SAInstrumentSDK

The SAInstrumentSDK is meant to provide a direct interface between your instrument and SpatialAnalyzer. It uses the same protocols as instrument interfaces written for SA by its creators.

# Prerequisites

* Knowledge of the C++ programming language is necessary.
* Familiarity with MFC (Microsoft Foundation Classes) is highly recommended.
* Visual Studio v.2019 is required.
* A SpatialAnalyzer license, and SpatialAnalyzer installed on the pc you’ll be developing your interface on are require
* The basics of running SA, such as adding an instrument to the job, will be necessary.

# The User Sample

The SAInstSDKUserSample is meant to serve as the primary documentation.

The sample code is in the form of an MFC Dialog Based Application. Its UI is comprised of buttons to perform the common actions that an instrument interface for SpatialAnalyzer needs to perform. The button handlers serve as examples and document the pertinent functions and their signatures for communicating data to SA.

For receiving data and commands from SA, the user sample includes SAMessenger, a class derived from the base Messenger, found in the sdkIncludes folder. In that folder, the Messenger.h defines the exported virtual methods that are overridden in the sample’s SAMessenger. The user sample has a folder, Test Files, which contains sample SA job files and SA MP (Measure Plan) scripts which can be used to test the implementation of the SAMessenger. (More on that later)

Also in the sdkIncludes class are the files CmdLineSocket.h and Instrument.h. CmdLineSocket defines communication protocol methods and variables that Instrument imports and uses, and Instrument.h itself includes both Messenger and CmdLineSocket. Instrument.h is therefore the only header of the 3 that must be included in the app’s pch.

# Deploying the Sample

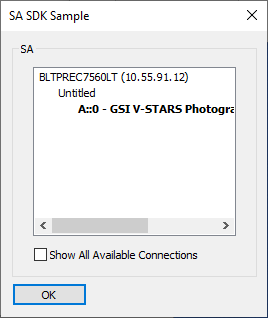
Open the Sample in Visual Studio. Open SAInstSDKUserSampleDlg.cpp. Find the CSAInstSDKUserSampleDlg constructor. Note that the Instrument member, sa, is constructed with a new Messenger object. That is how we let SA know what Instrument to send all messages, commands, and data to when we start using SA MP commands, etc. later on. Now find the method CSAInstSDKUserSampleDlg::Logon(). This is the method that performs the initial connection to SA, and nothing else can happen until that connection is made. Note the call to sa.Logon(…) and the code comment following that line. The first argument, the saIndex integer, is the key. That index tells SA what instrument model/type we are connecting to. The model with that index has to have been added to SA before attempting this connection. As the comment next to the sa.Logon call shows, the index 2 corresponds to the GSI Photogrammetry Instrument. Ensure that your call to sa.Logon has 2 as its saIndex argument and build the sample in the Release U x64 configuration. You’ll notice studio is set to write the output executable to  
SAInstrumentSDK\SAInstSDKUserSample\x64\Release U (64-bit)\SAInstSDKUserSample.exe  
Copy that file to the 64 bit folder of your SA install, for example  
C:\Program Files (x86)\New River Kinematics\SpatialAnalyzer 2023.2.0703.5 (alpha)\x64  
Now find the dll that is deployed with the sdk

SAInstrumentSDK\SAInstSDKUserSample\\_Lib&DLL\SAInstrumentSDK64uvc19.dll

And also copy it to your SA install’s 64 bit folder  
C:\Program Files (x86)\New River Kinematics\SpatialAnalyzer 2023.2.0703.5 (alpha)\x64

And your sdk sample is now deployed in SA.

Open SA and add the GSI V-STARS Photogrammetry System instrument to the job. You’ll see a simple model of a single camera appear in the SA graphics, and in the Instruments node in the SA TreeBar (tree view along the left side of the SA window).

Now double click on your deployed SAInstSDKUserSample.exe. The sample dialog will appear. Click on the [Log On] button and you will see  


This is the SA Instrument Interface LogOn dialog provided by SAInstrumentSDK64uvc19.dll. Note that the GSI V-STARS instrument that you added to SA (A::0 is Collection::InstrumentNumber) is bolded in the dialog window. The highlight means there is an open port in SA (no instrument connected yet) that matches the instrument index you called sa.Logon with as described above. Also note that the dialog’s title is “SA []”, where [] is the interfaceName string which is the second argument you called sa.Logon with.

Select the A::0 – GSI V-STARS… and click OK (or simply double click on the A::0…) and the sample will be connected to SA. Note the camera model in SA will show a green ring around it, and in SA’s tree view, if you expand the Instruments under Collection A, and hover over the 0-GSI V-STARS… indicator, you’ll see “Live” appended.

We could now explore the other buttons in the sample dialog, or more specifically their button click handler methods in the Sample code, which demonstrate various measurement types and the sending of data to SA. But first, we will explore the ways that we can make the sample deployment more like a fully supported instrument interface in SA.

There is a folder in the SA install directory called SAInstrumentSDK.  In that directory, there is a file called SDKInstDefs.xml.

Its default contents serve as a guide to fill in the blanks, and to instruct SA not to show the new instrument anywhere.  Those default contents are:

<sdkInstruments>

<PhotoGramm>

<exeName>

YourExe.exe

</exeName>

<uiName>

yourNameToShowInUI

</uiName>

</PhotoGramm>

<PhotoGramm2>

<exeName>

YourExe.exe

</exeName>

<uiName>

yourNameToShowInUI

</uiName>

</PhotoGramm2>

</sdkInstruments>

In the above state of the xml, there is no mention of the "generic instrument" anywhere in SA.  However, to make sdk interfaces show up in SA,

1. Build the sdk sample app and the SAInstrumentSDK64uvc19.dll in the SA install directory's **\x64** subdirectory, and
2. change the exeName and uiName in the xml file as follows:

<sdkInstruments>

<PhotoGramm>

<exeName>

SAInstSDKUserSample.exe

</exeName>

<uiName>

InstSDK 1

</uiName>

</PhotoGramm>

<PhotoGramm2>

<exeName>

SAInstSDKUserSample2.exe

</exeName>

<uiName>

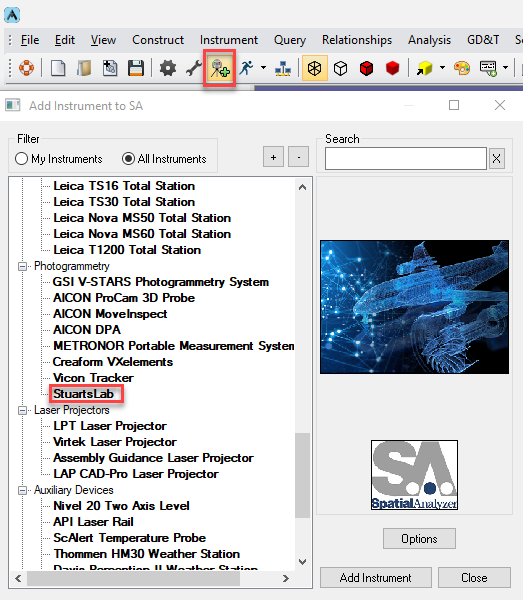
InstSDK 2

</uiName>

</PhotoGramm2>

</sdkInstruments>

then I see the following in the add instrument dialog:



# The Instrument in SA’s UI

O

# The Sample Files

O

Measurements will be triggered by the external start/stop sig- nal on the trigger board. However, the measurement rate will be taken based on internal settings and is not synchronized to an external signal.

* + - **External (External Clock with Start/Stop Signal).** The mea- surement will be controlled by a start/stop signal on the trig- ger board. One transition of the clock signal (positive or nega- tive depends on the configuration) triggers a measurement if the Start/Stop signal is active.

*Start / Stop Active Level*

* + - **Low/High.** The start/stop signal can be set either low or high active (for example, low active means that events are being generated as long as the start/stop signal remains low).

*Start / Stop Source*

* + - **Ignored/Active.** This setting controls the subsequent response to the external trigger after a measurement operation has started. If ignored, the measurement will continue regardless of other triggers until the profile is stopped, while if active, the following trigger changes will start / stop the measurement.

*Clock Transmission*

* + - **Negative/Positive.** This defines the change in clock signal used for the trigger (negative transition or positive transition).

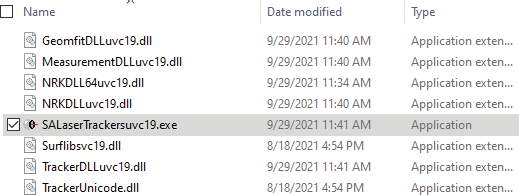
*Minimal Time Delay*

* + - **Delay Value.** This defines the maximum rate at which mea- surements can be taken (minimal delay between two consecu- tive measurements). Additional trigger signals sent faster than this preset delay will be ignored.

# Running the Tracker Interface Separately

One of the unique features about SA’s architecture is that the instru- ment interface can be run separately from SA. This provides a means to run multiple trackers independently on different machines while connect to a single SA for data storage. Doing so also provides the ability to separate the persistence files for individual trackers, as the persistence file will be saved in the directory as where the tracker in- terface is launched, as opposed to the ***C:\Analyzer Data\Persistence*** folder.

In order to run the SA Laser Tracker process separately some addition- al support files are required. These include the following files (Figure 3-102):

**Figure 3-102.** Required Files to run the SA Laser Tracker process independently from SA.

# Additional Connections

The AT960 can be used with a number of peripheral devices. For more information refer to the following quickstart guides:

* + “Hexagon AS1 Scanner” on page 121
  + “Leica Absolute Scanner (LAS) 20-8” on page 128
  + “Leica T-Scan Interface” on page 131