



ATFX SIGNAL READER API (C#, PYTHON, MATLAB, LABVIEW)

May 16, 2022
Document ver. 2.5
© Crystal Instruments Corporation

Contents

ATFX SIGNAL READER API (C#, PYTHON, MATLAB, LABVIEW)	4
ATFX API PACKAGE	7
Package Contents	7
How to Install the ATFX API.....	7
ATFX API C# CODE EXAMPLES	8
Opening a ATFX File – Start Here	8
What is a Recording vs. Signal?	8
Finding the Signal for a particular channel.....	9
What is a Frame?	10
An end-to-end code example	11
Additional File Components - .TS and .GPS	12
Opening a TS or GPS File	13
Reading the Record Properties.....	13
Calling individual property	14
GetProperties.....	15
Reading the GPS Data	15
Extracting the Date and Time of a Recording	18
Reading the Channel Table Data	20
Reading the Signal Properties.....	21
Using a List to Store and Recall Signals.....	22
Basic Signal Information	23
Advance Signal Information.....	25
Advance Generated Time	28
Reading the Data Values of a Signal Frame	30
Reading other Signal Parameters	32
Reading Merge Information.....	36
ATFX API FUNCTION LIST	38
List of Available Modules	38
Recording Manager Module	38
ODS Recording Module	40
ODS Signal Module.....	42
DateTimeNano Module	45
Property Glossary.....	46
RecordingProperty	46

SignalProperties	47
NVHParameterSet Parameter Keys	48
AoEnvironment.....	49
NVHMeasurement	50
NVHEnvironment.....	50
ATFX API CODING LANGUAGES	51
C# Demo Program	51
Python Demo Script.....	57
Importing C# DLL files	57
Python Script Code Example	58
LabVIEW Demo Script.....	62
Importing C# DLL files	62
LabVIEW Block Diagram Example	63
Matlab Demo Script.....	66
Importing C# DLL files	66
Matlab Script Code Example	67
POST ANALYSIS SOFTWARE INTEGRATES ATFX API	68
The Feature that Utilizes ATFX Reader API in PA Software	69
END USER LICENSE AGREEMENT FOR CRYSTAL INSTRUMENTS SOFTWARE	71

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form, for any purpose, without the written permission of Crystal Instruments Corporation (“Crystal Instruments”).

By installing, copying or using the Software, the user agrees to be bound by the terms of the Crystal Instruments End User License Agreement which is a legally binding agreement between the user (“the Licensee”) and Crystal Instruments for the Crystal Instruments software, which includes software components, tools, and written documentation (“Software”).

Crystal Instruments makes no warranties on the Software, whether express or implied, nor implied warranties of merchantability or fitness for a particular purpose. Crystal Instruments does not warrant your data, that the software will meet your requirements, or that the operation will be reliable or error free. The Licensee of the Software assumes the entire risk of use of the Software and the results obtained from the use of the software. Crystal Instruments shall not be liable for any incidental or consequential damages, including loss of data, lost profits, the cost of cover, or other special or indirect damages.

Copyright © 2005-2022 Crystal Instruments Corporation. All rights reserved.

All trademarks and registered trademarks used herein are the property of their respective holders.

ATFX Signal Reader API (C#, python, matlab, LabView)

The Crystal Instruments (CI) ATFX ODS Signal Reader Application Programming Interface (API) consists of 2 Windows Dynamic-Linked Libraries (DLL) providing third-party applications an interface to access the signal data stored in the ASAM Transport Format XML (ATFX) files.

ATFX files are formatted according to the Association for Standardization of Automation and Measuring Systems (ASAM) Open Data Services (ODS) standardization. This is a standard dedicated for storing vibration data and its different forms. CI software natively stores its data using the ATFX format, for both signals and recordings.

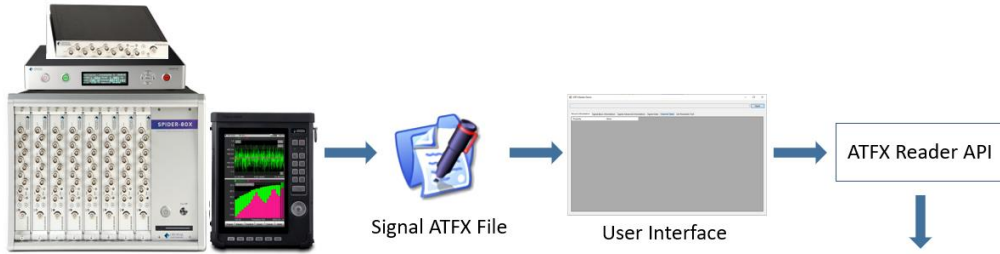
For details about the ATFX ODS format please refer to the official website:

<https://www.asam.net/standards/detail/ods/wiki/>

The .atfx files are xml-based files which store the signal data along with all the attributes of the signal data including data and time or recording, length of recording, number of channels, channel parameters (e.g., input channel sensor and sensitivities), geographic coordinates, sampling rate, high pass filter, etc. The .atfx files are well-defined for storing both raw time data as well as processed spectral data, calculated from methods including Fourier Transform, Frequency Response Functions, Cross-Power Spectrum, Octave Spectrum, etc.

There are 2 additional file types that the .atfx file references that contains the raw data: .ts and .gps. The .ts file is a TimeStamp recording that contains an accurate measure of when a recording was saved with accuracy down to nanoseconds. The .gps file is a GPS recording that contains locational data of where a recording was saved (e.g., latitude, longitude, altitude).

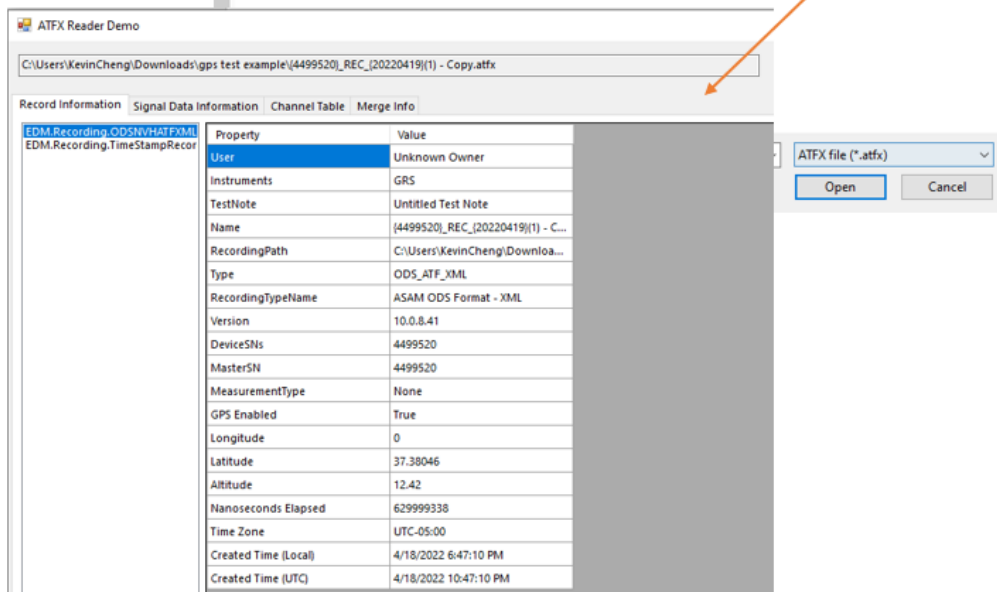
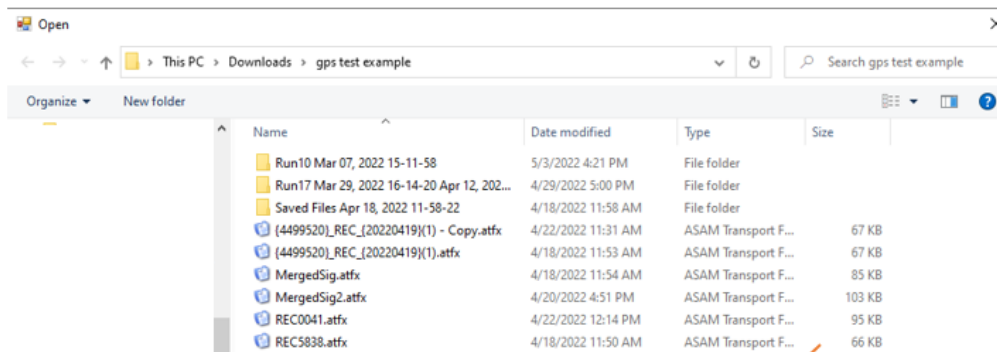
The Signal Reader API provides end-users with a streamlined file reading and browsing library to decode ATFX, TS and GPS files. Users can integrate the API with their own custom developed application. Currently, we support Windows-based programs, ideally written in C#. The same API also supports Python, MatLab and LabView.



CI Data Acquisition Devices

Record Information	Signal Data Information	Signal Data	Channel																																																								
<table border="1"> <tr><td>Device</td><td>GR5</td></tr> <tr><td>MeasurementType</td><td>VOL,Position</td></tr> <tr><td>SignalType</td><td>Time</td></tr> <tr><td>LocationName</td><td>1110131_31319.MA</td></tr> <tr><td>Signature</td><td>BlockC10</td></tr> <tr><td>SamplingRate</td><td>1.111MHz</td></tr> <tr><td>Resolution</td><td>100</td></tr> </table>	Device	GR5	MeasurementType	VOL,Position	SignalType	Time	LocationName	1110131_31319.MA	Signature	BlockC10	SamplingRate	1.111MHz	Resolution	100	<table border="1"> <tr><td>Device</td><td>GR5</td></tr> <tr><td>MeasurementType</td><td>VOL,Position</td></tr> <tr><td>SignalType</td><td>Time</td></tr> <tr><td>LocationName</td><td>1110131_31319.MA</td></tr> <tr><td>Signature</td><td>BlockC10</td></tr> <tr><td>SamplingRate</td><td>1.111MHz</td></tr> <tr><td>Resolution</td><td>100</td></tr> </table>	Device	GR5	MeasurementType	VOL,Position	SignalType	Time	LocationName	1110131_31319.MA	Signature	BlockC10	SamplingRate	1.111MHz	Resolution	100	<table border="1"> <tr><td>Device</td><td>GR5</td></tr> <tr><td>MeasurementType</td><td>VOL,Position</td></tr> <tr><td>SignalType</td><td>Time</td></tr> <tr><td>LocationName</td><td>1110131_31319.MA</td></tr> <tr><td>Signature</td><td>BlockC10</td></tr> <tr><td>SamplingRate</td><td>1.111MHz</td></tr> <tr><td>Resolution</td><td>100</td></tr> </table>	Device	GR5	MeasurementType	VOL,Position	SignalType	Time	LocationName	1110131_31319.MA	Signature	BlockC10	SamplingRate	1.111MHz	Resolution	100	<table border="1"> <tr><td>Device</td><td>GR5</td></tr> <tr><td>MeasurementType</td><td>VOL,Position</td></tr> <tr><td>SignalType</td><td>Time</td></tr> <tr><td>LocationName</td><td>1110131_31319.MA</td></tr> <tr><td>Signature</td><td>BlockC10</td></tr> <tr><td>SamplingRate</td><td>1.111MHz</td></tr> <tr><td>Resolution</td><td>100</td></tr> </table>	Device	GR5	MeasurementType	VOL,Position	SignalType	Time	LocationName	1110131_31319.MA	Signature	BlockC10	SamplingRate	1.111MHz	Resolution	100
Device	GR5																																																										
MeasurementType	VOL,Position																																																										
SignalType	Time																																																										
LocationName	1110131_31319.MA																																																										
Signature	BlockC10																																																										
SamplingRate	1.111MHz																																																										
Resolution	100																																																										
Device	GR5																																																										
MeasurementType	VOL,Position																																																										
SignalType	Time																																																										
LocationName	1110131_31319.MA																																																										
Signature	BlockC10																																																										
SamplingRate	1.111MHz																																																										
Resolution	100																																																										
Device	GR5																																																										
MeasurementType	VOL,Position																																																										
SignalType	Time																																																										
LocationName	1110131_31319.MA																																																										
Signature	BlockC10																																																										
SamplingRate	1.111MHz																																																										
Resolution	100																																																										
Device	GR5																																																										
MeasurementType	VOL,Position																																																										
SignalType	Time																																																										
LocationName	1110131_31319.MA																																																										
Signature	BlockC10																																																										
SamplingRate	1.111MHz																																																										
Resolution	100																																																										

Signal Data



Open

This PC > Downloads > gps test example

Search gps test example

Organize New folder

Name	Date modified	Type	Size
Run10 Mar 07, 2022 15-11-58	5/3/2022 4:21 PM	File folder	
Run17 Mar 29, 2022 16-14-20 Apr 12, 202...	4/29/2022 5:00 PM	File folder	
Saved Files Apr 18, 2022 11-58-22	4/18/2022 11:58 AM	File folder	
(4499520)_REC_(20220419)(1) - Copy.ts	4/18/2022 11:52 AM	TypeScript File	1 KB
(4499520)_REC_(20220419)(1).ts	4/18/2022 11:52 AM	TypeScript File	1 KB
MergedSig_ch1.ts	4/18/2022 11:52 AM	TypeScript File	1 KB
MergedSig_ch2.ts	4/18/2022 11:50 AM	TypeScript File	1 KB
REC5838.ts	4/18/2022 11:50 AM	TypeScript File	1 KB

ATFX Reader Demo

C:\Users\KevinCheng\Downloads\gps test example\4499520_REC_(20220419)(1) - Copy.ts

Record Information Signal Data Information Channel Table Merge Info

Property	Value
User	Unknown User
Instruments	GRS
TestNote	Untitled Test Note
Name	(4499520)_REC_(20220419)(1) - C...
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	TimeStamp
RecordingTypeName	Time Stamp Format
Version	10.0.8.44
MasterSN	0
MeasurementType	None

EDM.Recording.TimeStampRecord

TS file (*.ts)

Open Cancel

Open

This PC > Downloads > gps test example

Search gps test example

Organize New folder

Name	Date modified	Type	Size
Run10 Mar 07, 2022 15-11-58	5/3/2022 4:21 PM	File folder	
Run17 Mar 29, 2022 16-14-20 Apr 12, 202...	4/29/2022 5:00 PM	File folder	
Saved Files Apr 18, 2022 11-58-22	4/18/2022 11:58 AM	File folder	
REC0041.gps	4/22/2022 12:10 PM	GPS File	1 KB

ATFX Reader Demo

C:\Users\KevinCheng\Downloads\gps test example\REC0041.gps

Record Information Signal Data Information Channel Table Merge Info

Property	Value
User	Unknown User
Instruments	CoCo
TestNote	Untitled Test Note
Name	REC0041
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	GPS
RecordingTypeName	GPS Format
Version	10.0.8.44
MasterSN	0
MeasurementType	None

EDM.Recording.GPSRecording

GPS file (*.gps)

Open Cancel

ATFX API Package

Package Contents

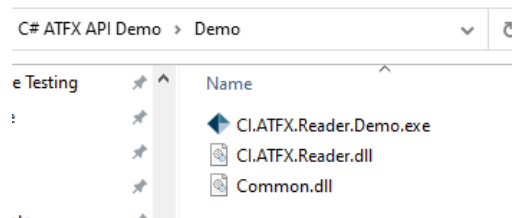
Crystal Instruments will provide a **zip file** that contains the following:

1. API DLL files
2. API user interface demo program - An executable file that calls ATFX reader API dlls to access information stored in Crystal Instruments ATFX files
 - a. Demo program source code written in C#, Python, LabVIEW and Matlab
3. API technical documents
 - a. API Class Methods Library
 - b. API Assembly Documentation

How to Install the ATFX API

Extract and place the zip file content anywhere on the computer. And the dll files can be moved anywhere, so long any custom scripts know the exact file path location of those dll files.

In order for the C# demo program to work, ensure the folder contains the CI.ATFX.Reader Demo file, CI.ATFX.Reader.dll, and Common.dll.



For the Python and Matlab scripts to work, please edit the scripts and change the file path location to point to the dll and recording files.

It is recommended to use Matlab version **R2021b** or later.

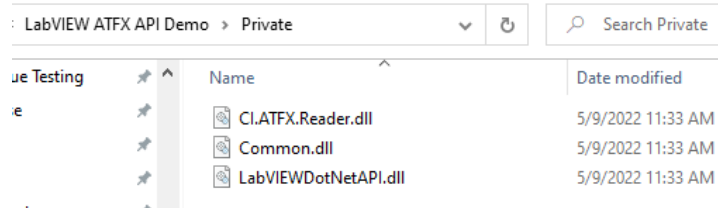
```
7  #---Pythonnet clr import
8  import clr
9  parentPath = "C:\\Users\\KevinCheng\\ATFX API Package v1.2\\"
10
11  clr.AddReference(parentPath + "CI.ATFX.Reader.dll")
12  clr.AddReference(parentPath + "Common.dll")

```

```
99  recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
100 recordingPathRegular = recordingPath + "SIG0020.atfx"
101 recordingPathTS = recordingPath + "{4499520}_REC_{20220419}(1).atfx"
102 recordingPathGPS = recordingPath + "REC0041.atfx"
```

```
1  % Copyright (C) 2022 by Crystal Instruments Corporation. All rights reserved.
2  % Load common and reader dll
3  NET.addAssembly('C:\\Users\\KevinCheng\\ATFX API Package v1.2\\Common.dll');
4  NET.addAssembly('C:\\Users\\KevinCheng\\ATFX API Package v1.2\\CI.ATFX.Reader.dll');
5
6  %create a atfx recording instance
7  rec = EDM.Recording.ODSNVHATFXMLRecording('C:\\Users\\KevinCheng\\Downloads\\gps test example\\{4499520}_REC_{20220419}(1).atfx');
```


For the LabVIEW ATFX API example to work, please use the latest version of LabVIEW, such as LabVIEW **2021** or **2021 SP1**. And use the provided dll files in the LabVIEW ATFX API Demo -> Private folder.



ATFX API C# Code Examples

The following sections are examples from our CI ATFX Reader C# Demo Program to help users understand how to utilize our API class methods. Some of the code snippets have been shortened compared to the actual Demo Program to provide a more concise explanation. These code samples can be used to quickstart custom software integration with the ATFX API.

There are 3 file types that the ATFX API can open: .atfx, .ts and .gps. The .atfx is the header file that references .dat, which contains the bulk of the data. It can also reference .ts and .gps files.

Opening a ATFX File – Start Here

To open an ATFX file, use the **RecordingManager** Class to call **OpenRecording**, which takes in a filename and outputs a **IRecording** object:

```
using EDM.RecordingInterface;
using EDM.Recording;

var recordingPath = "C:\Sig001.atfx";
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);
```

What is a Recording vs. Signal?

In our API, the **IRecording** object represents the ATFX file, and contains a list of **ISignal** objects. Each **ISignal** corresponds to a given channel and measurement method.

Concept	Class Type	Example
ATFX file record	< IRecording >	"C:\Sig001.atfx"
- Properties	< RecordingProperty >	
- Signals	List < ISignal >	
o Signals[0]	< ISignal >	Block(CH1)

○ Signals[1]	<ISignal>	Block(CH2)
○ Signals[2]	<ISignal>	APS(CH1)
○ Signals[3]	<ISignal>	APS(CH2)
○ ...		

For instance, in the example above, the first Signal stored in the ATFX file corresponds to a segment of Time Domain data acquired from Channel 1.

Note: in CI terminology, “Block” refers to a contiguous segment of time domain data (usually collected with sample size that is a power of 2), and “APS” refers to a contiguous segment of frequency domain data (usually calculated via FFT of a time block). These are the two most common types of signals in our software.

The example code below shows using the **IRecording.Signals** property to get a list of signals from a given ATFX record:

```
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);

// Get the list of signals from the recording
List<ISignal> signals = rec.Signals;
```

In addition, the **IRecording** object also supports the following properties:

Name	Type	Descriptions
Item	ISignal	Returns the ISignal object at a specified index
RecordingProperty	RecordingProperty	Returns a RecordingProperty object with metadata (ex: CreateTime, Serial Numbers, etc.)
SignalCount	int	Returns number of ISignal objects
Signals	List<ISignal>	This is where the actual data lives. Returns a list of ISignal objects

Finding the Signal for a particular channel

Once you have a list of signals, you will want to query the **ISignal.Name** of the signal to find the channel and measurement type you are looking for.

For instance, if you want the time block for channel 4, then you want to look for the signal with the name “Block(CH4)”

```
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);
```

```
// Get the list of signals from the recording
List<ISignal> signals = rec.Signals;

// To get the Channel 4 signal, select the signal whose name is 'Block(CH4)'
ISignal signalCh4 = signals.Where(sig => sig.Name == 'Block(CH4)').First();
```

What is a Frame?

A Frame is a **double[][]** array inside the ISignal object, that contains the numerical data (x-values, y-values) that you want to acquire. Most of the time, a Signal only has one Frame, but in the case of waterfall plots or 3D plots, there may be multiple frames.

Concept	Class Type	Example
Signal	<ISignal>	Block(CH1)
- Frame	<double[][]>	Signal.GetFrame(0)
o Frame[0]	<double[]>	Array of x-values
o Frame[1]	<double[]>	Array of y-values
o Frame[2]	<double[]>	Array of z-values (if applicable)

The Frame is formatted such that the first array is the x-values, the second array is the y-values, and (if applicable) the third array is the z-values.

The Frame size (int) is stored in the **ISignal.FrameSize** property. The full list of **ISignal** properties and methods is shown below:

Name	Type	Descriptions
Dimension	int	Get the signal dimension
FrameSize	int	Get the size of each frame
Name	string	Get the signal name
Properties	SignalProperties	Get the signal properties. Time domain and frequency domain signals have different signal properties. For time domain signals, Properties refer to SignalProperties. For frequency domain signals, Properties refer to FrequencyDomainSignalProperties.
Recording	IRecording	Get the signal recording
Type	SignalType	Get the signal type, time/frequency domain

Unknown 0
Time 1
Frequency 2
Trend 3

Name	Return Type	Descriptions
GetFrame(int)	Double[][]	Returns a double[][] with the data frame at that index A snapshot of measurement data consisting of X, Y and sometimes Z values.
GetParameter<T>(string)	T	Get the specified parameter by the given name.
GetParameterType(string)	string	Get the specified parameter data type by the given name.

An end-to-end code example

To summarize the above content, here is an example code that opens a recording, finds the signal for the “Channel 4” time domain data, and reads out the frame data:

```
using EDM.RecordingInterface;
using EDM.Recording;

var recordingPath = "C:\Sig001.atfx";
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);

// Get the list of signals from the recording
List<ISignal> signals = rec.Signals;

// To get the Channel 4 signal, select the signal whose name is 'Block(CH4)'
ISignal signalCh4 = signals.Where(sig => sig.Name == 'Block(CH4)').First();

// Get the frame, which is formatted like [[x1, x2, x3...], [y1, y2, y3...],...]
double[][] frame = signalCh4.GetFrame(0);
double[] xValues = frame[0];
double[] yValues = frame[1];

// If applicable
double[] zValues = frame[2];

// Size of the frame
int size = signalCh4.FrameSize;
```

Additional File Components - .TS and .GPS

An ATFX file may also come with a **.ts** and / or **.gps** where it lists the files as a **file component** inside the ATFX file.

```
13 <files>
14 <component>
15     <identifier>External_{4499520}_REC_{20220419}(1)</identifier>
16     <filename>{4499520}_REC_{20220419}(1).dat</filename>
17 </component>
18 <component>
19     <identifier>External_{4499520}_REC_{20220419}(1)</identifier>
20     <filename>{4499520}_REC_{20220419}(1).ts</filename>
21 </component>
22 </files>
```

```
13 <files>
14 <component>
15     <identifier>External_REC0041</identifier>
16     <filename>REC0041.dat</filename>
17 </component>
18 <component>
19     <identifier>External_REC0041</identifier>
20     <filename>REC0041.gps</filename>
21 </component>
22 </files>
```

In order to extract the data from these types of files users will need to import **ASAM.ODS.ATFXML**, which will allow access to **ExternalFileComponent** class.

```
using ASAM.ODS.ATFXML;
```

The file components in the ATFX file, which is now a IRecording object due to the previous step, is a **list of ExternalFileComponent** objects. Thus for each ExternalFileComponent, users will have to create a recording and add to a list to keep track of them.

The **CommonRecording.CreateRecording** will create a new IRecording object of a specific RecordingType, whether it is a **TimeStampRecording** or **GPSRecording**, allowing access to the specific type of data stored in a .ts or .gps file.

```
List<ExternalFileComponent> fileComponents = (rec as
ODSATFXMLRecording).ExternalFiles.FileComponents;
```

```
IRecording tsrec = CommonRecording.CreateRecording(fileComponents[0].FileName);
```

```
private void ShowRecordings(IRecording rec)
{
    List<ExternalFileComponent> fileComponents = (rec as
ODSATFXMLRecording).ExternalFiles.FileComponents;
    //Add the initial IRecording object
    lbRecordingDataInfo.Items.Add(rec);

    foreach (ExternalFileComponent externalfile in fileComponents)
    {
        if (!externalfile.FileName.EndsWith(".ts") &&
!externalfile.FileName.EndsWith(".gps"))
            continue;

        IRecording tsrec = CommonRecording.CreateRecording(externalfile.FileName);
        lbRecordingDataInfo.Items.Add(tsrec);
    }
}
```

```
}  
}
```

With a newly created recording of a .ts and / or .gps file, users can access their specific recording properties and signals from the IRecording properties. These signals also contain their own set of data and properties that can be stored in a list to keep track of.

```
private void ShowSignals(IRecording rec)  
{  
    foreach(ExternalFileComponent externalfile in (rec as  
    ODSATFXMLRecording).ExternalFiles.FileComponents)  
    {  
        if (!externalfile.FileName.EndsWith(".ts") &&  
        !externalfile.FileName.EndsWith(".gps"))  
            continue;  
  
        IRecording tsrec=CommonRecording.CreateRecording(externalfile.FileName);  
        foreach (ISignal sig in tsrec.Signals)  
        {  
            lbSignalDataInfo.Items.Add(sig);  
        }  
    }  
}
```

Opening a TS or GPS File

It is possible to open a .ts and .gps file, given that the **RecordingManager OpenRecording** will create a specific type of recording. It is similar to using **CommonRecording.CreateRecording**.

Thus all that is needed to do is find the file path of the .ts or .gps and send it to the **RecordingManager.Manager.OpenRecording**. Without having to access the ATFX external file components.

```
RecordingManager.Manager.OpenRecording(string filePath, out IRecording recording);
```

```
var recordingPath = "C:\Sig001.ts";  
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))  
{  
    // Grab data from IRecording  
}  
  
var recordingPath = "C:\Sig001.gps";  
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))  
{  
    // Grab data from IRecording  
}
```

Reading the Record Properties

To read the Record Properties, which contains the ATFX file record information, it is extracted directly from the **IRecording.RecordingProperty** using **GetProperties**. Or by calling the following properties in the **IRecording.RecordingProperty**.

Here are the **RecordingProperty** Class properties:

Name	Type	Descriptions
CreateTime	DateTime	When the file was recorded. It is not when the file is saved. This parameter can show the time accuracy as high as second. To obtain the starting recording time with better accuracy, please add "StartNanosecond" in integer that represents the additional nanoseconds elapsed.
Instruments	string	The product name used to record/save data to the file.
MasterSN	int	Serial number of the master module of the system when the file was created
MeasurementType	MeasurementConfigType	Measurement type of the file
RecordingName	string	Name of the recording file
DeviceSNs	string	Serial numbers of the 1 or many modules used in the recording
RecordingPath	string	Recording file save path
RecordingType	RecordingType	The type of recording based on its file extension
RecordingTypeName	string	Recording type name based on its file extension
SavingVersion	Version	EDM version number when the file was created.
TestNote	string	Test notes given by the user before the test ran
User	string	The EDM account name when the file was created.

Calling individual property

```

DateTime createTime = [IRecording object].RecordingProperty.CreateTime;
string instrument = [IRecording object].RecordingProperty.Instruments;
uint masterSN = [IRecording object].RecordingProperty.MasterSN;
etc.

```

GetProperties

The GetProperties function is useful in getting a list of various data types in the RecordingProperty class.

```
var properties = [IRecording object].RecordingProperty.GetProperties(BindingFlags  
bindingAttr);
```

```
private void ShowContents(DataGridView grid, object item)  
{  
    grid.Rows.Clear();  
  
    var props = item.GetType().GetProperties(BindingFlags.Instance |  
                                             BindingFlags.Public);  
  
    foreach (var prop in props)  
    {  
        //skip RecordingProperties property  
        if (prop.Name == "RecordingProperties") continue;  
        var content = prop.GetValue(item, null)?.ToString();  
        if (!string.IsNullOrEmpty(content))  
        {  
            grid.Rows.Add(prop.Name, content);  
        }  
    }  
}
```

```
var recordingPath = "C:\Sig001.atfx";  
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))  
{  
    ShowContents(dataGridRecord, rec.RecordingProperty);  
}
```

Property	Value
User	Admin
Instruments	Spider
TestNote	Random55/Run10
Name	SIG0010
RecordingPath	C:\Users\KevinCheng\Documen...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.30
CreateTime	3/7/2022 3:23:19 PM
MasterSN	2590976
UserAnnotation	Random55/Run10
MeasurementType	VCS_Random

Reading the GPS Data

To read the GPS data, it is extracted from the IRecording object as a **ODSNVHATFXMLRecording** object and locating the **Measurement** and **Environment** property. These properties are **AoMeasurement** and **AoEnvironment**, which can be converted into **NVHMeasurement** and **NVHEnvironment**.


```

ODSNVHATFXMLRecording nvhRec = rec as ODSNVHATFXMLRecording;
NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;
NVHEnvironment nvhEnvironment = nvhRec.Environment as NVHEnvironment;

```

In order to use NVHMeasurement and NVHEnvironment, users must import **ASAM.ODS.NVH**;

```
using ASAM.ODS.NVH;
```

Here are the **NVHMeasurement** Class properties:

Name	Type
Altitude	double
GPSEnabled	bool
Latitude	double
Longitude	double
MeasurementBegin	DateTime
MeasurementEnd	DateTime
NanoSecondElapsed	int

Here are the **NVHEnvironment** Class properties:

Name	Type
FirmwareVersion	string
InstruSoftwareVersion	string
HardwareVersion	string
BitwareVersion	string
TimeZone	string

Here are the **AoEnvironment** Class functions:

Name	Return Type	Descriptions
GetLocalTime(DateTime)	DateTime	Get time in local format
GetUTCTime(DateTime)	DateTime	Get time in UTC format

The code snippet below shows the extraction of GPS related data.

```
private void ShowGPSInfo(IRecording rec)
{
    if (rec is ODSNVHATFXMLRecording nvhRec)
    {
        NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;
        NVHEnvironment nvhEnvironment = nvhRec.Environment as NVHEnvironment;
        bool bGPS = nvhMeasurement.GPSEnabled;

        if (bGPS)
        {
            dgvRecInfo.Rows.Add("GPS Enabled", bGPS);
            double lng = nvhMeasurement.Longitude;
            double lat = nvhMeasurement.Latitude;
            double alt = nvhMeasurement.Altitude;
            double nano = nvhMeasurement.NanoSecondElapsed;

            dgvRecInfo.Rows.Add("Longitude", lng);
            dgvRecInfo.Rows.Add("Latitude", lat);
            dgvRecInfo.Rows.Add("Altitude", alt);
            dgvRecInfo.Rows.Add("Nanoseconds Elapsed", nano);
        }

        if (!String.IsNullOrEmpty(nvhRec.Environment.TimeZone))
        {
            dgvRecInfo.Rows.Add("Time Zone", nvhRec.Environment.TimeZone);
        }

        dgvRecInfo.Rows.Add("Created Time (Local)", nvhRec.RecordingProperty.CreateTime);
        dgvRecInfo.Rows.Add("Created Time (UTC)",
            nvhRec.Environment.GetUTCTime(nvhRec.RecordingProperty.CreateTime));

        if (!String.IsNullOrEmpty(nvhEnvironment.InstruSoftwareVersion))
        {
            dgvRecInfo.Rows.Add("Instrument Software Version",
                nvhEnvironment.InstruSoftwareVersion);
            dgvRecInfo.Rows.Add("Hardware Version", nvhEnvironment.HardwareVersion);
            dgvRecInfo.Rows.Add("Firmware Version", nvhEnvironment.FirmwareVersion);
            dgvRecInfo.Rows.Add("Bit Version", nvhEnvironment.BitVersion);
        }
    }
}
```

```
var recordingPath = "C:\\Sig001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
    ShowGPSInfo(rec);
}
```

Property	Value
User	Unknown Owner
Instruments	GRS
TestNote	Untitled Test Note
Name	{4499520}_REC_{20220419}(1) - C...
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.41
DeviceSNs	4499520
MasterSN	4499520
MeasurementType	None
GPS Enabled	True
Longitude	0
Latitude	37.38046
Altitude	12.42
Nanoseconds Elapsed	629999338
Time Zone	UTC-05:00
Created Time (Local)	4/18/2022 6:47:10 PM
Created Time (UTC)	4/18/2022 10:47:10 PM

Extracting the Date and Time of a Recording

To extract and read the time data that a recording has, users will have to import and use the **DateTimeNano** object, which is an extension of the **DateTime** that includes nanosecond data.

To use the DateTimeNano class, users will need to import **Common**.

```
using Common;
```

Here are the **DateTimeNano** Class properties, it shares similarities to DateTime, of which those are omitted:

Name	Type	Descriptions
IsNanoTime	DateTime	Gets whether nanoseconds exists / not equal to zero
TotalNanoSeconds	int	Get TotalSeconds in Nano Seconds
ms_us_ns	int	We use this NanoSeconds==0 Distinguish between normal time and nanosecond time Milisecond.Microsecond.Nanosecond 000/000/000

The following code snippet shows how to extract, create and display the DateTimeNano object properties.

```
private void ShowDateTimeNano(IRecording rec, bool isLocal)
{
```

```

if (rec is ODSNVHATFXMLRecording nvhRec)
{
    NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;
    NVHEnvironment nvhEnvironment = nvhRec.Environment as NVHEnvironment;
    DateTimeNano createTimeUTC;
    if (isLocal)
    {
        createTimeUTC = new DateTimeNano(nvhRec.RecordingProperty.CreateTime,
nvhMeasurement.NanoSecondElapsed);
    }
    else
    {
        createTimeUTC = new
DateTimeNano(nvhRec.Environment.GetUTCTime(nvhRec.RecordingProperty.CreateTime),
    nvhMeasurement.NanoSecondElapsed);
    }
    dgvRecInfo.Rows.Add("Year", createTimeUTC.Year);
    dgvRecInfo.Rows.Add("Month", createTimeUTC.Month);
    dgvRecInfo.Rows.Add("Day", createTimeUTC.Day);
    dgvRecInfo.Rows.Add("Hour", createTimeUTC.Hour);
    dgvRecInfo.Rows.Add("Minute", createTimeUTC.Minute);
    dgvRecInfo.Rows.Add("Second", createTimeUTC.Second);
    dgvRecInfo.Rows.Add("Millisecond", createTimeUTC.Millisecond);
    dgvRecInfo.Rows.Add("IsNanoTime", createTimeUTC.IsNanoTime);
    dgvRecInfo.Rows.Add("NanoSeconds", createTimeUTC.ms_us_ns);
    dgvRecInfo.Rows.Add("TotalNanosec", createTimeUTC.TotalNanosec);
    dgvRecInfo.Rows.Add("Date Time", createTimeUTC.DateTime);
    dgvRecInfo.Rows.Add("TimeOfDay", createTimeUTC.TimeOfDay);
    dgvRecInfo.Rows.Add("ToNanoString()", createTimeUTC.ToNanoString());

    int ms = (int)(createTimeUTC.ms_us_ns / 1e6);
    int us = (int)(createTimeUTC.ms_us_ns / 1e3 % 1e3);
    int ns = (int)(createTimeUTC.ms_us_ns % 1e3);
    string customFormat = string.Format("{0}/{1}/{2}/{3}/{4}/{5}/{6}/{7}/{8}",
createTimeUTC.Year, createTimeUTC.Month, createTimeUTC.Day, createTimeUTC.Hour,
createTimeUTC.Minute, createTimeUTC.Second, ms, us, ns);
    dgvRecInfo.Rows.Add("Custom Format: yyyy/mm/dd/hh/mm/ss/ms/us/ns", customFormat);
}
}

```

```

var recordingPath = "C:\Sig001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
    ShowDateTimeNano(rec, false);
}

```

Property	Value
Year	2022
Month	4
Day	18
Hour	22
Minute	47
Second	10
Millisecond	0
IsNanoTime	True

NanoSeconds	629999338
TotalNanosec	82030629999338
Date Time	4/18/2022 10:47:10 PM
TimeOfDay	22:47:10
ToNanoString()	4/18/2022 10:47:10 PM.629.999.338
Custom Format: yyyy/mm/dd/hh...	2022/4/18/22/47/10/629/999/338

Reading the Channel Table Data

To read the Channel Table data, it is extracted from the `IRecording` object as a **ODSNVHATFXMLRecording** object and locating the specific property, **ChnSensitivities**. (Apologies for the spelling error.) It can also be converted into a **NVHTestEquipmentPart**.

```
ODSNVHATFXMLRecording odsRec = rec as ODSNVHATFXMLRecording;
```

```
ChannelSensitivity eq in odsRec.ChnSensitivities[0];
```

```
NVHTestEquipmentPart channel = eq.EquipmentPart;
```

The `ODSNVHATFXMLRecording` and `ChannelSensitivity` class already comes with the importation of `EDM.Recording` and `EDM.RecordingInterface`.

However, there are also additional imports, such as the **ASAM.ODS.NVH**, that will be used in this section.

```
using ASAM.ODS.NVH;
```

Here are the **NVHTestEquipmentPart** Class properties:

Name	Type
ChannelID	int
ChannelStatus	int
ChannelType	int
DtType	int
EUFactor	double
EUName	string
InputRange	int
QuantityName	string
SampleRate	single
Sensitivity	single
SensorRange	single
SensorSN	string
Weighting	single

Below shows a much simpler way of extracting data directly from the NVHTestEquipmentPart object compared to the demo program.

```
private void ShowChannelTable(IRecording rec)
{
    foreach (ChannelSensitivity eq in odsRec.ChnSensitivities)
    {
        NVHTestEquipmentPart channel = eq.EquipmentPart;

        if (channel == null) continue;

        dataGridChannel.Rows.Add(channel.LabelTitle,
            channel.ChannelType.ToChannelTypeString(),
            channel.QuantityName,
            channel.EUName,
            $"{channel.Sensitivity}(mv/{channel.EUName})",
            channel.ChannelStatus.ToChannelStatusString(),
            channel.InputRange.ToChannelRangeString(),
            channel.SensorSN,
            channel.SensorRange,
            channel.Intergration.ToChannelIntegrationString(),
            channel.Weighting);
    }
}
```

```
var recordingPath = "C:\Sig001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
    ShowChannelTable(rec);
}
```

Location ID	Channel Type	Measurement Quantity	Engineering Unit	Sensitivity	Input Mode	Input Range	Sensor SN	Max. sensor range	Intergration	Control Weighting
Ch1	Control	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch2	Monitor	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch3	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch4	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch5	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch6	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch7	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch8	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1

Reading the Signal Properties

To read the Signal Properties, which contains the ATFX file signal property information, it is extracted directly from the **ISignal.Properties** using **GetProperties** or **GetFields**.

The ISignal interface already comes with the importation of EDM.RecordingInterface.

Here are the **ISignal** Class properties:

Name	Type	Descriptions
Dimension	int	Get the signal dimension

FrameSize	int	Get the size of each frame
Name	string	Get the signal name
Properties	SignalProperties	Get the signal properties. Time domain and frequency domain signals have different signal properties. For time domain signals, Properties refer to SignalProperties. For frequency domain signals, Properties refer to FrequencyDomainSignalProperties.
Recording	IRecording	Get the signal recording
Type	SignalType	Get the signal type, time/frequency domain Unknown 0 Time 1 Frequency 2 Trend 3

Name	Return Type	Descriptions
GetFrame(int)	Double[][]	Returns a double[][] with the data frame at that index A snapshot of measurement data consisting of X, Y and sometimes Z values.
GetParameter<T>(string)	T	Get the specified parameter by the given name.
GetParameterType(string)	string	Get the specified parameter data type by the given name.

Using a List to Store and Recall Signals

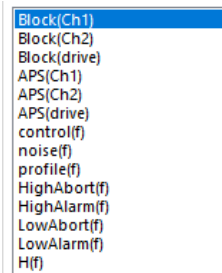
When working with the Signals list from IRecording object, it would be best to store it in a list to easily reference to it, especially when selecting which signal properties or data to display.

```
private void ShowSignals(IRecording rec)
{
    foreach (ISignal sig in rec.Signals)
    {
        lbSignalDataInfo.Items.Add(sig);
    }
}
```

```

var recordingPath = "C:\Sig001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))
{
    ShowSignals(rec);
}

```



Basic Signal Information

Here are the **SignalProperties** Class properties:

Name	Type	Descriptions
BlockSize	int	Get the block size Number of time data points captured in the signal
DeviceSN	string	The recording instrument serial numbers
Duration	string	Get the signal duration Amount of time covered by the signal
GeneratedTime	DateTimeNano	Get the signal generated time from instrument
Instruments	string	Get the instrument
MeasurementType	MesaurementConfigType	Get the MeasurementType
RecordingProperties	RecordingProperty	Get the RecordingProperties
SamplingRate	string	Get the sampling rate Number of data samples acquired per second
SignalName	string	Get the signal name
SignalType	SignalType	Get the signal type Unknown 0 Time 1 Frequency 2 Trend 3
SoftwareVersion	version	Get the software version

UnitX	string	Get the X unit
UnitY	string	Get the Y unit
UnitZ	string	Get the Z unit

Calling individual property

```
ISignal signal = [IRecording object].Signals[0];
Common.DateTimeNano dateTimeNano = signal.Properties.GeneratedTime;
MeasurementConfigType measureType = signal.Properties.MeasurementType;
SignalType type = signal.Properties.SignalType;
etc.
```

GetProperties

The GetProperties function is useful in getting a list of various data types in the SignalProperties class.

The following code snippets display the signal information.

```
var properties = [IRecording object].Item[0].GetProperties(BindingFlags bindingAttr);
```

```
private void ShowContents(DataGridView grid, object item)
{
    grid.Rows.Clear();

    var props = item.GetType().GetProperties(BindingFlags.Instance |
                                           BindingFlags.Public);

    foreach (var prop in props)
    {
        //skip RecordingProperties property
        if (prop.Name == "RecordingProperties") continue;
        var content = prop.GetValue(item, null)?.ToString();
        if (!string.IsNullOrEmpty(content))
        {
            grid.Rows.Add(prop.Name, content);
        }
    }
}
```

```
private void BtnSignalBasicInfo_Click(object sender, EventArgs e)
{
    if (lbSignalDataInfo.SelectedItem is ISignal signal)
    {
        ShowContents(dgvSignalDataInfo, signal.Properties);
    }
}
```

Record Information	Signal Data Information	Channel Table
Block(Ch1)	Property	Value
Block(Ch2)	UserAnnotation	Random57/Run12Rando...
Block(drive)	MeasurementType	VCS_Random
APS(Ch1)	SignalType	Time
APS(Ch2)	GeneratedTime	3/24/2022 1:48:58 PM
APS(drive)	SignalName	Block(Ch1)
control(f)	SamplingRate	5.12 kHz
noise(f)	BlockSize	1024
profile(f)	Duration	0.2 (s)
HighAbort(f)	UnitX	Time (s)
HighAlarm(f)	UnitY	m/s ²
LowAbort(f)	UnitZ	N/A
LowAlarm(f)	NvhType	NonEquidistant
H(f)	AcquisitionCalculateMeth...	Undefined
limit_notch(Ch1)	IsVCSignal	True
limit_notch(Ch2)		
limit_high_abort(Ch1)		
limit_high_abort(Ch2)		
limit_high_alarm(Ch1)		
limit_high_alarm(Ch2)		

Advance Signal Information

Here are the **DSASignalProperty** Class fields:

Name	Type	Descriptions
averageMode	int	average mode index when signal data saved
averageNumber	int	average number when signal data saved
blocksizeLine	string	block size line when signal data saved
elapsedTime	double	elapsed time when signal data saved
frequencyIndex	int	sample rate index when signal data saved
outputPeak	double	output peak when signal data saved
overlapRatioIndex	int	overlap ratio index when signal data saved
rpmTacho1	double	rpm tacho 1 when signal data saved
rpmTacho2	double	rpm tacho 2 when signal data saved
testLastSavedTime	DateTime	last saved time of the test
testName	string	test name
totalFrameNumber	int	total frame number(or current average number) when signal data saved
windowTypeIndex	int	window type index when signal data saved

And here are the **VCSSignalProperty** Class fields:

Name	Type	Descriptions
controlPeak	double	control peak (m/s ²) when data saved
controlRMS	double	current control RMS (m/s ²) when data saved
currentFrequency	double	current frequency when data saved (Sine)
curRepeat	int	current repeat times when data saved
displacementPkPk	double	displacement peak peak (m) when data saved
drivePK	double	current drive peak (voltage) when data saved
fullLevelElapsed	double	full level elapsed when data saved (time in Random/Sine/TDR, pulses in Shock system)
level	double	current VCS level when data saved
nextDrivePK	double	next predicted drive peak (voltage)
nextLevel	double	next predicted VCS level
pulseWidth	double	main pulse width in classic Shock
remaining	double	remaining time when data saved (time in Random/Sine/TDR, pulses in Shock system)
remainingCycle	double	remaining cycles when data saved (Sine)
sweepNumber	int	sweep number when data saved (Sine)
sweepRate	double	sweep rate when data saved (Sine)
sweepType	int	sweep type when data saved (Sine)
targetPeak	double	target peak (m/s ²) when data saved
targetRMS	double	target RMS (m/s ²) when data saved
testLastRunTime	DateTime	last run time of the test
testLastSavedTime	DateTime	last saved time of the test
testName	string	test name
totalCycle	double	total cycles when data saved (Sine)

totalElapsed	double	total elapsed time when data saved (time in Random/Sine/TDR, pulses in Shock system)
totalRepeat	int	total repeat times when data saved
velocityPk	double	velocity peak (m/s) when data saved

Calling individual field

```
ISignal signal = [IRecording object].Signals[0];
int avgMode = signal.Properties.dsaProperties.averageMode;
string name = signal.Properties.dsaProperties.testName;
double level = signal.Properties.vcsProperties.level;
double remaining = signal.Properties.vcsProperties.remaining;
string name = signal.Properties.vcsProperties.testName;
etc.
```

GetFields

Here is a code snippet for displaying the advance signal information, depending on if the signal comes from VCS or DSA.

For the showPublicField, it can be set to false to show the basic signal information or to true to show the advance signal information.

```
var fields = [IRecording object].Item[0].GetFields(BindingFlags bindingAttr);
```

```
private void ShowContents(DataGridView grid, object item, bool showPublicField = false)
{
    grid.Rows.Clear();

    if (showPublicField)
    {
        var fields = item.GetType().GetFields(BindingFlags.Instance |
                                                BindingFlags.Public);

        foreach (var field in fields)
        {
            //skip multiSineTonesInfo field
            if (field.Name == " multiSineTonesInfo") continue;
            var content = field.GetValue(item)?.ToString();
            if (!string.IsNullOrEmpty(content))
            {
                grid.Rows.Add(field.Name, content);
            }
        }
    }
}
```

```
private void BtnSignalAdvInfo_Click(object sender, EventArgs e)
{
```

```

if (lbSignalDataInfo.SelectedItem is ISignal signal)
{
    //if signal is a dsa signal, dsa properties should not be empty
    if (signal.Properties.dsaProperties != null)
    {
        ShowContents(dgvSignalDataInfo, signal.Properties.dsaProperties, true);
    }
    //if signal is a vcs signal, vcs properties should not be empty
    if (signal.Properties.vcsProperties != null)
    {
        ShowContents(dgvSignalDataInfo, signal.Properties.vcsProperties, true);
    }
}
}
}

```

Property	Value
testName	Random57
testLastSavedTime	3/24/2022 1:48:58 PM
testLastRunTime	3/24/2022 1:48:13 PM
level	1
drivePK	0.456419169902802
controlRMS	9.68552017211914
targetRMS	9.8128662109375
controlPeak	0
targetPeak	0
fullLevelElapsed	11
remaining	289.100006103516
totalElapsed	43.5499992370605
velocityPk	0.0249415971338749
displacementPkPk	0.000173337204614654
pulseWidth	0
DOF	64
currentFrequency	0
totalCycle	0
remainingCycle	0
sweepType	0

Advance Generated Time

The Generated Time property for Signal is a **DateTimeNano** object, which is imported from **Common**.

```
using Common;
```

Here are the **DateTimeNano** Class properties, it shares similarities to DateTime, of which those are omitted:

Name	Type	Descriptions
------	------	--------------

IsNanoTime	DateTime	Gets whether nanoseconds exists / not equal to zero
TotalNanoSeconds	int	Get TotalSeconds in Nano Seconds
ms_us_ns	int	We use this NanoSeconds==0 Distinguish between normal time and nanosecond time Milisecond.Microsecond.Nanosecond 000/000/000

Calling individual property

```
ISignal signal = [IRecording object].Signals[0];
uint ms_us_ns = signal.Properties.GeneratedTime.ms_us_ns;
ulong totalNanoSec = signal.Properties.GeneratedTime.TotalNanosec;
int seconds = signal.Properties.GeneratedTime.Second;
etc.
```

GetProperties

The GetProperties function is useful in getting a list of various data types in the DateTimeNano class.

```
DateTimeNano generatedTime = [ISignal object].Properties.GeneratedTime;
```

```
private void BtnShowGeneratedTime_Click(object sender, EventArgs e)
{
    lbSignalParameters.Visible = false;
    signalDataInfo = SignalDataInfo.SignalGeneratedTime;
    if (lbSignalDataInfo.SelectedItem is ISignal signal)
    {
        ShowContents(dgvSignalDataInfo, signal.Properties.GeneratedTime);
    }
}
```

Property	Value
Year	2022
Month	3
Day	29
Hour	16
Minute	14
Second	54
Millisecond	0
TimeOfDay	16:14:54
IsNanoTime	False
TotalNanosec	5849400000000

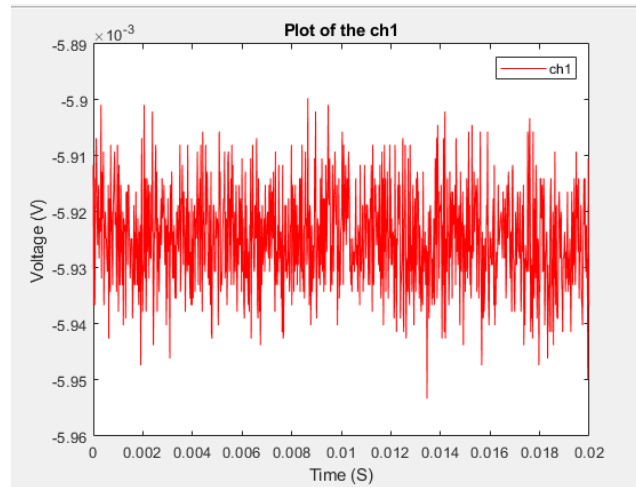
Reading the Data Values of a Signal Frame

A signal frame is a snapshot of measurement data that consists of X, Y and sometimes Z data. Each of these frames consists of an array with the size according to **Signal.FrameSize** property. Each signal usually has 1 Frame (unless it is a waterfall or 3D plot), and the **Signal.FrameCount** property describes how many frames are in the signal.

The X and Y formulate points in a chart where X can be Time or Frequency and Y can be a variety of engineering units, such as Voltage, Acceleration, Velocity, Displacement, Force, etc.

And the Z is generally the time since the device start measuring.

Thus, if a user were to graph the the X and Y data, they would get a plot graph like below.



A Frame object is stored inside a parent Signal object according the following structure:

Concept	Class Type	Example
Signal	<ISignal>	Block(CH1)
- Frame	<double[][]>	Signal.GetFrame(0)
o Frame[0]	<double[]>	Array of x-values
o Frame[1]	<double[]>	Array of y-values
o Frame[2]	<double[]>	Array of z-values (if applicable)

The Frame is formatted such that the first array is the x-values, the second array is the y-values, and (if applicable) the third array is the z-values.

More information about the Frame (e.g., Frame Size) can be queried from the **ISignal** parent object. The **ISignal** parent object for the Frame also supports the following additional properties:

Name	Type	Descriptions
------	------	--------------

Dimension	int	Get the signal dimension
FrameSize	int	Get the size of each frame
Name	string	Get the signal name
Properties	SignalProperties	Get the signal properties. Time domain and frequency domain signals have different signal properties. For time domain signals, Properties refer to SignalProperties. For frequency domain signals, Properties refer to FrequencyDomainSignalProperties.
Recording	IRecording	Get the signal recording
Type	SignalType	Get the signal type, time/frequency domain Unknown 0 Time 1 Frequency 2 Trend 3

Name	Return Type	Descriptions
GetFrame(int)	Double[][]	Returns a double[][] with the data frame at that index A snapshot of measurement data consisting of X, Y and sometimes Z values.
GetParameter<T>(string)	T	Get the specified parameter by the given name.
GetParameterType(string)	string	Get the specified parameter data type by the given name.

An end-to-end example of reading a Frame from a Signal, which can be read from a Recording:

```
using EDM.RecordingInterface;
using EDM.Recording;

var recordingPath = "C:\Sig001.atfx";
RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec);

// Get the list of signals from the recording
List<ISignal> signals = rec.Signals;
```



```

// To get the Channel 4 signal, select the signal whose name is 'Block(CH4)'
ISignal signalCh4 = signals.Where(sig => sig.Name == 'Block(CH4)').First();

// Get the frame, which is formatted like [[x1, x2, x3...], [y1, y2, y3...],...]
double[][] frame = signalCh4.GetFrame(0);
double[] xValues = frame[0];
double[] yValues = frame[1];

// If applicable
double[] zValues = frame[2];

// Size of the frame
int size = signalCh4.FrameSize;

```

Record Information	Signal Data Information	Channel Table
Block(Ch1)	X Data-Time (s)	Y Data-m/s ²
Block(Ch2)	0.000000E+000	-2.418288E+000
Block(drive)	1.953125E-004	1.084685E+001
APS(Ch1)	3.906250E-004	-1.259770E-001
APS(Ch2)	5.859375E-004	-8.884092E+000
APS(drive)	7.812500E-004	-7.022393E-001
control(f)	9.765625E-004	-9.082394E+000
noise(f)	1.171875E-003	-2.056571E+001
profile(f)	1.367188E-003	-2.594410E+001
HighAbort(f)	1.562500E-003	-1.424093E+001
HighAlarm(f)	1.757813E-003	4.717639E+000
LowAbort(f)	1.953125E-003	3.687933E+000
LowAlarm(f)	2.148438E-003	1.276019E+001
H(f)	2.343750E-003	1.329508E+001
limit_notch(Ch1)	2.539063E-003	-9.056539E+000
limit_notch(Ch2)	2.734375E-003	-1.155804E-001
limit_high_abort(Ch1)	2.929688E-003	1.046004E+001
limit_high_abort(Ch2)	3.125000E-003	-1.712991E+000
limit_high_alarm(Ch1)	3.320313E-003	-1.931287E+000
limit_high_alarm(Ch2)	3.515625E-003	-2.037931E+000
	3.710938E-003	-6.292362E+000

Reading other Signal Parameters

To read other Signal Parameters that isn't the general properties or frame data, it is extracted from the **IRecording.Signals** or **IRecording.Item[#]** using the **GetParameter<T>** with a **NVHParameterSet** parameter key.

There is a large list of fields in **NVHParameterSet**, thus it is recommended to find these fields in the CI ATEX Reader Class Methods.chm file under **ASAM.ODS.NVH -> NVHParameterSet Class -> NVHParameterSet Fields**.

In order to use the **NVHParameterSet** Class, users need to import **ASAM.ODS.NVH**.

There are also additional imports, such as the **Common.Spider** and **EDM.Utils**, that will be used in this section.

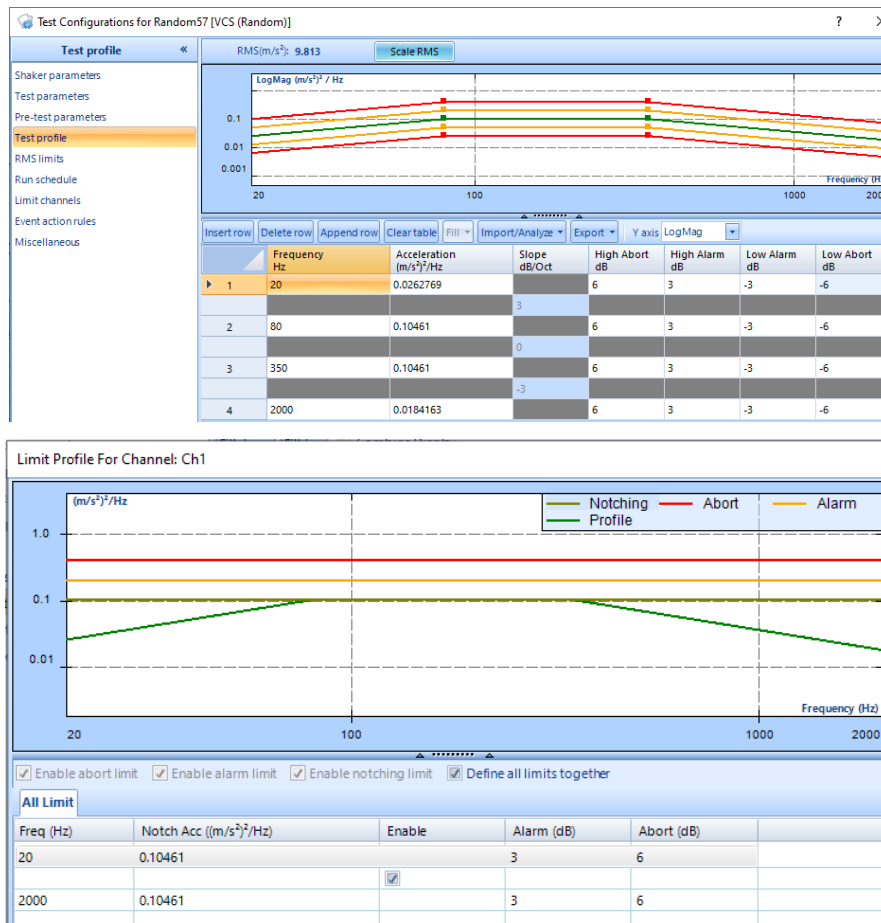
```

using ASAM.ODS.NVH;
using Common.Spider;
using EDM.Utils;

```

EDM.Utils will provide a **JSON deserialization** method used to deserialize the JSON string from the parameter values.

And depending on which parameter, such as **Test Profile**, the Common.Spider provides two classes, **ProfileElemDsp** and **SpiderChannelLimitElem**, to hold the deserialized JSON string data.



Reading and Displaying Individual Parameter Key

```
ISignal signal = [IRecording object].Signals[0];
```

```
string signalParam = signal.GetParameter<string>(NVHParameterSet.testProfile)
```

```
List<ProfileElemDsp> profile =
```

```
Utility.JsonDeserialize<List<ProfileElemDsp>>(signalParam);
```

```
string signalParam = signal.GetParameter<string>(NVHParameterSet.fullLevelElapsed);
```

```
string signalParam = signal.GetParameter<string>(NVHParameterSet.sampleRate);
```

etc.

Reading a Parameter Key Data Type

```
ISignal signal = [IRecording object].Signals[0];  
  
string sigParamType = sig.GetParameterType(NVHParameterSet.sampleRate);  
  
DT_FLOAT  
  
string sigParamType = sig.GetParameterType(NVHParameterSet.fullLevelElapsed);  
DT_DOUBLE
```

etc.

Reading and Displaying a List of Parameter Keys

Given that there is a list of parameters for each signal, it would be better to store the list of parameters into another list object for the user interface and other means of accessing the data.

```
private void ShowCandidateParameters()  
{  
    List<object> keyParameters = new List<object>();  
  
    foreach (var f in typeof(NVHParameterSet).GetFields(BindingFlags.Public |  
                                                         BindingFlags.Static))  
    {  
        if((f.Attributes & FieldAttributes.Literal) == FieldAttributes.Literal &&  
            f.FieldType == typeof(string))  
        {  
            object val = f.GetValue(null)?.ToString();  
            keyParameters.Add(val);  
        }  
    }  
  
    if (keyParameters.Count > 0)  
        lbSignalParameters.Items.AddRange(keyParameters.ToArray());  
    if (lbSignalParameters.Items.Count > 0)  
        lbSignalParameters.SelectedIndex = 0;  
}
```

Then, with the same as the previous Reading Signal sections, include the code snippet from **Reading the Signal Properties – Using a List to Store and Recall Signals** to read the list of signals from IRecording.

```
var recordingPath = "C:\\Sig001.atfx";  
if (RecordingManager.Manager.OpenRecording(recordingPath, out IRecording rec))  
{  
    ShowSignals(rec);  
    ShowCandidateParameters();  
}
```

Record Information	Signal Data Information	Channel Tabl
Block(Ch1)	Cl_sample rate	
Block(Ch2)	_Cl_testName	
Block(drive)	_Cl_testStatus	
APS(Ch1)	_Cl_testType	
APS(Ch2)	_Cl_testRunfolder	
APS(drive)	_Cl_spiderSN	
control(f)	_Cl_testStopReason	
noise(f)	_Cl_testSchedule	
profile(f)	_Cl_testProfile	
HighAbort(f)	_Cl_testAbortLimit	
HighAlarm(f)	_Cl_testAlarmLimit	
LowAbort(f)	_Cl_testNotchLimit	
LowAlarm(f)	_Cl_vcsLevel	
H(f)	_Cl_fullLevelElapsed	
limit_notch(Ch1)	_Cl_remaining	
limit_notch(Ch2)	_Cl_controlRMS	
limit_high_abort(Ch1)	_Cl_controlPeak	
limit_high_abort(Ch2)	_Cl_targetRMS	
limit_high_alarm(Ch1)	_Cl_targetPeak	
limit_high_alarm(Ch2)	_Cl_controlStrategy	
	Cl_lines	

```

private void ShowParameters(DataGridView grid, ISignal sig, string parameterKey)
{
    grid.Rows.Clear();
    clmSignalProp.HeaderText = "Data Type";
    clmSignalPropValue.HeaderText = "Value";

    if (sig != null && !string.IsNullOrEmpty(parameterKey))
    {
        string signalParam = sig.GetParameter<string>(parameterKey);
        string sigParamType = sig.GetParameterType(parameterKey);

        if (!string.IsNullOrEmpty(signalParam))
        {
            if (parameterKey == NVHParameterSet.testProfile)
            {
                clmSignalProp.HeaderText = "Frequency";
                clmSignalPropValue.HeaderText = "Profile";
                List<ProfileElemDsp> profile =
                    Utility.JsonDeserialize<List<ProfileElemDsp>>(signalParam);

                foreach (var entry in profile)
                {
                    grid.Rows.Add(entry.fFreq, entry.fProfile, entry.fHighAbort,
                        entry.fHighAlarm, entry.fLowAlarm, entry.fLowAbort);
                }
            }
            else if (parameterKey == NVHParameterSet.testAbortLimit ||
                parameterKey == NVHParameterSet.testAlarmLimit ||
                parameterKey == NVHParameterSet.testNotchLimit)
            {
                clmSignalProp.HeaderText = "Frequency";
                clmSignalPropValue.HeaderText = "Acc";
                SortedList<string, List<SpiderChannelLimitElem>> limits =
                    Utility.JsonDeserialize<SortedList<string,
                        List<SpiderChannelLimitElem>>>(signalParam);

                foreach (var channelLimit in limits)
                {
                    grid.Rows.Add(channelLimit.Key);
                }
            }
        }
    }
}

```


Name	Type
Altitude	double
GPSEnabled	bool
Latitude	double
Longitude	double
MeasurementBegin	DateTime
MeasurementEnd	DateTime
NanoSecondElapsed	int

The code snippet below shows the extraction and display of data.

```
private void ShowMergeInfo(IRecording rec)
{
    try
    {
        dgvMergeInfo.SuspendLayout();
        dgvMergeInfo.Rows.Clear();
        if (rec is ODSNVHATFXMLRecording atfxRec)
        {
            NVHMeasurement measurement = atfxRec.ODSInstance.Measurement as NVHMeasurement;
            if (measurement?.SigMap_SrcName.Count > 0)
            {
                if (!(rec is ODSNVHATFXMLRecording))
                    return;
                if ((rec as ODSNVHATFXMLRecording).ODSInstance.Measurement as NVHMeasurement ==
null)
                    return;

                if (measurement.SigMap_SrcName.Count == 0)
                {
                    dgvMergeInfo.Columns[0].Visible = false;
                    dgvMergeInfo.Columns[1].Visible = false;
                    for (int i = 0; i < rec.Signals.Count; i++)
                    {
                        dgvMergeInfo.Rows.Add(null, null, rec.RecordingProperty.RecordingName,
rec.Signals[i].Properties.SignalName);
                    }
                }
                else
                {
                    dgvMergeInfo.Columns[0].Visible = true;
                    dgvMergeInfo.Columns[1].Visible = true;
                    int counter = 0;
                    for (int i = 0; i < measurement.SigMap_SrcName.Count; i++)
                    {
                        int SigMapChCount = Convert.ToInt32(measurement.SigMap_ChCount[i]);
                        for (int j = 0; j < SigMapChCount; j++)
                        {
                            dgvMergeInfo.Rows.Add(measurement.SigMap_SrcName[i],
measurement.SigMap_ChName[j + counter],
```


CloseRecording	Method	Close the file
-----------------------	--------	----------------

1. OpenRecording

a. Description

Find and open the file based on the given file path. An IRecording object and the result are returned.

Parameters	Type	Description
recordingPath	String	The path where the file is located.
recording	IRecording	The variable which the returned object is store to.

b. Return

Type	Description
bool	true: the file is loaded false: failed to load the file

Example:

```
var recordingPath = @"C:\REC001.atfx";
if(RecordingManager.Manager.OpenRecording(recordingPath,out var rec))
{
    Console.WriteLine("Recording opened");
}
```

2. CloseRecording

a. Description

Find and close the file based on the given file path. The result is returned.

Parameters	Type	Description
recordingPath	string	The path where the file is located.

b. Return

Type	Description
bool	true: the file is closed false: failed to close the file

Example:

```
var recordingPath = @"C:\REC001.atfx";
if(RecordingManager.Manager.CloseRecording(recordingPath))
{
    Console.WriteLine("Recording closed");
}
```

ODS Recording Module

Name to Be Called	Type	Description
RecordingProperty	Property	Properties of the file
Signals	Property	Signals included in the file
ODSInstance	Property	ODS instances included in the file

The IRecording object can be converted to ODSRecording object before accessing its properties.

- 1) RecordingProperty
 - a. Descriptions

RecordingProperty contains properties of the file (the Recording object), listed below:

Attribute Name	Descriptions
User	The EDM account name when the file was created.
Instruments	The product name used to record/save data to the file.
TestNote	Test notes given by the user before the test ran
Name	File Name
RecordingPath	File Path
Version	EDM version number when the file was created.
CreateTime	This parameter defines when the signal was recorded. It is not when the file is saved. This parameter can show the time accuracy as high as second. To obtain the starting recording time with better accuracy, please add “NanoSecondElapsed” in integer that represents the additional nanoseconds elapsed.

MasterSN	Serial number of the master module of the system when the file was created
UserAnnotation	Annotation added by the user
MeasurementType	Measurement type of the file

Example:

```
var recordingPath = @"C:\REC001.atfx";
if(RecordingManager.Manager.OpenRecording(recordingPath,out var rec))
{
    Console.WriteLine(rec.RecordingProperty.User);
    Console.WriteLine(rec.RecordingProperty.Instruments);
    //can list more recording properties
}
```

2) Signals

a. Descriptions

It returns the list of signals saved in the file. Each signal can be accessed by the ODS Signal module.

Example:

```
var recordingPath = @"C:\REC001.atfx";
if(RecordingManager.Manager.OpenRecording(recordingPath,out var rec))
{
    foreach(var signal in rec.Signals)
    {
        Console.WriteLine($"{signal.Name}-{signal.Type}");
    }
}
```

3) ODSInstance

3.1 Descriptions

The ODSInstance attribute can be accessed only after the IRecording object returned by the Recording Manager module is converted to ODSRecording object.

Each ODS attributes can be accessed through the ODSInstance attribute, e.g. ODSInstance.Measurement.Equipments return the list of EquipmentPart, which corresponds to an input channel.

Example:

```

var recordingPath = @"C:\REC001.atfx";
if(RecordingManager.Manager.OpenRecording(recordingPath,out var rec) && rec is ODSRecording odsRec)
{
    //get measurement
    var measurement = odsRec.ODSInstance.Measurement;
    //get all ods parameter set
    var parameters = odsRec.ODSInstance.ParamSets;
    //get equipments
    var equipments = odsRec.ODSIntance.Environment.Equipments
    //get more ODS instance
}

```

ODS Signal Module

Name to Be Called	Type	Descriptions
Name	Attirbute	Signal Name
Type	Attirbute	Signal type, time/frequency domain
FrameCount	Attirbute	Total number of frames in the signal
FrameSize	Attirbute	Size of each frame
UnitX	Attirbute	Unit of X-axis
UnitY	Attirbute	Unit of Y-axis
Properties	Attirbute	Signal properties. Different signal types have different properties
GetFrame	Method	Return data of the specified frame of the signal A snapshot of measurement data consisting of X, Y and sometimes Z values.
GetParameter<T>	Method	Return the specified parameter by the given name.
GetParameterType	Method	Return the specified parameter data type by the given name.

1. Properties

a. Descriptions

Time domain and frequency domain signals have different signal properties.

For time domian signals, Properties refer to SignalProperties.

For frequency domian signals, Properties refer to FrequencyDomainSignalProperties.

Example:

```

var recordingPath = @"C:\REC001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out var rec))
{
    foreach (var signal in rec.Signals)
    {
        if (signal.Type == SignalType.Time)
        {
            Console.WriteLine(signal.Properties.BlockSize);
        }
        else if (signal.Type == SignalType.Frequency
            && signal.Properties is FrequencyDomainSignalProperties freqProps)
        {
            Console.WriteLine(freqProps.SpectrumAverageMode);
        }
    }
}
}

```

2. GetFrame

a. Descriptions

Return data of the specified frame of the signal

Parameters	Type	Descriptions
frameIndex	int	Index of the frame

b. Return

Type	Descriptions
double[][]	Signal data double[0] contains values of X-axis double[1] contains values of Y-axis double[2] contains values of Z-axis (if available)

Example:

```

var recordingPath = @"C:\REC001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out var rec))
{
    foreach (var signal in rec.Signals)
    {
        if (signal.Type == SignalType.Frequency)
        {
            for(var index = 0;index< (int)signal.FrameCount;index++)
            {
                var frameData = signal.GetFrame(index);
                Console.WriteLine($"X value length:{frameData[0].Length}");
                Console.WriteLine($"Y value length:{frameData[1].Length}");
                Console.WriteLine($"Z value length:{frameData[2].Length}");
            }
        }
    }
}

```

3. GetParameter<T>

a. Descriptions

Search through all ODS parameters for the one including the keyword (parameterKey). It will be returned if found.

Parameters	Type	Descriptions
T	Parameter type	Specifies the type of the object* to be returned
parameterKey	string	Keyword of the object* to be returned

*An object refers to an ODS parameter of the signal.

b. Return

Type	Descriptions
T	The type of the returned object* is determined by the object* found in ODS parameters. If it is not found according to the keyword, the original type is returned.

*An object refers to an ODS parameter of the signal.

Example:

```

var recordingPath = @"C:\REC001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out var rec))
{
    foreach (var signal in rec.Signals)
    {
        var samplingRate = signal.GetParameter<double>(NVHParameterSet.samplingRate);
        Console.WriteLine(samplingRate);
        var testName = signal.GetParameter<string>(NVHParameterSet.testName);
        Console.Write(testName);
    }
}

```

DateTimeNano Module

Constructors	Descriptions
DateTimeNano(DateTime, uint)	Using this Constructor with a IRecording.RecordingProperty.CreateTime and a NVHMeasurement.NanoSecondElapsed will create a DateTimeNano object that contains a DateTime with ms_us_ns.

Example:

```

var recordingPath = @"C:\REC001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out var rec))
{
    ODSNVHATFXMLRecording nvhRec = rec as ODSNVHATFXMLRecording;
    NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;
    DateTimeNano createTimeUTC = new DateTimeNano(nvhRec.Environment.GetUTCTime
(nvhRec.RecordingProperty.CreateTime), nvhMeasurement.NanoSecondElapsed);
}

```

Name to Be Called	Type	Descriptions
IsNanoTime	bool	Gets whether ms_us_ns exists / not equal to zero
TotalNanoSeconds	ulong	Get TotalSeconds in Nano Seconds
ToNanoString	string	Gets a string in the format of "DateTime Milisecond.Microsecond.Nanosecond"
ms_us_ns	uint	We use this NanoSeconds==0 Distinguish between normal time and nanosecond time Milisecond.Microsecond.Nanosecond

		000/000/000
--	--	-------------

Example:

```
var recordingPath = @"C:\REC001.atfx";
if (RecordingManager.Manager.OpenRecording(recordingPath, out var rec))
{
    ODSNVHATFXMLRecording nvhRec = rec as ODSNVHATFXMLRecording;
    NVHMeasurement nvhMeasurement = nvhRec.Measurement as NVHMeasurement;

    DateTimeNano createTimeUTC = new
DateTimeNano(nvhRec.Environment.GetUTCtime(nvhRec.RecordingProperty.CreateTime),
nvhMeasurement.NanoSecondElapsed);

    Console.WriteLine(createTimeUTC.IsNanoTime);
    Console.WriteLine(createTimeUTC.ms_us_ns);
    Console.WriteLine(createTimeUTC.TotalNanosec);
    Console.WriteLine(createTimeUTC.ToNanoString());
}
```

Property Glossary

The following properties and functions can be found in the chm file and are listed here for a quicker reference and to highlight the most important properties and functions for the ATFX API.

RecordingProperty

Property	Type	Description
CreateTime	DateTime	This parameter defines when the signal was recorded. It is not when the file is saved. This parameter can show the time accuracy as high as second. To obtain the starting recording time with better accuracy, please add “NanoSecondElapsed” in integer that represents the additional nanoseconds elapsed.
DeviceSNs	string	Serial numbers of the 1 or many modules used in the recording
Instruments	string	The product name used to record/save data to the file.
MasterSN	uint32	Serial number of the master module of the system when the file was created
MeasurementType	MeasurementConfigType	Measurement type of the file

Name	string	File Name
RecordingPath	string	File Path
RecordingTypeName	string	Recording Type Name based on its file extension
TestNote	string	Test notes given by the user before the test ran
Type	RecordingType	The type of recording based on its file extension Ex. ATRX, GPS, TS
User	string	The EDM account name when the file was created.
UserAnnotation	string	Annotation added by the user
Version	Version	EDM version number when the file was created.

SignalProperties

Property	Type	Description
BlockSize	uint64	Number of time data points captured in the signal
DeviceSN	string	The recording instrument serial numbers
Duration	string	Amount of time covered by the signal
GeneratedTime	DateTimeNano	The time when the data is saved
Instruments	string	The recording instruments used in measurement
IsVCSSignal	bool	Determines if VCS Signal from Random, Sine, Shock, or TWR
MeasurementType	MeasurementConfigType	Measurement type of the signal
NvhType	_NVHType	The Noise, Vibration, and Harshness Type of the signal
RecordingProperties	RecordingProperty	The recording property of the signal
SamplingRate	string	Number of data samples acquired per second
SignalName	string	Signal Name
SignalType	SignalType	Signal type, time/frequency domain

SoftwareVersion	Version	The software version of the recording instrument when the data is saved
UnitX	string	Engineering Unit of X-axis
UnitY	string	Engineering Unit of Y-axis
UnitZ	string	Engineering Unit of Z-axis

NVHParameterSet Parameter Keys

The following property list derived from the ISignal GetParameter<T> and GetParameterType where the functions gets the the value and data type of each parameter key.

Parameter Key	Type	Description
controlPeak	double	Control peak (m/s ²) when data saved
controlRMS	double	Current control RMS (m/s ²) when data saved
currentFrequency	float	Current frequency when data saved (Sine)
displacementPkPk	double	Displacement peak peak (m) when data saved
DOF	long	Degree Of Freedom
drivePK	double	Current drive peak (voltage) when data saved
fullLevelElapsed	double	Time since full level has elapsed in seconds Ex. 636.2
remaining	double	Time remaining in test schedule in seconds Ex. 299
sampleRate	float	Number of data samples acquired per second Ex. 5120
spiderSN	string	The recording device serial number Ex. "2590976"
spiderSystem	string	The recording instrument system configuration Ex. "SYS_2590976"
sweepCount	long	The test amount of times for sweep (Sine)
targetPeak	double	Target peak (m/s ²) when data saved
targetRMS	double	Target RMS (m/s ²) when data saved
testAbortLimit	string	The test abort limit profile

testAlarmLimit	string	The test alarm limit profile
testLastRunTime	string	Last run time of the test Ex. "03/07/2022 15:12:00"
testLastSavedTime	string	Last saved time of the test Ex. "03/07/2022 15:23:19"
testName	string	The test name Ex. "Random34", "Shock1"
testNotchLimit	string	The test notch limit profile
testProfile	string	The test profile
testSchedule	string	The test event schedule Ex. <pre> Loop number: 1 Level 25.00%, duration 00:00:10 Level 50.00%, duration 00:00:10 Level 75.00%, duration 00:00:10 Level 100.00%, duration 00:05:00 End loop My Report (Create Report) 2 </pre>
testStatus	string	The test status Ex. "Running", "Stopped"
testType	string	The test type Ex. "VCS_Random"
totalElapsed	double	Total elapsed time when data saved (time in Random/Sine/TDR, pulses in Shock system)
velocityPk	double	Velocity peak (m/s) when data saved

AoEnvironment

Property	Type	Description
TimeZone	string	The local timezone of where the recording instrument is Examples: "UTC-07:00", "UTC+05:45" Timezones are additional information, they do not change time values.

Function	Return Type	Description
GetLocalTime	DateTime	Get time in local format Ex. 3/18/2022 6:46:32 PM

GetUTCTime	DateTime	Get time in UTC format Ex. 3/18/2022 2:46:32 PM
-------------------	----------	--

NVHMeasurement

Property	Type	Description
Altitude	double	The measurement of altitude according to the device position
GPSEnabled	bool	Determines whether GPS location is on or off
Latitude	double	The measurement of latitude according to the device position
Longitude	double	The measurement of longitude according to the device position
MeasurementBegin	DateTime	The begin time of the measurement when the data is measured
MeasurementEnd	DateTime	The end time of the measurement when the data is measured
NanoSecondElapsed	uint32	The total elapsed time in nano seconds since measurement begin. This parameter can be used together with CreateTime to construct a complete recording starting time that has a format of: yyyy/mm/dd/hh/ss/ms/us/ns

NVHEnvironment

Property	Type	Description
TimeZone	string	The local timezone of where the recording instrument is Examples: "UTC-07:00","UTC+05:45" Timezones are additional information, they do not change time values.
InstruSoftwareVersion	string	The software version of the recording instrument when the data is saved
HardwareVersion	string	The hardware version of the recording instrument when the data is saved
FirmwareVersion	string	The firmware version of the recording instrument when the data is saved

BitVersion	string	The bit version of the recording instrument when the data is saved
-------------------	--------	--

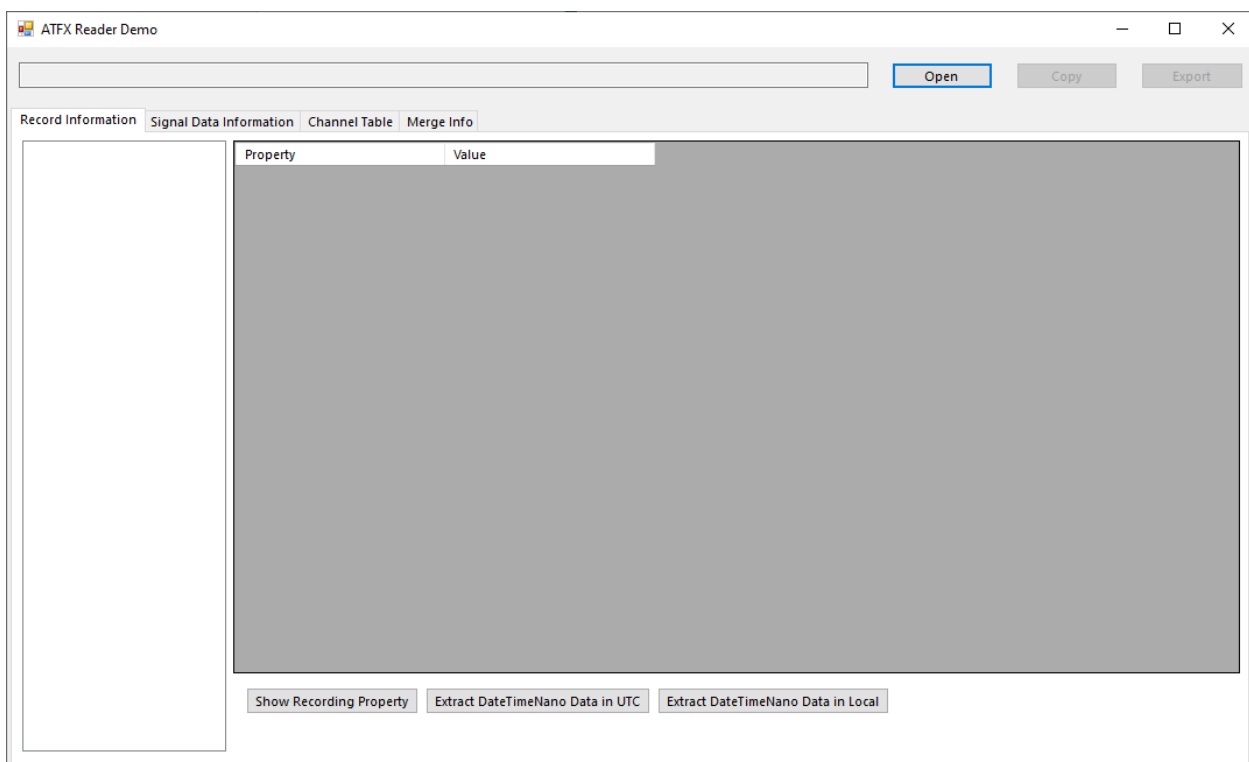
ATFX API Coding Languages

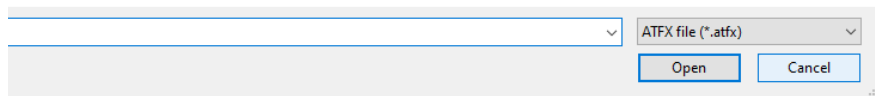
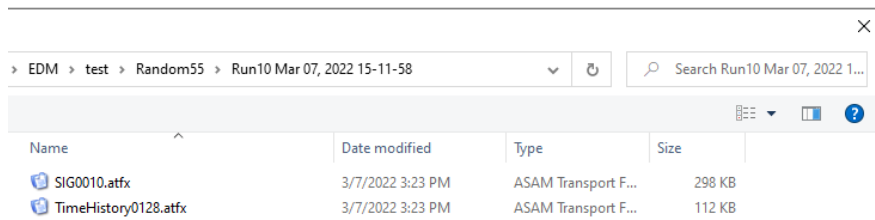
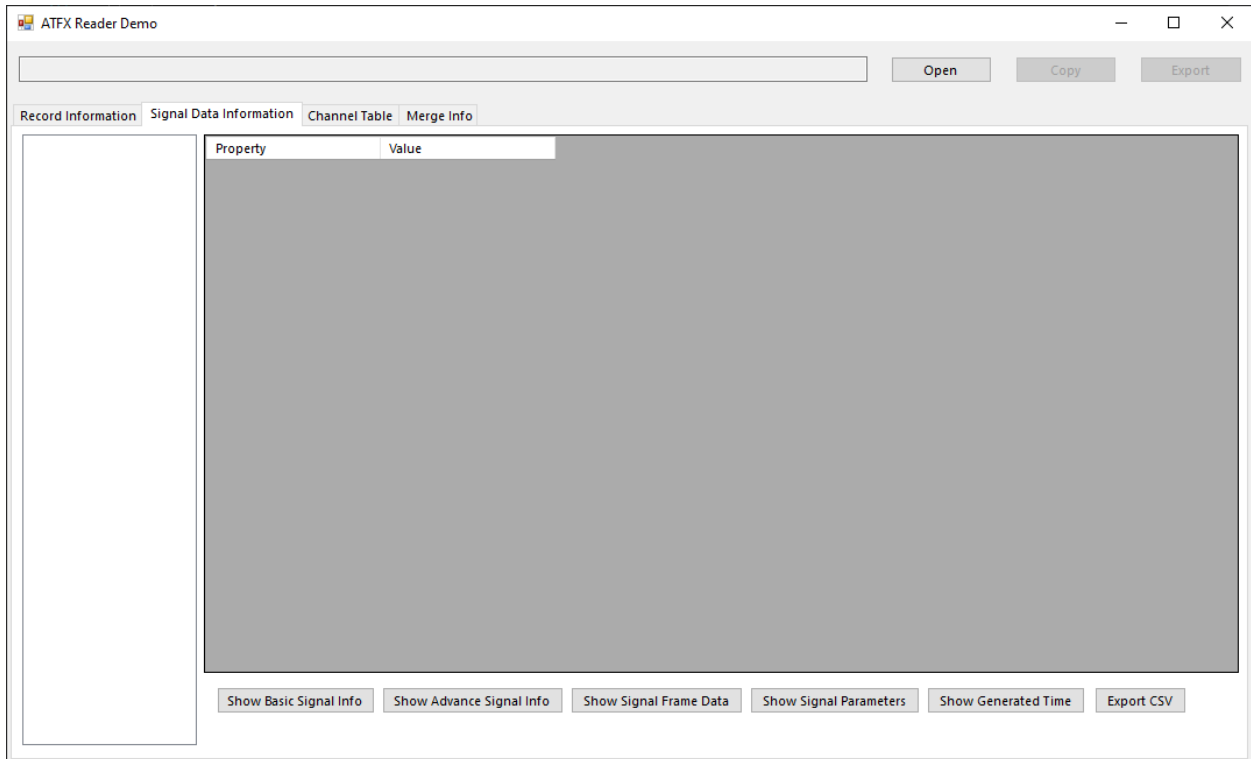
The ATFX API have C# DLL files that are used with the C# language, but there are ways to use the DLL files for other languages such as Python, LabVIEW and Matlab. The following section will demonstrate how to import the DLL files and how to call the functions and properties.

C# Demo Program

This is a demo program that demonstrates the API with a user interface that opens and displays the data stored in a ATFX file for the user to see. Instructions to how to import the DLL files and how to call the functions and properties are listed in the **API C# Demo Examples**.

Upon launching the demo program, click Open to select a ATFX file and the program will display the stored data.





ATFX Reader Demo

C:\Users\KevinCheng\Downloads\gps test example\Run10 Mar 07, 2022 15-11-58\SIG0010.atfx

Record Information | Signal Data Information | Channel Table | Merge Info

Property	Value
User	Admin
Instruments	Spider
TestNote	Random55/Run10
Name	SIG0010
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.30
DeviceSNs	2590976
MasterSN	2590976
UserAnnotation	Random55/Run10
MeasurementType	VCS_Random
Time Zone	UTC-05:00
Created Time (Local)	3/7/2022 3:23:19 PM
Created Time (UTC)	3/7/2022 8:23:19 PM

The below images show the various type of data stored in a ATFX file:

- 1) Record Information – Contains information regarding data format, the EDM version, spider device and so on.

Record Information | Signal Data Information | Channel Table | Merge Info

Property	Value
User	Admin
Instruments	Spider
TestNote	Random55/Run10
Name	SIG0010
RecordingPath	C:\Users\KevinCheng\Downloa...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.30
DeviceSNs	2590976
MasterSN	2590976
UserAnnotation	Random55/Run10
MeasurementType	VCS_Random
Time Zone	UTC-05:00
Created Time (Local)	3/7/2022 3:23:19 PM
Created Time (UTC)	3/7/2022 8:23:19 PM

- 2) DateTimeNano Data – Contains information regarding the recording create time and nanoseconds

Record Information		Signal Data Information	Channel Table	Merge Info
EDM.Recording.ODSNVHATFXML				
EDM.Recording.TimeStampRecor				
Property	Value			
Year	2022			
Month	4			
Day	18			
Hour	22			
Minute	47			
Second	10			
Millisecond	0			
IsNanoTime	True			
NanoSeconds	629999338			
TotalNanosec	82030629999338			
Date Time	4/18/2022 10:47:10 PM			
TimeOfDay	22:47:10			
ToNanoString()	4/18/2022 10:47:10 PM.629.999...			
Custom Format: yyyy/mm/dd/hh...	2022/4/18/22/47/10/629/999/338			

- 3) Signal Basic Information – Contains information regarding each signal properties, such as engineering units, signal block size, type and so on.

Record Information		Signal Data Information	Channel Table	Merge Info
Block(Ch1)				
Block(Ch2)				
Block(drive)				
APS(Ch1)				
APS(Ch2)				
APS(drive)				
control(f)				
noise(f)				
profile(f)				
HighAbort(f)				
HighAlarm(f)				
LowAbort(f)				
LowAlarm(f)				
H(f)				
Property	Value			
UserAnnotation	Random55/Run10			
MeasurementType	VCS_Random			
SignalType	Time			
GeneratedTime	3/7/2022 3:23:19 PM			
SamplingRate	5.12 kHz			
BlockSize	1024			
FrameCount	1			
Duration	0.2 (s)			
UnitX	S			
UnitY	m/s ²			
UnitZ	N/A			
Instruments	Spider			
DeviceSN	2590976			
SoftwareVersion	10.0.8.30			
NvhType	NonEquidistant			
AcquisitionCalculateMeth...	Undefined			
IsVCSignal	True			
IsLocalRecordSignal	False			

- 4) Signal Advanced Information – Contains more in-depth data values and properties of each signal.

Record Information	Signal Data Information	Channel Table	Merge Info
Block(Ch1)	Property	Value	
Block(Ch2)	testName	Random55	
Block(drive)	testLastSavedTime	3/7/2022 3:23:19 PM	
APS(Ch1)	testLastRunTime	3/7/2022 3:12:00 PM	
APS(Ch2)	level	1	
APS(drive)	drivePK	0.395702868700027	
control(f)	controlRMS	9.74654483795166	
noise(f)	targetRMS	9.8128662109375	
profile(f)	controlPeak	0	
HighAbort(f)	targetPeak	0	
HighAlarm(f)	fullLevelElapsed	636.200012207031	
LowAbort(f)	remaining	299.075012207031	
LowAlarm(f)	totalElapsed	675.049987792969	
H(f)	velocityPk	0.0252673029899597	
	displacementPkPk	0.000182511343155056	
	pulseWidth	0	
	DOF	32	
	currentFrequency	0	
	totalCycle	0	
	remainingCycle	0	
	sweepType	0	

5) Signal Data – Contains the signal frame data.

Record Information	Signal Data Information	Channel Table	Merge Info
Block(Ch1)	X Data-Time (S)	Y Data-m/s ²	
Block(Ch2)	0	0.266612559556961	
Block(drive)	0.000195312502910383	-1.51664209365845	
APS(Ch1)	0.000390625005820766	-5.16747570037842	
APS(Ch2)	0.000585937508731149	-18.7506160736084	
APS(drive)	0.000781250011641532	-18.1530494689941	
control(f)	0.000976562514551915	-18.3249702453613	
noise(f)	0.0011718750174623	-17.2090644836426	
profile(f)	0.00136718752037268	-1.64089691638947	
HighAbort(f)	0.00156250002328306	2.78795480728149	
HighAlarm(f)	0.00175781252619345	9.23579025268555	
LowAbort(f)	0.00195312502910383	13.5108318328857	
LowAlarm(f)	0.00214843753201421	1.53141975402832	
H(f)	0.0023437500349246	-5.12889099121094	
	0.00253906253783498	-15.2864398956299	
	0.00273437504074536	-17.5882568359375	
	0.00292968754365574	-1.06409096717834	
	0.00312500004656613	8.2636547088623	
	0.00332031254947651	8.24568557739258	
	0.00351562505238689	6.91971015930176	
	0.00371093755529728	20.5992469787598	

6) Signal Parameters – Contains a list of signal properties with the properties' names and the properties' values that users can call in custom programs.

Record Information	Signal Data Information	Channel Table	Merge Info
Block(Ch1)	CI_sampleRate		Data Type
Block(Ch2)	CI_testName		Value
Block(drive)	CI_testStatus		DT_FLOAT
APS(Ch1)	CI_testType		5120
APS(Ch2)	CI_testRunFolder		
APS(drive)	CI_spiderSN		
control(f)	CI_testStopReason		
noise(f)	CI_testSchedule		
profile(f)	CI_testProfile		
HighAbort(f)	CI_testAbortLimit		
HighAlarm(f)	CI_testAlarmLimit		
LowAbort(f)	CI_testNotchLimit		
LowAlarm(f)	CI_vcsLevel		
H(f)	CI_fullLevelElapsed		
	CI_remaining		
	CI_controlRMS		
	CI_controlPeak		

7) Signal Generate Time – Contains more advance information regarding a signal or atfx file generated time.

Record Information	Signal Data Information	Channel Table	Merge Info
Block(Ch1)	Property		Value
Block(Ch2)	Year		2022
Block(drive)	Month		3
APS(Ch1)	Day		7
APS(Ch2)	Hour		15
APS(drive)	Minute		23
control(f)	Second		19
noise(f)	Millisecond		0
profile(f)	TimeOfDay		15:23:19
HighAbort(f)	IsNanoTime		False
HighAlarm(f)	TotalNanosec		55399000000000
LowAbort(f)			
LowAlarm(f)			
H(f)			

8) Channel Table – Contains information regarding the signal test’s input channel table.

Record Information	Signal Data Information	Channel Table	Merge Info							
Location ID	Channel Type	Measurement Quantity	Engineering Unit	Sensitivity	Input Mode	Input Range	Sensor SN	Max. sensor range	Intergration	Control Weight
Ch1	Control	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch2	Monitor	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch3	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch4	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch5	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch6	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch7	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1
Ch8	Off	Acceleration	m/s ²	10.19716(mv/m/s ²)	AC_SingleEnd	AutoRange		20	No Integration	1

9) Merge Information – Contains information about mutple other atfx files if the file is merged.

C:\Users\KevinCheng\Downloads\gps test example\MergedSig2.atfx			
Record Information	Signal Data Information	Channel Table	Merge Info
Source File	Channel Label	Current File	Channel Label
{4499520}_REC_{...}	ch1	MergedSig2	ch1
REC5838.atfx	ch1	MergedSig2	ch2

Python Demo Script

Importing C# DLL files

In order to import C# DLL to be used in python, users will have to download a package called **Python.Net**. There are other packages that can also import C# related libraries, such as **IronPython**.

<https://github.com/pythonnet/pythonnet>

```
pip install pythonnet
```

After installing the pythonnet package, users can now import .NET Common Language Runtime, add references to the ATFX API DLL files and import them to the python script. The following code snippet below shows the importation of the ATFX API DLL files.

```
import clr
parentPath =
"C:\\MyStuff\\DevelopmentalVer\\bin\\AnyCPU\\Debug\\Utility\\CIATFXReader\\"

clr.AddReference(parentPath + "CI.ATFX.Reader.dll")
clr.AddReference(parentPath + "Common.dll")
clr.AddReference('System.Linq')
clr.AddReference('System.Collections')

from EDM.Recording import *
from EDM.RecordingInterface import *
from ASAM.ODS.NVH import *
from ASAM.ODS.ATFXML import *
from EDM.Utills import *
from Common import *
from Common.Spider import *
from System import *
from System.Diagnostics import *
from System.Reflection import *
from System.Text import *
from System.IO import *
```

Then users can call any functions and properties from the DLL files and use them accordingly.

Python Script Code Example

An example below shows how to open a recording and show its recording properties, GPS info and one of its signal properties.

```
#---Functions

def ShowContent(recording):
    props = recording.GetType().GetProperties(BindingFlags.Instance | BindingFlags.Public)
    for prop in props:
        content = prop.GetValue(recording, None)
        print(prop.Name, ": ", content)

def ShowGPSInfo(recording):
    if type(recording) is ODSNVHATFXMLRecording:
        nvhRec = recording
        nvhMeasurement = nvhRec.Measurement
        nvhEnvironment = nvhRec.Environment
        bGPS = nvhMeasurement.GPSEnabled
        if bGPS:
            print("GPS Enabled: ", bGPS)
            print("Longitude: ", nvhMeasurement.Longitude)
            print("Latitude: ", nvhMeasurement.Latitude)
            print("Altitude: ", nvhMeasurement.Altitude)
            print("Nanoseconds Elapsed: ", nvhMeasurement.NanoSecondElapsed)

        if not String.IsNullOrEmpty(nvhRec.Environment.TimeZone):
            print("Time Zone: ", nvhRec.Environment.TimeZone)

        print("Created Time (Local): ", nvhRec.RecordingProperty.CreateTime)
        print("Created Time (UTC): ",
nvhRec.Environment.GetUTCTime(nvhRec.RecordingProperty.CreateTime))

#---Main Code
```

```

print("Running Main Code")
recordingManager = RecordingManager

recordingPath = "C:\\Users\\KevinCheng\\Downloads\\gps test example\\"
recordingPathRegular = recordingPath + "SIG0020.atfx"
recordingPathTS = recordingPath + "{4499520}_REC_{20220419}(1).atfx"
recordingPathGPS = recordingPath + "REC0041.atfx"

#OpenRecording(string, out IRecording)
# dummy data is required for the OpenRecording for it to correctly return all data
dummyTest1, recording = RecordingManager.Manager.OpenRecording(recordingPathTS,
None)
print("\nRecording Properties\n")
ShowContent(recording.RecordingProperty)

print("\nRecording GPS Properties\n")
ShowGPSInfo(recording)

print("\nSignal 1 Properties\n")
ShowContent(recording.Signals[0].Properties)

print("\nSignal 1 Properties GeneratedTime\n")
ShowContent(recording.Signals[0].Properties.GeneratedTime)

```

Example Print Statements

```

Running Main Code

Recording Properties

User : Unknown Owner
Instruments : GRS
TestNote : Untitled Test Note

```

Name : {4499520}_REC_{20220419}(1)

RecordingPath : C:\Users\KevinCheng\Downloads\gps test example\{4499520}_REC_{20220419}(1).atfx

Type : 0

RecordingTypeName : ASAM ODS Format - XML

Version : 10.0.8.34

CreateTime : 4/18/2022 6:47:10 PM

DeviceSNs : 4499520

MasterSN : 4499520

UserAnnotation : None

MeasurementType : 0

Recording GPS Properties

GPS Enabled: True

Longitude: 0.0

Latitude: 37.38046

Altitude: 12.42

Nanoseconds Elapsed: 629999338

Time Zone: UTC-05:00

Created Time (Local): 4/18/2022 6:47:10 PM

Created Time (UTC): 4/18/2022 10:47:10 PM

Signal 1 Properties

RecordingProperties : EDM.RecordingInterface.RecordingProperty

UserAnnotation : None

MeasurementType : 0

SignalType : 1

GeneratedTime : 4/18/2022 6:47:10 PM.629.999.338

SignalName : ch1

SamplingRate : 51.20 kHz

BlockSize : 1793024
FrameCount : 1
Duration : 35.02 (s)
UnitX : S
UnitY : V
UnitZ : N/A
Instruments : GRS
DeviceSN : 4499520
SoftwareVersion : 10.0.8.34
NvhType : 0
AcquisitionCalculateMethod : 0
IsVCSSignal : False
IsLocalRecordSignal : False
DicTraceXMinMax :
System.Collections.Concurrent.ConcurrentDictionary`2[System.String,System.Collections.Generic.KeyValuePair`2[System.Double,System.Double]]
DicTraceRMS :
System.Collections.Concurrent.ConcurrentDictionary`2[System.String,System.Double]
DicTraceInComposite :
System.Collections.Concurrent.ConcurrentDictionary`2[System.String,System.Boolean]
DicTraceAVD :
System.Collections.Concurrent.ConcurrentDictionary`2[System.String,System.Boolean]

Signal 1 Properties GeneratedTime

Year : 2022
Month : 4
Day : 18
Hour : 18
Minute : 47
Second : 10
Millisecond : 0
TimeOfDay : 18:47:10
IsNanoTime : True

TotalNanosec : 67630629999338

The python script in the ATFX API package has more examples such as getting a list of signals and displaying the frame data of 1 signal and getting a list of recordings and displaying each recording properties.

LabVIEW Demo Script

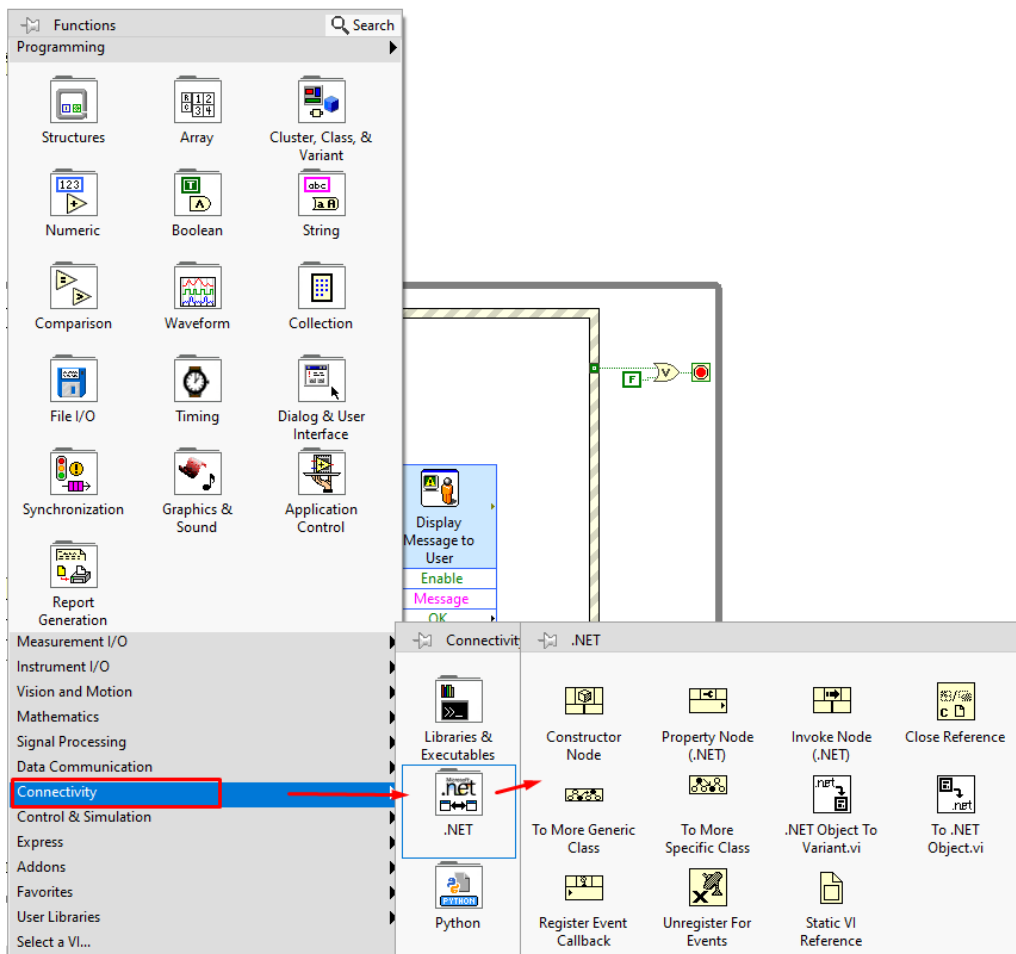
In order to open and run the provided LabVIEW Demo Script, it is recommended to use LabVIEW 2021 or 2021 SP1 version.

Importing C# DLL files

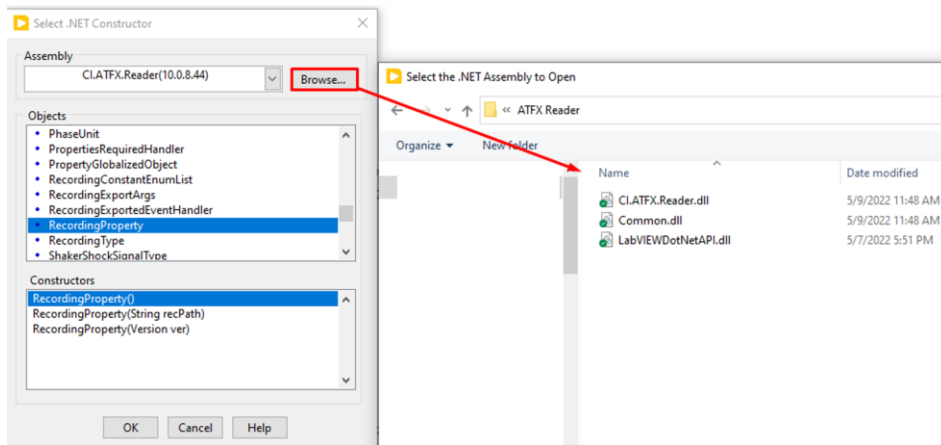
LabView comes with the combatility of importing C# dll files and articles on how to do so.

<https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YGggCAG&l=en-US>

Once the .vi file block diagram is up, users can right click the empty space and locate **Connectivity** -> **.NET** then any of the following nodes.

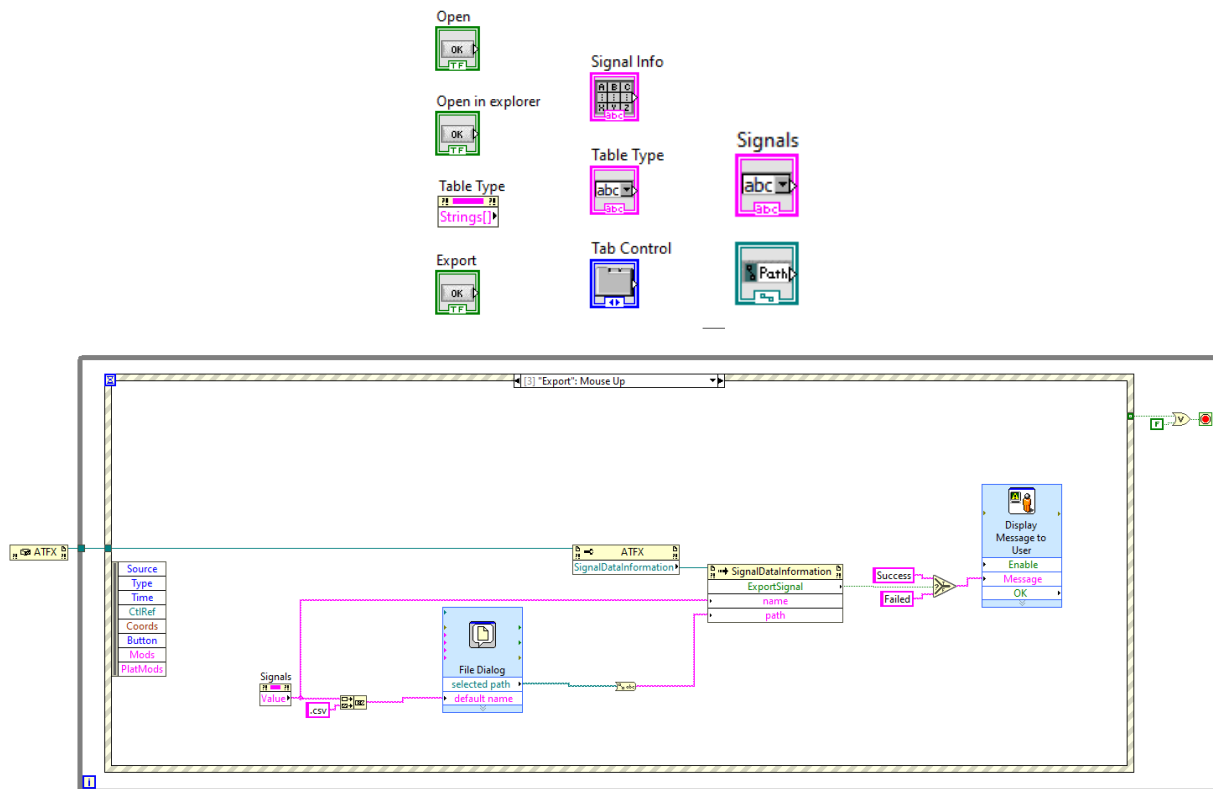


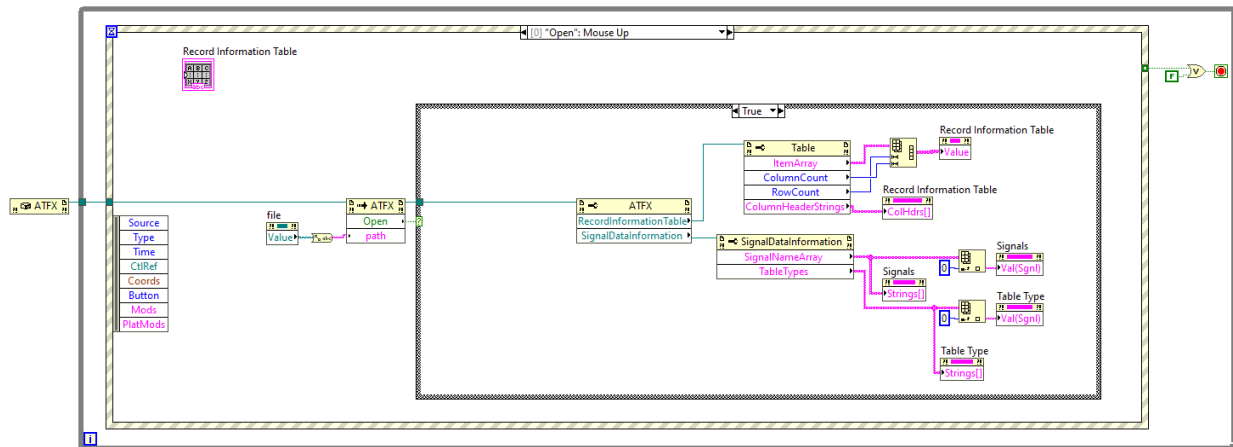
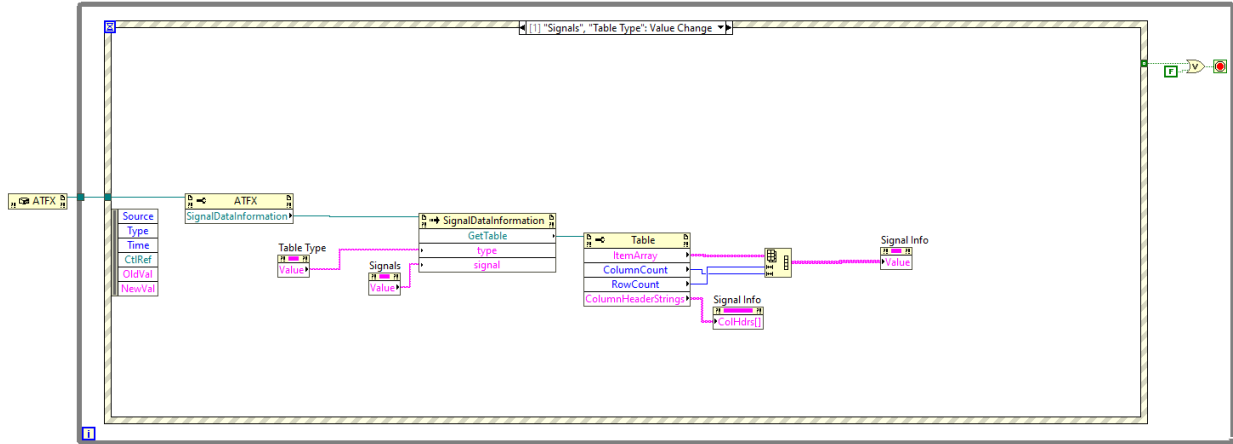
If the user selects the **Constructor Node** and place into the diagram, another window will pop up for selecting the .NET constructor reference. If the ATFX API dll files are not in the assembly list, then users can click **Browse** and add in the dll files.



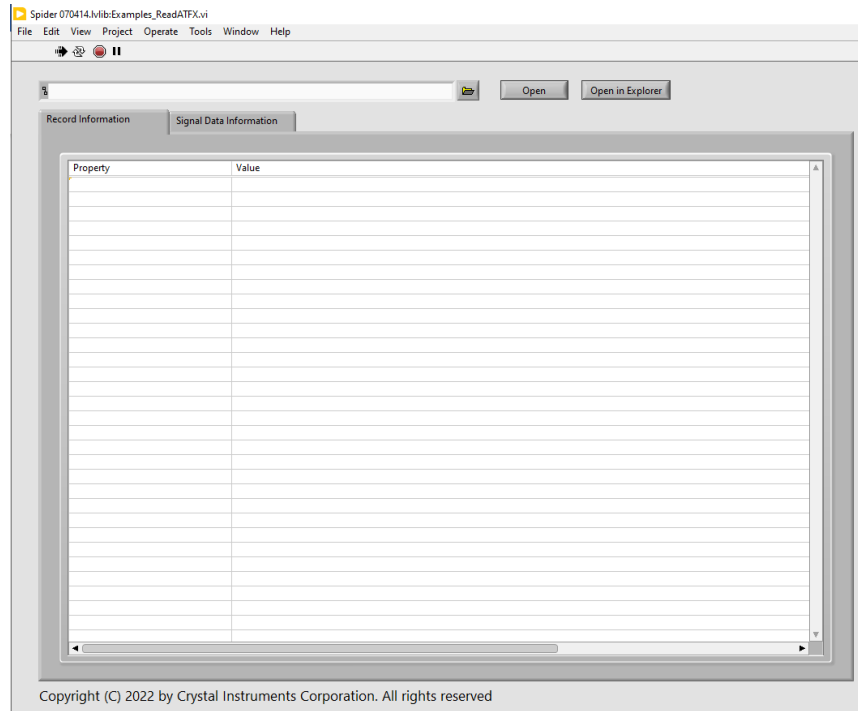
LabVIEW Block Diagram Example

The following shows the block diagram used to open the ATFX file and display its data from the **Examples_ReadATFX.vi** file.

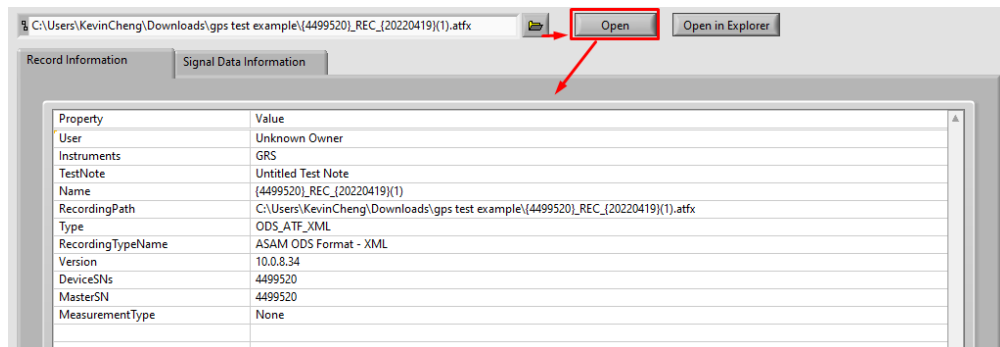




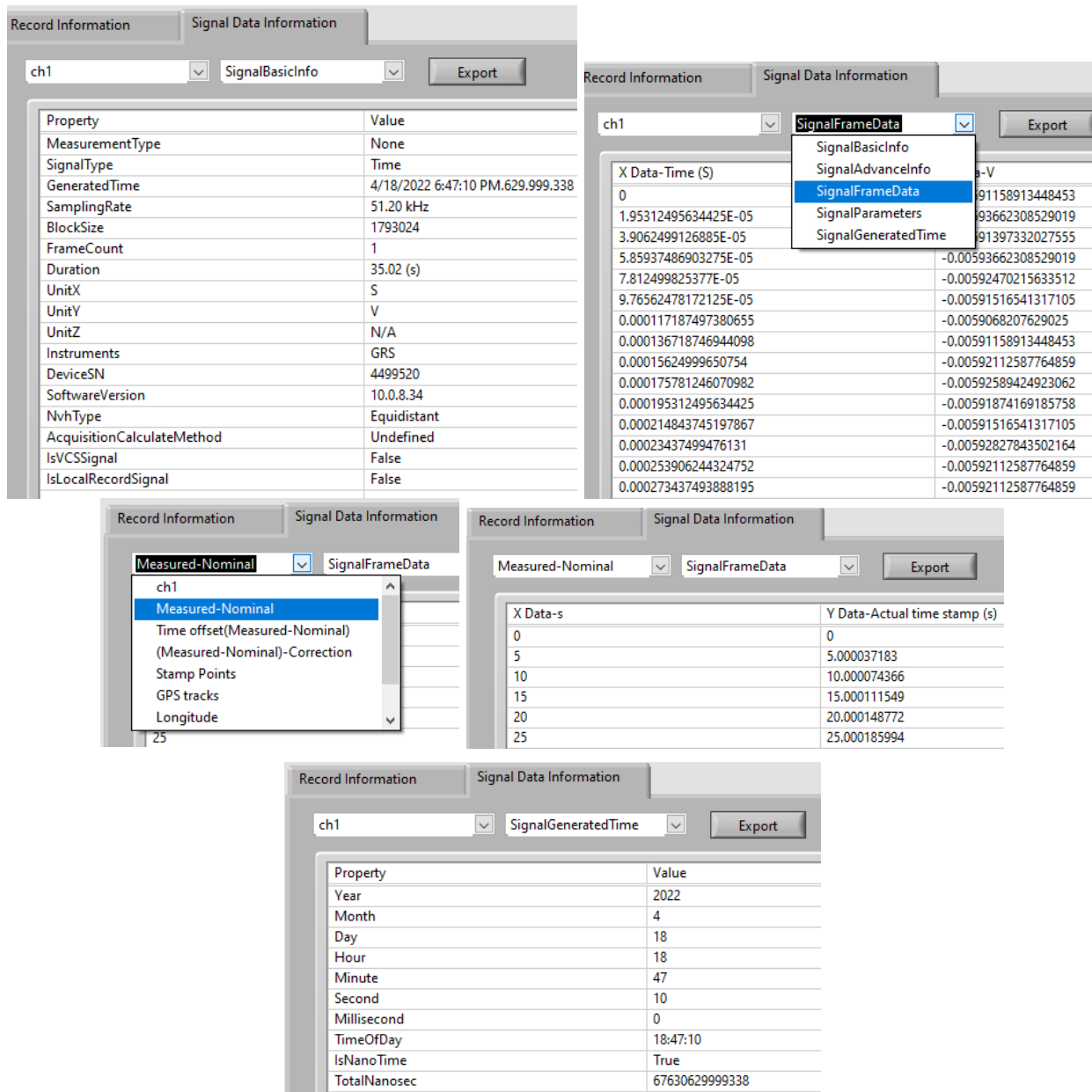
The following shows the GUI of the ATFX API LabView Reader and its usage.



Users open the file folder icon button to locate a atfx file, then click Open to extract and display the recording data.



Here is a display of the signal properties, frame data and generated time data.



Matlab Demo Script

In order to open and run the provided Matlab Demo Script, it is recommended to use Matlab **R2021b** or later version.

Importing C# DLL files

In the recent versions of Matlab allow loading DLL files by using **NET.addAssembly()**.

```
NET.addAssembly('C:\MyStuff\DevelopmentalVer\bin\AnyCPU\Debug\Utility\CIATFXReader\Common.dll');
```

```
NET.addAssembly('C:\MyStuff\DevelopmentalVer\bin\AnyCPU\Debug\Utility\CIATFXReader\CI.ATFX.Reader.dll');
```

Matlab Script Code Example

Then users can call any functions and properties similar to C#.

An example below shows how to open a recording and display its recording properties and signal frame data.

```
%create a atfx recording instance
rec = EDM.Recording.ODSNVHATFXMLRecording('C:\Users\KevinCheng\Downloads\gps
test example\REC0041.atfx');

%use item function to get signal instance
sig = Item(rec.Signals,0);

%display signal properties
disp(System.String.Format("Name:{0}",sig.Name));
disp(System.String.Format("X Unit:{0}",sig.Properties.xUnit));
disp(System.String.Format("Y Unit:{0}",sig.Properties.yUnit));
disp(System.String.Format("GPS Enable:{0}",rec.Measurement.GPSEnabled));
disp(System.String.Format("Longitude:{0}",rec.Measurement.Longitude));
disp(System.String.Format("Latitude:{0}",rec.Measurement.Latitude));
disp(System.String.Format("Altitude:{0}",rec.Measurement.Altitude));
disp(System.String.Format("Time zone:{0}",rec.Environment.TimeZone));
disp(System.String.Format("Created Time (Local):{0}",rec.RecordingProperty.CreateTime));
disp(System.String.Format("Created Time
(UTC):{0}",rec.Environment.GetUTCTime(rec.RecordingProperty.CreateTime)));
disp(System.String.Format("Nanoseconds
Elapsed:{0}",rec.Measurement.NanoSecondElapsed));

disp("display signal frame data");
%get signal frame
frame = sig.GetFrame(0);
%convert .Net double[][] array to matlab cell
matFrame = cell(frame);
```

```

%Long format, showing more decimal places
format long
%display the cell(frame) content
%celldisp(matFrame);
%convert back to mat array
xVals = cell2mat(matFrame(1));
yValues = cell2mat(matFrame(2));

%plot the signal
plot(xVals,yValues,'r');
xlabel(string(sig.Properties.xQuantity)+" (" +string(sig.Properties.xUnit)+")");
ylabel(string(sig.Properties.yQuantity)+" (" +string(sig.Properties.yUnit)+")");
title("Plot of the "+string(sig.Name));
legend(string(sig.Name))

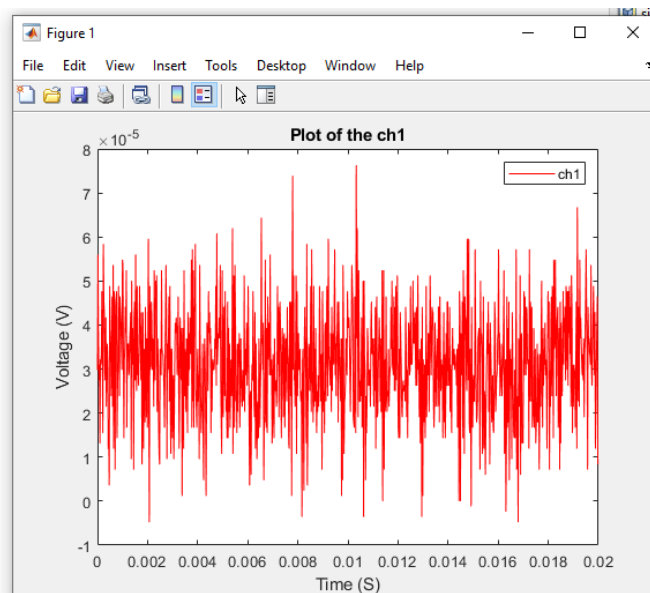
```

Example Output

```

Name:ch1
X Unit:S
Y Unit:V
GPS Enable:True
Longitude:0
Latitude:37.38038
Altitude:8.26
Time zone:UTC-05:00
Created Time (Local):3/23/2022 4:29:41 PM
Created Time (UTC):3/23/2022 8:29:41 PM
Nanoseconds Elapsed:815661371
display signal frame data
>>

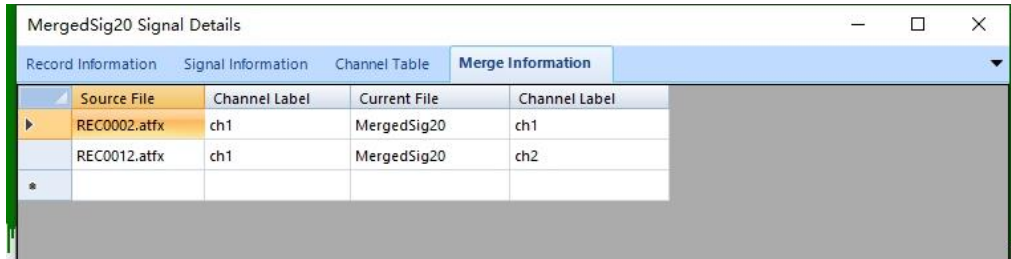
```



Post Analysis Software Integrates ATRX API

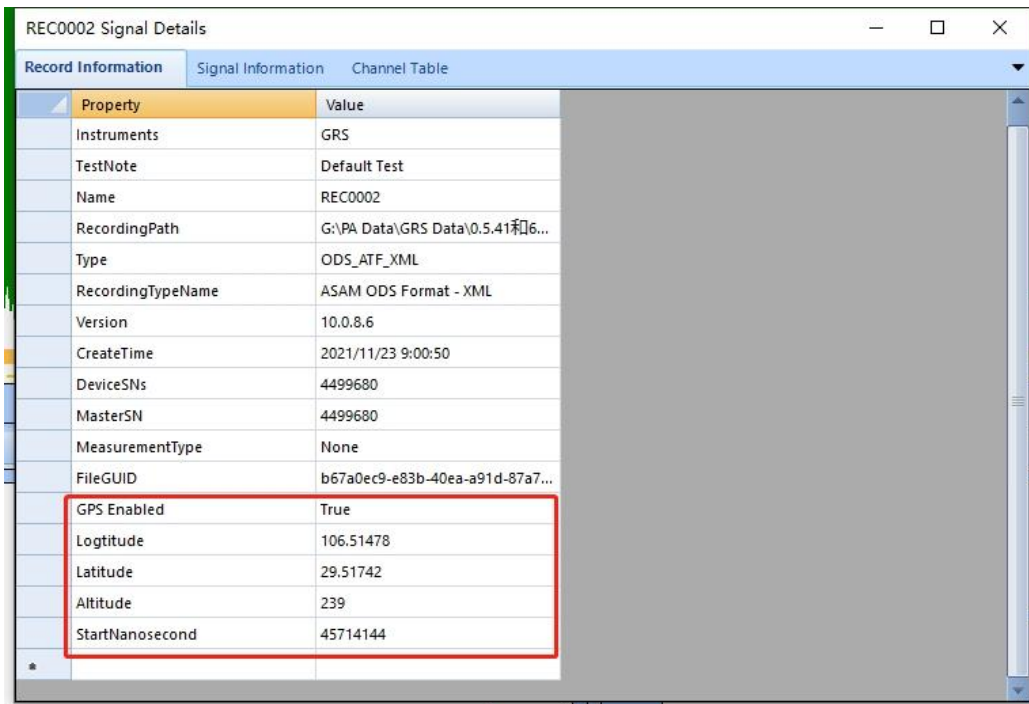
The Feature that Utilizes ATFX Reader API in PA Software

The following screenshots of the Post Analysis Software shows a feature that integrates ATFX Reader API, which reads and shows all the information in atfx files that are created by Crystal Instruments products. The ATFX Reader API not only can be integrated in software products of Crystal Instruments, but also can be licensed to users to customize their software.



MergedSig20 Signal Details

Source File	Channel Label	Current File	Channel Label
REC0002.atfx	ch1	MergedSig20	ch1
REC0012.atfx	ch1	MergedSig20	ch2



REC0002 Signal Details

Property	Value
Instruments	GRS
TestNote	Default Test
Name	REC0002
RecordingPath	G:\PA Data\GRS Data\0.5.41和6...
Type	ODS_ATF_XML
RecordingTypeName	ASAM ODS Format - XML
Version	10.0.8.6
CreateTime	2021/11/23 9:00:50
DeviceSNs	4499680
MasterSN	4499680
MeasurementType	None
FileGUID	b67a0ec9-e83b-40ea-a91d-87a7...
GPS Enabled	True
Longitude	106.51478
Latitude	29.51742
Altitude	239
StartNanosecond	45714144

REC0002 Signal Details

Record Information | **Signal Information** | Channel Table

Property	Value
MeasurementType	None
SignalType	Time
GeneratedTime	2021/11/23 9:00:50.045.706.211
SignalName	ch1
SamplingRate	102.40 kHz
BlockSize	76756992
FrameCount	1
Duration	749.58 (s)
UnitX	s
UnitY	V
UnitZ	N/A
Instruments	GRS
DeviceSN	4499680
SoftwareVersion	10.0.8.6
NvhType	Equidistant
AcquisitionCalculateMethod	Undefined
IsVCSignal	False
IsLocalRecordSignal	False
IsToleranceSignal	False

REC0002 Signal Details

Record Information | Signal Information | **Channel Table**

Ch.	Original sensitivity	Input mode	Hi-Pass filter	Range	Current sensitivity	Label
1	1000 mv/(V)	AC-Single End	1Hz	Auto	1000 mv/(V)	CH1
2	1000 mv/(V)	AC-Single End	1Hz	Auto	1000 mv/(V)	CH2
3	1000 mv/(V)	AC-Single End	1Hz	Auto	1000 mv/(V)	CH3
4	1000 mv/(V)	AC-Single End	1Hz	Auto	1000 mv/(V)	CH4

END USER LICENSE AGREEMENT FOR CRYSTAL INSTRUMENTS SOFTWARE

--- Updated May 11, 2022

IMPORTANT – READ CAREFULLY. This End User License Agreement (“the Agreement”) is a legally binding agreement between you (“the Licensee”) and Crystal Instruments Corporation (“Crystal Instruments”) for the Crystal Instruments EDM (Engineering Data Management) software, PA (Post Analyzer), EDM Cloud, CI Store, EDC (Embedded Device Control), various API, or the embedded software installed in CoCo, Spider and other series hardware, which includes software components and tools and written documentation (“Software”) that accompanies this Agreement. This Agreement contains **WARRANTY AND LIABILITY DISCLAIMERS**.

1. SCOPE OF THE LICENSE RIGHT

1.1 By installing, copying, or using the Software, the Licensee agrees to be bound by the terms of this Agreement.

1.2 Subject to the terms and conditions of this Agreement, Crystal Instruments hereby grants to the Licensee a non-exclusive, non-transferable, right to use the Software, as ordered by the Licensee, solely for the Licensee’s own use and solely with the Crystal Instruments hardware for which it is intended.

1.3 The Licensee shall not be entitled to copy or distribute the Software or parts thereof; publish the Software for others to copy; sell, rent, lease, or lend the Software; or transfer or assign the Software or the license rights to the Software to a third party in any other way whatsoever.

1.4 The Licensee shall, however, be entitled to make back-up copies of the Software to the extent that applicable law expressly permits. The use of the back-up copy shall be subject to the terms of this Agreement.

1.5 The Licensee shall ensure that the Software is stored in such a manner that third parties do not have access to it and that a third party does not come into possession of the Software in any other way. The Licensee shall make all employees who have access to the Software fully aware of this obligation.

2. CHANGES TO THE SOFTWARE

2.1 The Licensee shall not be entitled to make any changes to the Software, or reverse engineer, decompile, or disassemble the Software, except and only to the extent that applicable law expressly permits.

2.2 In the event of the Licensee or a third party interfering with or making any changes to the Software, Crystal Instruments may terminate the Agreement with immediate effect, and Crystal Instruments hereby disclaims any liability for the consequences of such interference or change.

3. INTELLECTUAL PROPERTY RIGHTS

3.1 The Software is protected by copyright law and other intellectual property laws. Crystal Instruments or its suppliers own all copyright and any other intellectual property rights in the Software. The Licensee shall respect Crystal Instruments’ and its suppliers’ rights and the Licensee shall be fully liable in the event of any violation of these rights, including unauthorized passing on of the Software or any part of it to a third party.

3.2 The Licensee shall not be entitled to break, change or delete any security codes or license keys, nor shall the Licensee be entitled to change or remove statements in the Software or on the media on which the Software is delivered regarding copyrights, trademarks, or any other proprietary notices.

3.3 Information and data supplied by Crystal Instruments with the Software, such as, but not limited to, user manuals and documentation, are proprietary to Crystal Instruments or its suppliers. Such information is furnished solely to assist the Licensee in the installation, operation and use of the Software and the Licensee agrees not to reproduce or copy such information, except as is reasonably necessary for proper use of the Software.

4. TRADEMARKS

4.1 The Licensee acknowledges Crystal Instruments’ and its suppliers’ sole ownership of any trademarks including service marks, logos and other proprietary marks submitted with the Software, and all associated goodwill. This Agreement does not grant the Licensee any rights to the trademarks of Crystal Instruments and its suppliers.

4.2 The Licensee agrees not to use the trademarks in any manner that will diminish or otherwise damage Crystal Instruments’ or its suppliers’ goodwill in the trademarks. The Licensee agrees not to adopt, use, or register any corporate name, trade name, trademark, domain name, service mark, certification mark, or other designation similar to, or containing in whole or in part, the trademarks of Crystal Instruments.

5. CLOUD SERVICE PROVIDED BY CRYSTAL INSTRUMENTS

5.1 Data Location When cloud service is enabled, Crystal Instruments Corporation may process and store the customer data anywhere Crystal Instruments Corporation or its agents maintain facilities and services.

5.1.1 Facilities All facilities used to store and process an application and customer data will adhere to reasonable security standards no less protective than the security standards at facilities where Crystal Instruments Corporation processes and stores its own information of a similar type.

5.2 Data Processing and Security

5.2.1 Scope of Processing By entering into this agreement, customer instructs Crystal Instruments Corporation to process customer personal data and other data related to its services only in accordance with applicable law: (a) to provide the cloud services; (b) as further specified by customer via customer’s use of the cloud services (including the admin console and other functionality of the services); (c) as documented in the form of this agreement, including these terms; and (d) as further documented in any other written instructions given by customer and acknowledged by Crystal Instruments Corporation as constituting instructions for purposes of these Terms.

5.2.2 Data Security Crystal Instruments Corporation will use third party technical measures to protect customer data against accidental or unlawful destruction, loss, alteration, unauthorized disclosure or access. Crystal Instruments Corporation is not responsible or liable for the deletion of or failure to store any customer data and other communications maintained or transmitted through use of the services. In addition, Crystal Instruments is not responsible or liable for unauthorized access of the customer data. Customer is solely responsible for securing and backing up data. Crystal

Instruments Corporation does not warrant that the operation of the software or the services will be error-free or uninterrupted. Neither the software nor the services are designed, manufactured, or intended for high risk activities.

5.2.3 Data Deletion

Deletion by Customer: Crystal Instruments Corporation will enable Customer to delete Customer Data during the Term in a manner consistent with the functionality of the Services.

Deletion on Termination. On expiry of the Term, Crystal Instruments would delete all Customer Data. Customer acknowledges and agrees that Customer will be responsible for exporting, before the Term expires, any Customer Data it wishes to retain afterwards.

5.3 Accounts Customer must have an account to use the services, and is responsible for the information it provides to create the account, the security of passwords for the account, and for any use of its account. If customer becomes aware of any unauthorized use of its password or its account, Customer will notify Crystal Instruments Corporation as promptly as possible. Crystal Instruments Corporation has no obligation to provide customer multiple accounts.

5.4 Payment Terms for Cloud Service

5.4.1 Free Quota Certain services are provided to customer without charge up to the fee threshold, as applicable.

5.4.2 Online Billing At the end of the applicable fee accrual period, Crystal Instruments Corporation will issue an electronic bill to customer for all charges accrued above the fee threshold based on (i) Customer's use of the Services during the previous fee accrual period; (ii) any additional units added; (iii) any committed purchases selected; and/or (iv) any package purchases selected. For use above the fee threshold, customer will be responsible for all fees up to the amount set in the account and will pay all fees in the currency set forth in the invoice. If customer elects to pay by credit card, debit card, or other non-invoiced form of payment, Crystal Instruments Corporation will charge (and customer will pay) all fees immediately at the end of the fee accrual period. If customer elects to pay by invoice (and Crystal Instruments Corporation agrees), all fees are due as set forth in the invoice. Customer's obligation to pay all fees is non-cancellable. Crystal Instruments Corporation's measurement of Customer's use of the services is final. Crystal Instruments Corporation has no obligation to provide multiple bills. Payments made via wire transfer must include the bank information provided by Crystal Instruments Corporation.

5.4.3 Payment Information Crystal Instruments Corporation will not store any payment related information on its facilities. All payment information, including recurring payments are stored at a third party facility. Crystal Instruments will not be responsible or liable for unauthorised access to this information.

5.4.4 Taxes for Cloud Services

(a) Customer is responsible for any taxes, and customer will pay Crystal Instruments Corporation for the services without any reduction for taxes. If Crystal Instruments Corporation is obligated to collect or pay taxes, the taxes will be invoiced to customer, unless customer provides Crystal Instruments Corporation with a timely and valid tax exemption certificate authorized by the appropriate taxing authority. In some states the sales tax is due on the total purchase price at the time of sale and must be invoiced and collected at the time of the sale. If customer is required by law to withhold any taxes from its payments to Crystal Instruments Corporation, customer must provide Crystal Instruments Corporation with an official tax receipt or other appropriate documentation to support such withholding. If under the applicable tax legislation the services are subject to local VAT and the customer is required to make a withholding of local VAT from amounts payable to Crystal Instruments Corporation, the value of services calculated in accordance with the above procedure will be increased (grossed up) by the customer for the respective amount of local VAT and the grossed up amount will be regarded as a VAT inclusive price. Local VAT amount withheld from the VAT-inclusive price will be remitted to the applicable local tax entity by the customer and customer will ensure that Crystal Instruments Corporation will receive payment for its services for the net amount as would otherwise be due (the VAT inclusive price less the local VAT withheld and remitted to applicable tax authority).

(b) If required under applicable law, customer will provide Crystal Instruments Corporation with applicable tax identification information that Crystal Instruments Corporation may require to ensure its compliance with applicable tax regulations and authorities in applicable jurisdictions. Customer will be liable to pay (or reimburse Crystal Instruments Corporation for any taxes, interest, penalties or fines arising out of any mis-declaration by the Customer).

5.4.5 Invoice Disputes and Refunds Any invoice disputes must be submitted prior to the payment due date. If the parties determine that certain billing inaccuracies are attributable to Crystal Instruments Corporation, Crystal Instruments Corporation will not issue a corrected invoice, but will instead issue a credit memo specifying the incorrect amount in the affected invoice. If the disputed invoice has not yet been paid, Crystal Instruments Corporation will apply the credit memo amount to the disputed invoice and Customer will be responsible for paying the resulting net balance due on that invoice. To the fullest extent permitted by law, customer waives all claims relating to fees unless claimed within thirty days after charged (this does not affect any customer rights with its credit card issuer). Refunds (if any) are at the discretion of Crystal Instruments Corporation and will only be in the form of credit for the services. Nothing in this Agreement obligates Crystal Instruments Corporation to extend credit to any party.

5.4.6 Delinquent Payments; Suspension Late payments may bear interest at the rate of 1.5% per month (or the highest rate permitted by law, if less) from the payment due date until paid in full. customer will be responsible for all reasonable expenses (including attorneys' fees) incurred by Crystal Instruments Corporation in collecting such delinquent amounts. If customer is late on payment for the services, Crystal Instruments Corporation may suspend the services or terminate the account(s) and services(s) for breach

5.5 Account Term & Termination

5.5.1 Account Term The term of the account will begin on the effective date and continue until the agreement is terminated.

5.5.2 Termination for Breach Crystal Instruments Corporation may terminate account for breach if: (i) the account(s) is in material breach of the agreement; or (ii) the customer ceases its business operations or becomes subject to insolvency proceedings and the proceedings are not dismissed within ninety days.

5.5.3 Termination for Convenience Customer may stop using the cloud service at any time. Customer may terminate the account(s) and services for its convenience at any time on prior written notice and upon termination, must cease use of the applicable services.

Crystal Instruments Corporation may terminate the account(s) or services for its convenience at any time without liability to Customer.

5.5.4 Effect of Termination If the account(s) or services(s) are terminated, then: (i) the rights granted by one party to the other will immediately cease; (ii) all fees owed by customer to Crystal Instruments Corporation are immediately due upon receipt of the final electronic bill; (iii) customer will delete the software, any application and any data; and (iv) upon request, each party will use commercially reasonable efforts to return or destroy all confidential information of the other party.

5.6 Customer Obligations for Cloud Services

5.6.1 Compliance Customer is solely responsible for account information and data and for making sure its usage of services is consistent with the terms of the services. Crystal Instruments Corporation reserves the right to review the data for compliance.

5.6.2 Restrictions

Customer will not, and will not allow third parties under its control to: (a) copy, modify, create a derivative work of, reverse engineer, decompile, translate, disassemble, or otherwise attempt to extract any or all of the source code of the services (except to the extent such restriction is expressly prohibited by applicable law); (b) sublicense, resell, or distribute any or all of the services; or (c) create multiple account(s) to simulate or act as a single account or otherwise access the services in a manner intended to avoid incurring fees or exceed usage limits or quotas;

5.6.3 Third Party Components

Third party components (which may include open source software) of the services may be subject to separate license agreements. To the limited extent a third party license expressly supersedes this agreement, that third party license governs customer's use of that third party component.

6. EXPORT RESTRICTIONS

The Software may be subject to the export control laws and regulations of the United States. The Licensee must comply with all domestic and international export control laws and regulations that apply to the Software. These laws include restrictions on destinations, end users, and end use.

7. THE LICENSEE'S CHOICE OF SOFTWARE

The Software is a standard product, which is delivered by Crystal Instruments with the functions that are specified in the accompanying documentation. Any assistance provided by Crystal Instruments in connection with the choice of the Software will be based on the Licensee's information about the Licensee's business provided to Crystal Instruments. The Licensee shall be responsible for both the completeness and the accuracy of such information. Crystal Instruments makes no representations or warranties as to whether the Software meets the functionality or other requirements of the Licensee and assumes no liability therefor.

8. WARRANTIES AND DISCLAIMERS

8.1 The Licensee shall be under obligation to examine and test the Software immediately after installation of the Software.

8.2 On condition that Crystal Instruments is fully paid for the Software that Customer purchased, Crystal Instruments warrants that the Software will be free of material defects for a period of 12 months after the delivery of the Software to Licensee (the "Warranty Period"). A defect in the Software shall be regarded as material if it has a material adverse effect on the functionality of the Software as a whole or if it prevents operation of the Software. Minor bugs or functions that can be improved are not viewed as a defect.

8.3 If the Licensee documents that there is a material defect in the Software, and notifies Crystal Instruments of the defect within the Warranty Period, Crystal Instruments will, at its discretion, without charge: (a) deliver a new version of the Software without the material defect, or (b) remedy the defect, or (c) provide Licensee with instructions for procedures or methods (workarounds) which result in the defect not having a significant effect on the Licensee's use of the Software. If Crystal Instruments fails to do any of the above within 30 days (or such longer period of time as is reasonably necessary given the nature of the defect), the Licensee may terminate this Agreement upon notice to Crystal Instruments, in which event Crystal Instruments will refund to Licensee a pro-rated portion of the license fee paid by Licensee for the Software (based on the portion of the Warranty Period remaining as of the date Licensee notified Crystal Instruments of the defect), provided Licensee returns to Crystal Instruments all the Licensee's versions and copies of the Software, and all manuals and accompanying documentation. This paragraph states the sole obligations of Crystal Instruments, and the sole remedy of Licensee, for defects in the Software, and the parties shall not be entitled to bring further claims against each other.

8.4 EXCEPT FOR THE EXPRESS WARRANTY IN SECTION 7.2 ABOVE, THE SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF ACCURACY, COMPATIBILITY WITH OTHER SOFTWARE OR HARDWARE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. CRYSTAL INSTRUMENTS DOES NOT WARRANT THAT THE OPERATION OF THE SOFTWARE WILL BE WITHOUT INTERRUPTIONS, DEFECT-FREE, OR ERROR-FREE OR THAT PRODUCT DEFECTS OR ERRORS CAN OR WILL BE REMEDIATED OR CORRECTED.

9. CONSENT TO USE OF DATA

Licensee agrees that Crystal Instruments and its affiliates may, through Internet connections established by the Software or otherwise, collect technical information related to Licensee's use of the Software, including but not limited to the serial numbers of Crystal Instruments hardware with which the Software is used, email addresses of users, and technical information relating to Licensee's computers, systems, application software, and peripherals. Licensee agrees that Crystal Instruments may use such information to facilitate the provision of Software updates and product support, to improve Crystal Instruments' products and/or services, or to provide products or services to Licensee. Crystal Instruments will not, however, publish or disclose such information in a form that may personally identify Licensee.

10. LIABILITY AND LIMITATION OF LIABILITY

10.1 CRYSTAL INSTRUMENTS SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING BUT NOT LIMITED TO LOSS OF EXPECTED PROFIT, LOSS OF DATA OR THEIR RECOVERY, LOSS OF GOODWILL OR ANY OTHER SIMILAR DAMAGES), UNDER ANY LEGAL THEORY, IN CONNECTION WITH THE USE OF THE SOFTWARE OR THE INABILITY TO USE THE SOFTWARE, REGARDLESS OF WHETHER CRYSTAL INSTRUMENTS HAS BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.

10.2 IN NO EVENT SHALL THE TOTAL LIABILITY OF CRYSTAL INSTRUMENTS TO LICENSEE ARISING OUT OF OR RELATING TO THE SOFTWARE EXCEED THE LICENSE FEE PAID BY LICENSEE FOR THE SOFTWARE.

10.3 Crystal Instruments shall not be liable for any errors, defects, or deficiencies which are not related to the Software, nor shall Crystal Instruments be liable for the integration or interaction between the Software and the Licensee's existing hardware and software. Crystal Instruments shall not be liable for the effect of any upgrades on existing hardware, software, or adjustments for the Software regardless of whether such adjustments were developed by Crystal Instruments.

10.4 Crystal Instruments shall have no liability of any nature relating to software or content of third parties that may be included in the Software.

10.5 The limitations in this Section 9 will apply even in the event of failure of essential purpose of any remedy.

11. GOVERNMENT USERS

The Software and related documentation are "Commercial Items", as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation", as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. The Software and documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein.

12. TERM AND TERMINATION

12.1 The term of this Agreement, and Licensee's license rights, which may be referred to the activation period of license, shall be as indicated in Licensee's order. Such term may be perpetual, or may be of limited duration in the event the Software is provided to Licensee for demonstration, evaluation or other similar purposes. Licensee acknowledges that if Licensee's rights are of limited duration, the license key provided to Licensee to enable use of the Software may cease to allow use of the Software after expiration of such activation period.

12.2 Upon termination of the Agreement for any reason, the Licensee is obliged to immediately return or destroy the Software and all copies thereof as directed by Crystal Instruments and, if requested by Crystal Instruments, to certify in writing as to the destruction or return of the Software and all copies thereof.

13. DEFAULTS

If the Licensee is in default of the Agreement, the Licensee's rights under the Agreement shall terminate with immediate effect, and the Licensee shall be under an obligation to return the Software, including any back-up copies and accompanying documentation, without a right to repayment. In addition, Crystal Instruments shall be entitled to damages for any loss, which Crystal Instruments may suffer, in accordance with the general rules of United States law, including all losses, damages, costs, expenses, etc., without any limitations, incurred or suffered by Crystal Instruments as a result of claims from any third party in relation to the Licensee's breach of the Agreement.

14. UPDATES AND RENEW

14.1 For one year after the delivery of the Software, Crystal Instruments will provide Licensee, free of charge, with any updates to the Software that Crystal Instruments makes generally available to its customers. Licensee may renew such right to receive updates, for additional periods of one year each, by paying Crystal Instruments the support renewal fee in effect at the time of such renewal. Licensee acknowledges that if Licensee elects not to renew the right to receive updates, the license key provided to Licensee to enable use of the Software may thereafter cease to allow installation and use of updates. Notwithstanding the above, Crystal Instruments may charge an additional license fee for any optional upgrades Crystal Instruments may release, which include significant new functionality and which Crystal Instruments does not make available without charge to its customers generally.

14.2 Crystal Instruments and the Licensee can agree on the other term about the period of software update after the sales.

14.3 Crystal Instruments has the rights to control the period of software update through various technical means including online activation or certain algorithm embedded in the license keys. The Licensee has no rights to reverse engineer, decompile, or disassemble the algorithm.

15. CHOICE OF LAW AND COURT OF JURISDICTION

15.1 The Agreement shall be governed by the laws of the State of California, and applicable United States federal law.

15.2 Any suit or proceeding arising out of this Agreement shall be brought only in a court located in Santa Clara County, California, and the parties submit to the exclusive jurisdiction and venue of such courts; provided, however, that Crystal Instruments may seek injunctive relief for any breach of this Agreement by Licensee in any court that would otherwise have jurisdiction over Licensee.

16. GENERAL PROVISIONS

16.1 Failure by Crystal Instruments to exercise or enforce any rights hereunder shall not be deemed to be a waiver of any such rights nor affect the exercise or enforcement thereof at any time or times thereafter.

16.2 If any provision or part of this Agreement is or is held by any court of competent jurisdiction to be unenforceable or invalid, such unenforceability or invalidity shall not affect the enforceability of any other provision.

16.3 This Agreement constitutes the entire agreement between the parties with respect to its subject matter and supersedes all prior or contemporaneous understandings regarding that subject matter. No amendment to or modification of this Agreement will be binding unless in writing and signed by an authorized officer of Crystal Instruments.

16.4 Licensee may not transfer or assign Licensee's rights under this Agreement to any third party without the prior written consent of Crystal Instruments, including by operation of law.

17. THIRD PARTY SOFTWARE LICENSE/NOTICES

Crystal Instruments Software uses a number of software products from 3rd parties that are under one of the following licenses, Apache License, GPL License, LGPL License and MIT License. Please contact Crystal Instruments to obtain the most updated list of 3rd party software that are incorporated in the Software.

License Type Definition

***Apache License**

Apache License is a free software license authored by the Apache Software Foundation (ASF). The Apache License requires preservation of the copyright notice and disclaimer. Like any free software license, the Apache License allows the user of the software the freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license, without concern for royalties.

The 2.0 version of the Apache License was approved by the ASF in 2004. The goals of this license revision have been to reduce the number of frequently asked questions, to allow the license to be reusable without modification by any project (including non-ASF projects), to allow the license to be included by reference instead of listed in every file, to clarify the license on submission of contributions, to require a patent license on contributions that necessarily infringe the contributor's own patents, and to move comments regarding Apache and other inherited attribution notices to a location outside the license terms

***GPL License**

The GNU General Public License (GNU GPL or GPL) is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project.

***LGPL License**

LGPL (formerly the GNU Library General Public License) is a free software license published by the Free Software Foundation (FSF). The LGPL allows developers and companies to use and integrate LGPL software into their own (even proprietary) software without being required (by the terms of a strong copyleft) to release the source code of their own software-parts.

***MIT License**

The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology (MIT). The MIT License is compatible with many copyleft licenses, such as the GNU General Public License (GNU GPL). Any software licensed under the terms of the MIT License can be integrated with software licensed under the terms of the GNU GPL.

--- Updated May 11, 2022