project file, the first time you open the project, it will automatically fix missing references and display a message informing you that it automatically replaced reference path of respective DLLs. You should accept the changes. There may also be a warning about loading the DLL's from new sources. These warnings can be reviewed and ignored.

GoSdkNet.dll and kApiNet.dll

The VIs in the Instr.lib folder are building blocks of the Gocator LabVIEW interface. These VIs call functions exposed in the Gocator SDK DLL (*GoSdkNet.dll*). By default, the *GoSdkNet.dll* and *GoSdk.dll* should be in LabVIEW's *Instr.lib* folder.

Setting up the Gocator to Communicate with LabVIEW

To communicate with LabVIEW, the Gocator needs to send Profile or Surface output over Ethernet using the Gocator protocol. The Gocator LabVIEW VIs can receive profile, profile intensity, surface, surface intensity, and measurement data. You enable these outputs on the Output page, Ethernet category, with **Gocator** selected as the protocol (see below).

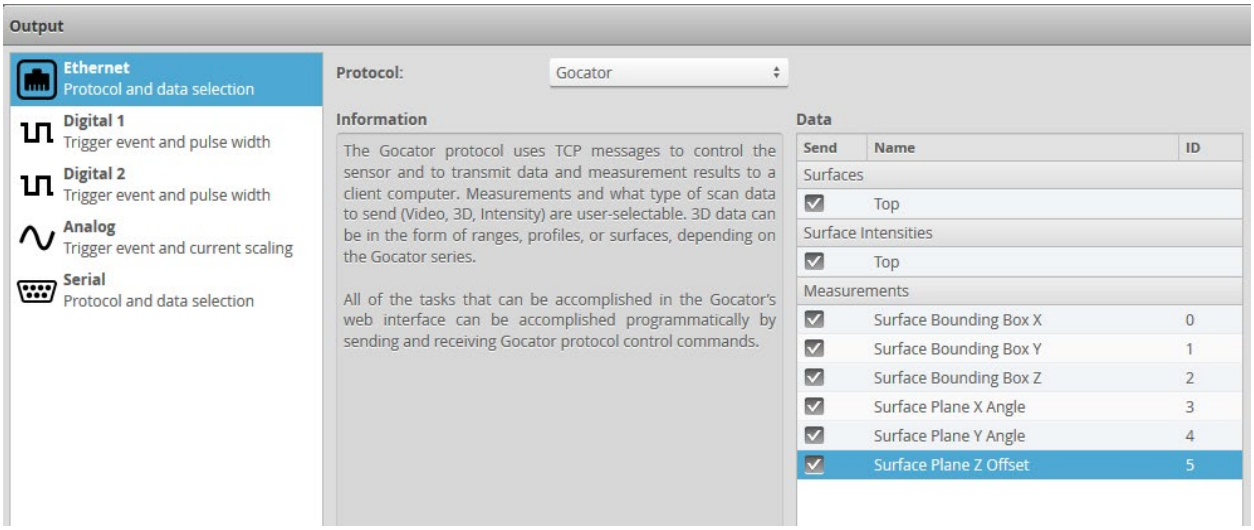


Figure 1. Enabling Output over Ethernet in the Output Page (Surface scan mode shown)

Gocator Demo LabVIEW Example

The *Gocator Demo* project is an example of how to interface and build LabVIEW VIs with a Gocator sensor using *GoSdkNet.dll* API. Open ***Main_Gocator.vi***.

Once LabVIEW loads the main example it opens the demo application front panel containing controls and indicators that display profile or surface data generated by the Gocator in real-time.

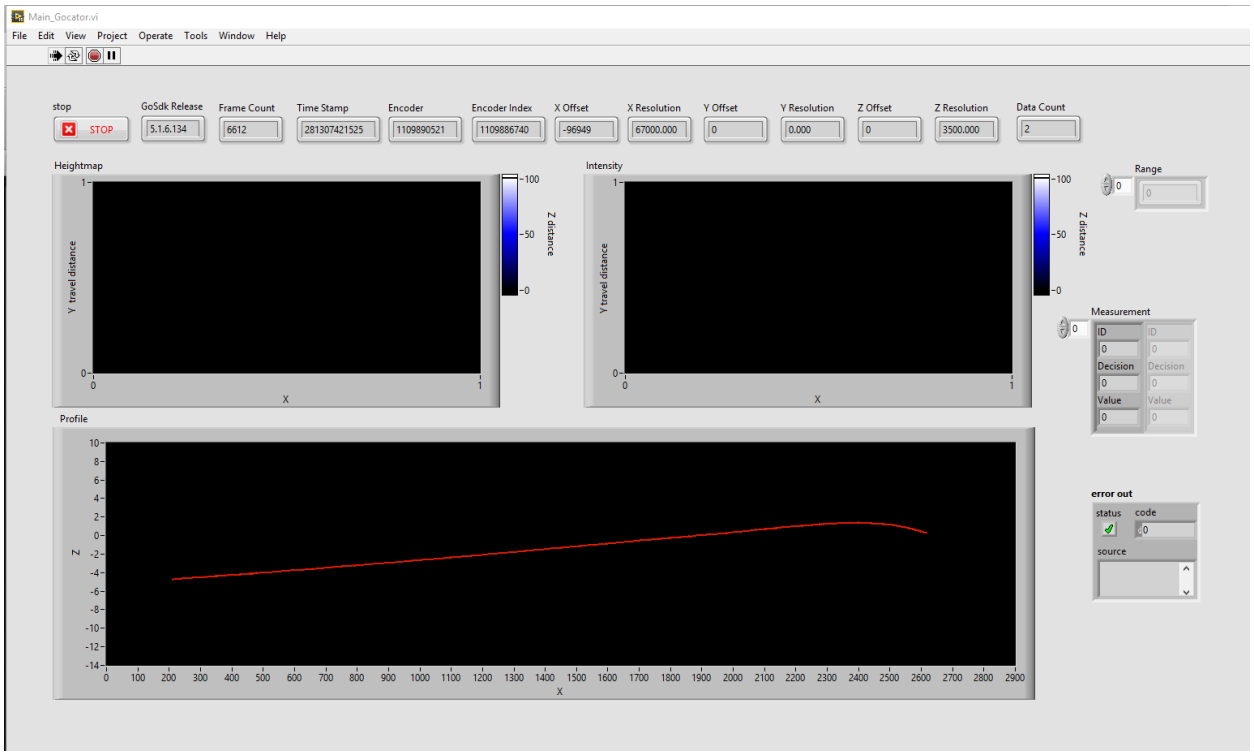
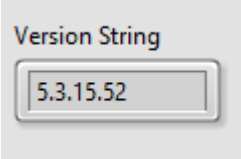


Figure 2. Gocator LabVIEW Main Panel

To confirm GoSdkNet.dll and GoSdk.dll are installed properly:

1. Run *Main_Gocato.vi*
2. The example program will connect to and start all Gocator sensors on the network.
3. The *GoSdk Release* indicator will report the *GoSdk.dll* release version used.

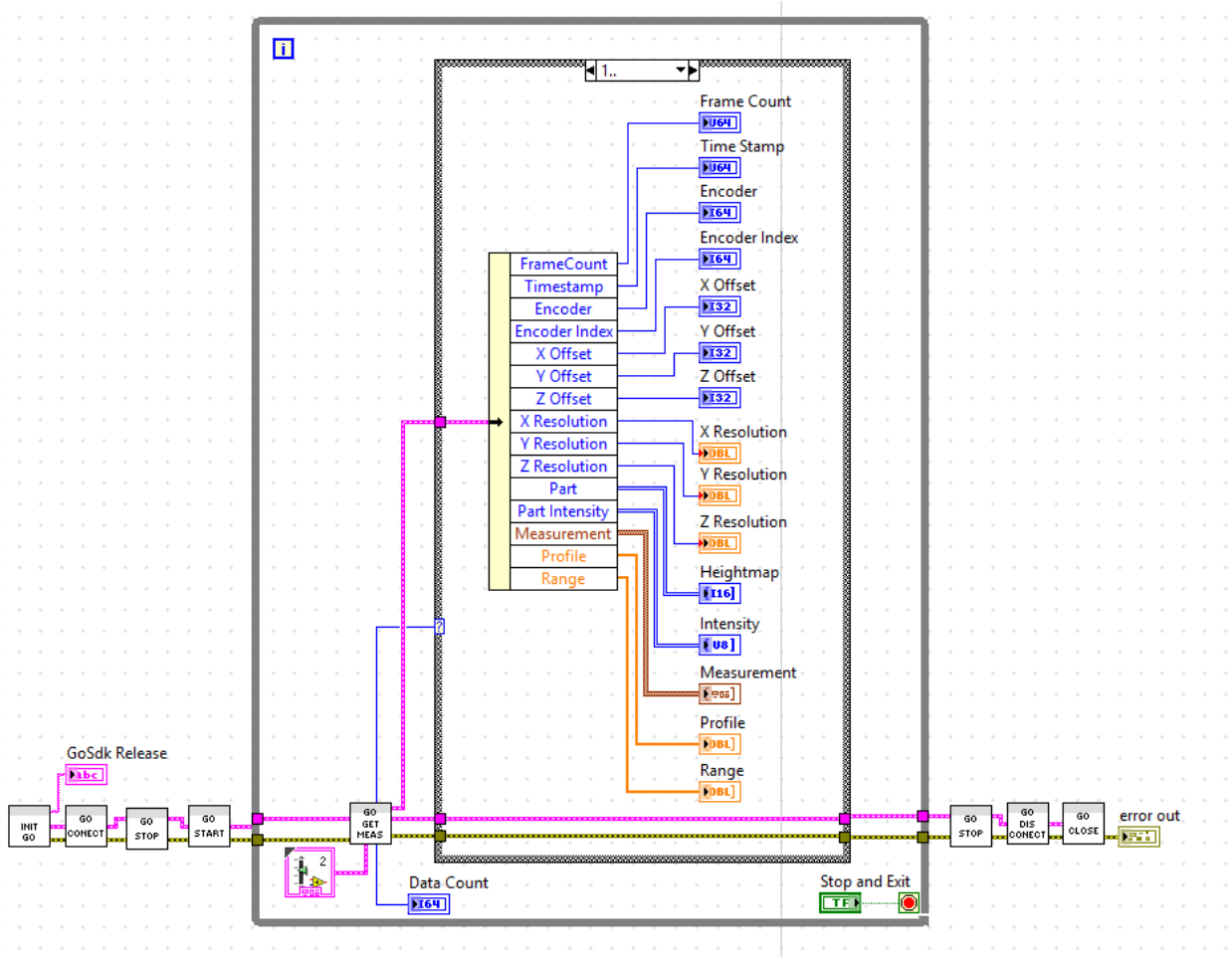


The laser line of sensor(s) should turn on once triggers start. In the browser interface ensure all desired Gocator outputs are enabled, i.e., Profile, Intensity, and Measurements. The front panel numerical indicators and graphical plots should start displaying the received real-time data. Use the **Stop** button to stop the sensor and exit the program.

Structure of the Gocator Demo LabVIEW Example

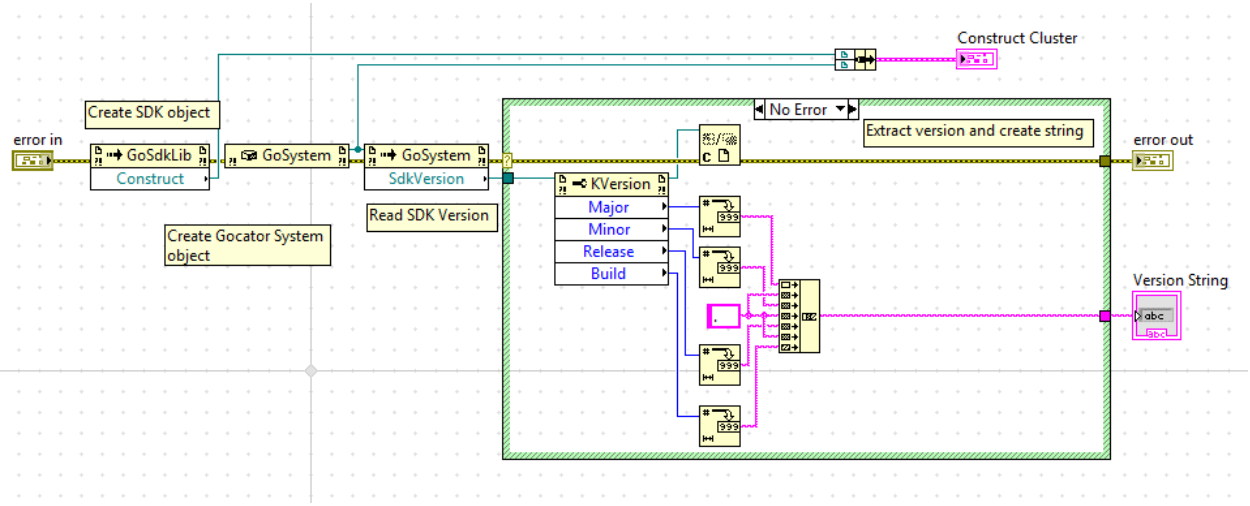
The *Main_Gocator.vi* example has two main sections, Panel Control and Gocator Result Processing.

It uses Gocator public VIs to initialize the Gocator system application interface (the *GoSdk.dll* API), which identifies and connects to all sensors and receives Gocator sensor result messages displaying real-time data.



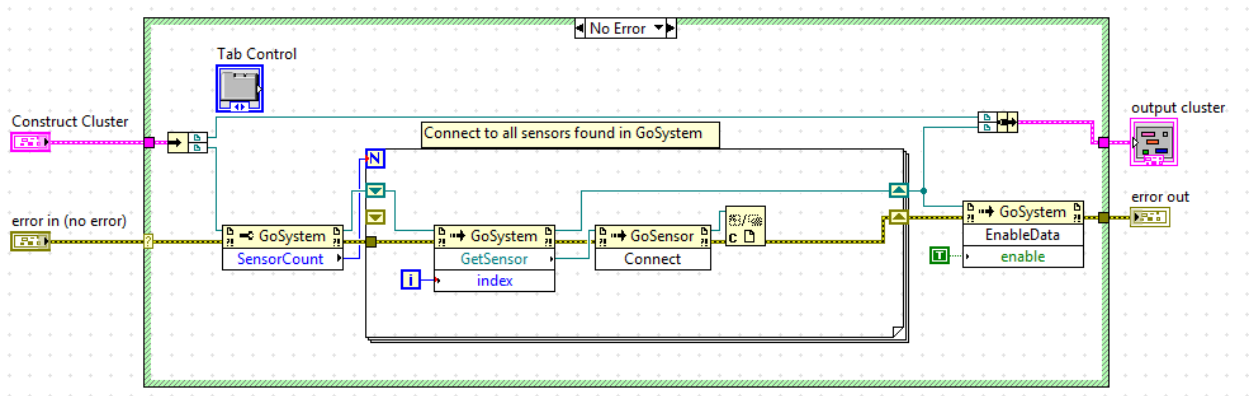
The Panel Control section is a sequence of system Gocator public VIs essential to writing LabVIEW applications using the .NET interface *GoSdkNet.dll* API. *Initialize.vi* is essential to construct *GoSdk* and *GoSystem* objects. *GoSystem* .NET object contains all the methods and properties necessary to interface with Gocator sensors and carry out data acquisition and measurement in real-time.

Initialize.vi



The main purpose of *Initialize.vi* is to set up the Gocator GoSdk API interface and create the GoSystem system object. During construction, a GoSystem object uses the Gocator Discovery Protocol to locate all Gocator sensors on the network. The output of this VI is a cluster containing both GoSdkLib and GoSystem .NET references which are subsequently used throughout the application and disposed of when the application exits. This VI also invokes the *GoSystem->SdkVersion()* method and formats the version into a string.

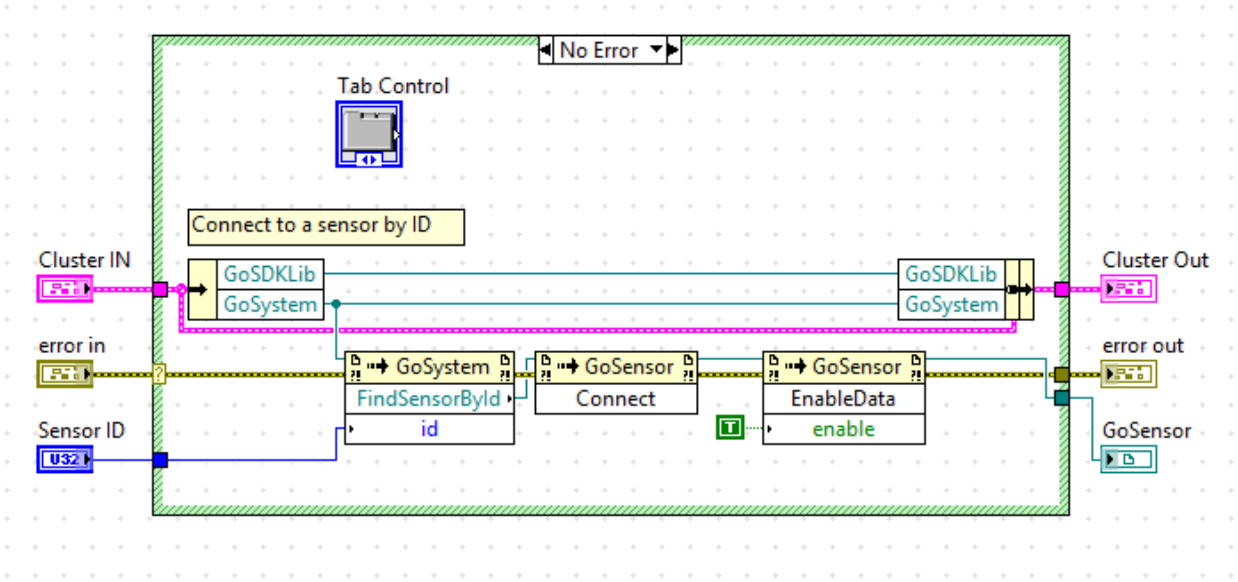
Connect.vi



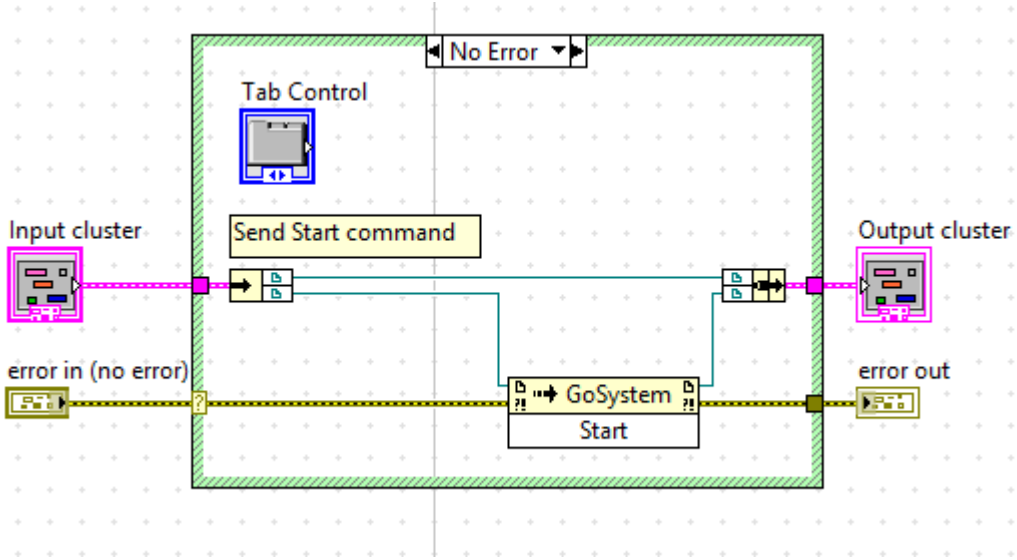
The purpose of this VI is to connect to all the sensors discovered during the construction of the *GoSystem* object in *Initialize.vi*, using the *GoSystem.SensorCount* property, and the *GetSensor(i)* and *Connect()* methods.

ConnectSensor.vi

If you want to connect to a specific sensor, identified by its ID (its serial number), you may want to design your own VI, where you invoke *GoSystem->FindSensorById*, *GoSensor->Connect* and *GoSensor->EnableData*. This will allow an application to start a specific sensor rather than the whole system using the *GoSensor->Start* method. For example:



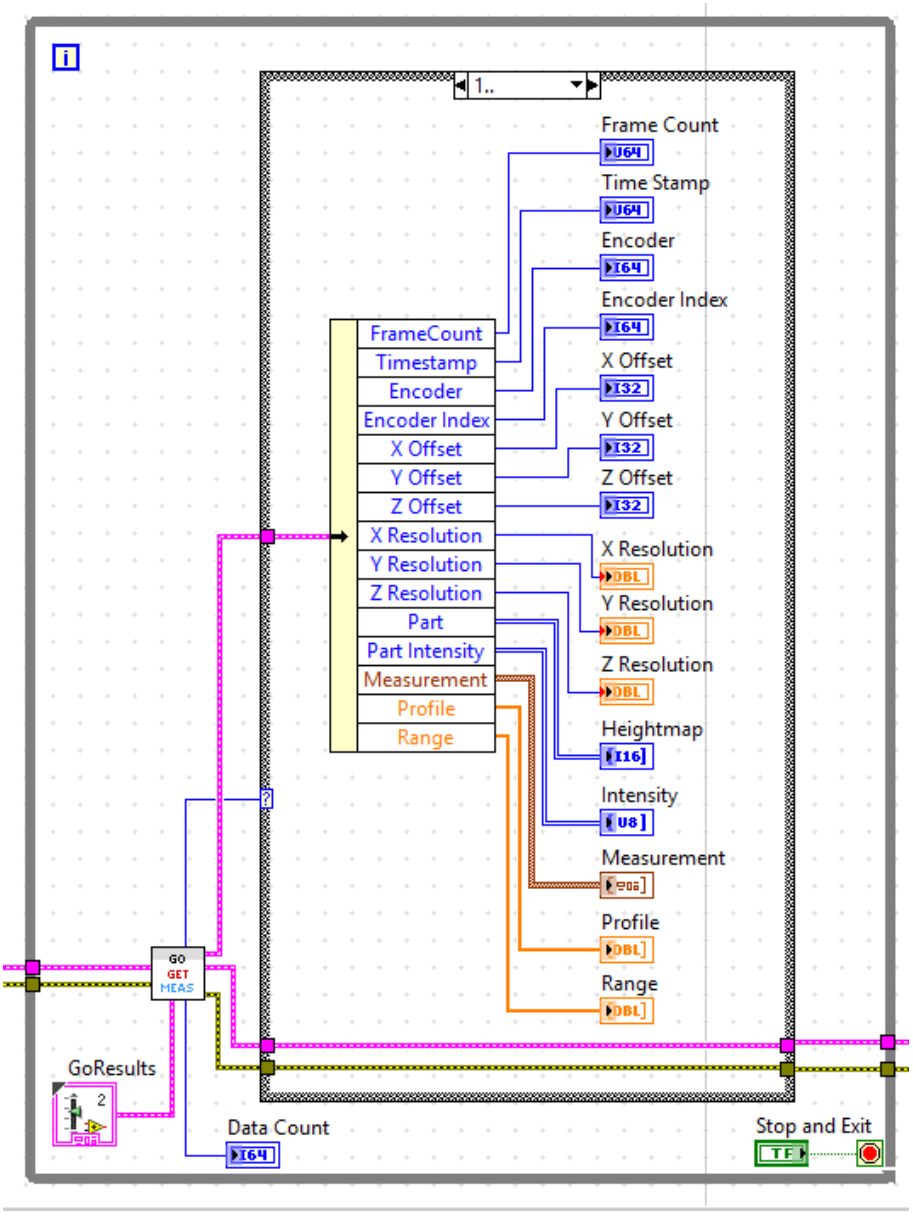
Start.vi (Stop.vi)



These are simple VIs invoking the GoSystem->Start() or ->Stop() method.

Data Acquisition and Display

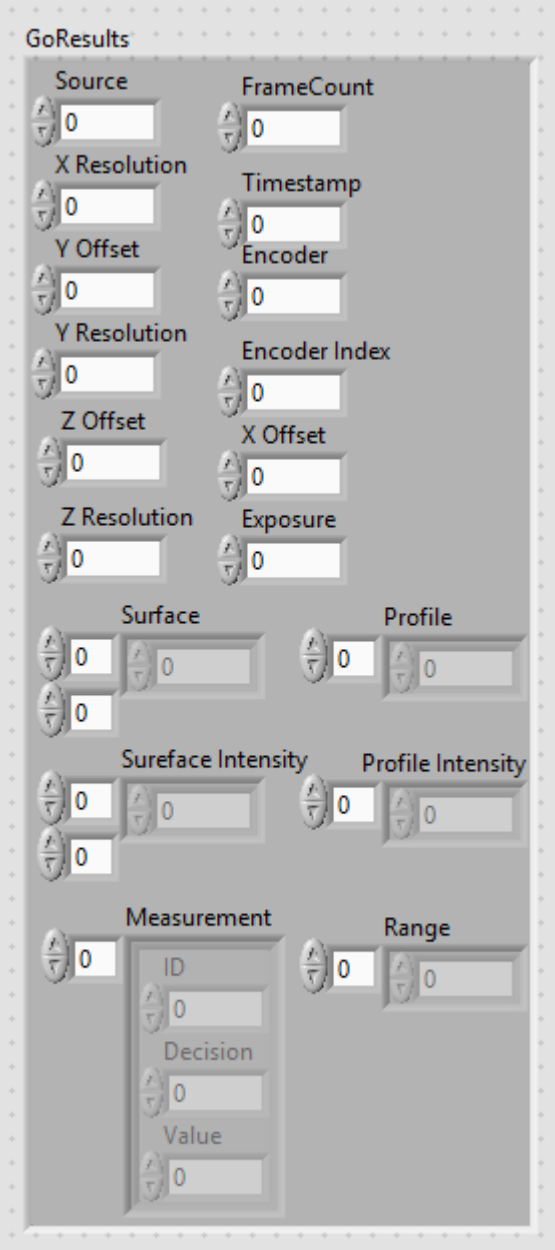
Once a Gocator sensor starts acquiring profiles/surfaces, the example uses a *while* structure that encloses ***Get_Measure.vi***, which queues up a copy of *Gocator Result* for every frame of data it receives. A *Gocator Result* can contain one or more output from the Gocator. Pull the *Gocator Result* data out of the queue periodically (with a while loop) and processes these results. Upon receiving a Gocator result message, a *case* structure parses and displays message properties and data specified in the Gocator output you configured previously in the sensor's web interface, such as profile, surface, measurement data and decisions.



When the application closes, it must properly dispose of previously allocated GoSdk API objects invoking *GoSystem*, *GoSdk* and other created .NET objects, and also close corresponding the LabVIEW references (see the *Close.vi* block diagram).

Gocator Result Cluster

The Gocator Result cluster is the data structure that carries profile, surface, intensity and measurement results between the user block diagrams and Gocator Public VIs.



Gocator Result Cluster

The structure is similar to the *Gocator Data Result* defined in the Gocator Protocol section in the Gocator User Manual.

Field	Type	Definitions
Source	U32	Profile Source of the data. 0 – Main sensor 1 – Buddy sensor 100 – Combined data
Timestamp	U64	Timestamp of the frame (us)
Encoder	I64	Encoder value (ticks)
Encoder Index	I64	Encoder value when the last index is triggered. (ticks)
Frame Count	U64	Frame count (frames).
Exposure	U64	Exposure value (us). This value is only available when profile is received.
X/Y/Z Offset and Resolution	Double	Scale and translation factors for converting pixel coordinates into real-world coordinates. See below for how these values are used.
Profile	1D array of I16	Array of z resampled profile data. Each element is a range value, 0x8000 represents NULL. Z in system coordinate = zOffset + zResolution * ranges[index] X in system coordinate = xOffset + xResolution * index
Profile Intensity	1D array of U8	Array of profile intensity values. Items in the array are arranged in the same order as items in the profile array. A value of 0 indicates no intensity data.
Surface	2D array of I16	2D Array of z resampled profile data. Each element is a range value, 0x8000 represents NULL. Z in system coordinate = zOffset + zResolution * ranges[indexY] [indexX]

		X in system coordinate = xOffset + xResolution * indexX. X in system coordinate = yOffset + yResolution * indexY.
Surface Intensity	2D array of U8	Array of profile intensity values. Items in the array are arranged in the same order as items in the profile array. A value of 0 indicates no intensity data.
Measurements	1D array of Measurement Data cluster	Each Measurement Data carries the result of one measurement tool. A Measurement Data has the following fields: Source – Measurement ID Decision – Measurement decision. 0 if the measurement falls outside of the decision limits or is invalid, 1 if the measurement falls within the decision limits. Value – Value of the measurement. Refer to *Measurement* under *Data Result* section in the Gocator User Manual for the possible values of Type, Source and Value.

Gocator Public VIs

VI	Icons	Explanation
Initialize.vi		Initializes the ***GoSdkLib*** environment and constructs ***GoSystem*** .NET objects. Users must call this VI before using any of the other Gocator public Vis.
Connect.vi		Establishes ***control***, ***data*** and ***health*** TCP connections to all Gocators on the network. This must be called before starting the Gocator.
Disconnect.vi		Closes control, data and health TCP connections to all Gocators on the network.

Start.vi		Start the Gocator system. All Gocators change from the ***Ready*** to ***Run*** state.
Stop.vi		Stop the Gocator system. All Gocators change from ***Run*** to ***Ready*** state.
Close.vi		Disposes of both ***GoSdkLib*** and ***GoSystem*** objects and closes respective LabVIEW .NET references.
Get_Measure.vi		This VI accepts a queue as the input. On every invocation it polls the Gocator for the latest result. If a new result is available, it queues the result. If no results are available, it waits until the next one comes. Internally it is an infinite loop. Wrap a while loop around this VI to receive multiple results.

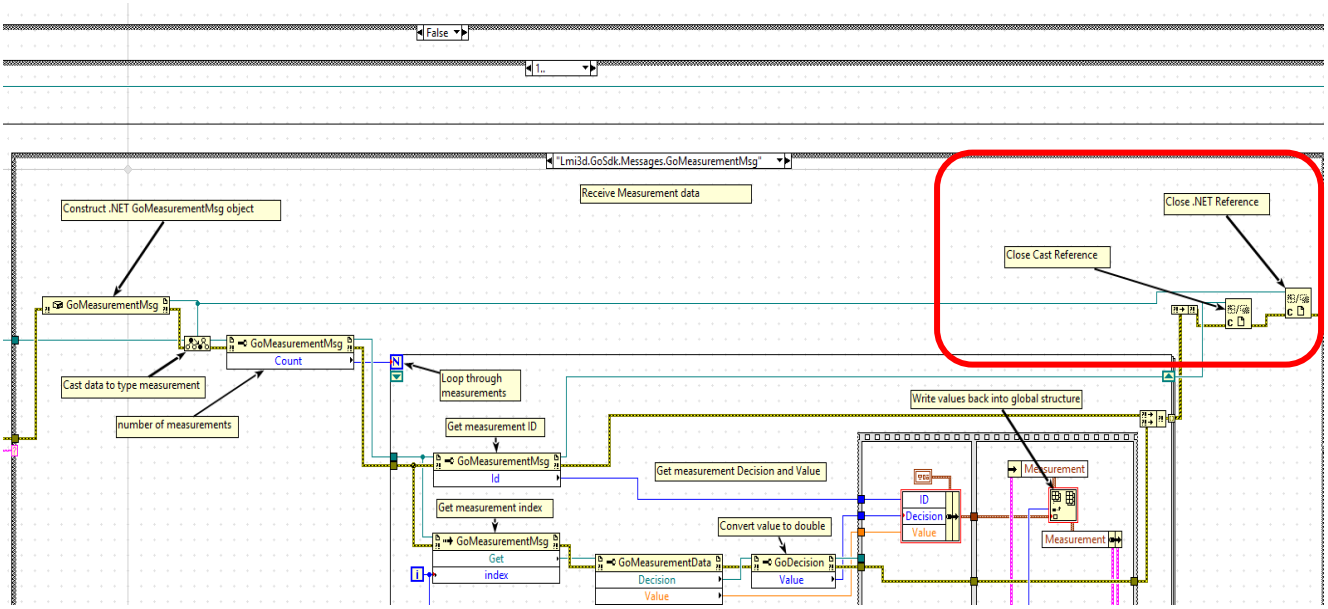
GoSdkNet Objects, Methods, Properties and References

Gocator Public VIs use LabVIEW Connectivity .NET Constructor nodes, Invoke nodes and Property nodes.

For example, to construct a **GoSystem** object, using the mouse, right-click on the block diagram to open the LabVIEW functions, select **Connectivity->.NET->Constructor Node**. This will place a constructor icon in the block diagram and open a dialog box that lets you select a .NET assembly. From the Assemblies drop-down list, select ***GoSdkNet.dll***, double-click on Lmi3d.GoSdk, scroll down the list and find GoSystem object and select the ***GoSystem()*** constructor. To select a *GoSystem* method, access the Invoke .NET node in the same way, and choose the previously created *GoSystem* object as input, which will automatically select the *GoSystem* reference and allow you to select the available GoSystem methods. A similar process applies to selecting *GoSystem* properties. Review the example LabVIEW code for more and the previously described Gocator public VIs (*Initialize.vi* etc.).

IMPORTANT: For proper LabVIEW application execution, make sure you always close both .NET **object references** and any **cast references** when you no longer need them. When you open a reference to an application, project, VI, or other reference source, LabVIEW allocates memory to store that reference. To free up the space in memory where LabVIEW stored the reference source, you must close the reference. It is always safe to close a reference when you no longer need it.

Knowing when to close a VI Server reference and when you can safely leave a reference open can be difficult. However, unless you are certain you can safely leave a reference open, always close references when you no longer need them.



Closing Object Reference and Cast Reference

Application Builder

Gocator Public VIs are a set of Virtual Instruments (VIs) that used the Gocator .NET API, *GoSdkNet.dll* and *kApiNet.dll*. The LabVIEW .NET connectivity functions can be used to invoke the methods and properties of the Gocator sensor system. The Gocator .NET interface supports sensors running 4.x and newer firmware, using the corresponding *GoSdkNet.dll* and *kApiNet.dll* APIs.

To build the Gocator LabVIEW application, use the Gocator Demo project as a template, replacing *Main_Gocator.vi* with your own VI. Make sure to place all of the DLLs from LabVIEW's Instr.lib directory into the Application directory (not a sub-directory). If the LabVIEW application fails to launch on a target PC, it's probably because the application can't find the DLLs. Ensure the DLLs are placed in the application directory. If issues persist, contact LMI support.