SkySonde Client Plugin System Documentation

Version 1.1, October 12th, 2021

Allen Jordan, allen.jordan@noaa.gov

SkySonde Client has support for adding external instrument plugins written by users. This allows new instruments to be attached to an iMet-1-RSB radiosonde (following the XDATA protocol) and their data will be collected, displayed, and stored with the other radiosonde fields in a CSV file. Any available plugins will be displayed on the configuration screen for enabling/disabling and setup fields:

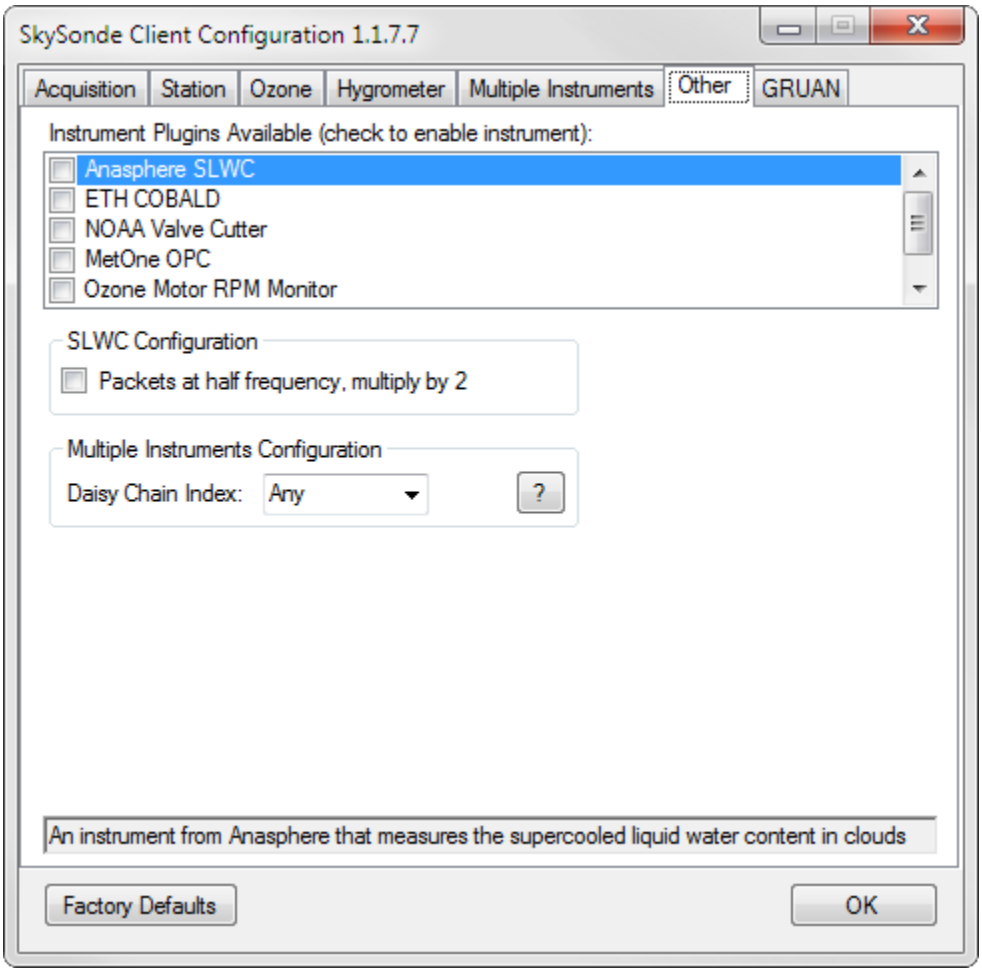


Figure 1: Other Instruments tab in SkySonde Client, showing the plugin instruments available.

Creating a New Plugin

Project Initialization

To create a custom instrument plugin, use Visual Studio 2019 or later. The free "Community" edition works fine:

https://visualstudio.microsoft.com/downloads/

Also check that the latest SkySonde Client is installed from the NOAA OZWV website:

https://gml.noaa.gov/ozwv/wvap/sw.html

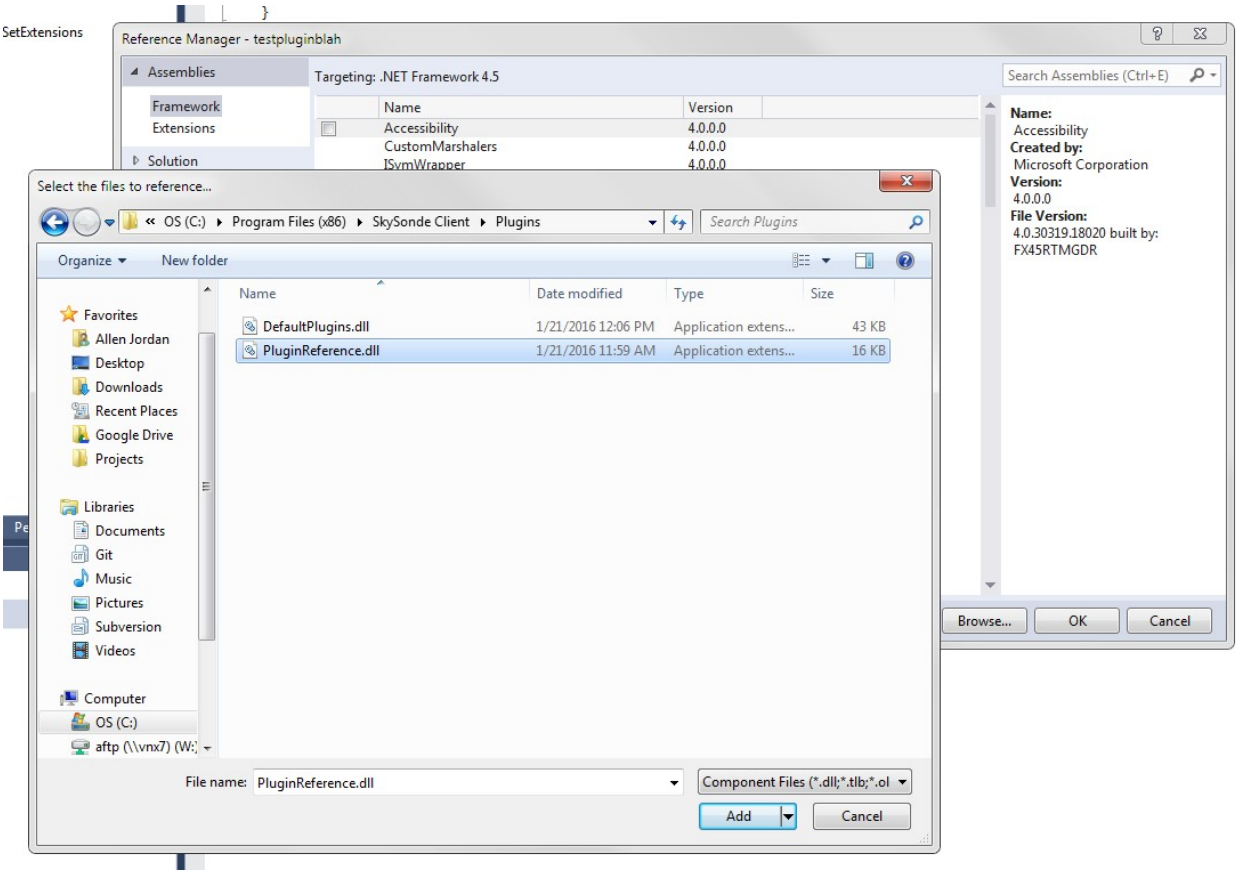
The following instructions are written for Visual Studio 2019 (though some screenshots are from older versions) with the .NET Framework 4.7.2 target framework. You can find example source code for creating a plugin here:

https://gml.noaa.gov/aftp/user/jordan/AllenTestPlugin.zip

I recommend modifying this example project to build a new plugin. Alternatively, to create a new plugin from scratch first start a new C# project in Visual Studio and choose the "Class Library (.NET Framework)" option. Any project name will work. Under the "Configure your new project" page that appears next, change the "Framework" to "**.NET Framework 4.7.2**". If it's not available, you will need to first install the "Developer Pack" version of this framework from here and restart Visual Studio:

https://dotnet.microsoft.com/download/dotnet-framework/net472

From the Solution Explorer, right-click on "References" and choose "Add Reference…". Click the "Browse…" button on the bottom right and navigate to SkySonde Client's installation folder (usually "C:\Program Files (x86)\SkySonde Client"). Now open the "Plugins" folder, select "PluginReference.dll", and press the "Add" button:



We will also need the System.Windows.Forms reference, available from the “Assemblies->Framework” area of the Reference Manager. Press “OK” to close the Reference Manager dialog and you should now see “PluginReference” added to the project’s list of references in the Solution Explorer.

To code the plugin, use the “Class1.cs” file Visual Studio automatically generated for this project. You can rename the file and class if needed. This class will extend the PluginBase class from the PluginReference library we referenced earlier. The basic plugin class template looks like this:

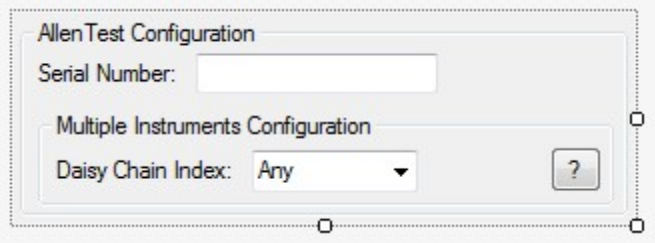
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using PluginReference;

namespace AllenTestPlugin
{
    public class AllenTestPlugin : PluginBase
    {
        /**
         * The name of the plugin's instrument, to be shown in the GUI.
         */
        override public string InstrumentName { get { return "AllenTest"; } }

        /**
         * A sentence or two describing the instrument in more detail.
         */
        override public string InstrumentDescription { get { return "A test instrument by Allen"; } }

        /**
         * Create and return a Windows Forms Panel containing any setup/configuration/metadata controls required by the plugin instrument.
         */
        override public Panel GetConfigPanel()
        {

        }

        /**
         * After the user has finished entering config values in the GUI and pressed "OK", this method will be called.
         * The plugin should parse its own config panel controls and store the results.
         *
         * @param selectedDaisyChainIndex       The user-selected daisy chain index for this instrument in case multiple duplicate instruments are attached.
         *                                      A value of 0 means to allow all daisy chain indices, no particular instrument has been selected so there is onl
         */
        override public void ParseConfigPanel(uint selectedDaisyChainIndex)
        {

        }
```

Plugin Configuration Control

Any setup information or metadata needed by your instrument will be collected in SkySonde Client's configuration dialog. You need to write a user control containing any fields required. Start by adding a new User Control to the project, naming it something like PluginConfig.cs. I start by placing a group box around the perimeter using the Windows Forms GUI designer, anchoring it to all sides, and give it some text describing the instrument like "AllenTest Configuration". Then any fields needed by the instrument are added.

Every plugin instrument should also add the "MultipleInstrumentsControl" user control to the bottom of their configuration area. This will allow more than one of the same plugin instrument to be attached to a single iMet, using a separate instance of SkySonde Client for each (and selecting the specific daisy chain index with this user control). To see this control in the Windows Forms designer, open the Toolbox and right-click in an unused area. Select "Choose Items…" and click the "Browse…" button. Navigate to the same SkySonde Client installation directory then the Plugins subdirectory and again select "PluginReference.dll" and press "Open". Now "MultipleInstrumentsControl" should be selected in the "Choose Toolbox Items" dialog. Press OK.

Find "MultipleInstrumentsControl" in the ToolBox and place it somewhere in the configuration control area. The final configuration control should look something like this:



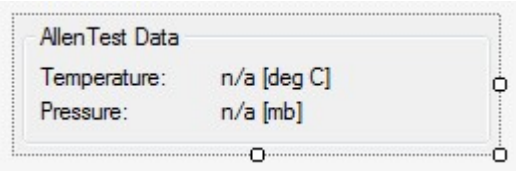
Now go to the code view for this user control. You need to write properties to access each of the controls:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AllenTestPlugin
{
    public partial class AllenTestConfig : UserControl
    {
        public int DaisyChainIndex
        {
            get { return multipleInstrumentsControl1.DaisyChainIndex; }
            set { multipleInstrumentsControl1.DaisyChainIndex = value; }
        }

        public string SerialNumber
        {
            get { return serialNumberTextBox.Text; }
            set { serialNumberTextBox.Text = value; }
        }

        public AllenTestConfig()
        {
            InitializeComponent();
        }
    }
}
```

Plugin Data View Control

We will also need a way to view incoming instrument data in real-time during a flight. Create a new User Control in the same way as the configuration control made in the last section. Name it something like “PluginDataView.cs”. I wrap a group box around everything and anchor it to

the sides again, making a nice border, and set the text to “Plugin Data” (using the instrument name). My example looks like this:



In the code view, we will need an UpdateData method and delegate, allowing SkySonde’s data collection thread to safely update the GUI fields as packets are parsed. It should look something like this:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AllenTestPlugin
{
    public partial class AllenTestDataView : UserControl
    {
        public AllenTestDataView()
        {
            InitializeComponent();
        }

        private delegate void UpdateDataDelegate(double temperature, double pressure);

        public void UpdateData(double temperature, double pressure)
        {
            if (this.InvokeRequired)
            {
                this.BeginInvoke(new UpdateDataDelegate(UpdateData), new object[] { temperature, pressure });
            }
            else
            {
                temperatureLabel.Text = string.Format("{0:0.00} [deg C]", temperature);
                pressureLabel.Text = string.Format("{0:0.00} [mb]", pressure);
            }
        }
    }
}
```

Note the use of BeginInvoke to safely call the UpdateData method again on the GUI thread instead of the data collection thread.

Writing the Plugin Code

The plugin code should contain instances of the configuration and data view controls we wrote earlier, and member variables to store any metadata and packet data fields. It should also

contain a separate Panel for each user control, set to null. The top of the class should look like this:

```
private AllenTestConfig configControl;
private Panel configPanel = null;
private string serialNumber;
private int daisyChainIndex;

private AllenTestDataView dataViewControl;
private Panel dataViewPanel = null;
private double temperature, pressure;
```

The following override methods/properties need to be written for a plugin to work. Note that the System.Xml and System.Xml.Linq namespaces should be included (with “using” statements) at the top of the plugin class’ file for some of the methods below.

- InstrumentName
  - The name of the plugin's instrument, to be shown in the GUI.
- InstrumentDescription
  - A sentence or two describing the instrument in more detail.
- GetConfigPanel
  - Create and return a Windows Forms Panel containing any setup/configuration/metadata controls required by the plugin instrument.
  - When called for the first time, initialize the configPanel. For future calls, just return the existing configPanel.
  - AutoSize is set so SkySonde can size it as needed
- ParseConfigPanel
  - After the user has finished entering config values in the GUI and pressed "OK", this method will be called.
  - The plugin should parse its own config panel controls and store the results.
- GetDataViewPanel
  - Create and return a Windows Forms Panel for displaying real-time data from the plugin's instrument. The ParsePacket method should update the panel's controls as data comes in.
  - When called for the first time, initialize the dataViewPanel. For future calls, just return the existing dataViewPanel.
  - DocStyle.Fill is used for docking the data view panel within SkySonde Client’s plugin data window.
- OutputRawconfigXML
  - Output XML elements using XmlTextWriter's WriteElementString method for saving any of the plugin instrument's metadata fields in the flight's rawconfig file.
  - Use XMLWriter’s WriteElementString method to output each of your metadata fields
  - Prefix your xml element names with the instrument name to keep them unique

- ParseRawconfig
    - For reprocessing flights, this method should parse the rawconfig xml file to restore any of the plugin's required metadata fields, and update the config GUI panel.
    - This should also set the plugin's "Enabled" property if an appropriate xml field is located for the plugin's instrument.
- ParsePacket
    - Parse a real-time data packet if it matches the plugin's expected format, and display the results on the config control.
    - This could be (but currently isn't) called from an alternate thread, so use BeginInvoke when updating the data view GUI panel.
    - Use the PluginHelper class from PluginReference for IntFromMSBHexString
- OutputCSVMetadataLines
    - Output instrument metadata lines to the top of the CSV file in the format:
        - name1:, value1
        - name2:, value2
- OutputCSVHeaderSegment
    - Output CSV header field names for the plugin's output fields. Please include units in square brackets at the end of the name. Always start with a leading comma, and finish without a comma. Example:
        - , fieldname1 [units1], fieldname2 [units2]
- OutputCSVRowSegment
    - Output the plugin's data matching the supplied UTC date time in a partial CSV row (or just output the latest received data).
    - The resulting string should be comma separated and begin with a comma, like this:
        - , data1, data2, data3

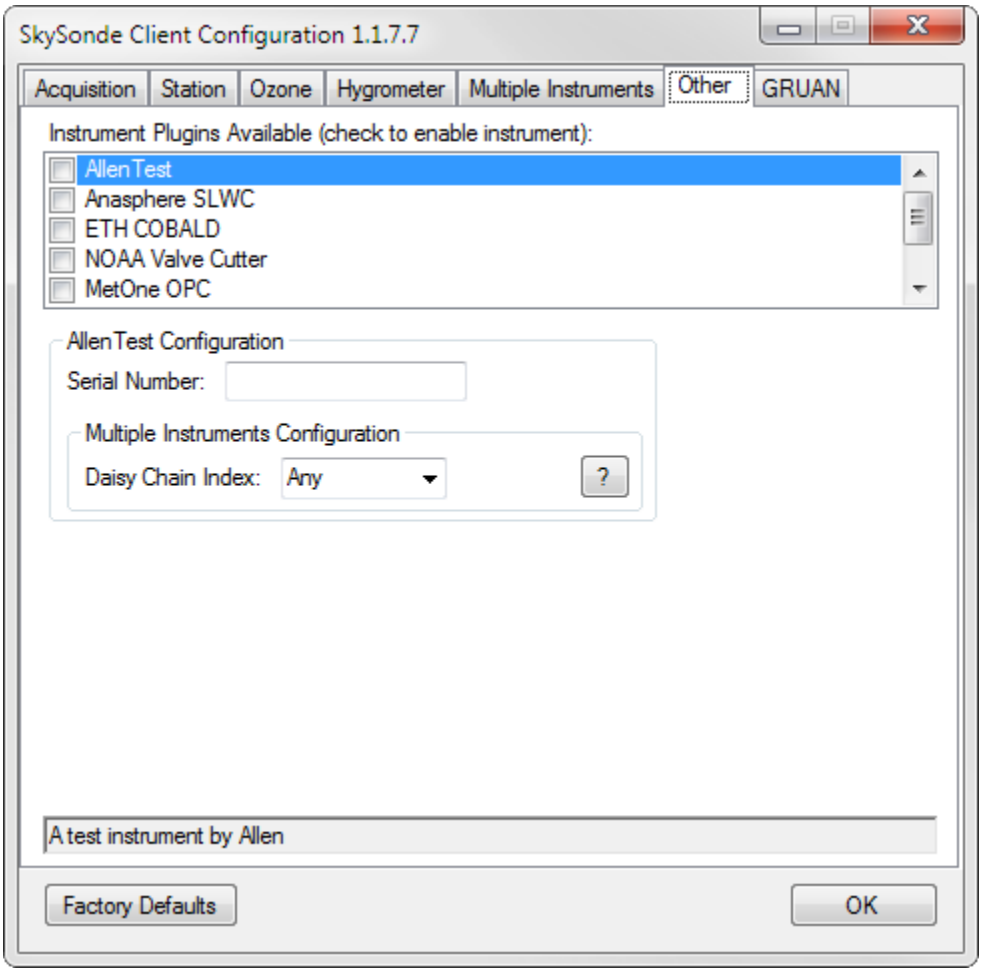
Building and Using

Now the plugin should compile without errors. Make sure the project is targeting .NET Framework 4.7.2 (check this under the project properties). Building the solution will produce a .dll file with your plugin's name in the build output directory. The "Release" build is preferable (change Visual Studio from Debug to Release build mode and recompile). My .dll file is located in the folder:

C:\Users\jordan\Documents\Visual Studio 2013\Projects\AllenTestPlugin\AllenTestPlugin\bin\Release

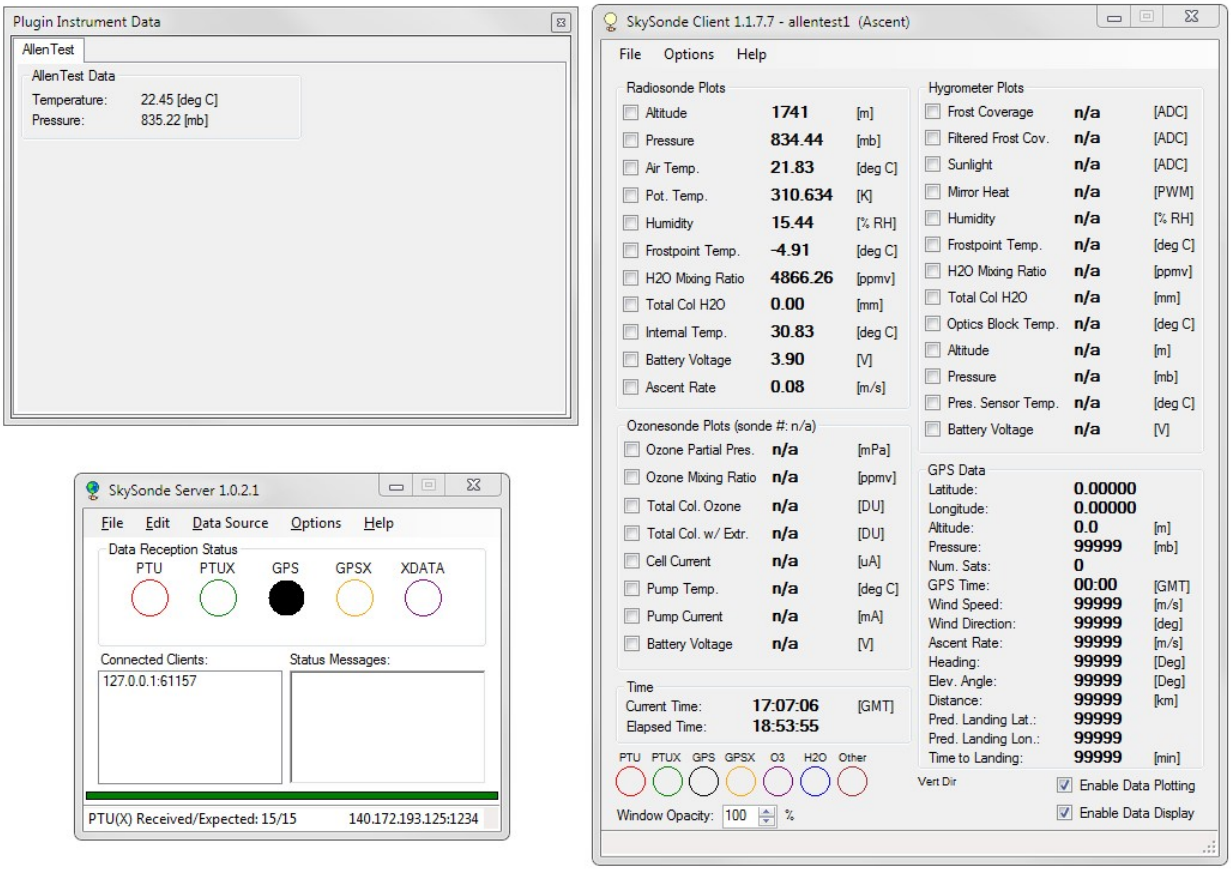
Copy your plugin's dll file into SkySonde Client's Plugins folder. On my machine, this is here: C:\Program Files (x86)\SkySonde Client\Plugins

Now open (or re-open, if already running) SkySonde Client. The “Other” tab should display your new plugin and its configuration control/panel:



After checking the box by the plugin name and typing in a serial number, then setting up the rest of the SkySonde Client config fields, pressing OK will show the plugin data window.

After connecting the plugin instrument to an iMet and starting transmission, the data are displayed and saved to the output CSV file:



This demo project is available for download on the NOAA OZWV Software Download website.